BITBYBIT

BITBYBIT

COS301: SOFTWARE ENGINEERING

TWITTER SUMMARISER: ARCHITECTURAL REQUIREMENTS

By submitting this assignment I confirm that I have read and am aware of the University of Pretoria's policy on academic dishonesty and plagiarism and I declare that the work submitted in this assignment is my own as delimited by the mentioned policies. I explicitly declare that no parts of this assignment have been copied from current or previous students' work or any other sources (including the internet), whether copyrighted or not. I understand that I will be subjected to disciplinary actions should it be found that the work I submit here does not comply with the said policies.

Tlholo Koma	20575085
Isheanesu Dzingirai	20536951
Jonah Gasura	20592061
Gabriel Shoderu	16186258
Batsirai Dzimati	20456078

Contents

List of Figures

1 Architectural Structure

1.1 Architectural design strategy

The main strategy implemented for the development of the Twitter Summariser architecture is the design based on quality requirements strategy, whereby the initial step is the identification of the system quality (non-functional) requirements and then design the architecture based on these established requirements.

1.2 Architectural quality requirements

The following quality requirements exist in order from highest to lowest priority.

- **QR1: Reliability**
  - Reliability is the most important quality requirement and is focused on how well the system maintains a certain level of quality over time and thus is the consistency of the subsystems within the system.
  - Reliability is crucial for the following subsystems:
    - Keyword Search:
      - A logged in user should always be allowed to search for a particular keyword on the home page
      - Tweets must always be returned for every keyword for which there are tweets that are linked to that particular keyword
      - Filters and sort criteria applied should always be correctly applied in the results returned from the search
    - Report Generation:
      - If tweets are successfully returned for a particular keyword, then a report should always be returned when the generate report action is triggered
    - Managing reports:
      - A user should always have access to reports they have created
      - A user should always have access to reports that have been shared to them
      - Any modifications made to a report should always be reflected when these changes are saved.
      - Any deleted reports should no longer be accessible once a user has deleted them
    - Sharing reports:
      - A report shared to a user needs to be in the same state that it was shared in when that user receives that report
- **QR2: Performance**
  - Performance is significant to ensure the system's ability to meet certain timing requirements.
  - Performance is crucial for the following subsystems:
    - Keyword Search:

        - Tweets should be correctly returned relatively quickly when a user searches for a particular keyword
    - Report Generation:
        - Once “generate report” is triggered, if tweets exist, the report must be returned in a reasonable amount of time
    - Exploring Reports:
        - Published reports should be returned to each user’s explore page in a reasonable amount of time
    - Sharing reports:
        - Sharing a report should be reflected in a reasonable amount of time.

- **QR3: Security**
  - Security is significant to ensure confidentiality, integrity and availability throughout different aspects of the Twitter Summariser system.
  - Security is crucial for the following subsystems:
    - Authentication:
        - The system should only allow registered users to log in to use the system – if unregistered, the user must first sign up and then log in to access the system. This ensures only authorised access to the Twitter Summariser system
    - Report Generation:
        - A user should be the only one with access to the draft report they generate unless they share it with other users and only then may those other people access these drafts
    - Managing reports:
        - The Owner of a report should be the only user with full access rights to that report – viewing, editing, publishing, “unpublishing” and deleting
        - The Editors of a report should only be allowed to view and edit that particular report
        - The Viewers of a report should only be allowed to view that particular report
    - Sharing reports:
        - The owner of a report should be the only one allowed to share a report that they have created that is not published
        - The owner of a report is the only one able to enable someone else to be an editor for that same report by sharing it to them for “Viewing Editing”
        - A viewer of a report shared to them may only view that report

- **QR4: Usability**
  - Usability is focused on user support and how easy it is for a user to perform certain tasks.
  - Usability is crucial for the following subsystems:
    - Keyword Search:
        - A user should not struggle to enter a keyword to search for tweets
        - A user should easily be able to apply filtering and sorting for the tweets returned for the keyword they search for

        - The user is limited in the sorting and filters the apply to prevent them from applying anything that does not exist and thus preventing faults
        - A user must be informed when any kind of loading is taking place to fetch the results for the search to prevent uncertainty
    * Managing reports:
        - A user should be able to easily edit documents they are authorised to edit
        - A user should be able to easily delete reports they are authorised to delete
    * Exploring reports:
        - A user should be able to easily explore published reports

- **QR5: Availability**
  - Availability is significant to ensure the system is always ready to perform particular tasks when it is required. This requirement is closely related to the Reliability requirement.
  - Availability is crucial for the following subsystems:
    * Authentication:
        - If a user is registered and they enter valid credentials when logging in, the system should always allow them to access the system
        - A valid user should never be denied access into the system when their credentials have been used to sign up
    * Exploring reports:
        - Publicly available should always be available for all Twitter Summariser users to explore

- **QR6: Modifiability**
  - Modifiability is significant to ensure the adaptability of the system and to minimize the risk of making changes to and within the system.
  - Modifiability is crucial for the following subsystems:
    * Keyword Search:
        - The subsystem should allow for changes to easily be made to the sort and filter options without negatively impacting any other elements of the system

1.3 Architectural styles

The system makes use of the following two (2) main architectural styles:

- **AS1: Layered Architecture**
  - The layered architecture consists of the following layers in the context of the Twitter Summariser:
    - Presentation Layer:
      - The Presentation layer mainly consists of the system's user interface that a user directly interacts with.
    - Business Layer:
      - The presentation layer consists of the components mainly involved in the main processing operations that take place in the system. These components include:
      - **Twitter API:** This is the interface provided by the Twitter application to access real Twitter data.
      - **Semantic Analysis Module:** This is the component responsible for assisting with semantic analysis operations when report text is automatically generated by the system
      - **Serverless Computing Service:** This is the service responsible for using the relevant modules and services to implement certain functionality within the system.
      - **Twitter Summariser API:** This is the main system API that handles user requests and triggers the necessary functions to be executed based on those requests
    - Database Layer:
      - The database layer mainly consists of the main system database where all the relevant data the system will need over time is stored and as well as a ”spare” backup database with the same data.
- **AS2: Client-Server Architecture**
  - The client-server architecture consists of the following layers in the context of the Twitter Summariser:
    - Client Side:
      - The client side of the system consists of the frontend element of the system that deals with direct user interaction.
    - Server Side
      - The server side of the system consists of the backend element of the system that deals with the main system processing and data access.

1.4 Architectural patterns

The system makes use of the following main architectural patterns to implement the chosen architectural styles:

- **AP1: Model View ViewModel (MVVM)**
  - The MVVM pattern is a specific type of the Model View Controller (MVC) pattern whereby the entry point for the system is the View unlike traditional MVC whereby the entry point of the system is the controller. MVVM clearly separates the frontend and backend of the system and allows them to interact with one another through the link known as the ViewModel. This pattern consists of the following layers in the context of the Twitter Summariser:
    - Model:
      - The database layer of the architecture is implemented as the model of the system in MVVM which consists of the database that holds all the system data.
    - View
      - The presentation layer of the architecture is implemented as the view of the system in MVVM which consists of the system user interface that the client directly interacts with.
    - ViewModel
      - The business layer of the architecture is implemented as the ViewModel of the system in MVVM which consists of the system's API that manages interactions with the relative services and modules as well as the Twitter API.
- **AP2: Active Redundancy**
  Within the database Layer of the system, the system implements the active redundancy pattern whereby there is an "active" backup of the original system database. This pattern is essential to achieve the most important quality requirement of the system - reliability - which is achieved by ensuring that in the event that some data cannot be retrieved from the main database, the backup will replace that entry and the system can still access the relevant data with minimum delay
- **AP3: Interceptor Validator**
  Within the Business Layer of the system, the system implements the interceptor pattern whereby there exists a "Validator" that cross checks and evaluates all data coming into the system. This pattern is essential to improve security of the system as data is not immediately stored before proper validation and in the case that there are attempts to make unauthorised access to the system this validator can "intercept" and prevent this action. The Twitter Summariser API is the validator for this system.

1.5 Architectural design

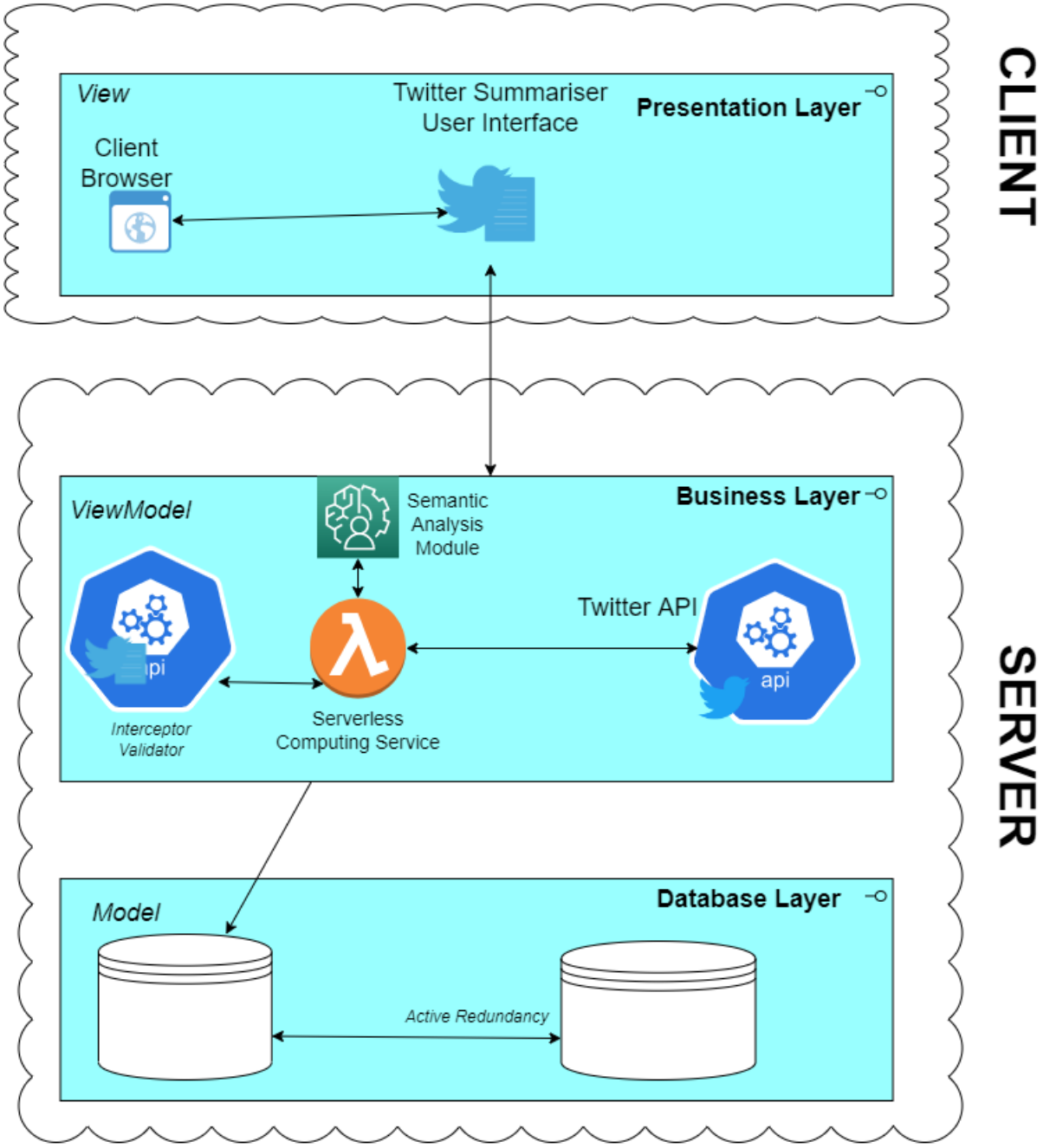

Figure 1: Twitter-Summarise Architectural Pattern

1.6 Architectural constraints

The main system architecture constraints include:

- User requests are restricted to components within the architecture
- System requests and responses are restricted to components within the architecture
- The client and server components of the Twitter Summariser must not negatively impact one another when modifications are made to the respectively
- The client perspective should be completely unaware of the logical components on the server side.

2 Technology Choice

2.1 Backend

The backend will comprise of functions written in Nodejs and TypeScript, which will then be run on AWS Lambda. AWS Lambda is the perfect tool to use as it allows the project to meet the requirements of being server-less, while also being well scalable enough to allow for potentially many requests and processing of data at a time. Node.js, with TypeScript, will make API calls to the Twitter API, and pre-process the data from the response before it goes to the NLP libraries/module for analysing, after which the results obtained will be further processed to allow for the report creation to be sent to the front-end. Twitter API, and pre-process the data from the response before it goes to the NLP libraries/module for analysing, after which the results obtained will be further processed to allow for the report creation to be sent to the front-end.

Pro:

- Scalable

Con:

- free tier is limited

2.2 AI Module

The optional AI module which is separate from the backend will be coded in Python (using AWS Comprehend as the cloud-based ML service). Python was chosen over JavaScript because in the system design it was stated that the AI module should be written in python. AWS Comprehend will be used along with Python instead of TensorFlow and PyTorch because it has a powerful machine learning model that can be used without any machine learning experience. This will be beneficial as most of the team members have zero to little experience in machine learning and natural language processing algorithms. Since the duration of the project is small, it is better we spent more time developing the Twitter Summarizer and ensuring it meets all the specified requirements than spending most of the limited amount of time we have developing and training models. Using AWS Comprehend will greatly assist in completing our system optional requirement

2.3 Frontend

The available technology choices for the user interface implementation were React and Vue. The chosen technology for the frontend interface of the system was React as it has the following advantages over Vue:

Pros:

- The current Twitter user interface is built on React and thus supports this system as it is closely related to Twitter and thus contributes to the system's consistency.
- React is highly compatible with Node.js and the backend of the system will mainly be implemented using Node.js and it is essential that the front-end implementation is compatible to the backend.

- React was developed before Vue and thus there exist more resources for research purposes when implementing the system.

Cons:

- React uses a syntax extension known as JSX that limits development progress.
- The technology is constantly changing and thus can become a barrier over time.

React implements the View and ViewModel aspects of the Model View ViewModel architecture which is the main architectural patter used in the system's architecture.

2.4 Database

The technology used for database is AWS DynamoDB. AWS DynamoDB is a NoSQL database service offered by Amazon that supports key-value and document data structure.

Pros

- DynamoDB is scaleable.
- DynamoDB ensures data security as it backs up data.

Cons

- DynamoDB does not have no triggers.
- The concept of foreign keys does not exist in DynamoDB.
- DynamoDB does not have server-side scripts.

Because the architecture includes Active Redundancy, DynamoDB is a suitable database service that can create a separate copy of the database automatically to achieve reliability.