

Architecture

August 1st, 2023

Group Members





				
Priyul Mahabeer	Ashir Butt	Jaimen Govender	Dharshan Pillay	Edwin Sen-Hong Chang
u20421169			u19027487	u20424575

Table of Contents

Group Members.....	1
Table of Contents.....	2
Architectural Design Strategy.....	3
Architectural Quality Requirements.....	3
Security.....	3
Reliability.....	4
Scalability.....	5
Usability.....	5
Maintainability.....	6
Chosen Architectural Design Based on Architectural Quality Requirements.....	7
Component-Based Architecture (CBA).....	7
Microservices Architecture.....	7
Three-Tiered Architecture.....	8
Architectural Design Diagram.....	9
Architectural Constraints.....	10

Architectural Design Strategy

We aim to apply the strategy of design based on quality requirements for our system. To ensure the success of this project and our system, we have analyzed the client's specification requirements and user stories, resulting in a substantial and comprehensive collection of quality requirements for the system. These quality requirements serve as a crucial cornerstone in the architectural design process.

Architectural Quality Requirements

Security

A key pillar of the architectural design for our service request system is its robust and comprehensive security framework, which is structured around a role-based access control system. This ensures that users can only execute actions and access information that aligns with their roles and responsibilities within the system, thereby maintaining integrity and confidentiality. The role-based model is multi-tiered, allowing internal users to possess one or more roles that dictate their permissible operations within the system. The tiering of roles is hierarchically organized, enabling different levels of access to system functionalities. This ensures that users with higher roles can execute advanced operations, whereas lower-level roles are restricted to basic functionalities. Such a strategy effectively mitigates the risk of accidental or intentional misuse of the system, and safeguards its core operations. In the context of data visibility, users are constrained to viewing only the tickets pertinent to their assigned group(s), thereby further strengthening the security framework of the system. This stratified approach to data access is essential in preventing unauthorized access to sensitive data, and maintains the principle of least privilege, a cornerstone of information security. To fortify these security measures, our system employs token-based authorization techniques. This token-based system provides secure, controlled access to our APIs, thus enabling secure transactions of data between client-side and server-side components of our system. Each token is generated and encrypted using advanced cryptographic methods, ensuring that they cannot be tampered with

or forged. Moreover, tokens are time-limited and are constantly refreshed, reducing the window of opportunity for any potential security breach. In addition to token authorization, the system incorporates stringent user authentication mechanisms. This involves multi-factor authentication that validates the identity of a user by requiring multiple evidence of authenticity, further reinforcing the overall system security. Security, as an architectural quality requirement, is therefore woven into every facet of our system's design and operation, providing a reliable and secure platform for managing service requests.

Reliability

The architectural design for our service request system is underpinned by an unyielding commitment to reliability. At the heart of our design philosophy is the ambition to build a system that minimizes downtime and swiftly mitigates any operational disruptions, thus providing users with a seamless and uninterrupted service experience. The system has been structured around a robust error-handling mechanism that quickly identifies and rectifies any anomalies in the system operations. This proactive error management approach is paired with stringent continuous monitoring and performance logging strategies that ensure optimal operation at all times. To further enhance the system's reliability, the architecture separates the client and internal staff portals. This ensures the functioning of one does not impact the other, thus providing an added layer of operational resilience. In a scenario where the internal staff portal encounters issues, clients can still access their portal without hindrance, and continue to interact with the system seamlessly. This separation aids in isolating potential issues, simplifies troubleshooting, and reduces downtime. In conclusion, our unwavering focus on reliability shapes our architectural design strategy at every level. We are committed to providing a service request system that users can rely on, irrespective of the scale of demand or operational challenges. Our holistic approach to reliability ensures that our system is resilient, adaptive, and dependable, providing a seamless user experience at all times.

Scalability

One of the key tenets guiding the architectural design of our application is the imperative to address scalability as a critical non-functional requirement. The goal is to construct an architecture that can seamlessly and efficiently accommodate a growing user base and increasing demands on system resources without compromising on performance or user experience. To achieve this paramount objective, our application is built upon a foundation of cutting-edge technologies, including MongoDB, Node.js, and Angular. These technologies have been carefully selected for their ability to foster a scalable ecosystem that can easily adapt to the evolving needs of our users and business requirements. While our application has been architected as a microservices-based system, we shall delve deeper into this architectural paradigm later in the document. Suffice it to say that the microservices architecture is another integral facet of our scalability strategy. By dividing our application into discrete and loosely coupled microservices, we create a cohesive ecosystem wherein each service can independently scale based on its specific demands. This modularity ensures that system growth can be orchestrated in a fine-grained manner, granting us unparalleled flexibility in managing our application's resources. In conclusion, our commitment to addressing scalability as a non-negotiable non-functional requirement manifests through the astute integration of MongoDB, Node.js, and Angular technologies. The orchestrated marriage of these tools, complemented by our microservices and component-based architectures, aspires to forge an application capable of handling increased user load and expanding business horizons, all while delivering a seamless and unparalleled user experience. As we progress in our architectural journey, these scalable underpinnings shall form the bedrock upon which we construct a robust and dynamic application.

Usability

In our relentless pursuit of creating an exceptional user experience, the notion of usability stands as a fundamental pillar guiding the design and development of our application. As a non-functional requirement of paramount importance, usability encapsulates our dedication to

crafting an intuitive, user-friendly, and delightful interface that fosters ease of interaction and a sense of empowerment for our valued users. The triumvirate of technologies comprising MongoDB, Node.js, and Angular underpinning our application provides the ideal foundation for translating our usability vision into reality. Together, they enable us to orchestrate an interface that seamlessly marries efficiency with elegance, all while ensuring a fluid and responsive user journey. Angular, the cornerstone of our front-end development, embraces a component-based architecture that fosters modularity and reusability. This architecture not only streamlines the development process but also facilitates consistent design patterns, leading to a coherent and familiar user interface. The result is an interface that exudes coherence and efficiency, where users can effortlessly navigate, comprehend, and interact with the application's features. Beyond the technological underpinnings, the forthcoming discussion will delve deeper into our component-based architecture, which further underscores our commitment to usability. By decoupling the application into self-contained, reusable components, we create a harmonious ecosystem where each component is thoughtfully designed to serve a specific function or feature. This meticulous organization ensures that users are presented with a clear and uncluttered interface, enhancing their cognitive ease and elevating the overall usability of our application.

Maintainability

Amidst our pursuit of creating a cutting-edge application that sets new standards in performance and user experience, we remain ever-mindful of the indispensable attribute of maintainability. As a non-functional requirement of paramount importance, maintainability underpins our commitment to developing an application that is not only exceptional in its present form but also sustains its excellence over time, with ease of updates, modifications, and enhancements. The harmonious amalgamation of MongoDB, Node.js, and Angular at the core of our application serves as a testament to our dedication to creating a maintainable ecosystem. Each technology brings with it a unique set of attributes that bolsters our ability to swiftly adapt and refine the application as needed, with minimal disruption to ongoing operations. Our architectural choice of a three-tiered design reinforces the principle of maintainability. While a

detailed exploration of this architecture is reserved for a later section, it is vital to acknowledge its impact on maintainability. By separating the presentation, business logic, and data layers, our application's components can be maintained independently, allowing developers to address specific functionalities or troubleshoot issues without interfering with other parts of the system.

Chosen Architectural Design Based on Architectural Quality Requirements

In order to satisfy our architectural quality requirements of security, reliability, scalability, usability, and maintainability, we have strategically chosen an architectural design that is characterized by three interdependent, yet distinct structures: Component-Based Architecture, Microservices Architecture, and a Three-tiered Architecture. These design paradigms will collaboratively function to ensure the fulfillment of our quality requirements.

Component-Based Architecture (CBA)

We have chosen a Component-Based Architecture (CBA) for our service request system, aiming to enhance usability and maintainability. The CBA approach fosters the creation of a system built from self-contained, reusable, and interchangeable modules, each designed to execute a specific functionality. This architecture significantly enhances the system's usability by streamlining the interface and maintaining a consistent design pattern, thereby delivering a smooth, intuitive, and efficient user experience. Moreover, the decoupling of the application into independent components contributes to the system's maintainability. Developers can individually update, modify, or troubleshoot components without impacting the rest of the system, thereby simplifying maintenance, reducing costs, and facilitating quick responses to change requests or identified issues.

Microservices Architecture

The choice to implement a Microservices Architecture is primarily driven by the need to ensure scalability and reliability. Microservices allow our system to be divided into loosely coupled

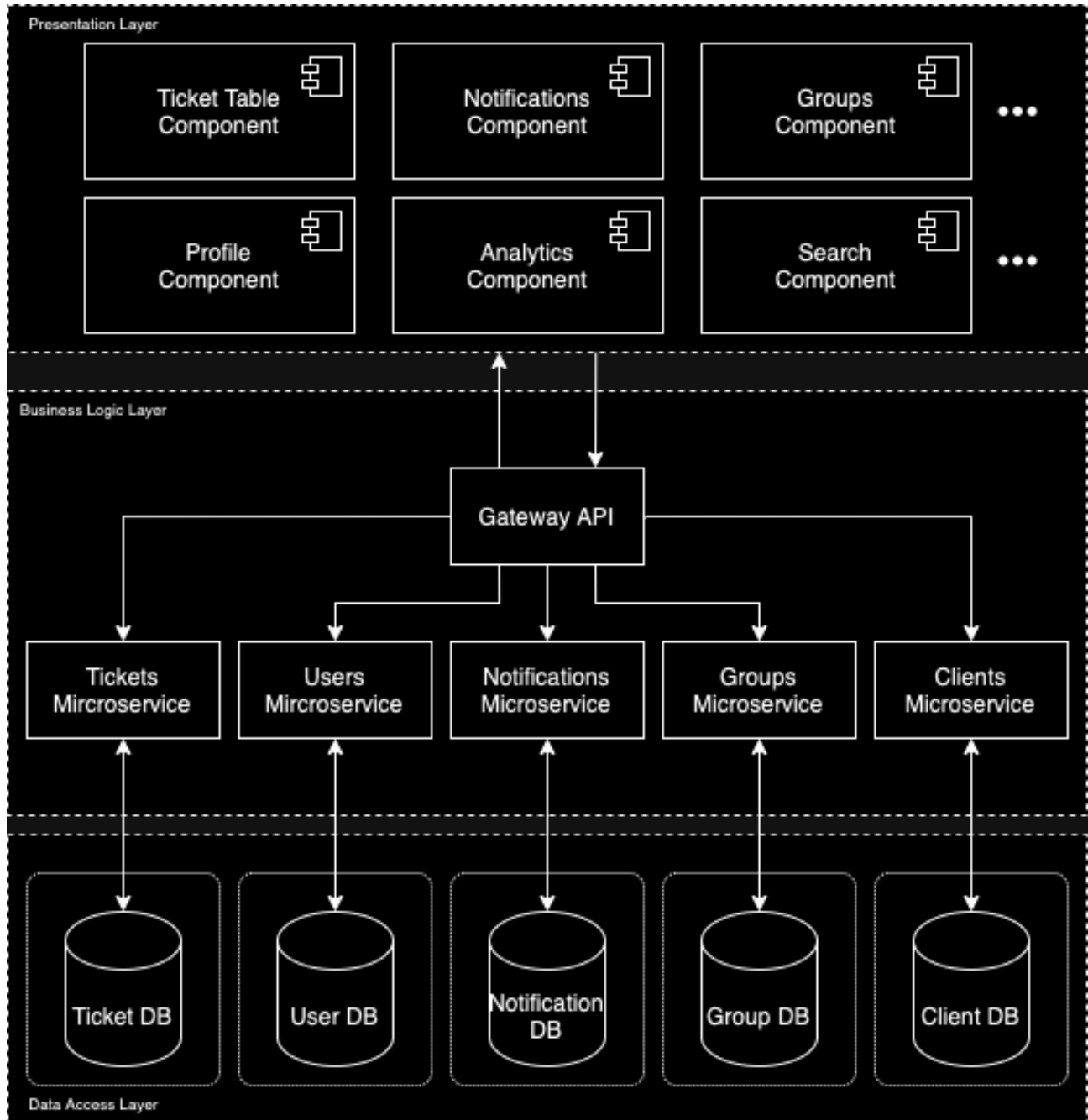
services, each performing a unique function. This architectural style facilitates independent scaling of services based on their individual demands, providing flexibility in resource allocation and ensuring efficient system growth. In addition to scalability, microservices enhance system reliability. Since each service operates independently, a failure in one service does not directly impact the others. This compartmentalization of potential errors significantly mitigates the risk of widespread system disruptions, thus augmenting system reliability.

Three-Tiered Architecture

Our system will also employ a Three-tiered Architecture, an approach that essentially separates the presentation, business logic, and data access layers. This architectural design enhances security, maintainability, and scalability. The clear separation of concerns enhances security by restricting access to the data layer and business logic from the presentation layer, thereby limiting the potential for unauthorized access or malicious activities. From a maintainability perspective, the decoupling of the system into distinct layers allows for isolated updates, modifications, or troubleshooting, without affecting the rest of the system. This isolation simplifies maintenance and fosters rapid and efficient responses to identified issues or required changes. Finally, in terms of scalability, this architecture enables the independent scaling of each layer based on its specific demands, ensuring optimal resource allocation and system performance.

In conclusion, our chosen architectural design, built on the principles of Component-Based Architecture, Microservices Architecture, and a Three-tiered Architecture, aligns with our commitment to meeting the architectural quality requirements of security, reliability, scalability, usability, and maintainability. This carefully curated combination of architectures aims to ensure a robust, user-friendly, and enduring service request system.

Architectural Design Diagram



Architectural Constraints

Client Requirements:

User interface constraints: The client may have specific user interface design guidelines or branding requirements that need to be followed.

Deployment Constraints:

Hosting environment: The system may need to be deployed on a specific infrastructure, such as on-premises servers or a cloud platform

Scalability requirements: The system may have to handle a high number of concurrent users or accommodate future growth.

Performance and Security:

Performance constraints: The system may have specific performance requirements, such as response time targets or throughput expectations. These constraints can influence architectural choices regarding caching, load balancing, or data storage optimizations.

Security requirements: The system has specific security requirements, such as encryption, access control.

Technology Choices

MongoDB:

MongoDB is a NoSQL database that offers flexibility in data modeling and scalability. It stores data in a JSON-like format (BSON) and provides rich querying capabilities.

Express.js

Express.js is a web application framework for Node.js. It provides a minimalist and flexible approach to building web applications and APIs. Express.js simplifies routing, middleware management, and request handling, making it well-suited for creating server-side components in our architecture.

Angular:

Angular is used for building dynamic and interactive web applications. It follows the MVC architectural pattern and provides a declarative approach to building user interfaces. Angular simplifies data binding, dependency injection, and component-based development, enhancing code maintainability and reusability.

Node.js:

Node.js is a runtime environment that allows JavaScript to run on the server-side. It provides an event-driven, non-blocking I/O model, making it efficient for handling concurrent requests. Node.js is known for its scalability and high performance, making it a good fit for building the backend components of your application.