Test Suite Navigation

# of failed tests: 0/100
(0.00%)

# of skipped tests: 0/100
(0.00%)

# of passed tests: 100/100
(100.00%)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 |
| 37 | 38 | 39 | 40 | 41 | 42 |
| 43 | 44 | 45 | 46 | 47 | 48 |
| 49 | 50 | 51 | 52 | 53 | 54 |
| 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 |
| 67 | 68 | 69 | 70 | 71 | 72 |
| 73 | 74 | 75 | 76 | 77 | 78 |
| 79 | 80 | 81 | 82 | 83 | 84 |
| 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 |
| 97 | 98 | 99 | 100 | | |

Scenario: [1:7] **Fetch all person custom scripts without bearer token**

| | |
|---|---|
| **Test 1 : * def mainUrl = scriptsUrl** | **0.000014** |
| Test 2 : Given url mainUrl + '/type' | 0.000232 |
| Test 3 : And path 'person_authentication' | 0.000049 |
| Test 4 : When method GET | 0.004163 |
| Test 5 : Then status 401 | 0.000013 |
| **Test 6 : And print response** | **0.000365** |

Scenario: [2:15] **Fetch all person custom scripts**

| | |
|---|---|
| **Test 7 : * def mainUrl = scriptsUrl** | **0.000019** |
| Test 8 : Given url mainUrl + '/type' | 0.000213 |
| Test 9 : And header Authorization = 'Bearer ' + accessToken | 0.000053 |
| Test 10 : And path 'person_authentication' | 0.00004 |
| Test 11 : When method GET | 0.077179 |
| Test 12 : Then status 200 | 0.000009 |
| **Test 13 : And print response** | **0.009387** |

```
23:54:09.685 [print] {
  "entriesCount": 12,
  "entries": [
    {
      "internal": false,
      "level": 50,
      "programmingLanguage": "PYTHON",
      "description": "Twilio SMS authentication module",
      "locationType": "LDAP",
      "dn": "inum=09A0-93D6,ou=scripts,o=jans",
      "inum": "09A0-93D6",
      "script": "# Janssen Project software is available under the Apache 2.0 License (2004). See
http://www.apache.org/licenses/ for full text.\n# Copyright (c) 2020, Janssen Project\n#\n# Author: Gasmyr
Mougang\n\nfrom io.jans.service.cdi.util import CdiUtil\nfrom io.jans.as.server.security import Identity\nfrom
io.jans.model.custom.script.type.auth import PersonAuthenticationType\nfrom io.jans.as.server.service import
AuthenticationService\nfrom io.jans.as.server.service import UserService\nfrom io.jans.as.server.service import
SessionIdService\nfrom io.jans.as.server.util import ServerUtil\nfrom io.jans.util import StringHelper\nfrom
io.jans.util import ArrayHelper\nfrom java.util import Arrays\nfrom jakarta.faces.application import
FacesMessage\nfrom io.jans.jsf2.message import FacesMessages\n\nimport com.twilio.Twilio as Twilio\nimport
com.twilio.rest.api.v2010.account.Message as Message\nimport com.twilio.type.PhoneNumber as PhoneNumber\nimport
org.codehaus.jettison.json.JSONArray as JSONArray\n\n\nimport java\nimport random\nimport jarray\n\nclass
PersonAuthentication(PersonAuthenticationType):\n    def __init__(self, currentTimeMillis):\n
self.currentTimeMillis = currentTimeMillis\n        self.mobile_number = None\n        self.identity =
CdiUtil.bean(Identity)\n\n    def init(self, customScript, configurationAttributes):\n        print
\"=============================================\"\n        print \"===TWILIO SMS
INITIALIZATION=================\"\n        print \"=============================================\"\n
self.ACCOUNT_SID = None\n        self.AUTH_TOKEN = None\n        self.FROM_NUMBER = None\n\n        # Get
Custom Properties\n        try:\n            self.ACCOUNT_SID =
configurationAttributes.get(\"twilio_sid\").getValue2()\n        except:\n            print 'TwilioSMS, Missing
required configuration attribute \"twilio_sid\"'\n\n        try:\n            self.AUTH_TOKEN =
configurationAttributes.get(\"twilio_token\").getValue2()\n        except:\n            print'TwilioSMS,
Missing required configuration attribute \"twilio_token\"'\n        try:\n            self.FROM_NUMBER =
configurationAttributes.get(\"from_number\").getValue2()\n        except:\n            print'TwilioSMS, Missing
required configuration attribute \"from_number\"'\n\n        if None in (self.ACCOUNT_SID, self.AUTH_TOKEN,
self.FROM_NUMBER):\n            print \"twilio_sid, twilio_token, from_number is empty ... returning False\"\n
return False\n\n        print \"===TWILIO SMS INITIALIZATION DONE PROPERLY=====\"  \n        return True\n\n
def destroy(self, configurationAttributes):\n        print \"Twilio SMS. Destroy\"\n        print \"Twilio SMS.
Destroyed successfully\"\n        return True\n\n    def getApiVersion(self):\n        return 11\n        \n
def getAuthenticationMethodClaims(self, requestParameters):\n        return None\n        \n    def
isValidAuthenticationMethod(self, usageType, configurationAttributes):\n        return True\n\n    def
getAlternativeAuthenticationMethod(self, usageType, configurationAttributes):\n        return None\n\n    def
authenticate(self, configurationAttributes, requestParameters, step):\n        print
\"=============================================\"\n        print \"====TWILIO SMS
AUTHENCATION=================\"\n        print \"=============================================\"\n
userService = CdiUtil.bean(UserService)\n        authenticationService = CdiUtil.bean(AuthenticationService)\n
sessionIdService = CdiUtil.bean(SessionIdService)\n        facesMessages = CdiUtil.bean(FacesMessages)\n
facesMessages.setKeepMessages()\n\n        session_attributes =
self.identity.getSessionId().getSessionAttributes()\n        form_passcode =
ServerUtil.getFirstValue(requestParameters, \"passcode\")\n        form_name =
ServerUtil.getFirstValue(requestParameters, \"TwilioSmsloginForm\")\n\n        print \"TwilioSMS.
form_response_passcode: %s\" % str(form_passcode)\n\n        if step == 1:\n            print
\"=============================================\"\n            print \"=TWILIO SMS STEP 1 | Password
Authentication==\"\n            print \"=============================================\"\n
credentials = self.identity.getCredentials()\n            user_name = credentials.getUsername()\n
user_password = credentials.getPassword()\n            logged_in = False\n            if
StringHelper.isNotEmptyString(user_name) and StringHelper.isNotEmptyString(user_password):\n
logged_in = authenticationService.authenticate(user_name, user_password)\n\n            if not logged_in:\n
return False\n\n            # Get the Person's number and generate a code\n            foundUser = None\n
try:\n                foundUser = authenticationService.getAuthenticatedUser()\n            except:\n
print 'TwilioSMS, Error retrieving user %s from LDAP' % (user_name)\n                return False\n\n
try:\n                isVerified = foundUser.getAttribute(\"phoneNumberVerified\")\n                if
isVerified:\n                    self.mobile_number = foundUser.getAttribute(\"employeeNumber\")\n
if  self.mobile_number == None:\n                    self.mobile_number = foundUser.getAttribute(\"mobile\")\n
if  self.mobile_number == None:\n                    self.mobile_number =
foundUser.getAttribute(\"telephoneNumber\")\n                if  self.mobile_number == None:\n
print \"TwilioSMS, Error finding mobile number for user '%s'\" % user_name    \n                    \n
except:\n                facesMessages.add(FacesMessage.SEVERITY_ERROR, \"Failed to determine mobile phone
number\")\n                print 'TwilioSMS, Error finding mobile number for \"%s\". Exception: %s` %
(user_name, sys.exc_info()[1])`'\n                return False\n\n            # Generate Random six digit code
and store it in array\n            code = random.randint(100000, 999999)\n\n            # Get code and save it
in LDAP temporarily with special session entry\n            self.identity.setWorkingParameter(\"code\", code)\n
sessionId = sessionIdService.getSessionId() # fetch from persistence\n
sessionId.getSessionAttributes().put(\"code\", code)\n\n            try:\n
Twilio.init(self.ACCOUNT_SID, self.AUTH_TOKEN);\n                message =
Message.creator(PhoneNumber(self.mobile_number), PhoneNumber(self.FROM_NUMBER), str(code)).create();\n
print \"++++++++++++++++++++++++++++++++++++++++++++\"\n                print 'TwilioSMs, Message Sid: %s' %
(message.getSid())\n                print 'TwilioSMs, User phone: %s' % (self.mobile_number)\n
print \"++++++++++++++++++++++++++++++++++++++++++++\"\n
sessionId.getSessionAttributes().put(\"mobile_number\", self.mobile_number)\n
sessionId.getSessionAttributes().put(\"mobile\", self.mobile_number)\n
sessionIdService.updateSessionId(sessionId)\n
self.identity.setWorkingParameter(\"mobile_number\", self.mobile_number)\n
self.identity.getSessionId().getSessionAttributes().put(\"mobile_number\",self.mobile_number)\n
```

## Test Suite Navigation

# of failed tests: 0/100
(0.00%)

# of skipped tests: 0/100
(0.00%)

# of passed tests: 100/100
(100.00%)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 |
| 37 | 38 | 39 | 40 | 41 | 42 |
| 43 | 44 | 45 | 46 | 47 | 48 |
| 49 | 50 | 51 | 52 | 53 | 54 |
| 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 |
| 67 | 68 | 69 | 70 | 71 | 72 |
| 73 | 74 | 75 | 76 | 77 | 78 |
| 79 | 80 | 81 | 82 | 83 | 84 |
| 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 |
| 97 | 98 | 99 | 100 | | |

```
self.identity.setWorkingParameter(\"mobile\", self.mobile_number)\n
self.identity.getSessionId().getSessionAttributes().put(\"mobile\",self.mobile_number)\n                print
\"++++++++++++++++++++++++++++++++++++++++++++\"\n                print \"Number: %s\" %
(self.identity.getWorkingParameter(\"mobile_number\"))\n                print \"Mobile: %s\" %
(self.identity.getWorkingParameter(\"mobile\"))\n                print
\"++++++++++++++++++++++++++++++++++++++++++++\"\n                print
\"========================================\"\n                print \"===TWILIO SMS FIRST STEP DONE
PROPERLY==\"\n                print \"========================================\"\n                return True\n
except Exception, ex:\n                facesMessages.add(FacesMessage.SEVERITY_ERROR, \"Failed to send message
to mobile phone\")\n                print \"TwilioSMS. Error sending message to Twilio\"\n                print
\"TwilioSMS. Unexpected error:\", ex\n\n                return False\n        elif step == 2:\n            #
Retrieve the session attribute\n            print \"==============================================\"\n
print \"=TWILIO SMS STEP 2 | Password Authentication==\"\n            print
\"==============================================\"\n            code = session_attributes.get(\"code\")\n
print '=======> Session code is \"%s\"' % str(code)\n            sessionIdService =
CdiUtil.bean(SessionIdService)\n            sessionId = sessionIdService.getSessionId() # fetch from
persistence\n            code = sessionId.getSessionAttributes().get(\"code\")\n            print '=======>
Database code is \"%s\"' % str(code)\n            self.identity.setSessionId(sessionId)\n            print
\"==============================================\"\n            print \"TwilioSMS. Code: %s\" % str(code)\n
print \"==============================================\"\n            if code is None:\n                print
\"TwilioSMS. Failed to find previously sent code\"\n                return False\n\n            if
form_passcode is None:\n                print \"TwilioSMS. Passcode is empty\"\n                return
False\n\n            if len(form_passcode) != 6:\n                print \"TwilioSMS. Passcode from response is
not 6 digits: %s\" % form_passcode\n                return False\n\n            if form_passcode == code:\n
print \"TiwlioSMS, SUCCESS! User entered the same code!\"\n                print
\"========================================\"\n                print \"===TWILIO SMS SECOND STEP DONE
PROPERLY\"\n                print \"========================================\"\n                return True\n\n
print \"+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++\" \n            print
\"TwilioSMS. FAIL! User entered the wrong code! %s != %s\" % (form_passcode, code)\n            print
\"+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++\" \n
facesMessages.add(FacesMessage.SEVERITY_ERROR, \"Incorrect Twilio code, please try again.\")\n            print
\"==============================================\"\n            print \"===TWILIO SMS SECOND STEP FAILED:
INCORRECT CODE\"\n            print \"==============================================\"\n            return
False\n\n        print \"TwilioSMS. ERROR: step param not found or != (1|2)\"\n\n        return False\n\n
def prepareForStep(self, configurationAttributes, requestParameters, step):\n        if step == 1:\n
print \"TwilioSMS. Prepare for Step 1\"\n            return True\n        elif step == 2:\n            print
\"TwilioSMS. Prepare for Step 2\"\n            return True\n        return False\n\n    def
getExtraParametersForStep(self, configurationAttributes, step):\n        if step == 2:\n            return
Arrays.asList(\"code\")\n\n        return None\n\n    def getCountAuthenticationSteps(self,
configurationAttributes):\n        return 2\n\n    def getPageForStep(self, configurationAttributes, step):\n
if step == 2:\n            return \"/auth/otp_sms/otp_sms.xhtml\"\n\n        return \"\"\n        \n    def
getNextStep(self, configurationAttributes, requestParameters, step):\n        return -1\n\n    def
getLogoutExternalUrl(self, configurationAttributes, requestParameters):\n        print \"Get external logout
URL call\"\n        return None\n        \n    def logout(self, configurationAttributes, requestParameters):\n
return True\n",
      "enabled": false,
      "revision": 1,
      "moduleProperties": [
        {
          "value2": "interactive",
          "value1": "usage_type"
        },
        {
          "value2": "ldap",
          "value1": "location_type"
        }
      ],
      "scriptType": "PERSON_AUTHENTICATION",
      "name": "twilio_sms",
      "modified": false,
      "configurationProperties": [
        {
          "hide": false,
          "value1": "twilio_sid",
          "description": "Twilio account SID"
        },
        {
          "hide": false,
          "value1": "twilio_token",
          "description": "Twilio API token"
        },
        {
          "hide": false,
          "value1": "from_number",
          "description": "Twilio phone number with SMS capabilities"
        }
      ],
      "baseDn": "inum=09A0-93D6,ou=scripts,o=jans"
    },
    {
      "internal": false,
      "level": 45,
      "programmingLanguage": "PYTHON",
      "description": "SMPP SMS authentication module",
      "locationType": "LDAP",
      "dn": "inum=09A0-93D7,ou=scripts,o=jans",
      "inum": "09A0-93D7",
      "script": "# Janssen Project software is available under the Apache 2.0 License (2004). See
http://www.apache.org/licenses/ for full text.\n# Copyright (c) 2020, Janssen Project\n# Copyright (c) 2019,
Tele2\n\n# Author: Jose Gonzalez\n# Author: Gasmyr Mougang\n# Author: Stefan Andersson\n\nfrom java.util import
Arrays, Date\nfrom java.io import IOException\nfrom java.lang import Enum\n\nfrom io.jans.service.cdi.util
import CdiUtil\nfrom io.jans.as.server.security import Identity\nfrom io.jans.model.custom.script.type.auth
import PersonAuthenticationType\nfrom io.jans.as.server.service import AuthenticationService\nfrom
io.jans.as.server.service import UserService\nfrom io.jans.as.server.util import ServerUtil\nfrom io.jans.util
import ArrayHelper\nfrom io.jans.util import StringHelper\nfrom jakarta.faces.application import
FacesMessage\nfrom io.jans.jsf2.message import FacesMessages\n\nfrom org.jsmpp import InvalidResponseException,
PDUException\nfrom org.jsmpp.bean import Alphabet, BindType, ESMClass, GeneralDataCoding, MessageClass,
NumberingPlanIndicator, RegisteredDelivery, SMSCDeliveryReceipt, TypeOfNumber\nfrom org.jsmpp.extra import
NegativeResponseException, ResponseTimeoutException\nfrom org.jsmpp.session import BindParameter,
SMPPSession\nfrom org.jsmpp.util import AbsoluteTimeFormatter, TimeFormatter\nimport random\n\n\nclass
SmppAttributeError(Exception):\n    pass\n\n\nclass PersonAuthentication(PersonAuthenticationType):\n    def
__init__(self, currentTimeMillis):\n        self.currentTimeMillis = currentTimeMillis\n        self.identity =
CdiUtil.bean(Identity)\n\n    def get_and_parse_smpp_config(self, config, attribute, _type = None,  convert =
False, optional = False, default_desc = None):\n        try:\n            value =
config.get(attribute).getValue2()\n        except:\n            if default_desc:\n                default_desc
= \" using default '{}'\".format(default_desc)\n            else:\n                default_desc = \"\"\n\n
if optional:\n                raise SmppAttributeError(\"SMPP missing optional configuration attribute
'{}'{}\".format(attribute, default_desc))\n            else:\n                raise SmppAttributeError(\"SMPP
missing required configuration attribute '{}'\".format(attribute))\n\n        if _type and issubclass(_type,
Enum):\n            try:\n                return getattr(_type, value)\n            except AttributeError:\n
raise SmppAttributeError(\"SMPP could not find attribute '{}' in {}\".format(attribute, _type))\n\n        if
convert:\n            try:\n                value = int(value)\n            except AttributeError:\n
```

Test Suite Navigation

# of failed tests: 0/100 (0.00%)

# of skipped tests: 0/100 (0.00%)

# of passed tests: 100/100 (100.00%)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 |
| 37 | 38 | 39 | 40 | 41 | 42 |
| 43 | 44 | 45 | 46 | 47 | 48 |
| 49 | 50 | 51 | 52 | 53 | 54 |
| 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 |
| 67 | 68 | 69 | 70 | 71 | 72 |
| 73 | 74 | 75 | 76 | 77 | 78 |
| 79 | 80 | 81 | 82 | 83 | 84 |
| 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 |
| 97 | 98 | 99 | 100 | | |

```
try:\n                    value = int(value, 16)\n                except AttributeError:\n
raise SmppAttributeError(\"SMPP could not parse value '{}' of attribute '{}'\".format(value, attribute))\n\n
return value\n\n    def init(self, customScript, configurationAttributes):\n        print(\"SMPP
Initialization\")\n\n        self.TIME_FORMATTER = AbsoluteTimeFormatter()\n\n        self.SMPP_SERVER = None\n
self.SMPP_PORT = None\n\n        self.SYSTEM_ID = None\n        self.PASSWORD = None\n\n        # Setup some
good defaults for TON, NPI and source (from) address\n        # TON (Type of Number), NPI (Number Plan
Indicator)\n        self.SRC_ADDR_TON = TypeOfNumber.ALPHANUMERIC    # Alphanumeric\n        self.SRC_ADDR_NPI
= NumberingPlanIndicator.ISDN  # ISDN (E163/E164)\n        self.SRC_ADDR = \"Janssen OTP\"\n\n        # Don't
touch these unless you know what your doing, we don't handle number reformatting for\n        # any other type
than international.\n        self.DST_ADDR_TON = TypeOfNumber.INTERNATIONAL   # International\n
self.DST_ADDR_NPI = NumberingPlanIndicator.ISDN  # ISDN (E163/E164)\n\n        # Priority flag and data_coding
bits\n        self.PRIORITY_FLAG = 3  # Very Urgent (ANSI-136), Emergency (IS-95)\n
self.DATA_CODING_ALPHABET = Alphabet.ALPHA_DEFAULT  # SMS default alphabet\n
self.DATA_CODING_MESSAGE_CLASS = MessageClass.CLASS1  # EM (Mobile Equipment (mobile memory), normal
message\n\n        # Required server settings\n        try:\n            self.SMPP_SERVER =
self.get_and_parse_smpp_config(configurationAttributes, \"smpp_server\")\n        except SmppAttributeError as
e:\n            print(e)\n\n        try:\n            self.SMPP_PORT =
self.get_and_parse_smpp_config(configurationAttributes, \"smpp_port\", convert = True)\n        except
SmppAttributeError as e:\n            print(e)\n\n        if None in (self.SMPP_SERVER, self.SMPP_PORT):\n
print(\"SMPP smpp_server and smpp_port is empty, will not enable SMPP service\")\n            return False\n\n
# Optional system_id and password for bind auth\n        try:\n            self.SYSTEM_ID =
self.get_and_parse_smpp_config(configurationAttributes, \"system_id\", optional = True)\n        except
SmppAttributeError as e:\n            print(e)\n\n        try:\n            self.PASSWORD =
self.get_and_parse_smpp_config(configurationAttributes, \"password\", optional = True)\n        except
SmppAttributeError as e:\n            print(e)\n\n        if None in (self.SYSTEM_ID, self.PASSWORD):\n
print(\"SMPP Authentication disabled\")\n\n        # From number and to number settings\n        try:\n
self.SRC_ADDR_TON = self.get_and_parse_smpp_config(\n                configurationAttributes,\n
\"source_addr_ton\",\n                _type = TypeOfNumber,\n                optional = True,\n
default_desc = self.SRC_ADDR_TON\n            )\n        except SmppAttributeError as e:\n
print(e)\n\n        try:\n            self.SRC_ADDR_NPI = self.get_and_parse_smpp_config(\n
configurationAttributes,\n                \"source_addr_npi\",\n                _type =
NumberingPlanIndicator,\n                optional = True,\n                default_desc = self.SRC_ADDR_NPI\n
)\n        except SmppAttributeError as e:\n            print(e)\n\n        try:\n            self.SRC_ADDR =
self.get_and_parse_smpp_config(\n                configurationAttributes,\n                \"source_addr\",\n
optional = True,\n                default_desc = self.SRC_ADDR\n            )\n        except
SmppAttributeError as e:\n            print(e)\n\n        try:\n            self.DST_ADDR_TON =
self.get_and_parse_smpp_config(\n                configurationAttributes,\n                \"dest_addr_ton\",\n
_type = TypeOfNumber,\n                optional = True,\n                default_desc = self.DST_ADDR_TON\n
)\n        except SmppAttributeError as e:\n            print(e)\n\n        try:\n            self.DST_ADDR_NPI
= self.get_and_parse_smpp_config(\n                configurationAttributes,\n
\"dest_addr_npi\",\n                _type = NumberingPlanIndicator,\n                optional = True,\n
default_desc = self.DST_ADDR_NPI\n            )\n        except SmppAttributeError as e:\n
print(e)\n\n        # Priority flag and data coding, don't touch these unless you know what your doing...\n
try:\n            self.PRIORITY_FLAG = self.get_and_parse_smpp_config(\n
configurationAttributes,\n                \"priority_flag\",\n                convert = True,\n
optional = True,\n                default_desc = \"3 (Very Urgent, Emergency)\"\n            )\n        except
SmppAttributeError as e:\n            print(e)\n\n        try:\n            self.DATA_CODING_ALPHABET =
self.get_and_parse_smpp_config(\n                configurationAttributes,\n
\"data_coding_alphabet\",\n                _type = Alphabet,\n                optional = True,\n
default_desc = self.DATA_CODING_ALPHABET\n            )\n        except SmppAttributeError as e:\n
print(e)\n\n        try:\n            self.DATA_CODING_MESSAGE_CLASS = self.get_and_parse_smpp_config(\n
configurationAttributes,\n                \"data_coding_alphabet\",\n                _type = MessageClass,\n
optional = True,\n                default_desc = self.DATA_CODING_MESSAGE_CLASS\n            )\n        except
SmppAttributeError as e:\n            print(e)\n\n        print(\"SMPP Initialized successfully\")\n
return True\n\n    def destroy(self, configurationAttributes):\n        print(\"SMPP Destroy\")\n
print(\"SMPP Destroyed successfully\")\n        return True\n\n    def getApiVersion(self):\n        return
11\n        \n    def getAuthenticationMethodClaims(self, requestParameters):\n        return None\n        \n
def isValidAuthenticationMethod(self, usageType, configurationAttributes):\n        return True\n\n    def
getAlternativeAuthenticationMethod(self, usageType, configurationAttributes):\n        return None\n\n    def
authenticate(self, configurationAttributes, requestParameters, step):\n        userService =
CdiUtil.bean(UserService)\n        authenticationService = CdiUtil.bean(AuthenticationService)\n\n
facesMessages = CdiUtil.bean(FacesMessages)\n        facesMessages.setKeepMessages()\n\n
session_attributes = self.identity.getSessionId().getSessionAttributes()\n        form_passcode =
ServerUtil.getFirstValue(requestParameters, \"passcode\")\n\n        print(\"SMPP form_response_passcode:
{}\".format(str(form_passcode)))\n\n        if step == 1:\n            print(\"SMPP Step 1 Password
Authentication\")\n            credentials = self.identity.getCredentials()\n\n            user_name =
credentials.getUsername()\n            user_password = credentials.getPassword()\n\n            logged_in =
False\n            if StringHelper.isNotEmptyString(user_name) and
StringHelper.isNotEmptyString(user_password):\n                logged_in =
authenticationService.authenticate(user_name, user_password)\n\n            if not logged_in:\n
return False\n\n            # Get the Person's number and generate a code\n            foundUser = None\n
try:\n                foundUser = authenticationService.getAuthenticatedUser()\n            except:\n
print(\"SMPP Error retrieving user {} from LDAP\".format(user_name))\n                return False\n\n
mobile_number = None\n            try:\n                isVerified =
foundUser.getAttribute(\"phoneNumberVerified\")\n                if isVerified:\n
mobile_number = foundUser.getAttribute(\"employeeNumber\")\n                if not mobile_number:\n
mobile_number = foundUser.getAttribute(\"mobile\")\n                if not mobile_number:\n
mobile_number = foundUser.getAttribute(\"telephoneNumber\")\n                if not mobile_number:\n
facesMessages.add(FacesMessage.SEVERITY_ERROR, \"Failed to determine mobile phone number\")\n
print(\"SMPP Error finding mobile number for user '{}'\".format(user_name))\n                    return False\n
except Exception as e:\n                facesMessages.add(FacesMessage.SEVERITY_ERROR, \"Failed to determine
mobile phone number\")\n                print(\"SMPP Error finding mobile number for {}: {}\".format(user_name,
e))\n                return False\n\n            # Generate Random six digit code\n            code =
random.randint(100000, 999999)\n\n            # Get code and save it in LDAP temporarily with special session
entry\n            self.identity.setWorkingParameter(\"code\", code)\n\n
self.identity.setWorkingParameter(\"mobile_number\", mobile_number)\n
self.identity.getSessionId().getSessionAttributes().put(\"mobile_number\", mobile_number)\n            if not
self.sendMessage(mobile_number, str(code)):\n                facesMessages.add(FacesMessage.SEVERITY_ERROR,
\"Failed to send message to mobile phone\")\n                return False\n\n            return True\n
elif step == 2:\n            # Retrieve the session attribute\n            print(\"SMPP Step 2 SMS/OTP
Authentication\")\n            code = session_attributes.get(\"code\")\n            print(\"SMPP Code:
{}\".format(str(code)))\n\n            if code is None:\n                print(\"SMPP Failed to find previously
sent code\")\n                return False\n\n            if form_passcode is None:\n
print(\"SMPP Passcode is empty\")\n                return False\n\n            if len(form_passcode) != 6:\n
print(\"SMPP Passcode from response is not 6 digits: {}\".format(form_passcode))\n                return
False\n\n            if form_passcode == code:\n                print(\"SMPP SUCCESS! User entered the same
code!\")\n                return True\n\n            print(\"SMPP failed, user entered the wrong code! {} !=
{}\".format(form_passcode, code))\n            facesMessages.add(facesMessage.SEVERITY_ERROR, \"Incorrect SMS
code, please try again.\")\n            return False\n\n        print(\"SMPP ERROR: step param not found or !=
(1|2)\")\n        return False\n\n    def prepareForStep(self, configurationAttributes, requestParameters,
step):\n        if step == 1:\n            print(\"SMPP Prepare for Step 1\")\n            return True\n
elif step == 2:\n            print(\"SMPP Prepare for Step 2\")\n            return True\n\n        return
False\n\n    def getExtraParametersForStep(self, configurationAttributes, step):\n        if step == 2:\n
return Arrays.asList(\"code\")\n\n        return None\n\n    def getCountAuthenticationSteps(self,
configurationAttributes):\n        return 2\n\n    def getPageForStep(self, configurationAttributes, step):\n
if step == 2:\n            return \"/auth/otp_sms/otp_sms.xhtml\"\n\n        return \"\"\n\n    def
getNextStep(self, configurationAttributes, requestParameters, step):\n        return -1\n\n    def
getLogoutExternalUrl(self, configurationAttributes, requestParameters):\n        print \"Get external logout
URL call\"\n        return None\n\n    def logout(self, configurationAttributes, requestParameters):\n
return True\n\n    def sendMessage(self, number, code):\n        status = False\n        session =
SMPPSession()\n        session.setTransactionTimer(10000)\n\n        # We only handle international destination
number reformatting.\n        # All others may vary by configuration decisions taken on SMPP\n        # server
```

## Test Suite Navigation

# of failed tests: 0/100 (0.00%)

# of skipped tests: 0/100 (0.00%)

# of passed tests: 100/100 (100.00%)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 |
| 37 | 38 | 39 | 40 | 41 | 42 |
| 43 | 44 | 45 | 46 | 47 | 48 |
| 49 | 50 | 51 | 52 | 53 | 54 |
| 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 |
| 67 | 68 | 69 | 70 | 71 | 72 |
| 73 | 74 | 75 | 76 | 77 | 78 |
| 79 | 80 | 81 | 82 | 83 | 84 |
| 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 |
| 97 | 98 | 99 | 100 | | |

```
side which we have no clue about.\n        if self.DST_ADDR_TON == TypeOfNumber.INTERNATIONAL and
number.startswith(\"+\"):\n            number = number[1:]\n\n        try:\n            print(\"SMPP
Connecting\")\n            reference_id = session.connectAndBind(\n                self.SMPP_SERVER,\n
self.SMPP_PORT,\n                BindParameter(\n                    BindType.BIND_TX,\n
self.SYSTEM_ID,\n                    self.PASSWORD,\n                    None,\n
self.SRC_ADDR_TON,\n                    self.SRC_ADDR_NPI,\n                    None\n                )\n
)\n            print(\"SMPP Connected to server with system id {}\".format(reference_id))\n\n            try:\n
message_id = session.submitShortMessage(\n                    \"CMT\",\n
self.SRC_ADDR_TON,\n                    self.SRC_ADDR_NPI,\n                    self.SRC_ADDR,\n
self.DST_ADDR_TON,\n                    self.DST_ADDR_NPI,\n                    number,\n
ESMClass(),\n                    0,\n                    self.PRIORITY_FLAG,\n
self.TIME_FORMATTER.format(Date()),\n                    None,\n
RegisteredDelivery(SMSCDeliveryReceipt.DEFAULT),\n                    0,\n
GeneralDataCoding(\n                        self.DATA_CODING_ALPHABET,\n
self.DATA_CODING_MESSAGE_CLASS,\n                        False\n                    ),\n
0,\n                    code\n                )\n                print(\"SMPP Message '{}' sent to #{} with
message id {}\".format(code, number, message_id))\n                status = True\n            except
PDUException as e:\n                print(\"SMPP Invalid PDU parameter: {}\".format(e))\n            except
ResponseTimeoutException as e:\n                print(\"SMPP Response timeout: {}\".format(e))\n
except InvalidResponseException as e:\n                print(\"SMPP Receive invalid response: {}\".format(e))\n
except NegativeResponseException as e:\n                print(\"SMPP Receive negative response:
{}\".format(e))\n            except IOException as e:\n                print(\"SMPP IO error occured:
{}\".format(e))\n            finally:\n                session.unbindAndClose()\n        except IOException as
e:\n            print(\"SMPP Failed connect and bind to host: {}\".format(e))\n\n        return status\n",
      "enabled": false,
      "revision": 1,
      "moduleProperties": [
        {
          "value2": "interactive",
          "value1": "usage_type"
        },
        {
          "value2": "ldap",
          "value1": "location_type"
        }
      ],
      "scriptType": "PERSON_AUTHENTICATION",
      "name": "smpp",
      "modified": false,
      "configurationProperties": [
        {
          "hide": false,
          "value1": "smpp_server",
          "description": "IP or FQDN of SMPP server"
        },
        {
          "hide": false,
          "value1": "smpp_port",
          "description": "TCP port of the SMPP server"
        },
        {
          "hide": false,
          "value1": "system_id",
          "description": "Use if SMPP server requires authentication"
        },
        {
          "hide": false,
          "value1": "password",
          "description": "Use if SMPP server requires authentication"
        },
        {
          "hide": false,
          "value1": "source_addr_ton",
          "description": "Type of number, eg ALPHANUMERIC, INTERNATIONAL"
        },
        {
          "hide": false,
          "value1": "source_addr",
          "description": "From number/name"
        }
      ],
      "baseDn": "inum=09A0-93D7,ou=scripts,o=jans"
    },
    {
      "internal": false,
      "level": 30,
      "programmingLanguage": "PYTHON",
      "description": "Cert authentication module",
      "locationType": "LDAP",
      "dn": "inum=2124-0CF1,ou=scripts,o=jans",
      "inum": "2124-0CF1",
      "script": "#\n# Janssen Project software is available under the Apache 2.0 License (2004). See
http://www.apache.org/licenses/ for full text.\n# Copyright (c) 2020, Janssen Project\n#\n# Author: Yuriy
Movchan\n#\n\nfrom io.jans.service.cdi.util import CdiUtil\nfrom io.jans.model.custom.script.type.auth import
PersonAuthenticationType\nfrom jakarta.faces.context import FacesContext\nfrom io.jans.as.server.security
import Identity\nfrom io.jans.as.server.service import AuthenticationService\nfrom io.jans.as.server.service
import UserService\nfrom io.jans.util import StringHelper\nfrom io.jans.as.server.util import ServerUtil\nfrom
io.jans.as.common.service.common import EncryptionService\nfrom java.util import Arrays\nfrom
io.jans.as.common.cert.fingerprint import FingerprintHelper\nfrom io.jans.as.common.cert.validation import
GenericCertificateVerifier\nfrom io.jans.as.common.cert.validation import PathCertificateVerifier\nfrom
io.jans.as.common.cert.validation import OCSPCertificateVerifier\nfrom io.jans.as.common.cert.validation import
CRLCertificateVerifier\nfrom io.jans.as.common.cert.validation.model import ValidationStatus\nfrom
io.jans.as.server.util import CertUtil\nfrom io.jans.as.model.util import CertUtils\nfrom
io.jans.as.server.service.net import HttpService\nfrom org.apache.http.params import
CoreConnectionPNames\n\nimport sys\nimport base64\nimport urllib\n\nimport java\nimport json\n\nclass
PersonAuthentication(PersonAuthenticationType):\n    def __init__(self, currentTimeMillis):\n
self.currentTimeMillis = currentTimeMillis\n\n    def init(self, customScript, configurationAttributes):\n
print \"Cert. Initialization\"\n\n        if not
(configurationAttributes.containsKey(\"chain_cert_file_path\")):\n            print \"Cert. Initialization.
Property chain_cert_file_path is mandatory\"\n            return False\n\n        if not
(configurationAttributes.containsKey(\"map_user_cert\")):\n            print \"Cert. Initialization. Property
map_user_cert is mandatory\"\n            return False\n\n        chain_cert_file_path =
configurationAttributes.get(\"chain_cert_file_path\").getValue2()\n\n        self.chain_certs =
CertUtil.loadX509CertificateFromFile(chain_cert_file_path)\n        if self.chain_certs == None:\n
print \"Cert. Initialization. Failed to load chain certificates from '%s'\" % chain_cert_file_path\n
return False\n\n        print \"Cert. Initialization. Loaded '%d' chain certificates\" %
self.chain_certs.size()\n        \n        crl_max_response_size = 5 * 1024 * 1024  # 10Mb\n        if
configurationAttributes.containsKey(\"crl_max_response_size\"):\n            crl_max_response_size =
StringHelper.toInteger(configurationAttributes.get(\"crl_max_response_size\").getValue2(),
crl_max_response_size)\n            print \"Cert. Initialization. CRL max response size is '%d'\" %
crl_max_response_size\n\n        # Define array to order methods correctly\n        self.validator_types = [
'generic', 'path', 'ocsp', 'crl']\n        self.validators = { 'generic' : [GenericCertificateVerifier(),
```

Test Suite Navigation

# of failed tests: 0/100 (0.00%)

# of skipped tests: 0/100 (0.00%)

# of passed tests: 100/100 (100.00%)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 |
| 37 | 38 | 39 | 40 | 41 | 42 |
| 43 | 44 | 45 | 46 | 47 | 48 |
| 49 | 50 | 51 | 52 | 53 | 54 |
| 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 |
| 67 | 68 | 69 | 70 | 71 | 72 |
| 73 | 74 | 75 | 76 | 77 | 78 |
| 79 | 80 | 81 | 82 | 83 | 84 |
| 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 |
| 97 | 98 | 99 | 100 | | |

```
False],\n                            'path' : [PathCertificateVerifier(False), False],\n
'ocsp' : [OCSPCertificateVerifier(), False],\n                            'crl' :
[CRLCertificateVerifier(crl_max_response_size), False] }\n\n        for type in self.validator_types:\n
validator_param_name = \"use_%s_validator\" % type\n            if
configurationAttributes.containsKey(validator_param_name):\n                validator_status =
StringHelper.toBoolean(configurationAttributes.get(validator_param_name).getValue2(), False)\n
self.validators[type][1] = validator_status\n\n            print \"Cert. Initialization. Validation method '%s'
status: '%s'\" % (type, self.validators[type][1])\n\n        self.map_user_cert =
StringHelper.toBoolean(configurationAttributes.get(\"map_user_cert\").getValue2(), False)\n        print
\"Cert. Initialization. map_user_cert: '%s'\" % self.map_user_cert\n\n        self.enabled_recaptcha =
self.initRecaptcha(configurationAttributes)\n        print \"Cert. Initialization. enabled_recaptcha: '%s'\" %
self.enabled_recaptcha\n\n        print \"Cert. Initialized successfully\"\n\n        return True   \n\n    def
destroy(self, configurationAttributes):\n        print \"Cert. Destroy\"\n\n        for type in
self.validator_types:\n            self.validators[type][0].destroy()\n\n        print \"Cert. Destroyed
successfully\"\n\n        return True\n\n    def getApiVersion(self):\n        return 11\n\n    def
getAuthenticationMethodClaims(self, requestParameters):\n        return None\n\n    def
isValidAuthenticationMethod(self, usageType, configurationAttributes):\n        return True\n\n    def
getAlternativeAuthenticationMethod(self, usageType, configurationAttributes):\n        return None\n\n    def
authenticate(self, configurationAttributes, requestParameters, step):\n        identity =
CdiUtil.bean(Identity)\n        credentials = identity.getCredentials()\n\n        user_name =
credentials.getUsername()\n\n        userService = CdiUtil.bean(UserService)\n        authenticationService =
CdiUtil.bean(AuthenticationService)\n\n        if step == 1:\n            print \"Cert. Authenticate for step
1\"\n            login_button = ServerUtil.getFirstValue(requestParameters, \"loginForm:loginButton\")\n
if StringHelper.isEmpty(login_button):\n                print \"Cert. Authenticate for step 1. Form were
submitted incorrectly\"\n                return False\n            if self.enabled_recaptcha:\n
print \"Cert. Authenticate for step 1. Validating recaptcha response\"\n                recaptcha_response =
ServerUtil.getFirstValue(requestParameters, \"g-recaptcha-response\")\n\n                recaptcha_result =
self.validateRecaptcha(recaptcha_response)\n                print \"Cert. Authenticate for step 1.
recaptcha_result: '%s'\" % recaptcha_result\n                \n                return recaptcha_result\n\n
return True\n        elif step == 2:\n            print \"Cert. Authenticate for step 2\"\n\n            #
Validate if user selected certificate\n            cert_x509 = self.getSessionAttribute(\"cert_x509\")\n
if cert_x509 == None:\n                print \"Cert. Authenticate for step 2. User not selected any certs\"\n
identity.setWorkingParameter(\"cert_selected\", False)\n                    \n                # Return True to
inform user how to reset workflow\n                return True\n            else:\n
identity.setWorkingParameter(\"cert_selected\", True)\n            x509Certificate =
self.certFromString(cert_x509)\n\n            subjectX500Principal =
x509Certificate.getSubjectX500Principal()\n            print \"Cert. Authenticate for step 2. User selected
certificate with DN '%s'\" % subjectX500Principal\n            \n            # Validate certificates which user
selected\n            valid = self.validateCertificate(x509Certificate)\n            if not valid:\n
print \"Cert. Authenticate for step 2. Certificate DN '%s' is not valid\" % subjectX500Principal\n
identity.setWorkingParameter(\"cert_valid\", False)\n                \n                # Return True to inform
user how to reset workflow\n                return True\n\n
identity.setWorkingParameter(\"cert_valid\", True)\n            \n            # Calculate certificate
fingerprint\n            x509CertificateFingerprint = self.calculateCertificateFingerprint(x509Certificate)\n
identity.setWorkingParameter(\"cert_x509_fingerprint\", x509CertificateFingerprint)\n            print \"Cert.
Authenticate for step 2. Fingerprint is '%s' of certificate with DN '%s'\" % (x509CertificateFingerprint,
subjectX500Principal)\n            \n            # Attempt to find user by certificate fingerprint\n
cert_user_external_uid = \"cert:%s\" % x509CertificateFingerprint\n            print \"Cert. Authenticate for
step 2. Attempting to find user by jansExtUid attribute value %s\" % cert_user_external_uid\n\n
find_user_by_external_uid = userService.getUserByAttribute(\"jansExtUid\", cert_user_external_uid)\n
if find_user_by_external_uid == None:\n                print \"Cert. Authenticate for step 2. Failed to find
user\"\n                \n                if self.map_user_cert:\n                    print \"Cert.
Authenticate for step 2. Storing cert_user_external_uid for step 3\"\n
identity.setWorkingParameter(\"cert_user_external_uid\", cert_user_external_uid)\n                    return
True\n                else:\n                    print \"Cert. Authenticate for step 2. Mapping cert to user
account is not allowed\"\n                    identity.setWorkingParameter(\"cert_count_login_steps\", 2)\n
return False\n\n            foundUserName = find_user_by_external_uid.getUserId()\n            print \"Cert.
Authenticate for step 2. foundUserName: \" + foundUserName\n\n            logged_in = False\n
userService = CdiUtil.bean(UserService)\n            logged_in =
authenticationService.authenticate(foundUserName)\n        \n            print \"Cert. Authenticate for step 2.
Setting count steps to 2\"\n            identity.setWorkingParameter(\"cert_count_login_steps\", 2)\n\n
return logged_in\n        elif step == 3:\n            print \"Cert. Authenticate for step 3\"\n\n
cert_user_external_uid = self.getSessionAttribute(\"cert_user_external_uid\")\n            if
cert_user_external_uid == None:\n                print \"Cert. Authenticate for step 3. cert_user_external_uid
is empty\"\n                return False\n\n            user_password = credentials.getPassword()\n\n
logged_in = False\n            if (StringHelper.isNotEmptyString(user_name) and
StringHelper.isNotEmptyString(user_password)):\n                logged_in =
authenticationService.authenticate(user_name, user_password)\n\n            if (not logged_in):\n
return False\n\n            # Double check just to make sure. We did checking in previous step\n            #
Check if there is user which has cert_user_external_uid\n            # Avoid mapping user cert to more than one
IDP account\n            find_user_by_external_uid = userService.getUserByAttribute(\"jansExtUid\",
cert_user_external_uid)\n            if find_user_by_external_uid == None:\n                # Add
cert_user_external_uid to user's external GUID list\n                find_user_by_external_uid =
userService.addUserAttribute(user_name, \"jansExtUid\", cert_user_external_uid)\n                if
find_user_by_external_uid == None:\n                    print \"Cert. Authenticate for step 3. Failed to update
current user\"\n                    return False\n\n                return True\n            \n            return
True\n        else:\n            return False\n\n    def prepareForStep(self, configurationAttributes,
requestParameters, step):\n        print \"Cert. Prepare for step %d\" % step\n        identity =
CdiUtil.bean(Identity)\n        \n        if step == 1:\n            if self.enabled_recaptcha:\n
identity.setWorkingParameter(\"recaptcha_site_key\", self.recaptcha_creds['site_key'])\n        elif step ==
2:\n            # Store certificate in session\n            facesContext = CdiUtil.bean(FacesContext)\n
externalContext = facesContext.getExternalContext()\n            request = externalContext.getRequest()\n\n
# Try to get certificate from header X-ClientCert\n            clientCertificate =
externalContext.getRequestHeaderMap().get(\"X-ClientCert\")\n            if clientCertificate != None:\n
x509Certificate = self.certFromPemString(clientCertificate)\n
identity.setWorkingParameter(\"cert_x509\",  self.certToString(x509Certificate))\n                print \"Cert.
Prepare for step 2. Storing user certificate obtained from 'X-ClientCert' header\"\n                return
True\n\n            # Try to get certificate from attribute jakarta.servlet.request.X509Certificate\n
x509Certificates = request.getAttribute('jakarta.servlet.request.X509Certificate')\n            if
(x509Certificates != None) and (len(x509Certificates) > 0):\n
identity.setWorkingParameter(\"cert_x509\", self.certToString(x509Certificates[0]))\n                print
\"Cert. Prepare for step 2. Storing user certificate obtained from 'jakarta.servlet.request.X509Certificate'
attribute\"\n                return True\n\n        if step < 4:\n            return True\n        else:\n
return False\n\n    def getExtraParametersForStep(self, configurationAttributes, step):\n        return
Arrays.asList(\"cert_selected\", \"cert_valid\", \"cert_x509\", \"cert_x509_fingerprint\",
\"cert_count_login_steps\", \"cert_user_external_uid\")\n\n    def getCountAuthenticationSteps(self,
configurationAttributes):\n        cert_count_login_steps =
self.getSessionAttribute(\"cert_count_login_steps\")\n        if cert_count_login_steps != None:\n
return cert_count_login_steps\n        else:\n            return 3\n\n    def getPageForStep(self,
configurationAttributes, step):\n        if step == 1:\n            return \"/auth/cert/login.xhtml\"\n
if step == 2:\n            return \"/auth/cert/cert-login.xhtml\"\n        elif step == 3:\n
cert_selected = self.getSessionAttribute(\"cert_selected\")\n            if True != cert_selected:\n
return \"/auth/cert/cert-not-selected.xhtml\"\n\n            cert_valid =
self.getSessionAttribute(\"cert_valid\")\n            if True != cert_valid:\n                return
\"/auth/cert/cert-invalid.xhtml\"\n            \n            return \"/login.xhtml\"\n\n        return \"\"\n\n
def logout(self, configurationAttributes, requestParameters):\n        return True\n\n    def
processBasicAuthentication(self, credentials):\n        userService = CdiUtil.bean(UserService)\n
authenticationService = CdiUtil.bean(AuthenticationService)\n\n        user_name = credentials.getUsername()\n
user_password = credentials.getPassword()\n\n        logged_in = False\n        if
(StringHelper.isNotEmptyString(user_name) and StringHelper.isNotEmptyString(user_password)):\n
logged_in = authenticationService.authenticate(user_name, user_password)\n\n        if (not logged_in):\n
return None\n\n        find_user_by_uid = authenticationService.getAuthenticatedUser()\n        if
```

## Test Suite Navigation

# of failed tests: 0/100
(0.00%)

# of skipped tests: 0/100
(0.00%)

# of passed tests: 100/100
(100.00%)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 |
| 37 | 38 | 39 | 40 | 41 | 42 |
| 43 | 44 | 45 | 46 | 47 | 48 |
| 49 | 50 | 51 | 52 | 53 | 54 |
| 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 |
| 67 | 68 | 69 | 70 | 71 | 72 |
| 73 | 74 | 75 | 76 | 77 | 78 |
| 79 | 80 | 81 | 82 | 83 | 84 |
| 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 |
| 97 | 98 | 99 | 100 | | |

```
(find_user_by_uid == None):\n            print \"Cert. Process basic authentication. Failed to find user '%s'\"
% user_name\n            return None\n        \n        return find_user_by_uid\n\n    def
getSessionAttribute(self, attribute_name):\n        identity = CdiUtil.bean(Identity)\n\n        # Try to get
attribute value from Seam event context\n        if identity.isSetWorkingParameter(attribute_name):\n
return identity.getWorkingParameter(attribute_name)\n        \n        # Try to get attribute from persistent
session\n        session_id = identity.getSessionId()\n        if session_id == None:\n            return
None\n\n        session_attributes = session_id.getSessionAttributes()\n        if session_attributes ==
None:\n            return None\n\n        if session_attributes.containsKey(attribute_name):\n
return session_attributes.get(attribute_name)\n\n        return None\n\n    def
calculateCertificateFingerprint(self, x509Certificate):\n        print \"Cert. Calculate fingerprint for
certificate DN '%s'\" % x509Certificate.getSubjectX500Principal()\n        \n        publicKey =
x509Certificate.getPublicKey()\n        \n        # Use oxAuth implementation\n        fingerprint =
FingerprintHelper.getPublicKeySshFingerprint(publicKey)\n        \n        return fingerprint      \n\n    def
validateCertificate(self, x509Certificate):\n        subjectX500Principal =
x509Certificate.getSubjectX500Principal()\n\n        print \"Cert. Validating certificate with DN '%s'\" %
subjectX500Principal\n        \n        validation_date = java.util.Date()\n\n        for type in
self.validator_types:\n            if self.validators[type][1]:\n                result = self.validators[type]
[0].validate(x509Certificate, self.chain_certs, validation_date)\n                print \"Cert. Validate
certificate: '%s'. Validation method '%s' result: '%s'\" % (subjectX500Principal, type, result)\n
\n                if (result.getValidity() != ValidationStatus.CertificateValidity.VALID):\n
print \"Cert. Certificate: '%s' is invalid\" % subjectX500Principal\n                    return False\n
\n        return True\n\n    def certToString(self, x509Certificate):\n        if x509Certificate == None:\n
return None\n        return base64.b64encode(x509Certificate.getEncoded())\n\n    def certFromString(self,
x509CertificateEncoded):\n        x509CertificateDecoded = base64.b64decode(x509CertificateEncoded)\n
return CertUtils.x509CertificateFromBytes(x509CertificateDecoded)\n\n    def certFromPemString(self,
pemCertificate):\n        x509CertificateEncoded = pemCertificate.replace(\"-----BEGIN CERTIFICATE-----\",
\"\").replace(\"-----END CERTIFICATE-----\", \"\").strip()\n        return
self.certFromString(x509CertificateEncoded)\n\n    def initRecaptcha(self, configurationAttributes):\n
print \"Cert. Initialize recaptcha\"\n        if not
configurationAttributes.containsKey(\"credentials_file\"):\n            return False\n\n        cert_creds_file
= configurationAttributes.get(\"credentials_file\").getValue2()\n\n        # Load credentials from file\n
f = open(cert_creds_file, 'r')\n        try:\n            creds = json.loads(f.read())\n        except:\n
print \"Cert. Initialize recaptcha. Failed to load credentials from file: %s\" % cert_creds_file\n
return False\n        finally:\n            f.close()\n        \n        try:\n            recaptcha_creds =
creds[\"recaptcha\"]\n        except:\n            print \"Cert. Initialize recaptcha. Invalid credentials file
'%s' format:\" % cert_creds_file\n            return False\n        \n        self.recaptcha_creds = None\n
if recaptcha_creds[\"enabled\"]:\n            print \"Cert. Initialize recaptcha. Recaptcha is enabled\"\n\n
encryptionService = CdiUtil.bean(EncryptionService)\n\n            site_key = recaptcha_creds[\"site_key\"]\n
secret_key = recaptcha_creds[\"secret_key\"]\n\n            try:\n                site_key =
encryptionService.decrypt(site_key)\n            except:\n                # Ignore exception. Value is not
encrypted\n                print \"Cert. Initialize recaptcha. Assuming that 'site_key' in not encrypted\"\n\n
try:\n                secret_key = encryptionService.decrypt(secret_key)\n            except:\n
# Ignore exception. Value is not encrypted\n                print \"Cert. Initialize recaptcha. Assuming that
'secret_key' in not encrypted\"\n\n            \n            self.recaptcha_creds = { 'site_key' : site_key,
\"secret_key\" : secret_key }\n            print \"Cert. Initialize recaptcha. Recaptcha is configured
correctly\"\n\n            return True\n        else:\n            print \"Cert. Initialize recaptcha.
Recaptcha is disabled\"\n\n        return False\n\n    def validateRecaptcha(self, recaptcha_response):\n
print \"Cert. Validate recaptcha response\"\n\n        facesContext = CdiUtil.bean(FacesContext)\n
request = facesContext.getExternalContext().getRequest()\n\n        remoteip =
ServerUtil.getIpAddress(request)\n        print \"Cert. Validate recaptcha response. remoteip: '%s'\" %
remoteip\n\n        httpService = CdiUtil.bean(HttpService)\n\n        http_client =
httpService.getHttpsClient()\n        http_client_params = http_client.getParams()\n
http_client_params.setIntParameter(CoreConnectionPNames.CONNECTION_TIMEOUT, 15 * 1000)\n        \n
recaptcha_validation_url = \"https://www.google.com/recaptcha/api/siteverify\"\n
recaptcha_validation_request = urllib.urlencode({ \"secret\" : self.recaptcha_creds['secret_key'], \"response\"
: recaptcha_response, \"remoteip\" : remoteip })\n        recaptcha_validation_headers = { \"Content-type\" :
\"application/x-www-form-urlencoded\", \"Accept\" : \"application/json\" }\n\n        try:\n
http_service_response = httpService.executePost(http_client, recaptcha_validation_url, None,
recaptcha_validation_headers, recaptcha_validation_request)\n            http_response =
http_service_response.getHttpResponse()\n        except:\n            print \"Cert. Validate recaptcha
response. Exception: \", sys.exc_info()[1]\n            return False\n\n        try:\n            if not
httpService.isResponseStastusCodeOk(http_response):\n                print \"Cert. Validate recaptcha response.
Get invalid response from validation server: \", str(http_response.getStatusLine().getStatusCode())\n
httpService.consume(http_response)\n                return False\n    \n            response_bytes =
httpService.getResponseContent(http_response)\n            response_string =
httpService.convertEntityToString(response_bytes)\n            httpService.consume(http_response)\n
finally:\n            http_service_response.closeConnection()\n\n        if response_string == None:\n
print \"Cert. Validate recaptcha response. Get empty response from validation server\"\n            return
False\n        \n        response = json.loads(response_string)\n        \n        return
response[\"success\"]\n\n    def getNextStep(self, configurationAttributes, requestParameters, step):\n
return -1\n\n    def getLogoutExternalUrl(self, configurationAttributes, requestParameters):\n        print
\"Get external logout URL call\"\n        return None",
      "enabled": false,
      "revision": 1,
      "moduleProperties": [
        {
          "value2": "ldap",
          "value1": "location_type"
        },
        {
          "value2": "interactive",
          "value1": "usage_type"
        }
      ],
      "scriptType": "PERSON_AUTHENTICATION",
      "name": "cert",
      "modified": false,
      "configurationProperties": [
        {
          "hide": false,
          "value2": "/etc/certs/chain_cert.pem",
          "value1": "chain_cert_file_path"
        },
        {
          "hide": false,
          "value2": "/etc/certs/cert_creds.json",
          "value1": "credentials_file"
        },
        {
          "hide": false,
          "value2": "true",
          "value1": "map_user_cert"
        },
        {
          "hide": false,
          "value2": "true",
          "value1": "use_generic_validator"
        },
        {
          "hide": false,
          "value2": "true",
          "value1": "use_path_validator"
```

Test Suite Navigation

# of failed tests: 0/100
(0.00%)

# of skipped tests: 0/100
(0.00%)

# of passed tests: 100/100
(100.00%)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 |
| 37 | 38 | 39 | 40 | 41 | 42 |
| 43 | 44 | 45 | 46 | 47 | 48 |
| 49 | 50 | 51 | 52 | 53 | 54 |
| 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 |
| 67 | 68 | 69 | 70 | 71 | 72 |
| 73 | 74 | 75 | 76 | 77 | 78 |
| 79 | 80 | 81 | 82 | 83 | 84 |
| 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 |
| 97 | 98 | 99 | 100 | | |

```
        },
        {
          "hide": false,
          "value2": "false",
          "value1": "use_ocsp_validator"
        },
        {
          "hide": false,
          "value2": "false",
          "value1": "use_crl_validator"
        },
        {
          "hide": false,
          "value2": "10485760",
          "value1": "crl_max_response_size"
        }
      ],
      "baseDn": "inum=2124-0CF1,ou=scripts,o=jans"
    },
    {
      "internal": false,
      "level": 40,
      "programmingLanguage": "PYTHON",
      "description": "OTP Validation of passwords using Yubicloud authentication module",
      "locationType": "LDAP",
      "dn": "inum=24FD-B96E,ou=scripts,o=jans",
      "inum": "24FD-B96E",
      "script": "# Janssen Project software is available under the Apache License (2004). See
http://www.apache.org/licenses/ for full text.\n# Copyright (c) 2020, Janssen Project\n#\n# Author: Yuriy
Movchan, Arunmozhi\n#\n\nfrom io.jans.service.cdi.util import CdiUtil\nfrom io.jans.as.server.security import
Identity\nfrom io.jans.model.custom.script.type.auth import PersonAuthenticationType\nfrom
io.jans.as.server.service import UserService\nfrom io.jans.util import StringHelper\n\nimport java\n\nimport
urllib2\nimport urllib\nimport uuid\n\n\nclass PersonAuthentication(PersonAuthenticationType):\n    def
__init__(self, currentTimeMillis):\n        self.currentTimeMillis = currentTimeMillis\n\n    def init(self,
customScript, configurationAttributes):\n        print \"Yubicloud. Initialization\"\n\n        self.api_server
= configurationAttributes.get(\"yubicloud_uri\").getValue2()\n        self.api_key =
configurationAttributes.get(\"yubicloud_api_key\").getValue2()\n        self.client_id =
configurationAttributes.get(\"yubicloud_id\").getValue2()\n\n        return True\n\n    def destroy(self,
configurationAttributes):\n        print \"Yubicloud. Destroyed successfully\"\n        return True\n\n    def
getApiVersion(self):\n        return 11\n        \n    def getAuthenticationMethodClaims(self,
requestParameters):\n        return None\n        \n    def isValidAuthenticationMethod(self, usageType,
configurationAttributes):\n        return True\n\n    def getAlternativeAuthenticationMethod(self, usageType,
configurationAttributes):\n        return None\n\n    def authenticate(self, configurationAttributes,
requestParameters, step):\n        if (step == 1):\n            print \"Yubicloud. Authenticate for step
1\"\n\n            identity = CdiUtil.bean(Identity)\n            credentials = identity.getCredentials()\n\n
username = credentials.getUsername()\n            otp = credentials.getPassword()\n\n            # Validate otp
length\n            if len(otp) < 32 or len(otp) > 48:\n                print \"Yubicloud. Invalid OTP
length\"\n                return False\n\n            user_service = CdiUtil.bean(UserService)\n
user = user_service.getUser(username)\n\n            public_key = user.getAttribute('yubikeyId')\n\n
# Match the user with the yubikey\n            if public_key not in otp:\n                print \"Yubicloud.
Public Key not matching OTP\"\n                return False\n\n            data = \"\"\n            try:\n
nonce = str(uuid.uuid4()).replace(\"-\", \"\")\n                params = urllib.urlencode({\"id\":
self.client_id, \"otp\": otp, \"nonce\": nonce})\n                url = \"https://\" + self.api_server +
\"/wsapi/2.0/verify/?\" + params\n                f = urllib2.urlopen(url)\n                data = f.read()\n
except Exception as e:\n                print \"Yubicloud. Exception \", e\n\n            if 'status=OK' in
data:\n                user_service.authenticate(username)\n                print \"Yubicloud. Authentication
Successful\"\n                return True\n\n            print \"Yubicloud. End of Step 1. Returning False.\"\n
return False\n        else:\n            return False\n\n    def prepareForStep(self, configurationAttributes,
requestParameters, step):\n        if (step == 1):\n            print \"Yubicloud. Prepare for Step 1\"\n
return True\n        else:\n            return False\n\n    def getExtraParametersForStep(self,
configurationAttributes, step):\n        return None\n\n    def getCountAuthenticationSteps(self,
configurationAttributes):\n        return 1\n\n    def getPageForStep(self, configurationAttributes, step):\n
return \"\"\n\n    def getNextStep(self, configurationAttributes, requestParameters, step):\n        return
-1\n\n    def getLogoutExternalUrl(self, configurationAttributes, requestParameters):\n        print \"Get
external logout URL call\"\n        return None\n\n    def logout(self, configurationAttributes,
requestParameters):\n        return True\n",
      "enabled": false,
      "revision": 1,
      "moduleProperties": [
        {
          "value2": "interactive",
          "value1": "usage_type"
        },
        {
          "value2": "ldap",
          "value1": "location_type"
        }
      ],
      "scriptType": "PERSON_AUTHENTICATION",
      "name": "yubicloud",
      "modified": false,
      "configurationProperties": [
        {
          "hide": false,
          "value2": "api.yubico.com",
          "value1": "yubicloud_uri"
        },
        {
          "hide": false,
          "value1": "yubicloud_api_key"
        },
        {
          "hide": false,
          "value1": "yubicloud_id"
        }
      ],
      "baseDn": "inum=24FD-B96E,ou=scripts,o=jans"
    },
    {
      "internal": false,
      "level": 20,
      "programmingLanguage": "PYTHON",
      "description": "Basic (with user locking) authentication module",
      "locationType": "LDAP",
      "dn": "inum=4BBE-C6A8,ou=scripts,o=jans",
      "inum": "4BBE-C6A8",
      "script": "# Janssen Project software is available under the Apache 2.0 License (2004). See
http://www.apache.org/licenses/ for full text.\n# Copyright (c) 2020, Janssen Project\n#\n# Author: Yuriy
Movchan\n# Author: Gasmyr Mougang\n#\n\nfrom io.jans.service.cdi.util import CdiUtil\nfrom
io.jans.as.server.security import Identity\nfrom io.jans.model.custom.script.type.auth import
PersonAuthenticationType\nfrom io.jans.as.server.service import AuthenticationService\nfrom
io.jans.as.server.service import UserService\nfrom io.jans.service import CacheService\nfrom io.jans.util
```

Test Suite Navigation

# of failed tests: 0/100
(0.00%)

# of skipped tests: 0/100
(0.00%)

# of passed tests: 100/100
(100.00%)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 |
| 37 | 38 | 39 | 40 | 41 | 42 |
| 43 | 44 | 45 | 46 | 47 | 48 |
| 49 | 50 | 51 | 52 | 53 | 54 |
| 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 |
| 67 | 68 | 69 | 70 | 71 | 72 |
| 73 | 74 | 75 | 76 | 77 | 78 |
| 79 | 80 | 81 | 82 | 83 | 84 |
| 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 |
| 97 | 98 | 99 | 100 | | |

```
import StringHelper\nfrom io.jans.orm.exception import AuthenticationException\nfrom jakarta.faces.application
import FacesMessage\nfrom io.jans.jsf2.message import FacesMessages\nfrom java.time import LocalDateTime,
Duration\nfrom java.time.format import DateTimeFormatter\n\nimport java\nimport datetime\nimport json\n\nclass
PersonAuthentication(PersonAuthenticationType):\n    def __init__(self, currentTimeMillis):\n
self.currentTimeMillis = currentTimeMillis\n\n    def init(self, customScript,  configurationAttributes):\n
print \"Basic (lock account). Initialization\"\n\n        self.invalidLoginCountAttribute =
\"jansCountInvalidLogin\"\n        if configurationAttributes.containsKey(\"invalid_login_count_attribute\"):\n
self.invalidLoginCountAttribute = configurationAttributes.get(\"invalid_login_count_attribute\").getValue2()\n
else:\n            print \"Basic (lock account). Initialization. Using default attribute\"\n\n
self.maximumInvalidLoginAttemps = 3\n        if
configurationAttributes.containsKey(\"maximum_invalid_login_attemps\"):\n
self.maximumInvalidLoginAttemps =
StringHelper.toInteger(configurationAttributes.get(\"maximum_invalid_login_attemps\").getValue2())\n
else:\n            print \"Basic (lock account). Initialization. Using default number attempts\"\n\n
self.lockExpirationTime = 180\n        if configurationAttributes.containsKey(\"lock_expiration_time\"):\n
self.lockExpirationTime =
StringHelper.toInteger(configurationAttributes.get(\"lock_expiration_time\").getValue2())\n        else:\n
print \"Basic (lock account). Initialization. Using default lock expiration time\"\n\n\n        print \"Basic
(lock account). Initialized successfully. invalid_login_count_attribute: '%s', maximum_invalid_login_attemps:
'%s', lock_expiration_time: '%s'\" % (self.invalidLoginCountAttribute, self.maximumInvalidLoginAttemps,
self.lockExpirationTime)\n\n        return True   \n\n    def destroy(self, configurationAttributes):\n
print \"Basic (lock account). Destroy\"\n        print \"Basic (lock account). Destroyed successfully\"\n
return True\n\n    def getApiVersion(self):\n        return 11\n\n    def getAuthenticationMethodClaims(self,
requestParameters):\n        return None\n        \n    def isValidAuthenticationMethod(self, usageType,
configurationAttributes):\n        return True\n\n    def getAlternativeAuthenticationMethod(self, usageType,
configurationAttributes):\n        return None\n\n    def authenticate(self, configurationAttributes,
requestParameters, step):\n        authenticationService = CdiUtil.bean(AuthenticationService)\n\n        if
step == 1:\n            print \"Basic (lock account). Authenticate for step 1\"\n            facesMessages =
CdiUtil.bean(FacesMessages)\n            facesMessages.setKeepMessages()\n            identity =
CdiUtil.bean(Identity)\n            credentials = identity.getCredentials()\n            user_name =
credentials.getUsername()\n            user_password = credentials.getPassword()\n            cacheService =
CdiUtil.bean(CacheService)\n            userService = CdiUtil.bean(UserService)\n\n\n            logged_in =
False\n            if (StringHelper.isNotEmptyString(user_name) and
StringHelper.isNotEmptyString(user_password)):\n                try:\n                    logged_in =
authenticationService.authenticate(user_name, user_password)\n                except AuthenticationException:\n
print \"Basic (lock account). Authenticate. Failed to authenticate user '%s'\" % user_name\n\n            if
logged_in:\n                self.setUserAttributeValue(user_name, self.invalidLoginCountAttribute,
StringHelper.toString(0))\n            else:\n                countInvalidLoginArributeValue =
self.getUserAttributeValue(user_name, self.invalidLoginCountAttribute)\n                userSatus =
self.getUserAttributeValue(user_name, \"jansStatus\")\n                print \"Current user '%s' status is
'%s'\" % ( user_name, userSatus )\n\n                countInvalidLogin =
StringHelper.toInteger(countInvalidLoginArributeValue, 0)\n\n                if countInvalidLogin <
self.maximumInvalidLoginAttemps:\n                    countInvalidLogin = countInvalidLogin + 1\n
remainingAttempts = self.maximumInvalidLoginAttemps - countInvalidLogin\n\n                    print
\"Remaining login count attempts '%s' for user '%s'\" % ( remainingAttempts, user_name )\n\n
self.setUserAttributeValue(user_name, self.invalidLoginCountAttribute,
StringHelper.toString(countInvalidLogin))\n                    if remainingAttempts > 0 and userSatus ==
\"active\":\n                        facesMessages.add(FacesMessage.SEVERITY_INFO,
StringHelper.toString(remainingAttempts)+\" more attempt(s) before account is LOCKED!\")\n\n                if
(countInvalidLogin >= self.maximumInvalidLoginAttemps) and ((userSatus == None) or (userSatus ==
\"active\")):\n                    print \"Basic (lock account). Locking '%s' for '%s' seconds\" % ( user_name,
self.lockExpirationTime)\n                    self.lockUser(user_name)\n                    return False\n\n
if (countInvalidLogin >= self.maximumInvalidLoginAttemps) and userSatus == \"inactive\":\n
print \"Basic (lock account). User '%s' is locked. Checking if we can unlock him\" % user_name\n
\n                    unlock_and_authenticate = False\n\n                    object_from_store =
cacheService.get(None, \"lock_user_\" + user_name)\n                    if object_from_store == None:\n
# Object in cache was expired. We need to unlock user\n                        print \"Basic (lock account).
User locking details for user '%s' not exists\" % user_name\n                        unlock_and_authenticate =
True\n                    else:\n                        # Analyze object from cache\n
user_lock_details = json.loads(object_from_store)\n\n                        user_lock_details_locked =
user_lock_details['locked']\n                        user_lock_details_created = user_lock_details['created']\n
user_lock_details_created_date = LocalDateTime.parse(user_lock_details_created,
DateTimeFormatter.ISO_LOCAL_DATE_TIME)\n                        user_lock_details_created_diff =
Duration.between(user_lock_details_created_date, LocalDateTime.now()).getSeconds()\n
print \"Basic (lock account). Get user '%s' locking details. locked: '%s', Created: '%s', Difference in
seconds: '%s'\" % ( user_name, user_lock_details_locked, user_lock_details_created,
user_lock_details_created_diff )\n\n                        if user_lock_details_locked and
user_lock_details_created_diff >= self.lockExpirationTime:\n                            print \"Basic (lock
account). Unlocking user '%s' after lock expiration\" % user_name\n
unlock_and_authenticate = True\n\n                    if unlock_and_authenticate:\n
self.unLockUser(user_name)\n                        self.setUserAttributeValue(user_name,
self.invalidLoginCountAttribute, StringHelper.toString(0))\n                        logged_in =
authenticationService.authenticate(user_name, user_password)\n                        if not logged_in:\n
# Update number of attempts \n                            self.setUserAttributeValue(user_name,
self.invalidLoginCountAttribute, StringHelper.toString(1))\n                            if
self.maximumInvalidLoginAttemps == 1:\n                                # Lock user if maximum count login
attempts is 1 \n                                self.lockUser(user_name)\n
return False\n\n\n            return logged_in\n        else:\n            return False\n\n    def
prepareForStep(self, configurationAttributes, requestParameters, step):\n        if step == 1:\n
print \"Basic (lock account). Prepare for Step 1\"\n            return True\n        else:\n            return
False\n\n    def getExtraParametersForStep(self, configurationAttributes, step):\n        return None\n\n
def getCountAuthenticationSteps(self, configurationAttributes):\n        return 1\n\n    def
getPageForStep(self, configurationAttributes, step):\n        return \"\"\n        \n    def getNextStep(self,
configurationAttributes, requestParameters, step):\n        return -1\n\n    def getLogoutExternalUrl(self,
configurationAttributes, requestParameters):\n        print \"Get external logout URL call\"\n        return
None\n\n    def logout(self, configurationAttributes, requestParameters):\n        return True\n\n    def
getUserAttributeValue(self, user_name, attribute_name):\n        if StringHelper.isEmpty(user_name):\n
return None\n\n        userService = CdiUtil.bean(UserService)\n\n        find_user_by_uid =
userService.getUser(user_name, attribute_name)\n        if find_user_by_uid == None:\n            return
None\n\n        custom_attribute_value = userService.getCustomAttribute(find_user_by_uid, attribute_name)\n
if custom_attribute_value == None:\n            return None\n        \n        attribute_value =
custom_attribute_value.getValue()\n\n        print \"Basic (lock account). Get user attribute. User's '%s'
attribute '%s' value is '%s'\" % (user_name, attribute_name, attribute_value)\n\n        return
attribute_value\n\n    def setUserAttributeValue(self, user_name, attribute_name, attribute_value):\n        if
StringHelper.isEmpty(user_name):\n            return None\n\n        userService =
CdiUtil.bean(UserService)\n\n        find_user_by_uid = userService.getUser(user_name)\n        if
find_user_by_uid == None:\n            return None\n        \n
userService.setCustomAttribute(find_user_by_uid, attribute_name, attribute_value)\n        updated_user =
userService.updateUser(find_user_by_uid)\n\n        print \"Basic (lock account). Set user attribute. User's
'%s' attribute '%s' value is '%s'\" % (user_name, attribute_name, attribute_value)\n\n        return
updated_user\n\n    def lockUser(self, user_name):\n        if StringHelper.isEmpty(user_name):\n
return None\n\n        userService = CdiUtil.bean(UserService)\n        cacheService=
CdiUtil.bean(CacheService)\n        facesMessages = CdiUtil.bean(FacesMessages)\n
facesMessages.setKeepMessages()\n\n        find_user_by_uid = userService.getUser(user_name)\n        if
(find_user_by_uid == None):\n            return None\n\n        status_attribute_value =
userService.getCustomAttribute(find_user_by_uid, \"gluuStatus\")\n        if status_attribute_value != None:\n
user_status = status_attribute_value.getValue()\n            if StringHelper.equals(user_status,
\"inactive\"):\n                print \"Basic (lock account). Lock user. User '%s' locked already\" %
user_name\n                return\n        \n        userService.setCustomAttribute(find_user_by_uid,
\"gluuStatus\", \"inactive\")\n        updated_user = userService.updateUser(find_user_by_uid)\n\n
object_to_store = json.dumps({'locked': True, 'created': LocalDateTime.now().toString()}, separators=
(',',':'))\n\n        cacheService.put(StringHelper.toString(self.lockExpirationTime),
```

Test Suite Navigation

# of failed tests: 0/100
(0.00%)

# of skipped tests: 0/100
(0.00%)

# of passed tests: 100/100
(100.00%)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 |
| 37 | 38 | 39 | 40 | 41 | 42 |
| 43 | 44 | 45 | 46 | 47 | 48 |
| 49 | 50 | 51 | 52 | 53 | 54 |
| 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 |
| 67 | 68 | 69 | 70 | 71 | 72 |
| 73 | 74 | 75 | 76 | 77 | 78 |
| 79 | 80 | 81 | 82 | 83 | 84 |
| 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 |
| 97 | 98 | 99 | 100 | | |

```
\"lock_user_\"+user_name, object_to_store);\n        facesMessages.add(FacesMessage.SEVERITY_ERROR, \"Your
account is locked. Please try again after \" + StringHelper.toString(self.lockExpirationTime) + \" secs\")\n\n
print \"Basic (lock account). Lock user. User '%s' locked\" % user_name\n\n    def unLockUser(self,
user_name):\n        if StringHelper.isEmpty(user_name):\n            return None\n\n        userService =
CdiUtil.bean(UserService)\n        cacheService= CdiUtil.bean(CacheService)\n\n        find_user_by_uid =
userService.getUser(user_name)\n        if (find_user_by_uid == None):\n            return None\n\n
object_to_store = json.dumps({'locked': False, 'created': LocalDateTime.now().toString()}, separators=
(',',':'))\n        cacheService.put(StringHelper.toString(self.lockExpirationTime), \"lock_user_\"+user_name,
object_to_store);\n\n        userService.setCustomAttribute(find_user_by_uid, \"jansStatus\", \"active\")\n
userService.setCustomAttribute(find_user_by_uid, self.invalidLoginCountAttribute, None)\n        updated_user =
userService.updateUser(find_user_by_uid)\n\n\n        print \"Basic (lock account). Lock user. User '%s'
unlocked\" % user_name\n",
      "enabled": true,
      "revision": 1,
      "moduleProperties": [
        {
          "value2": "ldap",
          "value1": "location_type"
        },
        {
          "value2": "interactive",
          "value1": "usage_type"
        }
      ],
      "scriptType": "PERSON_AUTHENTICATION",
      "name": "basic_lock",
      "modified": false,
      "configurationProperties": [
        {
          "hide": false,
          "value2": "oxCountInvalidLogin",
          "value1": "invalid_login_count_attribute"
        },
        {
          "hide": false,
          "value2": "3",
          "value1": "maximum_invalid_login_attemps"
        },
        {
          "hide": false,
          "value2": "120",
          "value1": "lock_expiration_time"
        }
      ],
      "baseDn": "inum=4BBE-C6A8,ou=scripts,o=jans"
    },
    {
      "internal": false,
      "level": 40,
      "programmingLanguage": "PYTHON",
      "description": "HOTP/TOPT authentication module",
      "locationType": "LDAP",
      "dn": "inum=5018-D4BF,ou=scripts,o=jans",
      "inum": "5018-D4BF",
      "script": "# Janssen Project software is available under the Apache 2.0 License (2004). See
http://www.apache.org/licenses/ for full text.\n# Copyright (c) 2020, Janssen Project\n#\n# Author: Yuriy
Movchan\n#\n\n# Requires the following custom properties and values:\n#   otp_type: totp/hotp\n#   issuer:
Janssen Inc\n#   otp_conf_file: /etc/certs/otp_configuration.json\n#\n# These are non mandatory custom
properties and values:\n#   label: Janssen OTP\n#   qr_options: { width: 400, height: 400 }\n#
registration_uri: https://ce-dev.jans.org/identity/register\n\nimport jarray\nimport json\nimport sys\nfrom
com.google.common.io import BaseEncoding\nfrom com.lochbridge.oath.otp import HOTP\nfrom
com.lochbridge.oath.otp import HOTPValidator\nfrom com.lochbridge.oath.otp import HmacShaAlgorithm\nfrom
com.lochbridge.oath.otp import TOTP\nfrom com.lochbridge.oath.otp.keyprovisioning import
OTPAuthURIBuilder\nfrom com.lochbridge.oath.otp.keyprovisioning import OTPKey\nfrom
com.lochbridge.oath.otp.keyprovisioning.OTPKey import OTPType\nfrom java.security import SecureRandom\nfrom
java.util import Arrays\nfrom java.util.concurrent import TimeUnit\nfrom jakarta.faces.application import
FacesMessage\nfrom io.jans.jsf2.message import FacesMessages\nfrom io.jans.model.custom.script.type.auth import
PersonAuthenticationType\nfrom io.jans.as.server.security import Identity\nfrom io.jans.as.server.service
import AuthenticationService\nfrom io.jans.as.server.service import SessionIdService\nfrom
io.jans.as.server.service import UserService\nfrom io.jans.as.server.util import ServerUtil\nfrom
io.jans.service.cdi.util import CdiUtil\nfrom io.jans.util import StringHelper\n\nclass
PersonAuthentication(PersonAuthenticationType):\n    def __init__(self, currentTimeMillis):\n
self.currentTimeMillis = currentTimeMillis\n\n    def init(self, customScript, configurationAttributes):\n
print \"OTP. Initialization\"\n\n        if not configurationAttributes.containsKey(\"otp_type\"):\n
print \"OTP. Initialization. Property otp_type is mandatory\"\n            return False\n        self.otpType =
configurationAttributes.get(\"otp_type\").getValue2()\n\n        if not self.otpType in [\"hotp\", \"totp\"]:\n
print \"OTP. Initialization. Property value otp_type is invalid\"\n            return False\n\n        if not
configurationAttributes.containsKey(\"issuer\"):\n            print \"OTP. Initialization. Property issuer is
mandatory\"\n            return False\n        self.otpIssuer =
configurationAttributes.get(\"issuer\").getValue2()\n\n        self.customLabel = None\n        if
configurationAttributes.containsKey(\"label\"):\n            self.customLabel =
configurationAttributes.get(\"label\").getValue2()\n\n        self.customQrOptions = {}\n        if
configurationAttributes.containsKey(\"qr_options\"):\n            self.customQrOptions =
configurationAttributes.get(\"qr_options\").getValue2()\n\n        self.registrationUri = None\n        if
configurationAttributes.containsKey(\"registration_uri\"):\n            self.registrationUri =
configurationAttributes.get(\"registration_uri\").getValue2()\n\n        validOtpConfiguration =
self.loadOtpConfiguration(configurationAttributes)\n        if not validOtpConfiguration:\n            return
False\n\n        print \"OTP. Initialized successfully\"\n        return True\n\n    def destroy(self,
configurationAttributes):\n        print \"OTP. Destroy\"\n        print \"OTP. Destroyed successfully\"\n
return True\n\n    def getApiVersion(self):\n        return 11\n        \n    def
getAuthenticationMethodClaims(self, requestParameters):\n        return None\n\n    def getNextStep(self,
configurationAttributes, requestParameters, step):\n        print \"getNextStep Invoked\"\n        # If user
not pass current step change step to previous\n        identity = CdiUtil.bean(Identity)\n
retry_current_step = identity.getWorkingParameter(\"retry_current_step\")\n        if retry_current_step:\n
print \"OTP. Get next step. Retrying current step %s\" % step\n            # Remove old QR code\n
#identity.setWorkingParameter(\"super_gluu_request\", \"timeout\")\n            resultStep = step\n
return resultStep\n        return -1\n\n    def isValidAuthenticationMethod(self, usageType,
configurationAttributes):\n        return True\n\n    def getAlternativeAuthenticationMethod(self, usageType,
configurationAttributes):\n        return None\n\n    def authenticate(self, configurationAttributes,
requestParameters, step):\n        authenticationService = CdiUtil.bean(AuthenticationService)\n\n
identity = CdiUtil.bean(Identity)\n        credentials = identity.getCredentials()\n\n
self.setRequestScopedParameters(identity)\n\n        if step == 1:\n            print \"OTP. Authenticate for
step 1\"\n            authenticated_user = self.processBasicAuthentication(credentials)\n            if
authenticated_user == None:\n                return False\n\n            otp_auth_method = \"authenticate\"\n
# Uncomment this block if you need to allow user second OTP registration\n            #enrollment_mode =
ServerUtil.getFirstValue(requestParameters, \"loginForm:registerButton\")\n            #if
StringHelper.isNotEmpty(enrollment_mode):\n            #    otp_auth_method = \"enroll\"\n\n            if
otp_auth_method == \"authenticate\":\n                user_enrollments =
self.findEnrollments(authenticated_user.getUserId())\n                if len(user_enrollments) == 0:\n
otp_auth_method = \"enroll\"\n                    print \"OTP. Authenticate for step 1. There is no OTP
enrollment for user '%s'. Changing otp_auth_method to '%s'\" % (authenticated_user.getUserId(),
otp_auth_method)\n\n            if otp_auth_method == \"enroll\":\n                print \"OTP. Authenticate
```

## Test Suite Navigation

# of failed tests: 0/100
(0.00%)

# of skipped tests: 0/100
(0.00%)

# of passed tests: 100/100
(100.00%)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 |
| 37 | 38 | 39 | 40 | 41 | 42 |
| 43 | 44 | 45 | 46 | 47 | 48 |
| 49 | 50 | 51 | 52 | 53 | 54 |
| 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 |
| 67 | 68 | 69 | 70 | 71 | 72 |
| 73 | 74 | 75 | 76 | 77 | 78 |
| 79 | 80 | 81 | 82 | 83 | 84 |
| 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 |
| 97 | 98 | 99 | 100 | | |

```
for step 1. Setting count steps: '%s'\" % 3\n
identity.setWorkingParameter(\"otp_count_login_steps\", 3)\n\n            print \"OTP. Authenticate for step 1.
otp_auth_method: '%s'\" % otp_auth_method\n            identity.setWorkingParameter(\"otp_auth_method\",
otp_auth_method)\n\n            return True\n        elif step == 2:\n            print \"OTP. Authenticate for
step 2\"\n\n            authenticationService = CdiUtil.bean(AuthenticationService)\n            user =
authenticationService.getAuthenticatedUser()\n            if user == None:\n                print \"OTP.
Authenticate for step 2. Failed to determine user name\"\n                return False\n\n
session_id_validation = self.validateSessionId(identity)\n            if not session_id_validation:\n
return False\n\n            # Restore state from session\n
identity.setWorkingParameter(\"retry_current_step\", False)\n            otp_auth_method =
identity.getWorkingParameter(\"otp_auth_method\")\n            if otp_auth_method == 'enroll':\n
auth_result = ServerUtil.getFirstValue(requestParameters, \"auth_result\")\n                if not
StringHelper.isEmpty(auth_result):\n                    # defect fix #1225  - Retry the step, show QR code
again\n                    if auth_result == 'timeout':\n\t\t\t\t\t\tprint \"OTP. QR-code timeout. Authenticate
for step %s. Reinitializing current step\" %
step\n\t\t\t\t\t\tidentity.setWorkingParameter(\"retry_current_step\", True)\n\t\t\t\t\t\treturn True\n\n
print \"OTP. Authenticate for step 2. User not enrolled OTP\"\n                    return False\n\n
print \"OTP. Authenticate for step 2. Skipping this step during enrollment\"\n                return True\n\n
otp_auth_result = self.processOtpAuthentication(requestParameters, user.getUserId(), identity,
otp_auth_method)\n            print \"OTP. Authenticate for step 2. OTP authentication result: '%s'\" %
otp_auth_result\n\n            return otp_auth_result\n        elif step == 3:\n            print \"OTP.
Authenticate for step 3\"\n\n            authenticationService = CdiUtil.bean(AuthenticationService)\n
user = authenticationService.getAuthenticatedUser()\n            if user == None:\n                print \"OTP.
Authenticate for step 2. Failed to determine user name\"\n                return False\n\n
session_id_validation = self.validateSessionId(identity)\n            if not session_id_validation:\n
return False\n\n            # Restore state from session\n            otp_auth_method =
identity.getWorkingParameter(\"otp_auth_method\")\n            if otp_auth_method != 'enroll':\n
return False\n\n            otp_auth_result = self.processOtpAuthentication(requestParameters,
user.getUserId(), identity, otp_auth_method)\n            print \"OTP. Authenticate for step 3. OTP
authentication result: '%s'\" % otp_auth_result\n\n            return otp_auth_result\n        else:\n
return False\n\n    def prepareForStep(self, configurationAttributes, requestParameters, step):\n
identity = CdiUtil.bean(Identity)\n        credentials = identity.getCredentials()\n\n
self.setRequestScopedParameters(identity)\n\n        if step == 1:\n            print \"OTP. Prepare for step
1\"\n\n            return True\n        elif step == 2:\n            print \"OTP. Prepare for step 2\"\n\n
session_id_validation = self.validateSessionId(identity)\n            if not session_id_validation:\n
return False\n\n            otp_auth_method = identity.getWorkingParameter(\"otp_auth_method\")\n
print \"OTP. Prepare for step 2. otp_auth_method: '%s'\" % otp_auth_method\n\n            if otp_auth_method ==
'enroll':\n                authenticationService = CdiUtil.bean(AuthenticationService)\n                user =
authenticationService.getAuthenticatedUser()\n                if user == None:\n                    print
\"OTP. Prepare for step 2. Failed to load user enty\"\n                    return False\n\n                if
self.otpType == \"hotp\":\n                    otp_secret_key = self.generateSecretHotpKey()\n
otp_enrollment_request = self.generateHotpSecretKeyUri(otp_secret_key, self.otpIssuer,
user.getAttribute(\"displayName\"))\n                elif self.otpType == \"totp\":\n
otp_secret_key = self.generateSecretTotpKey()\n                    otp_enrollment_request =
self.generateTotpSecretKeyUri(otp_secret_key, self.otpIssuer, user.getAttribute(\"displayName\"))\n
else:\n                    print \"OTP. Prepare for step 2. Unknown OTP type: '%s'\" % self.otpType\n
return False\n\n                print \"OTP. Prepare for step 2. Prepared enrollment request for user: '%s'\" %
user.getUserId()\n                identity.setWorkingParameter(\"otp_secret_key\",
self.toBase64Url(otp_secret_key))\n                identity.setWorkingParameter(\"otp_enrollment_request\",
otp_enrollment_request)\n\n            return True\n        elif step == 3:\n            print \"OTP. Prepare
for step 3\"\n\n            session_id_validation = self.validateSessionId(identity)\n            if not
session_id_validation:\n                return False\n\n            otp_auth_method =
identity.getWorkingParameter(\"otp_auth_method\")\n            print \"OTP. Prepare for step 3.
otp_auth_method: '%s'\" % otp_auth_method\n\n            if otp_auth_method == 'enroll':\n
return True\n\n        return False\n\n    def getExtraParametersForStep(self, configurationAttributes,
step):\n        return Arrays.asList(\"otp_auth_method\", \"otp_count_login_steps\", \"otp_secret_key\",
\"otp_enrollment_request\",\"retry_current_step\")\n\n    def getCountAuthenticationSteps(self,
configurationAttributes):\n        identity = CdiUtil.bean(Identity)\n\n        if
identity.isSetWorkingParameter(\"otp_count_login_steps\"):\n            return StringHelper.toInteger(\"%s\" %
identity.getWorkingParameter(\"otp_count_login_steps\"))\n        else:\n            return 2\n\n    def
getPageForStep(self, configurationAttributes, step):\n        if step == 2:\n            identity =
CdiUtil.bean(Identity)\n\n            otp_auth_method = identity.getWorkingParameter(\"otp_auth_method\")\n
print \"OTP. Gep page for step 2. otp_auth_method: '%s'\" % otp_auth_method\n\n            if otp_auth_method
== 'enroll':\n                return \"/auth/otp/enroll.xhtml\"\n            else:\n                return
\"/auth/otp/otplogin.xhtml\"\n        elif step == 3:\n            return \"/auth/otp/otplogin.xhtml\"\n\n
return \"\"\n\n\n    def getLogoutExternalUrl(self, configurationAttributes, requestParameters):\n        print
\"Get external logout URL call\"\n        return None\n\n    def logout(self, configurationAttributes,
requestParameters):\n        return True\n\n    def setRequestScopedParameters(self, identity):\n        if
self.registrationUri != None:\n            identity.setWorkingParameter(\"external_registration_uri\",
self.registrationUri)\n\n        if self.customLabel != None:\n
identity.setWorkingParameter(\"qr_label\", self.customLabel)\n\n
identity.setWorkingParameter(\"qr_options\", self.customQrOptions)\n\n    def loadOtpConfiguration(self,
configurationAttributes):\n        print \"OTP. Load OTP configuration\"\n        if not
configurationAttributes.containsKey(\"otp_conf_file\"):\n            return False\n\n        otp_conf_file =
configurationAttributes.get(\"otp_conf_file\").getValue2()\n\n        # Load configuration from file\n        f
= open(otp_conf_file, 'r')\n        try:\n            otpConfiguration = json.loads(f.read())\n
except:\n            print \"OTP. Load OTP configuration. Failed to load configuration from file:\",
otp_conf_file\n            return False\n        finally:\n            f.close()\n\n        # Check
configuration file settings\n        try:\n            self.hotpConfiguration = otpConfiguration[\"hotp\"]\n
self.totpConfiguration = otpConfiguration[\"totp\"]\n            \n            hmacShaAlgorithm =
self.totpConfiguration[\"hmacShaAlgorithm\"]\n            hmacShaAlgorithmType = None\n\n            if
StringHelper.equalsIgnoreCase(hmacShaAlgorithm, \"sha1\"):\n                hmacShaAlgorithmType =
HmacShaAlgorithm.HMAC_SHA_1\n            elif StringHelper.equalsIgnoreCase(hmacShaAlgorithm, \"sha256\"):\n
hmacShaAlgorithmType = HmacShaAlgorithm.HMAC_SHA_256\n            elif
StringHelper.equalsIgnoreCase(hmacShaAlgorithm, \"sha512\"):\n                hmacShaAlgorithmType =
HmacShaAlgorithm.HMAC_SHA_512\n            else:\n                print \"OTP. Load OTP configuration. Invalid
TOTP HMAC SHA algorithm: '%s'\" % hmacShaAlgorithm\n                \n
self.totpConfiguration[\"hmacShaAlgorithmType\"] = hmacShaAlgorithmType\n        except:\n            print
\"OTP. Load OTP configuration. Invalid configuration file '%s' format. Exception: '%s'\" % (otp_conf_file,
sys.exc_info()[1])\n            return False\n        \n\n        return True\n\n    def
processBasicAuthentication(self, credentials):\n        userService = CdiUtil.bean(UserService)\n
authenticationService = CdiUtil.bean(AuthenticationService)\n\n        user_name = credentials.getUsername()\n
user_password = credentials.getPassword()\n\n        logged_in = False\n        if
StringHelper.isNotEmptyString(user_name) and StringHelper.isNotEmptyString(user_password):\n
logged_in = authenticationService.authenticate(user_name, user_password)\n\n        if not logged_in:\n
return None\n\n        find_user_by_uid = authenticationService.getAuthenticatedUser()\n        if
find_user_by_uid == None:\n            print \"OTP. Process basic authentication. Failed to find user '%s'\" %
user_name\n            return None\n        \n        return find_user_by_uid\n\n    def findEnrollments(self,
user_name, skipPrefix = True):\n        result = []\n\n        userService = CdiUtil.bean(UserService)\n
user = userService.getUser(user_name, \"jansExtUid\")\n        if user == None:\n            print \"OTP. Find
enrollments. Failed to find user\"\n            return result\n        \n        user_custom_ext_attribute =
userService.getCustomAttribute(user, \"jansExtUid\")\n        if user_custom_ext_attribute == None:\n
return result\n\n        otp_prefix = \"%s:\" % self.otpType\n        \n        otp_prefix_length =
len(otp_prefix) \n        for user_external_uid in user_custom_ext_attribute.getValues():\n            index =
user_external_uid.find(otp_prefix)\n            if index != -1:\n                if skipPrefix:\n
enrollment_uid = user_external_uid[otp_prefix_length:]\n                else:\n
enrollment_uid = user_external_uid\n\n                result.append(enrollment_uid)\n        \n        return
result\n\n    def validateSessionId(self, identity):\n        session =
CdiUtil.bean(SessionIdService).getSessionId()\n        if session == None:\n            print \"OTP. Validate
session id. Failed to determine session_id\"\n            return False\n\n        otp_auth_method =
identity.getWorkingParameter(\"otp_auth_method\")\n        if not otp_auth_method in ['enroll',
'authenticate']:\n            print \"OTP. Validate session id. Failed to authenticate user. otp_auth_method:
```

## Test Suite Navigation

# of failed tests: 0/100
(0.00%)

# of skipped tests: 0/100
(0.00%)

# of passed tests: 100/100
(100.00%)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 |
| 37 | 38 | 39 | 40 | 41 | 42 |
| 43 | 44 | 45 | 46 | 47 | 48 |
| 49 | 50 | 51 | 52 | 53 | 54 |
| 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 |
| 67 | 68 | 69 | 70 | 71 | 72 |
| 73 | 74 | 75 | 76 | 77 | 78 |
| 79 | 80 | 81 | 82 | 83 | 84 |
| 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 |
| 97 | 98 | 99 | 100 | | |

```
'%s'\" % otp_auth_method\n            return False\n\n        return True\n\n    def
processOtpAuthentication(self, requestParameters, user_name, identity, otp_auth_method):\n        facesMessages
= CdiUtil.bean(FacesMessages)\n        facesMessages.setKeepMessages()\n\n        userService =
CdiUtil.bean(UserService)\n\n        otpCode = ServerUtil.getFirstValue(requestParameters,
\"loginForm:otpCode\")\n        if StringHelper.isEmpty(otpCode):\n
facesMessages.add(FacesMessage.SEVERITY_ERROR, \"Failed to authenticate. OTP code is empty\")\n
print \"OTP. Process OTP authentication. otpCode is empty\"\n\n            return False\n        \n        if
otp_auth_method == \"enroll\":\n            # Get key from session\n            otp_secret_key_encoded =
identity.getWorkingParameter(\"otp_secret_key\")\n            if otp_secret_key_encoded == None:\n
print \"OTP. Process OTP authentication. OTP secret key is invalid\"\n                return False\n
\n            otp_secret_key = self.fromBase64Url(otp_secret_key_encoded)\n\n            if self.otpType ==
\"hotp\":\n                validation_result = self.validateHotpKey(otp_secret_key, 1, otpCode)\n
\n                if (validation_result != None) and validation_result[\"result\"]:\n                    print
\"OTP. Process HOTP authentication during enrollment. otpCode is valid\"\n                    # Store HOTP
Secret Key and moving factor in user entry\n                    otp_user_external_uid = \"hotp:%s;%s\" % (
otp_secret_key_encoded, validation_result[\"movingFactor\"] )\n\n                    # Add
otp_user_external_uid to user's external GUID list\n                    find_user_by_external_uid =
userService.addUserAttribute(user_name, \"jansExtUid\", otp_user_external_uid, True)\n                    if
find_user_by_external_uid != None:\n                        return True\n\n                    print \"OTP.
Process HOTP authentication during enrollment. Failed to update user entry\"\n            elif self.otpType ==
\"totp\":\n                validation_result = self.validateTotpKey(otp_secret_key, otpCode,user_name)\n
if (validation_result != None) and validation_result[\"result\"]:\n                    print \"OTP. Process
TOTP authentication during enrollment. otpCode is valid\"\n                    # Store TOTP Secret Key and
moving factor in user entry\n                    otp_user_external_uid = \"totp:%s\" %
otp_secret_key_encoded\n\n                    # Add otp_user_external_uid to user's external GUID list\n
find_user_by_external_uid = userService.addUserAttribute(user_name, \"jansExtUid\", otp_user_external_uid,
True)\n                    if find_user_by_external_uid != None:\n                        return True\n\n
print \"OTP. Process TOTP authentication during enrollment. Failed to update user entry\"\n        elif
otp_auth_method == \"authenticate\":\n            user_enrollments = self.findEnrollments(user_name)\n\n
if len(user_enrollments) == 0:\n                print \"OTP. Process OTP authentication. There is no OTP
enrollment for user '%s'\" % user_name\n                facesMessages.add(FacesMessage.SEVERITY_ERROR, \"There
is no valid OTP user enrollments\")\n                return False\n\n            if self.otpType == \"hotp\":\n
for user_enrollment in user_enrollments:\n                    user_enrollment_data =
user_enrollment.split(\";\")\n                    otp_secret_key_encoded = user_enrollment_data[0]\n\n
# Get current moving factor from user entry\n                    moving_factor =
StringHelper.toInteger(user_enrollment_data[1])\n                    otp_secret_key =
self.fromBase64Url(otp_secret_key_encoded)\n\n                    # Validate TOTP\n
validation_result = self.validateHotpKey(otp_secret_key, moving_factor, otpCode)\n                    if
(validation_result != None) and validation_result[\"result\"]:\n                        print \"OTP. Process
HOTP authentication during authentication. otpCode is valid\"\n                        otp_user_external_uid =
\"hotp:%s;%s\" % ( otp_secret_key_encoded, moving_factor )\n                        new_otp_user_external_uid =
\"hotp:%s;%s\" % ( otp_secret_key_encoded, validation_result[\"movingFactor\"] )\n    \n
# Update moving factor in user entry\n                        find_user_by_external_uid =
userService.replaceUserAttribute(user_name, \"jansExtUid\", otp_user_external_uid, new_otp_user_external_uid,
True)\n                        if find_user_by_external_uid != None:\n                            return True\n
\n                        print \"OTP. Process HOTP authentication during authentication. Failed to update user
entry\"\n            elif self.otpType == \"totp\":\n                for user_enrollment in user_enrollments:\n
otp_secret_key = self.fromBase64Url(user_enrollment)\n\n                    # Validate TOTP\n
validation_result = self.validateTotpKey(otp_secret_key, otpCode, user_name)\n                    if
(validation_result != None) and validation_result[\"result\"]:\n                        print \"OTP. Process
TOTP authentication during authentication. otpCode is valid\"\n                        return True\n\n
facesMessages.add(FacesMessage.SEVERITY_ERROR, \"Failed to authenticate. OTP code is invalid\")\n        print
\"OTP. Process OTP authentication. OTP code is invalid\"\n\n        return False\n\n    # Shared HOTP/TOTP
methods\n    def generateSecretKey(self, keyLength):\n        bytes = jarray.zeros(keyLength, \"b\")\n
secureRandom = SecureRandom()\n        secureRandom.nextBytes(bytes)\n        \n        return bytes\n    \n
# HOTP methods\n    def generateSecretHotpKey(self):\n        keyLength =
self.hotpConfiguration[\"keyLength\"]\n        \n        return self.generateSecretKey(keyLength)\n\n    def
generateHotpKey(self, secretKey, movingFactor):\n        digits = self.hotpConfiguration[\"digits\"]\n\n
hotp = HOTP.key(secretKey).digits(digits).movingFactor(movingFactor).build()\n        \n        return
hotp.value()\n\n    def validateHotpKey(self, secretKey, movingFactor, totpKey):\n        lookAheadWindow =
self.hotpConfiguration[\"lookAheadWindow\"]\n        digits = self.hotpConfiguration[\"digits\"]\n\n
htopValidationResult = HOTPValidator.lookAheadWindow(lookAheadWindow).validate(secretKey, movingFactor, digits,
totpKey)\n        if htopValidationResult.isValid():\n            return { \"result\": True, \"movingFactor\":
htopValidationResult.getNewMovingFactor() }\n\n        return { \"result\": False, \"movingFactor\": None }\n\n
def generateHotpSecretKeyUri(self, secretKey, issuer, userDisplayName):\n        digits =
self.hotpConfiguration[\"digits\"]\n\n        secretKeyBase32 = self.toBase32(secretKey)\n        otpKey =
OTPKey(secretKeyBase32, OTPType.HOTP)\n        label = issuer + \" %s\" % userDisplayName\n\n        otpAuthURI
= OTPAuthURIBuilder.fromKey(otpKey).label(label).issuer(issuer).digits(digits).build()\n\n        return
otpAuthURI.toUriString()\n\n    # TOTP methods\n    def generateSecretTotpKey(self):\n        keyLength =
self.totpConfiguration[\"keyLength\"]\n        \n        return self.generateSecretKey(keyLength)\n\n    def
generateTotpKey(self, secretKey):\n        digits = self.totpConfiguration[\"digits\"]\n        timeStep =
self.totpConfiguration[\"timeStep\"]\n        hmacShaAlgorithmType =
self.totpConfiguration[\"hmacShaAlgorithmType\"]\n\n        totp =
TOTP.key(secretKey).digits(digits).timeStep(TimeUnit.SECONDS.toMillis(timeStep)).hmacSha(hmacShaAlgorithmType).bui
        \n        return totp.value()\n\n    def validateTotpKey(self, secretKey, totpKey, user_name):\n
localTotpKey = self.generateTotpKey(secretKey)\n        cachedOTP = self.getCachedOTP(user_name)\n\n        if
StringHelper.equals(localTotpKey, totpKey) and not StringHelper.equals(localTotpKey, cachedOTP):\n
userService = CdiUtil.bean(UserService)\n            if cachedOTP is None:\n
userService.addUserAttribute(user_name, \"jansOTPCache\",localTotpKey)\n            else :\n
userService.replaceUserAttribute(user_name, \"jansOTPCache\", cachedOTP, localTotpKey)\n            print
\"OTP. Caching OTP: '%s'\" % localTotpKey\n            return { \"result\": True }\n        return {
\"result\": False }\n\t\n    def getCachedOTP(self, user_name):\n        userService =
CdiUtil.bean(UserService)\n        user = userService.getUser(user_name, \"jansOTPCache\")\n        if user is
None:\n            print \"OTP. Get Cached OTP. Failed to find OTP\"\n            return None\n
customAttribute = userService.getCustomAttribute(user, \"jansOTPCache\")\n        \n        if customAttribute
is None:\n            print \"OTP. Custom attribute is null\"\n            return None\n        user_cached_OTP
= customAttribute.getValue()\n        if user_cached_OTP is None:\n            print \"OTP. no OTP is present
in LDAP\"\n            return None\n        \n        print \"OTP.Cached OTP: '%s'\" % user_cached_OTP\n
return user_cached_OTP\n        \n    def generateTotpSecretKeyUri(self, secretKey, issuer, userDisplayName):\n
digits = self.totpConfiguration[\"digits\"]\n        timeStep = self.totpConfiguration[\"timeStep\"]\n\n
secretKeyBase32 = self.toBase32(secretKey)\n        otpKey = OTPKey(secretKeyBase32, OTPType.TOTP)\n
label = issuer + \" %s\" % userDisplayName\n\n        otpAuthURI =
OTPAuthURIBuilder.fromKey(otpKey).label(label).issuer(issuer).digits(digits).timeStep(TimeUnit.SECONDS.toMillis(ti
        return otpAuthURI.toUriString()\n\n    # Utility methods\n    def toBase32(self, bytes):\n
return BaseEncoding.base32().omitPadding().encode(bytes)\n\n    def toBase64Url(self, bytes):\n        return
BaseEncoding.base64Url().encode(bytes)\n\n    def fromBase64Url(self, chars):\n        return
BaseEncoding.base64Url().decode(chars)\n",
      "enabled": false,
      "revision": 1,
      "moduleProperties": [
        {
          "value2": "ldap",
          "value1": "location_type"
        },
        {
          "value2": "interactive",
          "value1": "usage_type"
        }
      ],
      "scriptType": "PERSON_AUTHENTICATION",
      "name": "otp",
      "modified": false,
```

## Test Suite Navigation

# of failed tests: 0/100
(0.00%)

# of skipped tests: 0/100
(0.00%)

# of passed tests: 100/100
(100.00%)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 |
| 37 | 38 | 39 | 40 | 41 | 42 |
| 43 | 44 | 45 | 46 | 47 | 48 |
| 49 | 50 | 51 | 52 | 53 | 54 |
| 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 |
| 67 | 68 | 69 | 70 | 71 | 72 |
| 73 | 74 | 75 | 76 | 77 | 78 |
| 79 | 80 | 81 | 82 | 83 | 84 |
| 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 |
| 97 | 98 | 99 | 100 | | |

```
      "configurationProperties": [
        {
          "hide": false,
          "value2": "totp",
          "value1": "otp_type"
        },
        {
          "hide": false,
          "value2": "/etc/certs/otp_configuration.json",
          "value1": "otp_conf_file"
        },
        {
          "hide": false,
          "value2": "Gluu Inc",
          "value1": "issuer"
        },
        {
          "hide": false,
          "value2": "Gluu OTP",
          "value1": "label"
        },
        {
          "hide": false,
          "value2": "{ size: 400, mSize: 0.05 }",
          "value1": "qr_options"
        },
        {
          "hide": false,
          "value2": "https://jans.server3/identity/register",
          "value1": "registration_uri"
        }
      ],
      "baseDn": "inum=5018-D4BF,ou=scripts,o=jans"
    },
    {
      "internal": false,
      "level": 50,
      "programmingLanguage": "PYTHON",
      "description": "DUO authentication module",
      "locationType": "LDAP",
      "dn": "inum=5018-F9CF,ou=scripts,o=jans",
      "inum": "5018-F9CF",
      "script": "# Janssen Project software is available under the Apache 2.0 License (2004). See
http://www.apache.org/licenses/ for full text.\n# Copyright (c) 2020, Janssen Project\n#\n# Author: Yuriy
Movchan\n#\n\nfrom io.jans.service.cdi.util import CdiUtil\nfrom io.jans.as.server.security import
Identity\nfrom io.jans.model.custom.script.type.auth import PersonAuthenticationType\nfrom
io.jans.as.server.service import AuthenticationService\nfrom io.jans.as.server.service import UserService\nfrom
io.jans.service import MailService\nfrom io.jans.util import ArrayHelper\nfrom io.jans.util import
StringHelper\nfrom java.util import Arrays\n\nimport duo_web\nimport json\n\nclass
PersonAuthentication(PersonAuthenticationType):\n    def __init__(self, currentTimeMillis):\n
self.currentTimeMillis = currentTimeMillis\n\n    def init(self, customScript, configurationAttributes):\n
print \"Duo. Initialization\"\n\n        duo_creds_file =
configurationAttributes.get(\"duo_creds_file\").getValue2()\n        # Load credentials from file\n        f =
open(duo_creds_file, 'r')\n        try:\n            creds = json.loads(f.read())\n        except:\n
print \"Duo. Initialization. Failed to load creds from file:\", duo_creds_file\n            return False\n
finally:\n            f.close()\n\n        self.ikey = str(creds[\"ikey\"])\n        self.skey =
str(creds[\"skey\"])\n        self.akey = str(creds[\"akey\"])\n\n        self.use_duo_group = False\n
if (configurationAttributes.containsKey(\"duo_group\")):\n            self.duo_group =
configurationAttributes.get(\"duo_group\").getValue2()\n            self.use_duo_group = True\n
print \"Duo. Initialization. Using Duo only if user belong to group:\", self.duo_group\n\n
self.use_audit_group = False\n        if (configurationAttributes.containsKey(\"audit_group\")):\n
self.audit_group = configurationAttributes.get(\"audit_group\").getValue2()\n\n            if (not
configurationAttributes.containsKey(\"audit_group_email\")):\n                print \"Duo. Initialization.
Property audit_group_email is not specified\"\n                return False\n\n            self.audit_email =
configurationAttributes.get(\"audit_group_email\").getValue2()\n            self.use_audit_group = True\n\n
print \"Duo. Initialization. Using audito group:\", self.audit_group\n            \n        if
(self.use_duo_group or self.use_audit_group):\n            if (not
configurationAttributes.containsKey(\"audit_attribute\")):\n                print \"Duo. Initialization.
Property audit_attribute is not specified\"\n                return False\n            else:\n
self.audit_attribute = configurationAttributes.get(\"audit_attribute\").getValue2()\n\n        print \"Duo.
Initialized successfully\"\n        return True   \n\n    def destroy(self, configurationAttributes):\n
print \"Duo. Destroy\"\n        print \"Duo. Destroyed successfully\"\n        return True\n\n    def
getApiVersion(self):\n        return 11\n        \n    def getAuthenticationMethodClaims(self,
requestParameters):\n        return None\n        \n    def isValidAuthenticationMethod(self, usageType,
configurationAttributes):\n        return True\n\n    def getAlternativeAuthenticationMethod(self, usageType,
configurationAttributes):\n        return None\n\n    def authenticate(self, configurationAttributes,
requestParameters, step):\n        duo_host = configurationAttributes.get(\"duo_host\").getValue2()\n\n
authenticationService = CdiUtil.bean(AuthenticationService)\n\n        identity = CdiUtil.bean(Identity)\n\n
if (step == 1):\n            print \"Duo. Authenticate for step 1\"\n\n            # Check if user
authenticated already in another custom script\n            user =
authenticationService.getAuthenticatedUser()\n            if user == None:\n                credentials =
identity.getCredentials()\n                user_name = credentials.getUsername()\n                user_password
= credentials.getPassword()\n    \n                logged_in = False\n                if
(StringHelper.isNotEmptyString(user_name) and StringHelper.isNotEmptyString(user_password)):\n
userService = CdiUtil.bean(UserService)\n                    logged_in =
authenticationService.authenticate(user_name, user_password)\n    \n                if (not logged_in):\n
return False\n    \n            user = authenticationService.getAuthenticatedUser()\n\n            if
(self.use_duo_group):\n                print \"Duo. Authenticate for step 1. Checking if user belong to Duo
group\"\n                is_member_duo_group = self.isUserMemberOfGroup(user, self.audit_attribute,
self.duo_group)\n                if (is_member_duo_group):\n                    print \"Duo. Authenticate for
step 1. User '\" + user.getUserId() + \"' member of Duo group\"\n                    duo_count_login_steps =
2\n                else:\n                    self.processAuditGroup(user)\n
duo_count_login_steps = 1\n\n                identity.setWorkingParameter(\"duo_count_login_steps\",
duo_count_login_steps)\n\n            return True\n        elif (step == 2):\n            print \"Duo.
Authenticate for step 2\"\n            user = authenticationService.getAuthenticatedUser()\n            if user
== None:\n                print \"Duo. Authenticate for step 2. Failed to determine user name\"\n
return False\n\n            user_name = user.getUserId()\n\n            sig_response_array =
requestParameters.get(\"sig_response\")\n            if ArrayHelper.isEmpty(sig_response_array):\n
print \"Duo. Authenticate for step 2. sig_response is empty\"\n                return False\n\n
duo_sig_response = sig_response_array[0]\n\n            print \"Duo. Authenticate for step 2. duo_sig_response:
\" + duo_sig_response\n\n            authenticated_username = duo_web.verify_response(self.ikey, self.skey,
self.akey, duo_sig_response)\n\n            print \"Duo. Authenticate for step 2. authenticated_username: \" +
authenticated_username + \", expected user_name: \" + user_name\n\n            if (not
StringHelper.equals(user_name, authenticated_username)):\n                return False\n\n
self.processAuditGroup(user)\n\n            return True\n        else:\n            return False\n\n    def
prepareForStep(self, configurationAttributes, requestParameters, step):\n        identity =
CdiUtil.bean(Identity)\n        authenticationService = CdiUtil.bean(AuthenticationService)\n\n        duo_host
= configurationAttributes.get(\"duo_host\").getValue2()\n\n        if (step == 1):\n            print \"Duo.
Prepare for step 1\"\n\n            return True\n        elif (step == 2):\n            print \"Duo. Prepare
for step 2\"\n\n            user = authenticationService.getAuthenticatedUser()\n            if (user ==
None):\n                print \"Duo. Prepare for step 2. Failed to determine user name\"\n
return False\n            user_name = user.getUserId()\n\n            duo_sig_request =
```

## Test Suite Navigation

# of failed tests: 0/100 (0.00%)

# of skipped tests: 0/100 (0.00%)

# of passed tests: 100/100 (100.00%)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 |
| 37 | 38 | 39 | 40 | 41 | 42 |
| 43 | 44 | 45 | 46 | 47 | 48 |
| 49 | 50 | 51 | 52 | 53 | 54 |
| 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 |
| 67 | 68 | 69 | 70 | 71 | 72 |
| 73 | 74 | 75 | 76 | 77 | 78 |
| 79 | 80 | 81 | 82 | 83 | 84 |
| 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 |
| 97 | 98 | 99 | 100 | | |

```
duo_web.sign_request(self.ikey, self.skey, self.akey, user_name)\n            print \"Duo. Prepare for step 2.
duo_sig_request: \" + duo_sig_request\n            \n            identity.setWorkingParameter(\"duo_host\",
duo_host)\n            identity.setWorkingParameter(\"duo_sig_request\", duo_sig_request)\n\n            return
True\n        else:\n            return False\n\n    def getExtraParametersForStep(self,
configurationAttributes, step):\n        if step == 2:\n            return
Arrays.asList(\"duo_count_login_steps\", \"cas2_user_uid\")\n\n        return None\n\n    def
getCountAuthenticationSteps(self, configurationAttributes):\n        identity = CdiUtil.bean(Identity)\n
if (identity.isSetWorkingParameter(\"duo_count_login_steps\")):\n            return
int(identity.getWorkingParameter(\"duo_count_login_steps\"))\n\n        return 2\n\n    def
getPageForStep(self, configurationAttributes, step):\n        if (step == 2):\n            return
\"/auth/duo/duologin.xhtml\"\n        return \"\"\n\n    def getNextStep(self, configurationAttributes,
requestParameters, step):\n        return -1\n\n    def getLogoutExternalUrl(self, configurationAttributes,
requestParameters):\n        print \"Get external logout URL call\"\n        return None\n\n    def
logout(self, configurationAttributes, requestParameters):\n        return True\n\n    def
isUserMemberOfGroup(self, user, attribute, group):\n        is_member = False\n        member_of_list =
user.getAttributeValues(attribute)\n        if (member_of_list != None):\n            for member_of in
member_of_list:\n                if StringHelper.equalsIgnoreCase(group, member_of) or
member_of.endswith(group):\n                    is_member = True\n                    break\n\n        return
is_member\n\n    def processAuditGroup(self, user):\n        if (self.use_audit_group):\n            is_member
= self.isUserMemberOfGroup(user, self.audit_attribute, self.audit_group)\n            if (is_member):\n
print \"Duo. Authenticate for processAuditGroup. User '\" + user.getUserId() + \"' member of audit group\"\n
print \"Duo. Authenticate for processAuditGroup. Sending e-mail about user '\" + user.getUserId() + \"' login
to\", self.audit_email\n                \n                # Send e-mail to administrator\n
user_id = user.getUserId()\n                mailService = CdiUtil.bean(MailService)\n                subject =
\"User log in: \" + user_id\n                body = \"User log in: \" + user_id\n
mailService.sendMail(self.audit_email, subject, body)\n",
      "enabled": false,
      "revision": 1,
      "moduleProperties": [
        {
          "value2": "interactive",
          "value1": "usage_type"
        },
        {
          "value2": "ldap",
          "value1": "location_type"
        }
      ],
      "scriptType": "PERSON_AUTHENTICATION",
      "name": "duo",
      "modified": false,
      "configurationProperties": [
        {
          "hide": false,
          "value2": "/etc/certs/duo_creds.json",
          "value1": "duo_creds_file"
        },
        {
          "hide": false,
          "value2": "api-random.duosecurity.com",
          "value1": "duo_host"
        }
      ],
      "baseDn": "inum=5018-F9CF,ou=scripts,o=jans"
    },
    {
      "internal": false,
      "level": 70,
      "programmingLanguage": "PYTHON",
      "description": "Fido2 authentication module",
      "locationType": "LDAP",
      "dn": "inum=8BAF-80D7,ou=scripts,o=jans",
      "inum": "8BAF-80D7",
      "script": "# Janssen Project software is available under the Apache 2.0 License (2004). See
http://www.apache.org/licenses/ for full text.\n# Copyright (c) 2020, Janssen Project\n#\n# Author: Yuriy
Movchan\n#\n\nfrom io.jans.model.custom.script.type.auth import PersonAuthenticationType\nfrom
io.jans.fido2.client import Fido2ClientFactory\nfrom io.jans.as.server.security import Identity\nfrom
io.jans.as.server.service import AuthenticationService\nfrom io.jans.as.server.service import UserService\nfrom
io.jans.as.server.service import SessionIdService\nfrom io.jans.as.server.util import ServerUtil\nfrom
io.jans.service.cdi.util import CdiUtil\nfrom io.jans.util import StringHelper\nfrom java.util import
Arrays\nfrom java.util.concurrent.locks import ReentrantLock\nfrom jakarta.ws.rs import
ClientErrorException\nfrom jakarta.ws.rs.core import Response\n\nimport java\nimport sys\nimport json\n\nclass
PersonAuthentication(PersonAuthenticationType):\n    def __init__(self, currentTimeMillis):\n
self.currentTimeMillis = currentTimeMillis\n\n    def init(self, customScript, configurationAttributes):\n
print \"Fido2. Initialization\"\n\n        if not configurationAttributes.containsKey(\"fido2_server_uri\"):\n
print \"fido2_server_uri. Initialization. Property fido2_server_uri is not specified\"\n            return
False\n\n        self.fido2_server_uri = configurationAttributes.get(\"fido2_server_uri\").getValue2()\n\n
self.fido2_domain = None\n        if configurationAttributes.containsKey(\"fido2_domain\"):\n
self.fido2_domain = configurationAttributes.get(\"fido2_domain\").getValue2()\n\n
self.metaDataLoaderLock = ReentrantLock()\n        self.metaDataConfiguration = None\n\n        print \"Fido2.
Initialized successfully\"\n        return True\n\n    def destroy(self, configurationAttributes):\n
print \"Fido2. Destroy\"\n        print \"Fido2. Destroyed successfully\"\n        return True\n\n    def
getApiVersion(self):\n        return 11\n\n    def isValidAuthenticationMethod(self, usageType,
configurationAttributes):\n        return True\n\n    def getAlternativeAuthenticationMethod(self, usageType,
configurationAttributes):\n        return None\n\n    def authenticate(self, configurationAttributes,
requestParameters, step):\n        authenticationService = CdiUtil.bean(AuthenticationService)\n\n
identity = CdiUtil.bean(Identity)\n        credentials = identity.getCredentials()\n\n        user_name =
credentials.getUsername()\n\n        if step == 1:\n            print \"Fido2. Authenticate for step 1\"\n
identity.setWorkingParameter(\"platformAuthenticatorAvailable\",ServerUtil.getFirstValue(requestParameters,
\"loginForm:platformAuthenticator\"))\n\n            user_password = credentials.getPassword()\n
logged_in = False\n            if StringHelper.isNotEmptyString(user_name) and
StringHelper.isNotEmptyString(user_password):\n                userService = CdiUtil.bean(UserService)\n
logged_in = authenticationService.authenticate(user_name, user_password)\n\n            if not logged_in:\n
return False\n\n            return True\n        elif step == 2:\n            print \"Fido2. Authenticate for
step 2\"\n\n            token_response = ServerUtil.getFirstValue(requestParameters, \"tokenResponse\")\n
if token_response == None:\n                print \"Fido2. Authenticate for step 2. tokenResponse is empty\"\n
return False\n\n            auth_method = ServerUtil.getFirstValue(requestParameters, \"authMethod\")\n
if auth_method == None:\n                print \"Fido2. Authenticate for step 2. authMethod is empty\"\n
return False\n\n            authenticationService = CdiUtil.bean(AuthenticationService)\n            user =
authenticationService.getAuthenticatedUser()\n            if user == None:\n                print \"Fido2.
Prepare for step 2. Failed to determine user name\"\n                return False\n\n            if auth_method
== 'authenticate':\n                print \"Fido2. Prepare for step 2. Call Fido2 in order to finish
authentication flow\"\n                assertionService =
Fido2ClientFactory.instance().createAssertionService(self.metaDataConfiguration)\n
assertionStatus = assertionService.verify(token_response)\n                authenticationStatusEntity =
assertionStatus.readEntity(java.lang.String)\n\n                if assertionStatus.getStatus() !=
Response.Status.OK.getStatusCode():\n                    print \"Fido2. Authenticate for step 2. Get invalid
authentication status from Fido2 server\"\n                    return False\n\n                return True\n
elif auth_method == 'enroll':\n                print \"Fido2. Prepare for step 2. Call Fido2 in order to finish
registration flow\"\n                attestationService =
Fido2ClientFactory.instance().createAttestationService(self.metaDataConfiguration)\n
attestationStatus = attestationService.verify(token_response)\n\n                if
```

## Test Suite Navigation

# of failed tests: 0/100
(0.00%)

# of skipped tests: 0/100
(0.00%)

# of passed tests: 100/100
(100.00%)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 |
| 37 | 38 | 39 | 40 | 41 | 42 |
| 43 | 44 | 45 | 46 | 47 | 48 |
| 49 | 50 | 51 | 52 | 53 | 54 |
| 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 |
| 67 | 68 | 69 | 70 | 71 | 72 |
| 73 | 74 | 75 | 76 | 77 | 78 |
| 79 | 80 | 81 | 82 | 83 | 84 |
| 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 |
| 97 | 98 | 99 | 100 | | |

```
attestationStatus.getStatus() != Response.Status.OK.getStatusCode():\n                    print \"Fido2.
Authenticate for step 2. Get invalid registration status from Fido2 server\"\n                    return
False\n\n            return True\n            else:\n                print \"Fido2. Prepare for step 2.
Authentication method is invalid\"\n                return False\n\n            return False\n        else:\n
return False\n\n    def prepareForStep(self, configurationAttributes, requestParameters, step):\n
identity = CdiUtil.bean(Identity)\n\n        if step == 1:\n            return True\n        elif step == 2:\n
print \"Fido2. Prepare for step 2\"\n\n            session = CdiUtil.bean(SessionIdService).getSessionId()\n
if session == None:\n                print \"Fido2. Prepare for step 2. Failed to determine session_id\"\n
return False\n\n            authenticationService = CdiUtil.bean(AuthenticationService)\n            user =
authenticationService.getAuthenticatedUser()\n            if user == None:\n                print \"Fido2.
Prepare for step 2. Failed to determine user name\"\n                return False\n\n            userName =
user.getUserId()\n\n            metaDataConfiguration = self.getMetaDataConfiguration()\n\n
assertionResponse = None\n            attestationResponse = None\n\n            # Check if user have registered
devices\n            count = CdiUtil.bean(UserService).countFido2RegisteredDevices(userName,
self.fido2_domain)\n            if count > 0:\n                print \"Fido2. Prepare for step 2. Call Fido2
endpoint in order to start assertion flow\"\n\n                try:\n                    assertionService =
Fido2ClientFactory.instance().createAssertionService(metaDataConfiguration)\n
assertionRequest = json.dumps({'username': userName}, separators=(',', ':'))\n
assertionResponse = assertionService.authenticate(assertionRequest).readEntity(java.lang.String)\n
# if device has only platform authenticator and assertion is expecting a security key\n                    if
\"internal\" in assertionResponse:\n
identity.setWorkingParameter(\"platformAuthenticatorAvailable\", \"true\")\n                    else:\n
identity.setWorkingParameter(\"platformAuthenticatorAvailable\", \"false\")\n\n                except
ClientErrorException, ex:\n                    print \"Fido2. Prepare for step 2. Failed to start assertion
flow. Exception:\", sys.exc_info()[1]\n                    return False\n            else:\n
print \"Fido2. Prepare for step 2. Call Fido2 endpoint in order to start attestation flow\"\n\n
try:\n                    attestationService =
Fido2ClientFactory.instance().createAttestationService(metaDataConfiguration)\n
platformAuthenticatorAvailable = identity.getWorkingParameter(\"platformAuthenticatorAvailable\") == \"true\"\n
basic_json = {'username': userName, 'displayName': userName, 'attestation' : 'direct'}\n
print \"% s\" % identity.getWorkingParameter(\"platformAuthenticatorAvailable\")\n                    if
platformAuthenticatorAvailable is True:\n                        # the reason behind userVerification =
discouraged  -->
https://chromium.googlesource.com/chromium/src/+/master/content/browser/webauth/uv_preferred.md\n
platform_json = {\"authenticatorSelection\":{\"authenticatorAttachment\":\"platform\",\"requireResidentKey\" :
\"false\", \"userVerification\" : \"discouraged\" } }\n
basic_json.update(platform_json)\n\n                    # also need to add this --> excludeCredentials :
[//registered ids]\n                    print \" basic_json %s\" % basic_json\n\n
attestationRequest = json.dumps(basic_json)\n                    #, separators=(',', ':'))\n\n
attestationResponse = attestationService.register(attestationRequest).readEntity(java.lang.String)\n
except ClientErrorException, ex:\n                    print \"Fido2. Prepare for step 2. Failed to start
attestation flow. Exception:\", sys.exc_info()[1]\n                    return False\n\n
identity.setWorkingParameter(\"fido2_assertion_request\", ServerUtil.asJson(assertionResponse))\n
identity.setWorkingParameter(\"fido2_attestation_request\", ServerUtil.asJson(attestationResponse))\n
print \"Fido2. Prepare for step 2. Successfully start flow with next requests.\\nfido2_assertion_request:
'%s'\\nfido2_attestation_request: '%s'\" % ( assertionResponse, attestationResponse )\n\n            return
True\n        elif step == 3:\n            print \"Fido2. Prepare for step 3\"\n\n            return True\n
else:\n            return False\n\n    def getExtraParametersForStep(self, configurationAttributes, step):\n
return Arrays.asList( \"platformAuthenticatorAvailable\")\n\n    def getCountAuthenticationSteps(self,
configurationAttributes):\n        return 2\n\n    def getNextStep(self, configurationAttributes,
requestParameters, step):\n        return -1\n\n    def getPageForStep(self, configurationAttributes, step):\n
if step == 1:\n            return \"/auth/fido2/step1.xhtml\"\n        elif step == 2:\n            identity =
CdiUtil.bean(Identity)\n            if identity.getWorkingParameter(\"platformAuthenticatorAvailable\") ==
\"true\":\n            return \"/auth/fido2/platform.xhtml\"\n            else:\n            return
\"/auth/fido2/secKeys.xhtml\"\n        return \"\"\n\n    def logout(self, configurationAttributes,
requestParameters):\n        return True\n\n    def getAuthenticationMethodClaims(self, requestParameters):\n
return None\n\n    def getLogoutExternalUrl(self, configurationAttributes, requestParameters):\n        print
\"Get external logout URL call\"\n        return None\n\n    def getMetaDataConfiguration(self):\n        if
self.metaDataConfiguration != None:\n            return self.metaDataConfiguration\n\n
self.metaDataLoaderLock.lock()\n        # Make sure that another thread not loaded configuration already\n
if self.metaDataConfiguration != None:\n            return self.metaDataConfiguration\n\n        try:\n
print \"Fido2. Initialization. Downloading Fido2 metadata\"\n            self.fido2_server_metadata_uri =
self.fido2_server_uri + \"/.well-known/fido2-configuration\"\n\n            metaDataConfigurationService =
Fido2ClientFactory.instance().createMetaDataConfigurationService(self.fido2_server_metadata_uri)\n\n
max_attempts = 10\n            for attempt in range(1, max_attempts + 1):\n                try:\n
self.metaDataConfiguration =
metaDataConfigurationService.getMetadataConfiguration().readEntity(java.lang.String)\n
return self.metaDataConfiguration\n                except ClientErrorException, ex:\n                    #
Detect if last try or we still get Service Unavailable HTTP error\n                    if (attempt ==
max_attempts) or (ex.getResponse().getResponseStatus() != Response.Status.SERVICE_UNAVAILABLE):\n
raise ex\n\n                    java.lang.Thread.sleep(3000)\n                    print \"Attempting to load
metadata: %d\" % attempt\n        finally:\n            self.metaDataLoaderLock.unlock()\n",
      "enabled": false,
      "revision": 1,
      "moduleProperties": [
        {
          "value2": "interactive",
          "value1": "usage_type"
        },
        {
          "value2": "ldap",
          "value1": "location_type"
        }
      ],
      "scriptType": "PERSON_AUTHENTICATION",
      "name": "fido2",
      "modified": false,
      "configurationProperties": [
        {
          "hide": false,
          "value2": "https://jans.server3",
          "value1": "fido2_server_uri"
        }
      ],
      "baseDn": "inum=8BAF-80D7,ou=scripts,o=jans"
    },
    {
      "internal": false,
      "level": 60,
      "programmingLanguage": "PYTHON",
      "description": "Super Gluu authentication module",
      "locationType": "LDAP",
      "dn": "inum=92F0-BF9E,ou=scripts,o=jans",
      "inum": "92F0-BF9E",
      "script": "# Janssen Project software is available under the Apache 2.0 License (2004). See
http://www.apache.org/licenses/ for full text.\n# Copyright (c) 2020, Janssen Project\n#\n# Author: Yuriy
Movchan\n#\n\nfrom com.google.android.gcm.server import Sender, Message\nfrom com.notnoop.apns import
APNS\nfrom java.util import Arrays\nfrom org.apache.http.params import CoreConnectionPNames\nfrom
io.jans.service.cdi.util import CdiUtil\nfrom io.jans.as.server.security import Identity\nfrom
io.jans.model.custom.script.type.auth import PersonAuthenticationType\nfrom io.jans.as.server.model.config
import ConfigurationFactory\nfrom io.jans.as.server.service import AuthenticationService\nfrom
io.jans.as.server.service import SessionIdService\nfrom io.jans.as.server.service.fido.u2f import
```

## Test Suite Navigation

# of failed tests: 0/100
(0.00%)

# of skipped tests: 0/100
(0.00%)

# of passed tests: 100/100
(100.00%)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 |
| 37 | 38 | 39 | 40 | 41 | 42 |
| 43 | 44 | 45 | 46 | 47 | 48 |
| 49 | 50 | 51 | 52 | 53 | 54 |
| 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 |
| 67 | 68 | 69 | 70 | 71 | 72 |
| 73 | 74 | 75 | 76 | 77 | 78 |
| 79 | 80 | 81 | 82 | 83 | 84 |
| 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 |
| 97 | 98 | 99 | 100 | | |

```
DeviceRegistrationService\nfrom io.jans.as.server.service.net import HttpService\nfrom io.jans.as.server.util
import ServerUtil\nfrom io.jans.util import StringHelper\nfrom io.jans.as.common.service.common import
EncryptionService\nfrom io.jans.as.server.service import UserService\nfrom io.jans.service import
MailService\nfrom io.jans.as.server.service.push.sns import PushPlatform\nfrom
io.jans.as.server.service.push.sns import PushSnsService\nfrom io.jans.notify.client import NotifyClientFactory
\nfrom java.util import Arrays, HashMap, IdentityHashMap, Date\nfrom java.time import ZonedDateTime\nfrom
java.time.format import DateTimeFormatter\n\nimport datetime\nimport urllib\n\nimport sys\nimport json\n\nclass
PersonAuthentication(PersonAuthenticationType):\n    def __init__(self, currentTimeMillis):\n
self.currentTimeMillis = currentTimeMillis\n\n    def init(self, customScript, configurationAttributes):\n
print \"Super-Gluu. Initialization\"\n\n        if not
configurationAttributes.containsKey(\"authentication_mode\"):\n            print \"Super-Gluu. Initialization.
Property authentication_mode is mandatory\"\n            return False\n\n        self.applicationId = None\n
if configurationAttributes.containsKey(\"application_id\"):\n            self.applicationId =
configurationAttributes.get(\"application_id\").getValue2()\n\n        self.registrationUri = None\n        if
configurationAttributes.containsKey(\"registration_uri\"):\n            self.registrationUri =
configurationAttributes.get(\"registration_uri\").getValue2()\n\n        authentication_mode =
configurationAttributes.get(\"authentication_mode\").getValue2()\n        if
StringHelper.isEmpty(authentication_mode):\n            print \"Super-Gluu. Initialization. Failed to determine
authentication_mode. authentication_mode configuration parameter is empty\"\n            return False\n
\n        self.oneStep = StringHelper.equalsIgnoreCase(authentication_mode, \"one_step\")\n        self.twoStep
= StringHelper.equalsIgnoreCase(authentication_mode, \"two_step\")\n\n        if not (self.oneStep or
self.twoStep):\n            print \"Super-Gluu. Initialization. Valid authentication_mode values are one_step
and two_step\"\n            return False\n        \n        self.enabledPushNotifications =
self.initPushNotificationService(configurationAttributes)\n\n        self.androidUrl = None\n        if
configurationAttributes.containsKey(\"supergluu_android_download_url\"):\n            self.androidUrl =
configurationAttributes.get(\"supergluu_android_download_url\").getValue2()\n\n        self.IOSUrl = None\n
if configurationAttributes.containsKey(\"supergluu_ios_download_url\"):\n            self.IOSUrl =
configurationAttributes.get(\"supergluu_ios_download_url\").getValue2()\n\n        self.customLabel = None\n
if configurationAttributes.containsKey(\"label\"):\n            self.customLabel =
configurationAttributes.get(\"label\").getValue2()\n\n        self.customQrOptions = {}\n        if
configurationAttributes.containsKey(\"qr_options\"):\n            self.customQrOptions =
configurationAttributes.get(\"qr_options\").getValue2()\n\n        self.use_super_gluu_group = False\n
if configurationAttributes.containsKey(\"super_gluu_group\"):\n            self.super_gluu_group =
configurationAttributes.get(\"super_gluu_group\").getValue2()\n            self.use_super_gluu_group = True\n
print \"Super-Gluu. Initialization. Using super_gluu only if user belong to group: %s\" %
self.super_gluu_group\n\n        self.use_audit_group = False\n        if
configurationAttributes.containsKey(\"audit_group\"):\n            self.audit_group =
configurationAttributes.get(\"audit_group\").getValue2()\n\n            if (not
configurationAttributes.containsKey(\"audit_group_email\")):\n                print \"Super-Gluu.
Initialization. Property audit_group_email is not specified\"\n                return False\n\n
self.audit_email = configurationAttributes.get(\"audit_group_email\").getValue2()\n
self.use_audit_group = True\n\n            print \"Super-Gluu. Initialization. Using audit group: %s\" %
self.audit_group\n            \n        if self.use_super_gluu_group or self.use_audit_group:\n            if
not configurationAttributes.containsKey(\"audit_attribute\"):\n                print \"Super-Gluu.
Initialization. Property audit_attribute is not specified\"\n                return False\n            else:\n
self.audit_attribute = configurationAttributes.get(\"audit_attribute\").getValue2()\n\n        print \"Super-
Gluu. Initialized successfully. oneStep: '%s', twoStep: '%s', pushNotifications: '%s', customLabel: '%s'\" %
(self.oneStep, self.twoStep, self.enabledPushNotifications, self.customLabel)\n\n        return True   \n\n
def destroy(self, configurationAttributes):\n        print \"Super-Gluu. Destroy\"\n\n
self.pushAndroidService = None\n        self.pushAppleService = None\n\n        print \"Super-Gluu. Destroyed
successfully\"\n        return True\n\n    def getApiVersion(self):\n        return 11\n        \n    def
getAuthenticationMethodClaims(self, requestParameters):\n        return None\n        \n    def
isValidAuthenticationMethod(self, usageType, configurationAttributes):\n        return True\n\n    def
getAlternativeAuthenticationMethod(self, usageType, configurationAttributes):\n        return None\n\n    def
authenticate(self, configurationAttributes, requestParameters, step):\n        authenticationService =
CdiUtil.bean(AuthenticationService)\n\n        identity = CdiUtil.bean(Identity)\n        credentials =
identity.getCredentials()\n\n        session_attributes = identity.getSessionId().getSessionAttributes()\n\n
client_redirect_uri = self.getApplicationUri(session_attributes)\n        if client_redirect_uri == None:\n
print \"Super-Gluu. Authenticate. redirect_uri is not set\"\n            return False\n\n
self.setRequestScopedParameters(identity, step)\n\n        # Validate form result code and initialize QR code
regeneration if needed (retry_current_step = True)\n
identity.setWorkingParameter(\"retry_current_step\", False)\n        form_auth_result =
ServerUtil.getFirstValue(requestParameters, \"auth_result\")\n        if
StringHelper.isNotEmpty(form_auth_result):\n            print \"Super-Gluu. Authenticate for step %s. Get
auth_result: '%s'\" % (step, form_auth_result)\n            if form_auth_result in ['error']:\n
return False\n\n            if form_auth_result in ['timeout']:\n                if ((step == 1) and
self.oneStep) or ((step == 2) and self.twoStep):        \n                    print \"Super-Gluu. Authenticate
for step %s. Reinitializing current step\" % step\n
identity.setWorkingParameter(\"retry_current_step\", True)\n                    return False\n\n
userService = CdiUtil.bean(UserService)\n        deviceRegistrationService =
CdiUtil.bean(DeviceRegistrationService)\n        if step == 1:\n            print \"Super-Gluu. Authenticate
for step 1\"\n\n            user_name = credentials.getUsername()\n            if self.oneStep:\n
session_device_status = self.getSessionDeviceStatus(session_attributes, user_name)\n                if
session_device_status == None:\n                    return False\n\n                u2f_device_id =
session_device_status['device_id']\n\n                validation_result =
self.validateSessionDeviceStatus(client_redirect_uri, session_device_status)\n                if
validation_result:\n                    print \"Super-Gluu. Authenticate for step 1. User successfully
authenticated with u2f_device '%s'\" % u2f_device_id\n                else:\n                    return False\n
\n                if not session_device_status['one_step']:\n                    print \"Super-Gluu.
Authenticate for step 1. u2f_device '%s' is not one step device\" % u2f_device_id\n                    return
False\n                    \n                # There are two steps only in enrollment mode\n                if
session_device_status['enroll']:\n                    return validation_result\n\n
identity.setWorkingParameter(\"super_gluu_count_login_steps\", 1)\n\n                user_inum =
session_device_status['user_inum']\n\n                u2f_device =
deviceRegistrationService.findUserDeviceRegistration(user_inum, u2f_device_id, \"jansId\")\n                if
u2f_device == None:\n                    print \"Super-Gluu. Authenticate for step 1. Failed to load u2f_device
'%s'\" % u2f_device_id\n                    return False\n\n                logged_in =
authenticationService.authenticate(user_name)\n                if not logged_in:\n                    print
\"Super-Gluu. Authenticate for step 1. Failed to authenticate user '%s'\" % user_name\n
return False\n\n                print \"Super-Gluu. Authenticate for step 1. User '%s' successfully
authenticated with u2f_device '%s'\" % (user_name, u2f_device_id)\n                \n                return
True\n            elif self.twoStep:\n                authenticated_user =
self.processBasicAuthentication(credentials)\n                if authenticated_user == None:\n
return False\n\n                if (self.use_super_gluu_group):\n                    print \"Super-Gluu.
Authenticate for step 1. Checking if user belong to super_gluu group\"\n
is_member_super_gluu_group = self.isUserMemberOfGroup(authenticated_user, self.audit_attribute,
self.super_gluu_group)\n                    if (is_member_super_gluu_group):\n                        print
\"Super-Gluu. Authenticate for step 1. User '%s' member of super_gluu group\" %
authenticated_user.getUserId()\n                        super_gluu_count_login_steps = 2\n
else:\n                        if self.use_audit_group:\n
self.processAuditGroup(authenticated_user, self.audit_attribute, self.audit_group)\n
super_gluu_count_login_steps = 1\n    \n
identity.setWorkingParameter(\"super_gluu_count_login_steps\", super_gluu_count_login_steps)\n
\n                    if super_gluu_count_login_steps == 1:\n                        return True\n    \n
auth_method = 'authenticate'\n                enrollment_mode = ServerUtil.getFirstValue(requestParameters,
\"loginForm:registerButton\")\n                if StringHelper.isNotEmpty(enrollment_mode):\n
auth_method = 'enroll'\n                \n                if auth_method == 'authenticate':\n
user_inum = userService.getUserInum(authenticated_user)\n                    u2f_devices_list =
deviceRegistrationService.findUserDeviceRegistrations(user_inum, client_redirect_uri, \"jansId\")\n
if u2f_devices_list.size() == 0:\n                        auth_method = 'enroll'\n                        print
\"Super-Gluu. Authenticate for step 1. There is no U2F '%s' user devices associated with application '%s'.
Changing auth_method to '%s'\" % (user_name, client_redirect_uri, auth_method)\n    \n                    print
```

## Test Suite Navigation

# of failed tests: 0/100
(0.00%)

# of skipped tests: 0/100
(0.00%)

# of passed tests: 100/100
(100.00%)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 |
| 37 | 38 | 39 | 40 | 41 | 42 |
| 43 | 44 | 45 | 46 | 47 | 48 |
| 49 | 50 | 51 | 52 | 53 | 54 |
| 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 |
| 67 | 68 | 69 | 70 | 71 | 72 |
| 73 | 74 | 75 | 76 | 77 | 78 |
| 79 | 80 | 81 | 82 | 83 | 84 |
| 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 |
| 97 | 98 | 99 | 100 | | |

```
\"Super-Gluu. Authenticate for step 1. auth_method: '%s'\" % auth_method\n               \n
identity.setWorkingParameter(\"super_gluu_auth_method\", auth_method)\n\n               return True\n\n
return False\n        elif step == 2:\n            print \"Super-Gluu. Authenticate for step 2\"\n\n
user = authenticationService.getAuthenticatedUser()\n            if (user == None):\n                print
\"Super-Gluu. Authenticate for step 2. Failed to determine user name\"\n                return False\n
user_name = user.getUserId()\n\n            session_attributes =
identity.getSessionId().getSessionAttributes()\n\n            session_device_status =
self.getSessionDeviceStatus(session_attributes, user_name)\n            if session_device_status == None:\n
return False\n\n            u2f_device_id = session_device_status['device_id']\n\n            # There are two
steps only in enrollment mode\n            if self.oneStep and session_device_status['enroll']:\n
authenticated_user = self.processBasicAuthentication(credentials)\n                if authenticated_user ==
None:\n                    return False\n\n                user_inum =
userService.getUserInum(authenticated_user)\n                \n                attach_result =
deviceRegistrationService.attachUserDeviceRegistration(user_inum, u2f_device_id)\n\n                print
\"Super-Gluu. Authenticate for step 2. Result after attaching u2f_device '%s' to user '%s': '%s'\" %
(u2f_device_id, user_name, attach_result) \n\n                return attach_result\n            elif
self.twoStep:\n                if user_name == None:\n                    print \"Super-Gluu. Authenticate for
step 2. Failed to determine user name\"\n                    return False\n\n                validation_result
= self.validateSessionDeviceStatus(client_redirect_uri, session_device_status, user_name)\n                if
validation_result:\n                    print \"Super-Gluu. Authenticate for step 2. User '%s' successfully
authenticated with u2f_device '%s'\" % (user_name, u2f_device_id)\n                else:\n
return False\n                    \n                    super_gluu_request =
json.loads(session_device_status['super_gluu_request'])\n                    auth_method =
super_gluu_request['method']\n                    if auth_method in ['enroll', 'authenticate']:\n
if validation_result and self.use_audit_group:\n                            user =
authenticationService.getAuthenticatedUser()\n                            self.processAuditGroup(user,
self.audit_attribute, self.audit_group)\n\n                        return validation_result\n\n
print \"Super-Gluu. Authenticate for step 2. U2F auth_method is invalid\"\n\n            return False\n
else:\n            return False\n\n    def prepareForStep(self, configurationAttributes, requestParameters,
step):\n        identity = CdiUtil.bean(Identity)\n        session_attributes =
identity.getSessionId().getSessionAttributes()\n\n        client_redirect_uri =
self.getApplicationUri(session_attributes)\n        if client_redirect_uri == None:\n            print \"Super-
Gluu. Prepare for step. redirect_uri is not set\"\n            return False\n\n
self.setRequestScopedParameters(identity, step)\n\n        if step == 1:\n            print \"Super-Gluu.
Prepare for step 1\"\n            if self.oneStep:\n                session =
CdiUtil.bean(SessionIdService).getSessionId()\n                if session == None:\n                    print
\"Super-Gluu. Prepare for step 2. Failed to determine session_id\"\n                    return False\n\n
issuer = CdiUtil.bean(ConfigurationFactory).getConfiguration().getIssuer()\n
super_gluu_request_dictionary = {'app': client_redirect_uri,\n                                   'issuer':
issuer,\n                                   'state': session.getId(),\n
'created': DateTimeFormatter.ISO_OFFSET_DATE_TIME.format(ZonedDateTime.now().withNano(0))}\n\n
self.addGeolocationData(session_attributes, super_gluu_request_dictionary)\n\n
super_gluu_request = json.dumps(super_gluu_request_dictionary, separators=(',',':'))\n                print
\"Super-Gluu. Prepare for step 1. Prepared super_gluu_request:\", super_gluu_request\n    \n
identity.setWorkingParameter(\"super_gluu_request\", super_gluu_request)\n            elif self.twoStep:\n
identity.setWorkingParameter(\"display_register_action\", True)\n\n            return True\n        elif step
== 2:\n            print \"Super-Gluu. Prepare for step 2\"\n            if self.oneStep:\n
return True\n\n            authenticationService = CdiUtil.bean(AuthenticationService)\n            user =
authenticationService.getAuthenticatedUser()\n            if user == None:\n                print \"Super-Gluu.
Prepare for step 2. Failed to determine user name\"\n                return False\n\n            if
session_attributes.containsKey(\"super_gluu_request\"):\n               super_gluu_request =
session_attributes.get(\"super_gluu_request\")\n               if not
StringHelper.equalsIgnoreCase(super_gluu_request, \"timeout\"):\n                   print \"Super-Gluu. Prepare
for step 2. Request was generated already\"\n                   return True\n            \n            session
= CdiUtil.bean(SessionIdService).getSessionId()\n            if session == None:\n                print
\"Super-Gluu. Prepare for step 2. Failed to determine session_id\"\n                return False\n\n
auth_method = session_attributes.get(\"super_gluu_auth_method\")\n            if
StringHelper.isEmpty(auth_method):\n                print \"Super-Gluu. Prepare for step 2. Failed to determine
auth_method\"\n                return False\n\n            print \"Super-Gluu. Prepare for step 2. auth_method:
'%s'\" % auth_method\n            \n            issuer =
CdiUtil.bean(ConfigurationFactory).getAppConfiguration().getIssuer()\n            super_gluu_request_dictionary
= {'username': user.getUserId(),\n                               'app': client_redirect_uri,\n
'issuer': issuer,\n                               'method': auth_method,\n
'state': session.getId(),\n                               'created':
DateTimeFormatter.ISO_OFFSET_DATE_TIME.format(ZonedDateTime.now().withNano(0))}\n\n
self.addGeolocationData(session_attributes, super_gluu_request_dictionary)\n\n            super_gluu_request =
json.dumps(super_gluu_request_dictionary, separators=(',',':'))\n            print \"Super-Gluu. Prepare for
step 2. Prepared super_gluu_request:\", super_gluu_request\n\n
identity.setWorkingParameter(\"super_gluu_request\", super_gluu_request)\n
identity.setWorkingParameter(\"super_gluu_auth_method\", auth_method)\n\n            if auth_method in
['authenticate']:\n                self.sendPushNotification(client_redirect_uri, user, super_gluu_request)\n\n
return True\n        else:\n            return False\n\n    def getNextStep(self, configurationAttributes,
requestParameters, step):\n        # If user not pass current step change step to previous\n        identity =
CdiUtil.bean(Identity)\n        retry_current_step = identity.getWorkingParameter(\"retry_current_step\")\n
if retry_current_step:\n            print \"Super-Gluu. Get next step. Retrying current step\"\n\n            #
Remove old QR code\n            identity.setWorkingParameter(\"super_gluu_request\", \"timeout\")\n\n
resultStep = step\n            return resultStep\n\n        return -1\n\n    def
getExtraParametersForStep(self, configurationAttributes, step):\n        if step == 1:\n            if
self.oneStep:        \n                return Arrays.asList(\"super_gluu_request\")\n            elif
self.twoStep:\n                return Arrays.asList(\"display_register_action\")\n        elif step == 2:\n
return Arrays.asList(\"super_gluu_auth_method\", \"super_gluu_request\")\n        \n        return None\n\n
def getCountAuthenticationSteps(self, configurationAttributes):\n        identity = CdiUtil.bean(Identity)\n
if identity.isSetWorkingParameter(\"super_gluu_count_login_steps\"):\n            return
identity.getWorkingParameter(\"super_gluu_count_login_steps\")\n        else:\n            return 2\n\n    def
getPageForStep(self, configurationAttributes, step):\n        if step == 1:\n            if self.oneStep:
\n                return \"/auth/super-gluu/login.xhtml\"\n        elif step == 2:\n            if
self.oneStep:\n                return \"/login.xhtml\"\n            else:\n                identity =
CdiUtil.bean(Identity)\n                authmethod = identity.getWorkingParameter(\"super_gluu_auth_method\")\n
print \"Super-Gluu. authmethod '%s'\" % authmethod\n                if authmethod == \"enroll\":\n
return \"/auth/super-gluu/login.xhtml\"\n                else:\n                    return \"/auth/super-
gluu/login.xhtml\"\n\n        return \"\"\n\n    def getLogoutExternalUrl(self, configurationAttributes,
requestParameters):\n        print \"Get external logout URL call\"\n        return None\n\n    def
logout(self, configurationAttributes, requestParameters):\n        return True\n\n    def
processBasicAuthentication(self, credentials):\n        authenticationService =
CdiUtil.bean(AuthenticationService)\n\n        user_name = credentials.getUsername()\n        user_password =
credentials.getPassword()\n\n        logged_in = False\n        if StringHelper.isNotEmptyString(user_name) and
StringHelper.isNotEmptyString(user_password):\n            logged_in =
authenticationService.authenticate(user_name, user_password)\n\n        if not logged_in:\n            return
None\n\n        find_user_by_uid = authenticationService.getAuthenticatedUser()\n        if find_user_by_uid ==
None:\n            print \"Super-Gluu. Process basic authentication. Failed to find user '%s'\" % user_name\n
return None\n        \n        return find_user_by_uid\n\n    def validateSessionDeviceStatus(self,
client_redirect_uri, session_device_status, user_name = None):\n        userService =
CdiUtil.bean(UserService)\n        deviceRegistrationService = CdiUtil.bean(DeviceRegistrationService)\n\n
u2f_device_id = session_device_status['device_id']\n\n        u2f_device = None\n        if
session_device_status['enroll'] and session_device_status['one_step']:\n            u2f_device =
deviceRegistrationService.findOneStepUserDeviceRegistration(u2f_device_id)\n            if u2f_device ==
None:\n                print \"Super-Gluu. Validate session device status. There is no one step u2f_device
'%s'\" % u2f_device_id\n                return False\n        else:\n            # Validate if user has
specified device_id enrollment\n            user_inum = userService.getUserInum(user_name)\n\n            if
session_device_status['one_step']:\n                user_inum = session_device_status['user_inum']\n    \n
u2f_device = deviceRegistrationService.findUserDeviceRegistration(user_inum, u2f_device_id)\n            if
u2f_device == None:\n                print \"Super-Gluu. Validate session device status. There is no u2f_device
```

## Test Suite Navigation

# of failed tests: 0/100 (0.00%)

# of skipped tests: 0/100 (0.00%)

# of passed tests: 100/100 (100.00%)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 |
| 37 | 38 | 39 | 40 | 41 | 42 |
| 43 | 44 | 45 | 46 | 47 | 48 |
| 49 | 50 | 51 | 52 | 53 | 54 |
| 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 |
| 67 | 68 | 69 | 70 | 71 | 72 |
| 73 | 74 | 75 | 76 | 77 | 78 |
| 79 | 80 | 81 | 82 | 83 | 84 |
| 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 |
| 97 | 98 | 99 | 100 | | |

```
'%s' associated with user '%s'\" % (u2f_device_id, user_inum)\n                return False\n\n        if not
StringHelper.equalsIgnoreCase(client_redirect_uri, u2f_device.application):\n            print \"Super-Gluu.
Validate session device status. u2f_device '%s' associated with other application '%s'\" % (u2f_device_id,
u2f_device.application)\n                return False\n            \n        return True\n\n    def
getSessionDeviceStatus(self, session_attributes, user_name):\n        print \"Super-Gluu. Get session device
status\"\n\n        if not session_attributes.containsKey(\"super_gluu_request\"):\n            print \"Super-
Gluu. Get session device status. There is no Super-Gluu request in session attributes\"\n            return
None\n\n        # Check session state extended\n        if not
session_attributes.containsKey(\"session_custom_state\"):\n            print \"Super-Gluu. Get session device
status. There is no session_custom_state in session attributes\"\n            return None\n\n
session_custom_state = session_attributes.get(\"session_custom_state\")\n        if not
StringHelper.equalsIgnoreCase(\"approved\", session_custom_state):\n            print \"Super-Gluu. Get session
device status. User '%s' not approve or not pass U2F authentication. session_custom_state: '%s'\" % (user_name,
session_custom_state)\n            return None\n\n        # Try to find device_id in session attribute\n
if not session_attributes.containsKey(\"oxpush2_u2f_device_id\"):\n            print \"Super-Gluu. Get session
device status. There is no u2f_device associated with this request\"\n            return None\n\n        # Try
to find user_inum in session attribute\n        if not
session_attributes.containsKey(\"oxpush2_u2f_device_user_inum\"):\n            print \"Super-Gluu. Get session
device status. There is no user_inum associated with this request\"\n            return None\n        \n
enroll = False\n        if session_attributes.containsKey(\"oxpush2_u2f_device_enroll\"):\n            enroll =
StringHelper.equalsIgnoreCase(\"true\", session_attributes.get(\"oxpush2_u2f_device_enroll\"))\n\n
one_step = False\n        if session_attributes.containsKey(\"oxpush2_u2f_device_one_step\"):\n
one_step = StringHelper.equalsIgnoreCase(\"true\", session_attributes.get(\"oxpush2_u2f_device_one_step\"))\n
\n        super_gluu_request = session_attributes.get(\"super_gluu_request\")\n        u2f_device_id =
session_attributes.get(\"oxpush2_u2f_device_id\")\n        user_inum =
session_attributes.get(\"oxpush2_u2f_device_user_inum\")\n\n        session_device_status =
{\"super_gluu_request\": super_gluu_request, \"device_id\": u2f_device_id, \"user_inum\" : user_inum,
\"enroll\" : enroll, \"one_step\" : one_step}\n        print \"Super-Gluu. Get session device status.
session_device_status: '%s'\" % (session_device_status)\n        \n        return session_device_status\n\n
def initPushNotificationService(self, configurationAttributes):\n        print \"Super-Gluu. Initialize
Native/SNS/Gluu notification services\"\n\n        self.pushSnsMode = False\n        self.pushGluuMode =
False\n        if configurationAttributes.containsKey(\"notification_service_mode\"):\n
notificationServiceMode = configurationAttributes.get(\"notification_service_mode\").getValue2()\n
if StringHelper.equalsIgnoreCase(notificationServiceMode, \"sns\"):\n                return
self.initSnsPushNotificationService(configurationAttributes)\n            elif
StringHelper.equalsIgnoreCase(notificationServiceMode, \"gluu\"):\n                return
self.initGluuPushNotificationService(configurationAttributes)\n\n        return
self.initNativePushNotificationService(configurationAttributes)\n\n    def
initNativePushNotificationService(self, configurationAttributes):\n        print \"Super-Gluu. Initialize
native notification services\"\n        \n        creds =
self.loadPushNotificationCreds(configurationAttributes)\n        if creds == None:\n            return False\n
\n        try:\n            android_creds = creds[\"android\"][\"gcm\"]\n            ios_creds = creds[\"ios\"]
[\"apns\"]\n        except:\n            print \"Super-Gluu. Initialize native notification services. Invalid
credentials file format\"\n            return False\n        \n        self.pushAndroidService = None\n
self.pushAppleService = None\n        if android_creds[\"enabled\"]:\n            self.pushAndroidService =
Sender(android_creds[\"api_key\"]) \n            print \"Super-Gluu. Initialize native notification services.
Created Android notification service\"\n            \n        if ios_creds[\"enabled\"]:\n
p12_file_path = ios_creds[\"p12_file_path\"]\n            p12_password = ios_creds[\"p12_password\"]\n\n
try:\n                encryptionService = CdiUtil.bean(EncryptionService)\n                p12_password =
encryptionService.decrypt(p12_password)\n            except:\n                # Ignore exception. Password is
not encrypted\n                print \"Super-Gluu. Initialize native notification services. Assuming that
'p12_password' password in not encrypted\"\n\n            apnsServiceBuilder =
APNS.newService().withCert(p12_file_path, p12_password)\n            if ios_creds[\"production\"]:\n
self.pushAppleService = apnsServiceBuilder.withProductionDestination().build()\n            else:\n
self.pushAppleService = apnsServiceBuilder.withSandboxDestination().build()\n\n
self.pushAppleServiceProduction = ios_creds[\"production\"]\n\n            print \"Super-Gluu. Initialize
native notification services. Created iOS notification service\"\n\n        enabled = self.pushAndroidService
!= None or self.pushAppleService != None\n\n        return enabled\n\n    def
initSnsPushNotificationService(self, configurationAttributes):\n        print \"Super-Gluu. Initialize SNS
notification services\"\n        self.pushSnsMode = True\n\n        creds =
self.loadPushNotificationCreds(configurationAttributes)\n        if creds == None:\n            return False\n
\n        try:\n            sns_creds = creds[\"sns\"]\n            android_creds = creds[\"android\"]
[\"sns\"]\n            ios_creds = creds[\"ios\"][\"sns\"]\n        except:\n            print \"Super-Gluu.
Initialize SNS notification services. Invalid credentials file format\"\n            return False\n        \n
self.pushAndroidService = None\n        self.pushAppleService = None\n        if not
(android_creds[\"enabled\"] or ios_creds[\"enabled\"]):\n            print \"Super-Gluu. Initialize SNS
notification services. SNS disabled for all platforms\"\n            return False\n\n        sns_access_key =
sns_creds[\"access_key\"]\n        sns_secret_access_key = sns_creds[\"secret_access_key\"]\n        sns_region
= sns_creds[\"region\"]\n\n        encryptionService = CdiUtil.bean(EncryptionService)\n\n        try:\n
sns_secret_access_key = encryptionService.decrypt(sns_secret_access_key)\n        except:\n            # Ignore
exception. Password is not encrypted\n            print \"Super-Gluu. Initialize SNS notification services.
Assuming that 'sns_secret_access_key' in not encrypted\"\n        \n        pushSnsService =
CdiUtil.bean(PushSnsService)\n        pushClient = pushSnsService.createSnsClient(sns_access_key,
sns_secret_access_key, sns_region)\n\n        if android_creds[\"enabled\"]:\n
self.pushAndroidService = pushClient\n            self.pushAndroidPlatformArn =
android_creds[\"platform_arn\"]\n            print \"Super-Gluu. Initialize SNS notification services. Created
Android notification service\"\n\n        if ios_creds[\"enabled\"]:\n            self.pushAppleService =
pushClient \n            self.pushApplePlatformArn = ios_creds[\"platform_arn\"]\n
self.pushAppleServiceProduction = ios_creds[\"production\"]\n            print \"Super-Gluu. Initialize SNS
notification services. Created iOS notification service\"\n\n        enabled = self.pushAndroidService != None
or self.pushAppleService != None\n\n        return enabled\n\n    def initGluuPushNotificationService(self,
configurationAttributes):\n        print \"Super-Gluu. Initialize Gluu notification services\"\n\n
self.pushGluuMode = True\n\n        creds = self.loadPushNotificationCreds(configurationAttributes)\n        if
creds == None:\n            return False\n        \n        try:\n            gluu_conf = creds[\"gluu\"]\n
android_creds = creds[\"android\"][\"gluu\"]\n            ios_creds = creds[\"ios\"][\"gluu\"]\n
except:\n            print \"Super-Gluu. Initialize Gluu notification services. Invalid credentials file
format\"\n            return False\n        \n        self.pushAndroidService = None\n
self.pushAppleService = None\n        if not (android_creds[\"enabled\"] or ios_creds[\"enabled\"]):\n
print \"Super-Gluu. Initialize Gluu notification services. Gluu disabled for all platforms\"\n
return False\n\n        gluu_server_uri = gluu_conf[\"server_uri\"]\n        notifyClientFactory =
NotifyClientFactory.instance()\n        metadataConfiguration = None\n        try:\n
metadataConfiguration =
notifyClientFactory.createMetaDataConfigurationService(gluu_server_uri).getMetadataConfiguration()\n
except:\n            print \"Super-Gluu. Initialize Gluu notification services. Failed to load metadata.
Exception: \", sys.exc_info()[1]\n            return False\n\n        gluuClient =
notifyClientFactory.createNotifyService(metadataConfiguration)\n        encryptionService =
CdiUtil.bean(EncryptionService)\n\n        if android_creds[\"enabled\"]:\n            gluu_access_key =
android_creds[\"access_key\"]\n            gluu_secret_access_key = android_creds[\"secret_access_key\"]\n
\n            try:\n                gluu_secret_access_key =
encryptionService.decrypt(gluu_secret_access_key)\n            except:\n                # Ignore exception.
Password is not encrypted\n                print \"Super-Gluu. Initialize Gluu notification services. Assuming
that 'gluu_secret_access_key' in not encrypted\"\n            \n            self.pushAndroidService =
gluuClient \n            self.pushAndroidServiceAuth = notifyClientFactory.getAuthorization(gluu_access_key,
gluu_secret_access_key);\n            print \"Super-Gluu. Initialize Gluu notification services. Created
Android notification service\"\n\n        if ios_creds[\"enabled\"]:\n            gluu_access_key =
ios_creds[\"access_key\"]\n            gluu_secret_access_key = ios_creds[\"secret_access_key\"]\n        \n
try:\n                gluu_secret_access_key = encryptionService.decrypt(gluu_secret_access_key)\n
except:\n                # Ignore exception. Password is not encrypted\n                print \"Super-Gluu.
Initialize Gluu notification services. Assuming that 'gluu_secret_access_key' in not encrypted\"\n
\n            self.pushAppleService = gluuClient \n            self.pushAppleServiceAuth =
notifyClientFactory.getAuthorization(gluu_access_key, gluu_secret_access_key);\n            print \"Super-Gluu.
Initialize Gluu notification services. Created iOS notification service\"\n\n        enabled =
```

## Test Suite Navigation

# of failed tests: 0/100
(0.00%)

# of skipped tests: 0/100
(0.00%)

# of passed tests: 100/100
(100.00%)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 |
| 37 | 38 | 39 | 40 | 41 | 42 |
| 43 | 44 | 45 | 46 | 47 | 48 |
| 49 | 50 | 51 | 52 | 53 | 54 |
| 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 |
| 67 | 68 | 69 | 70 | 71 | 72 |
| 73 | 74 | 75 | 76 | 77 | 78 |
| 79 | 80 | 81 | 82 | 83 | 84 |
| 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 |
| 97 | 98 | 99 | 100 | | |

```
self.pushAndroidService != None or self.pushAppleService != None\n\n        return enabled\n\n    def
loadPushNotificationCreds(self, configurationAttributes):\n        print \"Super-Gluu. Initialize notification
services\"\n        if not configurationAttributes.containsKey(\"credentials_file\"):\n            return
None\n\n        super_gluu_creds_file = configurationAttributes.get(\"credentials_file\").getValue2()\n\n
# Load credentials from file\n        f = open(super_gluu_creds_file, 'r')\n        try:\n            creds =
json.loads(f.read())\n        except:\n            print \"Super-Gluu. Initialize notification services. Failed
to load credentials from file:\", super_gluu_creds_file\n            return None\n        finally:\n
f.close()\n\n        return creds\n\n    def sendPushNotification(self, client_redirect_uri, user,
super_gluu_request):\n        try:\n            self.sendPushNotificationImpl(client_redirect_uri, user,
super_gluu_request)\n        except:\n            print \"Super-Gluu. Send push notification. Failed to send
push notification: \", sys.exc_info()[1]\n\n    def sendPushNotificationImpl(self, client_redirect_uri, user,
super_gluu_request):\n        if not self.enabledPushNotifications:\n            return\n\n        user_name =
user.getUserId()\n        print \"Super-Gluu. Send push notification. Loading user '%s' devices\" %
user_name\n\n        send_notification = False\n        send_notification_result = True\n\n        userService
= CdiUtil.bean(UserService)\n        deviceRegistrationService = CdiUtil.bean(DeviceRegistrationService)\n\n
user_inum = userService.getUserInum(user_name)\n\n        send_android = 0\n        send_ios = 0\n
u2f_devices_list = deviceRegistrationService.findUserDeviceRegistrations(user_inum, client_redirect_uri,
\"jansId\", \"jansDeviceData\", \"jansDeviceNotificationConf\")\n        if u2f_devices_list.size() > 0:\n
for u2f_device in u2f_devices_list:\n                device_data = u2f_device.getDeviceData()\n\n
# Device data which Super-Gluu gets during enrollment\n                if device_data == None:\n
continue\n\n                platform = device_data.getPlatform()\n                push_token =
device_data.getPushToken()\n                debug = False\n\n                if
StringHelper.equalsIgnoreCase(platform, \"ios\") and StringHelper.isNotEmpty(push_token):\n
# Sending notification to iOS user's device\n                    if self.pushAppleService == None:\n
print \"Super-Gluu. Send push notification. Apple native push notification service is not enabled\"\n
else:\n                        send_notification = True\n                        \n
title = \"Super Gluu\"\n                        message = \"Confirm your sign in request to: %s\" %
client_redirect_uri\n\n                        if self.pushSnsMode or self.pushGluuMode:\n
pushSnsService = CdiUtil.bean(PushSnsService)\n                            targetEndpointArn =
self.getTargetEndpointArn(deviceRegistrationService, pushSnsService, PushPlatform.APNS, user, u2f_device)\n
if targetEndpointArn == None:\n                            \treturn\n\n
send_notification = True\n    \n                            sns_push_request_dictionary = { \"aps\": \n
{ \"badge\": 0,\n                                            \"alert\" : {\"body\":
message, \"title\" : title},\n                                            \"category\":
\"ACTIONABLE\",\n                                            \"content-available\":
\"1\",\n                                            \"sound\": 'default'\n
},\n                                    \"request\" : super_gluu_request\n
}\n                        push_message = json.dumps(sns_push_request_dictionary, separators=(',',':'))\n
\n                        if self.pushSnsMode:\n                                apple_push_platform =
PushPlatform.APNS\n                                if not self.pushAppleServiceProduction:\n
apple_push_platform = PushPlatform.APNS_SANDBOX\n        \n
send_notification_result = pushSnsService.sendPushMessage(self.pushAppleService, apple_push_platform,
targetEndpointArn, push_message, None)\n                            if debug:\n
print \"Super-Gluu. Send iOS SNS push notification. token: '%s', message: '%s', send_notification_result: '%s',
apple_push_platform: '%s'\" % (push_token, push_message, send_notification_result, apple_push_platform)\n
elif self.pushGluuMode:\n                            send_notification_result =
self.pushAppleService.sendNotification(self.pushAppleServiceAuth, targetEndpointArn, push_message)\n
if debug:\n                                print \"Super-Gluu. Send iOS Gluu push notification. token:
'%s', message: '%s', send_notification_result: '%s'\" % (push_token, push_message, send_notification_result)\n
else:\n                        additional_fields = { \"request\" : super_gluu_request }\n    \n
msgBuilder = APNS.newPayload().alertBody(message).alertTitle(title).sound(\"default\")\n
msgBuilder.category('ACTIONABLE').badge(0)\n                            msgBuilder.forNewsstand()\n
msgBuilder.customFields(additional_fields)\n                            push_message = msgBuilder.build()\n
\n                        send_notification_result = self.pushAppleService.push(push_token, push_message)\n
if debug:\n                            print \"Super-Gluu. Send iOS Native push notification. token: '%s',
message: '%s', send_notification_result: '%s'\" % (push_token, push_message, send_notification_result)\n
send_ios = send_ios + 1\n\n                if StringHelper.equalsIgnoreCase(platform, \"android\") and
StringHelper.isNotEmpty(push_token):\n                    # Sending notification to Android user's device\n
if self.pushAndroidService == None:\n                        print \"Super-Gluu. Send native push notification.
Android native push notification service is not enabled\"\n                    else:\n
send_notification = True\n\n                        title = \"Super-Gluu\"\n                        if
self.pushSnsMode or self.pushGluuMode:\n                            pushSnsService =
CdiUtil.bean(PushSnsService)\n                            targetEndpointArn =
self.getTargetEndpointArn(deviceRegistrationService, pushSnsService, PushPlatform.GCM, user, u2f_device)\n
if targetEndpointArn == None:\n                            \treturn\n\n
send_notification = True\n    \n                            sns_push_request_dictionary = { \"collapse_key\":
\"single\",\n                                            \"content_available\": True,\n
\"time_to_live\": 60,\n                                            \"data\": \n
{ \"message\" : super_gluu_request,\n
\"title\" : title }\n                        }\n                            push_message =
json.dumps(sns_push_request_dictionary, separators=(',',':'))\n    \n                            if
self.pushSnsMode:\n                                send_notification_result =
pushSnsService.sendPushMessage(self.pushAndroidService, PushPlatform.GCM, targetEndpointArn, push_message,
None)\n                                if debug:\n                                    print \"Super-Gluu. Send
Android SNS push notification. token: '%s', message: '%s', send_notification_result: '%s'\" % (push_token,
push_message, send_notification_result)\n                            elif self.pushGluuMode:\n
send_notification_result = self.pushAndroidService.sendNotification(self.pushAndroidServiceAuth,
targetEndpointArn, push_message)\n                                if debug:\n
print \"Super-Gluu. Send Android Gluu push notification. token: '%s', message: '%s', send_notification_result:
'%s'\" % (push_token, push_message, send_notification_result)\n                            else:\n
msgBuilder = Message.Builder().addData(\"message\", super_gluu_request).addData(\"title\",
title).collapseKey(\"single\").contentAvailable(True)\n                            push_message =
msgBuilder.build()\n    \n                            send_notification_result =
self.pushAndroidService.send(push_message, push_token, 3)\n                            if debug:\n
print \"Super-Gluu. Send Android Native push notification. token: '%s', message: '%s',
send_notification_result: '%s'\" % (push_token, push_message, send_notification_result)\n
send_android = send_android + 1\n\n        print \"Super-Gluu. Send push notification. send_android: '%s',
send_ios: '%s'\" % (send_android, send_ios)\n\n    def getTargetEndpointArn(self, deviceRegistrationService,
pushSnsService, platform, user, u2fDevice):\n        targetEndpointArn = None\n                            \n
# Return endpoint ARN if it created already\n        notificationConf = u2fDevice.getDeviceNotificationConf()\n
if StringHelper.isNotEmpty(notificationConf):\n            notificationConfJson =
json.loads(notificationConf)\n            targetEndpointArn = notificationConfJson['sns_endpoint_arn']\n
if StringHelper.isNotEmpty(targetEndpointArn):\n                print \"Super-Gluu. Get target endpoint ARN.
There is already created target endpoint ARN\"\n                return targetEndpointArn\n\n        # Create
endpoint ARN        \n        pushClient = None\n        pushClientAuth = None\n        platformApplicationArn
= None\n        if platform == PushPlatform.GCM:\n            pushClient = self.pushAndroidService\n
if self.pushSnsMode:\n                platformApplicationArn = self.pushAndroidPlatformArn\n            if
self.pushGluuMode:\n                pushClientAuth = self.pushAndroidServiceAuth\n        elif platform ==
PushPlatform.APNS:\n            pushClient = self.pushAppleService\n            if self.pushSnsMode:\n
platformApplicationArn = self.pushApplePlatformArn\n            if self.pushGluuMode:\n
pushClientAuth = self.pushAppleServiceAuth\n        else:\n            return None\n\n        deviceData =
u2fDevice.getDeviceData()\n        pushToken = deviceData.getPushToken()\n        \n        print \"Super-Gluu.
Get target endpoint ARN. Attempting to create target endpoint ARN for user: '%s'\" % user.getUserId()\n
if self.pushSnsMode:\n            targetEndpointArn = pushSnsService.createPlatformArn(pushClient,
platformApplicationArn, pushToken, user)\n        else:\n            customUserData =
pushSnsService.getCustomUserData(user)\n            registerDeviceResponse =
pushClient.registerDevice(pushClientAuth, pushToken, customUserData);\n            if registerDeviceResponse !=
None and registerDeviceResponse.getStatusCode() == 200:\n                targetEndpointArn =
registerDeviceResponse.getEndpointArn()\n        \n        if StringHelper.isEmpty(targetEndpointArn):\n\t
print \"Super-Gluu. Failed to get endpoint ARN for user: '%s'\" % user.getUserId()\n        \treturn None\n\n
print \"Super-Gluu. Get target endpoint ARN. Create target endpoint ARN '%s' for user: '%s'\" %
(targetEndpointArn, user.getUserId())\n        \n        # Store created endpoint ARN in device entry\n
```

## Test Suite Navigation

# of failed tests: 0/100
(0.00%)

# of skipped tests: 0/100
(0.00%)

# of passed tests: 100/100
(100.00%)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 |
| 37 | 38 | 39 | 40 | 41 | 42 |
| 43 | 44 | 45 | 46 | 47 | 48 |
| 49 | 50 | 51 | 52 | 53 | 54 |
| 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 |
| 67 | 68 | 69 | 70 | 71 | 72 |
| 73 | 74 | 75 | 76 | 77 | 78 |
| 79 | 80 | 81 | 82 | 83 | 84 |
| 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 |
| 97 | 98 | 99 | 100 | | |

```
userInum = user.getAttribute(\"inum\")\n        u2fDeviceUpdate =
deviceRegistrationService.findUserDeviceRegistration(userInum, u2fDevice.getId())\n
u2fDeviceUpdate.setDeviceNotificationConf('{\"sns_endpoint_arn\" : \"%s\"}' % targetEndpointArn)\n
deviceRegistrationService.updateDeviceRegistration(userInum, u2fDeviceUpdate)\n\n        return
targetEndpointArn\n\n    def getApplicationUri(self, session_attributes):\n        if self.applicationId !=
None:\n            return self.applicationId\n            \n        if not
session_attributes.containsKey(\"redirect_uri\"):\n            return None\n\n        return
session_attributes.get(\"redirect_uri\")\n\n    def setRequestScopedParameters(self, identity, step):\n
downloadMap = HashMap()\n        if self.registrationUri != None:\n
identity.setWorkingParameter(\"external_registration_uri\", self.registrationUri)\n\n        if
self.androidUrl!= None and step == 1:\n            downloadMap.put(\"android\", self.androidUrl)\n\n        if
self.IOSUrl  != None and step == 1:\n            downloadMap.put(\"ios\", self.IOSUrl)\n            \n
if self.customLabel != None:\n            identity.setWorkingParameter(\"super_gluu_label\",
self.customLabel)\n            \n        identity.setWorkingParameter(\"download_url\", downloadMap)\n
identity.setWorkingParameter(\"super_gluu_qr_options\", self.customQrOptions)\n\n    def
addGeolocationData(self, session_attributes, super_gluu_request_dictionary):\n        if
session_attributes.containsKey(\"remote_ip\"):\n            remote_ip = session_attributes.get(\"remote_ip\")\n
if StringHelper.isNotEmpty(remote_ip):\n                print \"Super-Gluu. Prepare for step 2. Adding req_ip
and req_loc to super_gluu_request\"\n                super_gluu_request_dictionary['req_ip'] = remote_ip\n\n
remote_loc_dic = self.determineGeolocationData(remote_ip)\n                if remote_loc_dic == None:\n
print \"Super-Gluu. Prepare for step 2. Failed to determine remote location by remote IP '%s'\" % remote_ip\n
return\n\n                remote_loc = \"%s, %s, %s\" % ( remote_loc_dic['country'],
remote_loc_dic['regionName'], remote_loc_dic['city'] )\n                remote_loc_encoded =
urllib.quote(remote_loc.encode('utf-8'))\n                super_gluu_request_dictionary['req_loc'] =
remote_loc_encoded\n\n    def determineGeolocationData(self, remote_ip):\n        print \"Super-Gluu. Determine
remote location. remote_ip: '%s'\" % remote_ip\n        httpService = CdiUtil.bean(HttpService)\n\n
http_client = httpService.getHttpsClient()\n        http_client_params = http_client.getParams()\n
http_client_params.setIntParameter(CoreConnectionPNames.CONNECTION_TIMEOUT, 15 * 1000)\n        \n
geolocation_service_url = \"http://ip-api.com/json/%s?fields=49177\" % remote_ip\n
geolocation_service_headers = { \"Accept\" : \"application/json\" }\n\n        try:\n
http_service_response = httpService.executeGet(http_client, geolocation_service_url,
geolocation_service_headers)\n            http_response = http_service_response.getHttpResponse()\n
except:\n            print \"Super-Gluu. Determine remote location. Exception: \", sys.exc_info()[1]\n
return None\n\n        try:\n            if not httpService.isResponseStastusCodeOk(http_response):\n
print \"Super-Gluu. Determine remote location. Get invalid response from validation server: \",
str(http_response.getStatusLine().getStatusCode())\n                httpService.consume(http_response)\n
return None\n    \n            response_bytes = httpService.getResponseContent(http_response)\n
response_string = httpService.convertEntityToString(response_bytes)\n
httpService.consume(http_response)\n        finally:\n            http_service_response.closeConnection()\n\n
if response_string == None:\n            print \"Super-Gluu. Determine remote location. Get empty response from
location server\"\n            return None\n        \n        response = json.loads(response_string)\n
\n        if not StringHelper.equalsIgnoreCase(response['status'], \"success\"):\n            print \"Super-
Gluu. Determine remote location. Get response with status: '%s'\" % response['status']\n            return
None\n\n        return response\n\n    def isUserMemberOfGroup(self, user, attribute, group):\n
is_member = False\n        member_of_list = user.getAttributeValues(attribute)\n        if (member_of_list !=
None):\n            for member_of in member_of_list:\n                if StringHelper.equalsIgnoreCase(group,
member_of) or member_of.endswith(group):\n                    is_member = True\n                    break\n\n
return is_member\n\n    def processAuditGroup(self, user, attribute, group):\n        is_member =
self.isUserMemberOfGroup(user, attribute, group)\n        if (is_member):\n            print \"Super-Gluu.
Authenticate for processAuditGroup. User '%s' member of audit group\" % user.getUserId()\n            print
\"Super-Gluu. Authenticate for processAuditGroup. Sending e-mail about user '%s' login to %s\" %
(user.getUserId(), self.audit_email)\n            \n            # Send e-mail to administrator\n
user_id = user.getUserId()\n            mailService = CdiUtil.bean(MailService)\n            subject = \"User
log in: %s\" % user_id\n            body = \"User log in: %s\" % user_id\n
mailService.sendMail(self.audit_email, subject, body)\n",
      "enabled": false,
      "revision": 1,
      "moduleProperties": [
        {
          "value2": "ldap",
          "value1": "location_type"
        },
        {
          "value2": "interactive",
          "value1": "usage_type"
        }
      ],
      "scriptType": "PERSON_AUTHENTICATION",
      "name": "super_gluu",
      "modified": false,
      "configurationProperties": [
        {
          "hide": false,
          "value2": "{ size: 500, mSize: 0.05 }",
          "value1": "qr_options"
        },
        {
          "hide": false,
          "value2": "Super Gluu",
          "value1": "label"
        },
        {
          "hide": false,
          "value2": "https://jans.server3/identity/register",
          "value1": "registration_uri"
        },
        {
          "hide": false,
          "value2": "two_step",
          "value1": "authentication_mode"
        },
        {
          "hide": false,
          "value2": "gluu",
          "value1": "notification_service_mode"
        },
        {
          "hide": false,
          "value2": "/etc/certs/super_gluu_creds.json",
          "value1": "credentials_file"
        },
        {
          "hide": false,
          "value2": "https://play.google.com/store/apps/details?id=gluu.org.super.gluu&hl=en_US",
          "value1": "supergluu_android_download_url"
        },
        {
          "hide": false,
          "value2": "https://itunes.apple.com/us/app/super-gluu/id1093479646",
          "value1": "supergluu_ios_download_url"
        }
      ],
```

## Test Suite Navigation

# of failed tests: 0/100 (0.00%)

# of skipped tests: 0/100 (0.00%)

# of passed tests: 100/100 (100.00%)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 |
| 37 | 38 | 39 | 40 | 41 | 42 |
| 43 | 44 | 45 | 46 | 47 | 48 |
| 49 | 50 | 51 | 52 | 53 | 54 |
| 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 |
| 67 | 68 | 69 | 70 | 71 | 72 |
| 73 | 74 | 75 | 76 | 77 | 78 |
| 79 | 80 | 81 | 82 | 83 | 84 |
| 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 |
| 97 | 98 | 99 | 100 | | |

```
      "baseDn": "inum=92F0-BF9E,ou=scripts,o=jans"
    },
    {
      "internal": false,
      "aliases": [
        "basic_alias1",
        "basic_alias2"
      ],
      "level": 10,
      "programmingLanguage": "PYTHON",
      "description": "Sample authentication module",
      "locationType": "LDAP",
      "dn": "inum=A51E-76DA,ou=scripts,o=jans",
      "inum": "A51E-76DA",
      "script": "# Janssen Project software is available under the Apache 2.0 License (2004). See
http://www.apache.org/licenses/ for full text.\n# Copyright (c) 2020, Janssen Project\n#\n# Author: Yuriy
Movchan\n#\n\nfrom io.jans.service.cdi.util import CdiUtil\nfrom io.jans.as.server.security import
Identity\nfrom io.jans.model.custom.script.type.auth import PersonAuthenticationType\nfrom
io.jans.as.server.service import AuthenticationService\nfrom io.jans.util import StringHelper\n\nimport
java\n\nclass PersonAuthentication(PersonAuthenticationType):\n    def __init__(self, currentTimeMillis):\n
self.currentTimeMillis = currentTimeMillis\n\n    def init(self, customScript,  configurationAttributes):\n
print \"Basic. Initialization\"\n        print \"Basic. Initialized successfully\"\n        return True   \n\n
def destroy(self, configurationAttributes):\n        print \"Basic. Destroy\"\n        print \"Basic. Destroyed
successfully\"\n        return True\n        \n    def getAuthenticationMethodClaims(self,
requestParameters):\n        return None\n        \n    def getApiVersion(self):\n        return 11\n\n    def
isValidAuthenticationMethod(self, usageType, configurationAttributes):\n        return True\n\n    def
getAlternativeAuthenticationMethod(self, usageType, configurationAttributes):\n        return None\n\n    def
authenticate(self, configurationAttributes, requestParameters, step):\n        authenticationService =
CdiUtil.bean(AuthenticationService)\n\n        if (step == 1):\n            print \"Basic. Authenticate for
step 1\"\n\n            identity = CdiUtil.bean(Identity)\n            credentials =
identity.getCredentials()\n\n            user_name = credentials.getUsername()\n            user_password =
credentials.getPassword()\n\n            logged_in = False\n            if
(StringHelper.isNotEmptyString(user_name) and StringHelper.isNotEmptyString(user_password)):\n
logged_in = authenticationService.authenticate(user_name, user_password)\n\n            if (not logged_in):\n
return False\n\n            return True\n        else:\n            return False\n\n    def
prepareForStep(self, configurationAttributes, requestParameters, step):\n        if (step == 1):\n
print \"Basic. Prepare for Step 1\"\n            return True\n        else:\n            return False\n\n
def getExtraParametersForStep(self, configurationAttributes, step):\n        return None\n\n    def
getCountAuthenticationSteps(self, configurationAttributes):\n        return 1\n\n    def getPageForStep(self,
configurationAttributes, step):\n        return \"\"\n\n    def getNextStep(self, configurationAttributes,
requestParameters, step):\n        return -1\n\n    def getLogoutExternalUrl(self, configurationAttributes,
requestParameters):\n        print \"Get external logout URL call\"\n        return None\n\n    def
logout(self, configurationAttributes, requestParameters):\n        return True\n",
      "enabled": true,
      "revision": 1,
      "moduleProperties": [
        {
          "value2": "interactive",
          "value1": "usage_type"
        },
        {
          "value2": "ldap",
          "value1": "location_type"
        }
      ],
      "scriptType": "PERSON_AUTHENTICATION",
      "name": "basic",
      "modified": false,
      "baseDn": "inum=A51E-76DA,ou=scripts,o=jans"
    },
    {
      "internal": false,
      "level": 10,
      "programmingLanguage": "PYTHON",
      "description": "This script is a 2 in 1. It can be used to enable user to reset its password or to enable
2FA sending a token to user's email",
      "dn": "inum=B270-381E,ou=scripts,o=jans",
      "inum": "B270-381E",
      "script": "# coding: utf-8\n# Janssen Project software is available under the Apache License (2004). See
http://www.apache.org/licenses/ for full text.\n# Copyright (c) 2020, Janssen Project\n#\n# Author: Christian
Eland\n\n\nfrom org.xdi.oxauth.service import AuthenticationService\nfrom io.jans.as.server.service import
UserService\nfrom org.gluu.oxauth.auth import Authenticator\nfrom org.xdi.oxauth.security import Identity\nfrom
org.xdi.model.custom.script.type.auth import PersonAuthenticationType\nfrom org.xdi.service.cdi.util import
CdiUtil\nfrom org.xdi.util import StringHelper\nfrom org.xdi.oxauth.util import ServerUtil\nfrom
io.jans.as.common.service.common import ConfigurationService\nfrom io.jans.as.common.service.common import
EncryptionService\nfrom io.jans.jsf2.message import FacesMessages\nfrom jakarta.faces.application import
FacesMessage\nfrom io.jans.orm.exception import AuthenticationException\n\n#dealing with smtp server\nimport
smtplib\n\n#dealing with emails\nfrom email.mime.multipart import MIMEMultipart\nfrom email.mime.text import
MIMEText\n\n# This one is from core Java\nfrom java.util import Arrays\n\n# to generate string token\nimport
random\nimport string\n\n# regex\nimport re\n\nimport urllib\n\nimport java\n\nclass EmailValidator():\n
'''\n    Class to check e-mail format\n    '''\n    regex = '^\\w+([\\.-]?\\w+)*@\\w+([\\.-]?\\w+)*
(\\.\\w{2,3})+$'\n\n    def check(self, email):\n        '''\n        Check if email format is valid\n
returns: boolean\n        '''\n\n        if(re.search(self.regex,email)):\n            print \"Forgot Password
- %s is a valid email format\" % email\n            return True\n        else:\n            print \"Forgot
Password - %s is an invalid email format\" % email\n            return False\n\nclass Token:\n    #class that
deals with string token\n\n    def generateToken(self):\n        ''' method to generate token string\n
returns: String\n        '''\n        letters = string.ascii_lowercase\n\n        #token lenght\n        lenght
= 20\n\n        #generate token\n        token = ''.join(random.choice(letters) for i in range(lenght))\n
print \"Forgot Password - Generating token\"\n\n        return token\n\n\nclass EmailSender():\n    #class that
sends e-mail through smtp\n\n    def getSmtpConfig(self):\n        '''\n        get SMTP config from Gluu
Server\n        return dict\n        '''\n        \n        smtpconfig =
CdiUtil.bean(ConfigurationService).getConfiguration().getSmtpConfiguration()\n        \n        if smtpconfig
is None:\n            print \"Forgot Password - SMTP CONFIG DOESN'T EXIST - Please configure\"\n\n
else:\n            print \"Forgot Password - SMTP CONFIG FOUND\"\n            encryptionService =
CdiUtil.bean(EncryptionService)\n            smtp_config = {\n                'host' : smtpconfig.getHost(),\n
'port' : smtpconfig.getPort(),\n                'user' : smtpconfig.getUserName(),\n                'from' :
smtpconfig.getFromEmailAddress(),\n                'pwd_decrypted' :
encryptionService.decrypt(smtpconfig.getPassword()),\n                'req_ssl' : smtpconfig.isRequiresSsl(),\n
'requires_authentication' : smtpconfig.isRequiresAuthentication(),\n                'server_trust' :
smtpconfig.isServerTrust()\n            }\n\n        return smtp_config\n\n            \n\n    def
sendEmail(self,useremail,token):\n        '''\n        send token by e-mail to useremail\n        '''\n\n
# server connection \n        smtpconfig = self.getSmtpConfig()\n        \n        try:\n            s =
smtplib.SMTP(smtpconfig['host'], port=smtpconfig['port'])\n            \n\n            if
smtpconfig['requires_authentication']:\n                \n                if smtpconfig['req_ssl']:\n
s.starttls()\n                \n                s.login(smtpconfig['user'], smtpconfig['pwd_decrypted'])\n\n
\n            #message setup\n            msg = MIMEMultipart() #create message\n            \n
message = \"Here is your token: %s\" % token\n\n            msg['From'] = smtpconfig['from'] #sender\n
msg['To'] = useremail #recipient\n            msg['Subject'] = \"Password Reset Request\" #subject\n\n
#attach message body\n            msg.attach(MIMEText(message, 'plain'))\n\n            #send message via smtp
server\n            # send_message method is for python3 only s.send_message(msg)\n\n            #send email
(python2)\n            s.sendmail(msg['From'],msg['To'],msg.as_string())\n            \n            #after
sent, delete\n            del msg\n\n        except smtplib.SMTPAuthenticationError as err:\n            print
```

## Test Suite Navigation

# of failed tests: 0/100
(0.00%)

# of skipped tests: 0/100
(0.00%)

# of passed tests: 100/100
(100.00%)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 |
| 37 | 38 | 39 | 40 | 41 | 42 |
| 43 | 44 | 45 | 46 | 47 | 48 |
| 49 | 50 | 51 | 52 | 53 | 54 |
| 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 |
| 67 | 68 | 69 | 70 | 71 | 72 |
| 73 | 74 | 75 | 76 | 77 | 78 |
| 79 | 80 | 81 | 82 | 83 | 84 |
| 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 |
| 97 | 98 | 99 | 100 | | |

```
\"Forgot Password - SMTPAuthenticationError - %s - %s\" % (MY_ADDRESS,PASSWORD)\n            print err\n\n
except smtplib.smtplib.SMTPSenderRefused as err:\n            print \"Forgot Password - SMTPSenderRefused - \"
+ err\n\n\nclass PersonAuthentication(PersonAuthenticationType):\n\n\n    def __init__(self,
currentTimeMillis):\n        self.currentTimeMillis = currentTimeMillis\n\n    def init(self, customScript,
configurationAttributes):\n\n        print \"Forgot Password - Initialized successfully\"\n        return True
\n\n    def destroy(self, configurationAttributes):\n        print \"Forgot Password - Destroyed
successfully\"\n        return True\n\n    def getApiVersion(self):\n        # I'm not sure why is 11 and not
2\n        return 11\n\n    def getAuthenticationMethodClaims(self, requestParameters):\n        return
None\n\n    def isValidAuthenticationMethod(self, usageType, configurationAttributes):\n        return
True\n\n\n    def getAlternativeAuthenticationMethod(self, usageType, configurationAttributes):\n        return
None\n\n    def authenticate(self, configurationAttributes, requestParameters, step):\n        '''\n
Authenticates user\n        Step 1 will be defined according to SCRIPT_FUNCTION custom attribute\n
returns: boolean\n        '''\n\n        #gets custom attribute\n        sf =
configurationAttributes.get(\"SCRIPT_FUNCTION\").getValue2()\n\n        print \"Forgot Password - %s -
Authenticate for step %s\" % (sf, step)\n\n        identity = CdiUtil.bean(Identity)\n        credentials =
identity.getCredentials()\n        user_name = credentials.getUsername()\n        user_password =
credentials.getPassword()\n\n\n        if step == 1:\n\n            if sf == \"forgot_password\":\n\n
\n                authenticationService = CdiUtil.bean(AuthenticationService)\n\n                logged_in =
authenticationService.authenticate(user_name, user_password)\n\n                \n                if not
logged_in:\n\n                    \n                    email = ServerUtil.getFirstValue(requestParameters,
\"ForgotPasswordForm:useremail\")\n                    validator = EmailValidator()\n                    if not
validator.check(email):\n                        print \"Forgot Password - Email format invalid\"\n
return False\n\n                    else:\n                        print \"Forgot Password -Email format
valid\"\n    \n                        print \"Forgot Password - Entered email is %s\" % email\n
identity.setWorkingParameter(\"useremail\",email)\n                        \n                        # Just
trying to get the user by the email\n                        user_service = CdiUtil.bean(UserService)\n
user2 = user_service.getUserByAttribute(\"mail\", email)\n\n                        if user2 is not None:\n
\n                            print user2\n                            print \"Forgot Password - User with e-
mail %s found.\" % user2.getAttribute(\"mail\")\n                            \n                            # send
email\n                            new_token = Token()\n                            token =
new_token.generateToken()                            \n                            sender = EmailSender()\n
print \"Email: \" + email\n                            print \"Token: \" + token\n
sender.sendEmail(email,token)\n\n                            \n
identity.setWorkingParameter(\"token\", token)\n                            print
identity.getWorkingParameter(\"token\")\n                            \n    \n                            \n
else:\n                            print \"Forgot Password - User with e-mail %s not found\" % email\n\n
return True\n\n\n                else:\n                    # if user is already authenticated, returns
true.\n\n                    user = authenticationService.getAuthenticatedUser()\n                    print
\"Forgot Password - User %s is authenticated\" % user.getUserId()\n\n                    return True\n\n
if sf == \"email_2FA\":\n\n                try:\n                    # Just trying to get the user by the uid\n
authenticationService = CdiUtil.bean(AuthenticationService)\n                    logged_in =
authenticationService.authenticate(user_name, user_password)\n                    \n                    print
'email_2FA user_name: ' + str(user_name)\n                    \n                    user_service =
CdiUtil.bean(UserService)\n                    user2 = user_service.getUserByAttribute(\"uid\", user_name)\n\n
if user2 is not None:\n                        print \"user:\"\n                        print user2\n
print \"Forgot Password - User with e-mail %s found.\" % user2.getAttribute(\"mail\")\n
email = user2.getAttribute(\"mail\")\n                        uid = user2.getAttribute(\"uid\")\n\n
# send token\n                        # send email\n                        new_token = Token()\n
token = new_token.generateToken()                        \n                        sender = EmailSender()\n
print \"Email: \" + email\n                        print \"Token: \" + token\n
sender.sendEmail(email,token)\n\n                        identity.setWorkingParameter(\"token\", token)\n\n
return True\n\n                except AuthenticationException as err:\n                    print err\n
return False\n\n                \n    \n\n        if step == 2:\n            # step 2 user enters token\n
credentials = identity.getCredentials()\n            user_name = credentials.getUsername()\n
user_password = credentials.getPassword()\n            \n            authenticationService =
CdiUtil.bean(AuthenticationService)\n            logged_in = authenticationService.authenticate(user_name,
user_password)\n\n            # retrieves token typed by user\n            input_token =
ServerUtil.getFirstValue(requestParameters, \"ResetTokenForm:inputToken\")\n\n            print \"Forgot
Password - Token inputed by user is %s\" % input_token\n\n            token =
identity.getWorkingParameter(\"token\")\n            print \"Forgot Password - Retrieved token\"\n
email = identity.getWorkingParameter(\"useremail\")\n            print \"Forgot Password - Retrieved email\"
\n\n            # compares token sent and token entered by user\n            if input_token == token:\n
print \"Forgot Password - token entered correctly\"\n
identity.setWorkingParameter(\"token_valid\", True)\n                \n                return True\n\n
else:\n                print \"Forgot Password - wrong token\"\n                return False\n\n        \n
if step == 3:\n            # step 3 enters new password (only runs if custom attibute is forgot_password\n\n
user_service = CdiUtil.bean(UserService)\n\n            email = identity.getWorkingParameter(\"useremail\")\n
user2 = user_service.getUserByAttribute(\"mail\", email)\n\n\n            user_name = user2.getUserId()\n
\n            new_password = ServerUtil.getFirstValue(requestParameters, \"UpdatePasswordForm:newPassword\")\n
\n            print \"Forgot Password - New password submited\"\n            \n            # update user info with
new password\n            user2.setAttribute(\"userPassword\",new_password)\n            print \"Forgot
Password - user uid is %s\" % user_name\n            print \"Forgot Password - Updating user with new
password...\"\n            user_service.updateUser(user2)\n            print \"Forgot Password - User updated
with new password\"\n            # authenticates and login user\n            print \"Forgot Password - Loading
authentication service...\"\n            authenticationService2 = CdiUtil.bean(AuthenticationService)\n\n
print \"Forgot Password - Trying to authenticate user...\"\n            login =
authenticationService2.authenticate(user_name, new_password)\n            \n            return True\n\n    def
prepareForStep(self, configurationAttributes, requestParameters, step):\n        \n        print \"Forgot
Password - Preparing for step %s\" % step\n        \n        return True\n\n\n    # Return value is a
java.util.List<String> \n    def getExtraParametersForStep(self, configurationAttributes, step):\n
return Arrays.asList(\"token\",\"useremail\",\"token_valid\")\n\n\n    # This method determines how many steps
the authentication flow may have\n    # It doesn't have to be a constant value\n    def
getCountAuthenticationSteps(self, configurationAttributes):\n        \n        sf =
configurationAttributes.get(\"SCRIPT_FUNCTION\").getValue2()\n        \n\n        # if option is forgot_token\n
if sf == \"forgot_password\":\n            print \"Entered sf == forgot_password\"\n            return 3\n
\n        # if ption is email_2FA\n        if sf == \"email_2FA\":\n            print \"Entered if
sf=email_2FA\"\n            return 2\n\n        else:\n            print \"Forgot Password - Custom Script
Custom Property Incorrect, please check\"\n\n\n    # The xhtml page to render upon each step of the flow\n    #
returns a string relative to oxAuth webapp root\n    def getPageForStep(self, configurationAttributes, step):\n
\n        sf = configurationAttributes.get(\"SCRIPT_FUNCTION\").getValue2()\n\n        if step == 1:\n\n
if sf == \"forgot_password\":\n                return \"/auth/forgot_password/forgot.xhtml\"\n\n            if
sf == 'email_2FA':\n                return \"\"\n\n        if step == 2:\n            return
\"/auth/forgot_password/entertoken.xhtml\"\n\n        if step == 3:\n            if sf ==
\"forgot_password\":\n                return \"/auth/forgot_password/newpassword.xhtml\"\n\n    \n    def
getNextStep(self, configurationAttributes, requestParameters, step):\n        # Method used on version 2
(11?)\n        return -1\n    \n    def getLogoutExternalUrl(self, configurationAttributes,
requestParameters):\n        print \"Get external logout URL call\"\n        return None\n        \n    def
logout(self, configurationAttributes, requestParameters):\n        return True\n",
      "enabled": false,
      "revision": 1,
      "moduleProperties": [
        {
          "value2": "ldap",
          "value1": "SCRIPT_FUNCTION"
        }
      ],
      "scriptType": "PERSON_AUTHENTICATION",
      "name": "Forgot_Password_2FA_Token",
      "modified": false,
      "configurationProperties": [
        {
          "hide": false,
```

## Test Suite Navigation

# of failed tests: 0/100 (0.00%)

# of skipped tests: 0/100 (0.00%)

# of passed tests: 100/100 (100.00%)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 |
| 37 | 38 | 39 | 40 | 41 | 42 |
| 43 | 44 | 45 | 46 | 47 | 48 |
| 49 | 50 | 51 | 52 | 53 | 54 |
| 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 |
| 67 | 68 | 69 | 70 | 71 | 72 |
| 73 | 74 | 75 | 76 | 77 | 78 |
| 79 | 80 | 81 | 82 | 83 | 84 |
| 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 |
| 97 | 98 | 99 | 100 | | |

```
          "value2": "forgot_password",
          "value1": "SCRIPT_FUNCTION"
        }
      ],
      "baseDn": "inum=B270-381E,ou=scripts,o=jans"
    },
    {
      "internal": false,
      "level": 10,
      "programmingLanguage": "PYTHON",
      "description": "Agama Script",
      "locationType": "LDAP",
      "dn": "inum=BADA-BADA,ou=scripts,o=jans",
      "inum": "BADA-BADA",
      "script": "# Janssen Project software is available under the Apache 2.0 License (2004). See
http://www.apache.org/licenses/ for full text.\n# Copyright (c) 2020, Janssen Project\n#\nfrom io.jans.agama
import NativeJansFlowBridge\nfrom io.jans.agama.engine.misc import FlowUtils\nfrom io.jans.as.server.security
import Identity\nfrom io.jans.as.server.service import AuthenticationService\nfrom io.jans.jsf2.service import
FacesService\nfrom io.jans.jsf2.message import FacesMessages\nfrom io.jans.model.custom.script.type.auth import
PersonAuthenticationType\nfrom io.jans.orm import PersistenceEntryManager\nfrom io.jans.service.cdi.util import
CdiUtil\nfrom io.jans.util import StringHelper\n\nfrom jakarta.faces.application import FacesMessage\n\nimport
java\nimport sys\n\nclass PersonAuthentication(PersonAuthenticationType):\n    def __init__(self,
currentTimeMillis):\n        self.currentTimeMillis = currentTimeMillis\n\n    def init(self, customScript,
configurationAttributes):\n        print \"Agama. Initialization\"        \n        prop =
\"cust_param_name\"\n        self.cust_param_name = self.configProperty(configurationAttributes, prop)\n
\n        if self.cust_param_name == None:\n            print \"Agama. Custom parameter name not referenced via
property '%s'\" % prop\n            return False\n            \n        print \"Agama. Request param '%s' will
be used to pass flow inputs\" % self.cust_param_name\n        print \"Agama. Initialized successfully\"\n
return True\n\n    def destroy(self, configurationAttributes):\n        print \"Agama. Destroy\"\n        print
\"Agama. Destroyed successfully\"\n        return True\n        \n    def getAuthenticationMethodClaims(self,
requestParameters):\n        return None\n        \n    def getApiVersion(self):\n        return 11\n\n    def
isValidAuthenticationMethod(self, usageType, configurationAttributes):\n        return True\n\n    def
getAlternativeAuthenticationMethod(self, usageType, configurationAttributes):\n        return None\n\n    def
authenticate(self, configurationAttributes, requestParameters, step):\n\n        if step == 1:\n
print \"Agama. Authenticate for step 1\"\n            \n            try:\n                bridge =
CdiUtil.bean(NativeJansFlowBridge)\n                result = bridge.close()\n                \n
if result == None or not result.isSuccess():\n                    print \"Agama. Flow DID NOT finished
successfully\"\n                    return False\n                else:\n                    print \"Agama.
Flow finished successfully\"\n                    data = result.getData()\n                    userId =
data.get(\"userId\") if data != None else None\n                    \n                    if userId == None:\n
print \"Agama. No userId provided in flow result.\"                        \n
self.setMessageError(FacesMessage.SEVERITY_ERROR, \"Unable to determine identity of user\")\n
return False\n                    \n                    authenticated =
CdiUtil.bean(AuthenticationService).authenticate(userId)\n                    \n                    if not
authenticated:\n                        print \"Agama. Unable to authenticate %s\" % userId\n
return False\n            except:\n                print \"Agama. Exception: \", sys.exc_info()[1]\n
return False\n\n            return True\n\n\n    def prepareForStep(self, configurationAttributes,
requestParameters, step):\n        \n        if not CdiUtil.bean(FlowUtils).serviceEnabled():\n
print \"Agama. Please ENABLE Agama engine in auth-server configuration\"\n            return False\n\n
if step == 1:\n            print \"Agama. Prepare for Step 1\"\n\n            session =
CdiUtil.bean(Identity).getSessionId()\n            if session == None:\n                print \"Agama. Failed
to retrieve session_id\"\n                return False\n                \n            param =
session.getSessionAttributes().get(self.cust_param_name) \n            if param == None:\n                print
\"Agama. Request param '%s' is missing or has no value\" % self.cust_param_name\n                return False\n
\n            (qn, ins) = self.extractParams(param)\n            if qn == None:\n                print \"Agama.
Param '%s' is missing the name of the flow to be launched\" % self.cust_param_name\n                return
False\n            \n            try:\n                bridge = CdiUtil.bean(NativeJansFlowBridge)\n
running = bridge.prepareFlow(session.getId(), qn, ins)\n                \n                if running == None:\n
print \"Agama. Flow '%s' does not exist!\" % qn\n                    return False\n                elif
running:\n                    print \"Agama. A flow is already in course\"\n                    \n
print \"Agama. Redirecting to start/resume agama flow '%s'...\" % qn\n                \n
CdiUtil.bean(FacesService).redirectToExternalURL(bridge.getTriggerUrl())\n            except:\n
print \"Agama. An error occurred when launching flow '%s'. Check jans-auth logs\" % qn\n                print
\"Agama. Exception: \", sys.exc_info()[1]\n                return False\n            #except
java.lang.Throwable, ex:\n            #    ex.printStackTrace() \n            #    return False
\n            return True\n        \n    def getExtraParametersForStep(self, configurationAttributes, step):\n
return None\n\n    def getCountAuthenticationSteps(self, configurationAttributes):\n        return 1\n\n    def
getPageForStep(self, configurationAttributes, step):\n        # page referenced here is only used when a flow
is restarted\n        return \"/\" + CdiUtil.bean(NativeJansFlowBridge).scriptPageUrl()\n\n    def
getNextStep(self, configurationAttributes, requestParameters, step):\n        return -1\n\n    def
getLogoutExternalUrl(self, configurationAttributes, requestParameters):\n        return None\n\n    def
logout(self, configurationAttributes, requestParameters):\n        return True\n\n# Misc routines\n\n    def
configProperty(self, configProperties, name):\n        prop = configProperties.get(name)\n        return None
if prop == None else prop.getValue2()\n\n    def setMessageError(self, severity, msg):\n        facesMessages =
CdiUtil.bean(FacesMessages)\n        facesMessages.setKeepMessages()\n        facesMessages.clear()\n
facesMessages.add(severity, msg)\n        \n    def extractParams(self, param):\n\n        # param must be of
the form QN-INPUT where QN is the qualified name of the flow to launch\n        # INPUT is a JSON object that
contains the arguments to use for the flow call.\n        # The keys of this object should match the already
defined flow inputs. Ideally, and   \n        # depending on the actual flow implementation, some keys may not
even be required \n        # QN and INPUTS are separated by a hyphen\n        # INPUT must be properly URL-
encoded when HTTP GET is used\n        \n        i = param.find(\"-\")\n        if i == 0:\n            return
(None, None)\n        elif i == -1:\n            return (param, None)\n        else:\n            return
(param[:i], param[i+1:])\n",
      "enabled": false,
      "revision": 1,
      "moduleProperties": [
        {
          "value2": "interactive",
          "value1": "usage_type"
        },
        {
          "value2": "ldap",
          "value1": "location_type"
        }
      ],
      "scriptType": "PERSON_AUTHENTICATION",
      "name": "agama",
      "modified": false,
      "configurationProperties": [
        {
          "hide": false,
          "value2": "customParam1",
          "value1": "cust_param_name"
        }
      ],
      "baseDn": "inum=BADA-BADA,ou=scripts,o=jans"
    }
  ],
  "start": 0,
  "totalEntriesCount": 12
}
```

## Test Suite Navigation

# of failed tests: 0/100
(0.00%)

# of skipped tests: 0/100
(0.00%)

# of passed tests: 100/100
(100.00%)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 |
| 37 | 38 | 39 | 40 | 41 | 42 |
| 43 | 44 | 45 | 46 | 47 | 48 |
| 49 | 50 | 51 | 52 | 53 | 54 |
| 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 |
| 67 | 68 | 69 | 70 | 71 | 72 |
| 73 | 74 | 75 | 76 | 77 | 78 |
| 79 | 80 | 81 | 82 | 83 | 84 |
| 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 |
| 97 | 98 | 99 | 100 | | |

| | |
|---|---|
| Test 14 : And assert response.length != null | 0.000275 |
| Test 15 : And assert response.entries[0].scriptType == 'PERSON_AUTHENTICATION' | 0.00642 |

Scenario: [3:26] **Fetch the first three person custom scripts**

| | |
|---|---|
| **Test 16 : * def mainUrl = scriptsUrl** | **0.000025** |
| Test 17 : Given url mainUrl + '/type' | 0.000315 |
| Test 18 : And header Authorization = 'Bearer ' + accessToken | 0.000108 |
| Test 19 : And path 'person_authentication' | 0.000062 |
| Test 20 : And params ({ limit: 3}) | 0.005015 |
| Test 21 : When method GET | 0.077479 |
| **Test 22 : And print response** | **0.00183** |
| Test 23 : Then status 200 | 0.000008 |
| Test 24 : And assert response.entries.length == 3 | 0.011098 |
| Test 25 : And assert response.entries[0].scriptType == 'PERSON_AUTHENTICATION' | 0.000772 |

Scenario: [4:38] **Search person custom scripts given a serach pattern**

| | |
|---|---|
| **Test 26 : * def mainUrl = scriptsUrl** | **0.000011** |
| Test 27 : Given url mainUrl + '/type' | 0.00017 |
| Test 28 : And header Authorization = 'Bearer ' + accessToken | 0.000048 |
| Test 29 : And path 'person_authentication' | 0.000026 |
| Test 30 : And params ({ limit: 3,pattern:'fido2'}) | 0.002163 |
| Test 31 : When method GET | 0.043963 |
| **Test 32 : And print response** | **0.00078** |
| Test 33 : Then status 200 | 0.000012 |
| Test 34 : And assert response.entries.length <= 3 | 0.00313 |
| Test 35 : And assert response.entries[0].scriptType == 'PERSON_AUTHENTICATION' | 0.000158 |

Scenario: [5:51] **Create new Person Script**

| | |
|---|---|
| **Test 36 : * def mainUrl = scriptsUrl** | **0.000016** |
| Test 37 : Given url mainUrl + '/type' | 0.000212 |
| Test 38 : And header Authorization = 'Bearer ' + accessToken | 0.000057 |
| Test 39 : And path 'person_authentication' | 0.00004 |
| Test 40 : When method GET | 0.066942 |
| **Test 41 : And print response** | **0.009659** |
| Test 42 : Then status 200 | 0.000014 |
| Test 43 : And assert response.length != 0 | 0.003217 |
| Test 44 : And assert response.entries[0].scriptType == 'PERSON_AUTHENTICATION' | 0.0002 |
| Test 45 : Given url mainUrl | 0.000008 |
| Test 46 : And header Authorization = 'Bearer ' + accessToken | 0.000069 |
| Test 47 : And def testScript = response.entries[0] | 0.000144 |
| **Test 48 : And print "testScript before = "+testScript** | **0.006428** |
| Test 49 : And testScript.inum = null | 0.004258 |
| Test 50 : And testScript.dn = null | 0.002393 |
| Test 51 : And testScript.name = "Test_PERSON_AUTHENTICATION" | 0.003443 |
| Test 52 : And testScript.description = "Test_PERSON_AUTHENTICATION_description" | 0.00292 |
| **Test 53 : And print "testScript after = "+testScript** | **0.003482** |
| Test 54 : And request testScript | 0.000015 |
| Test 55 : When method POST | 0.073985 |
| **Test 56 : And print response** | **0.000824** |
| Test 57 : Then status 201 | 0.00001 |
| Test 58 : Then def result = response | 0.000012 |
| Test 59 : Then set result.name = 'UpdatedQAAddedPersonScript' | 0.002385 |
| Test 60 : Then def inum_before = result.inum | 0.002559 |
| Test 61 : Given url mainUrl | 0.000008 |
| Test 62 : And header Authorization = 'Bearer ' + accessToken | 0.000089 |
| Test 63 : And request result | 0.000004 |
| Test 64 : When method PUT | 0.086897 |
| **Test 65 : And print response** | **0.002138** |
| Test 66 : Then status 200 | 0.000016 |
| Test 67 : And assert response.name == 'UpdatedQAAddedPersonScript' | 0.036418 |
| Test 68 : And assert response.inum == inum_before | 0.013449 |
| Test 69 : Given url mainUrl + '/' +response.inum | 0.002804 |
| Test 70 : And header Authorization = 'Bearer ' + accessToken | 0.000113 |
| **Test 71 : And print response** | **0.000623** |
| Test 72 : When method DELETE | 0.044421 |
| Test 73 : Then status 204 | 0.000006 |

Scenario: [6:91] **Delete a non-existing person custom script by inum**

## Test Suite Navigation

# of failed tests: 0/100
(0.00%)

# of skipped tests: 0/100
(0.00%)

# of passed tests: 100/100
(100.00%)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 |
| 37 | 38 | 39 | 40 | 41 | 42 |
| 43 | 44 | 45 | 46 | 47 | 48 |
| 49 | 50 | 51 | 52 | 53 | 54 |
| 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 |
| 67 | 68 | 69 | 70 | 71 | 72 |
| 73 | 74 | 75 | 76 | 77 | 78 |
| 79 | 80 | 81 | 82 | 83 | 84 |
| 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 |
| 97 | 98 | 99 | 100 | | |

| | |
|---|---|
| **Test 74 : * def mainUrl = scriptsUrl** | **0.000014** |
| Test 75 : Given url mainUrl + '/1402.66633-8675-473e-a749' | 0.002222 |
| Test 76 : And header Authorization = 'Bearer ' + accessToken | 0.000102 |
| Test 77 : When method DELETE | 0.051093 |
| **Test 78 : And print response** | **0.000336** |
| Test 79 : Then status 404 | 0.000006 |

Scenario: [7:99] **Get a person custom script by inum(unexisting person script)**

| | |
|---|---|
| **Test 80 : * def mainUrl = scriptsUrl** | **0.000012** |
| Test 81 : Given url mainUrl + '/inum/53553532727272772' | 0.002131 |
| Test 82 : And header Authorization = 'Bearer ' + accessToken | 0.000063 |
| Test 83 : When method GET | 0.042601 |
| **Test 84 : And print response** | **0.000238** |
| Test 85 : Then status 404 | 0.000005 |

Scenario: [8:108] **Get a person custom script by inum**

| | |
|---|---|
| **Test 86 : * def mainUrl = scriptsUrl** | **0.000016** |
| Test 87 : Given url mainUrl + '/type' | 0.000331 |
| Test 88 : And header Authorization = 'Bearer ' + accessToken | 0.000102 |
| Test 89 : And path 'person_authentication' | 0.000037 |
| Test 90 : When method GET | 0.118615 |
| **Test 91 : And print response** | **0.008395** |
| Test 92 : Then status 200 | 0.000011 |
| **Test 93 : And print response.entries[0].inum** | **0.002649** |
| Test 94 : Given url mainUrl + '/inum/'+response.entries[0].inum | 0.002263 |
| Test 95 : And header Authorization = 'Bearer ' + accessToken | 0.000072 |
| **Test 96 : And print request** | **0.000803** |
| Test 97 : When method GET | 0.038356 |
| **Test 98 : And print response** | **0.000986** |
| Test 99 : Then status 200 | 0.000013 |
| Test 100 : And assert response.scriptType == 'PERSON_AUTHENTICATION' | 0.004444 |