Virtual Consultant

Design Report For Software Maintainability

Version 1.1

<05/10/2021>

Aditya Chandrasekhar (U1923951A) : Project Manager / Lead Developer
Aratrika Pal (U1922069F) : Backend Developer
Shruthi Srinivas (U1923611G) : Backend Developer / Release Engineer
Chong Zhe Ming (U1920757K) : Frontend Developer
Khush Kothari (U1922279J) : Frontend Developer
Kushal Sai Gunturi (U1923232F) : QA Manager
Yi Jia Xin, Joceline (U1920057J) : QA Engineer
Lim Yun Han Darren: (U1921275J) : Frontend Developer

VERSION HISTORY

Version #	Implemented By	Revision Date	Approved By	Approval Date	Reason
1.0	Kushal Sai Gunturi	23/9/2021	Aditya Chandrasekhar	25/9/2021	Initial Risk Management Plan draft
1.1	Aditya Chandrasekhar	02/10/2021	Aratrika Pal	05/10/2021	Final Project Plan, proof-read estimations and final formatting

Template Version: 11/30/06

TABLE OF CONTENTS

DESIGN STRATEGIES **1**
PRE-DEVELOPMENT PLANNING PHASE 1
DEVELOPMENT PROCESS 1
CORRECTION BY NATURE 1
CORRECTIVE SOFTWARE MAINTENANCE 1

1 DESIGN STRATEGIES

1.1 PRE-DEVELOPMENT PLANNING PHASE

Extensibility and Scalability were the core tenets of the application, and were the primary considerations even before the development of the application had started. We wanted to ensure that our application supported a growing feature-set and user-base as a primary consideration, not an afterthought. For this, we tried to anticipate the type of features that would later be extended, and also the ability for the backend to handle greater traffic flows.

Virtual Consultant was built with the idea in mind that different regions would require different feature-sets. Additionally, we wanted to extensively prototype and experiment with different variations of features to provide the best possible final-product for the user. To achieve this objective of flexibility, we ensured the system was built with OOP principles, to ensure extensibility throughout the process.

Although we would begin the application-pilot in Singapore, the objective is to expand to other regions. Therefore, Scalability of the application is important to ensure the application can handle hundreds of thousands more new users at once.

Micro-Service architecture, where the system consisted of smaller components (independent from each other) was chosen for our project, since we would easily be able to add or remove features without breaking the whole application.

1.2 DEVELOPMENT PROCESS

Application-testing was one of the crucial steps in our development process. The feedback from the QA team from testing (who also collected market real-world use data about the application) was constantly consulted with to refine the application.

This meant that the product went through several significant changes during the SDLC. To ensure our objectives of Extensibility and Scalability and facilitate these changes, we wrote the components for the system in a re-usable manner with separation of concerns. As we made changes to the system, the modules could be swapped easily without having to rewrite the whole system.

1.3 CORRECTION BY NATURE

We would use the feedback and the error-information to fix issues during testing. The different types are as follows:

1. Corrective Software Maintenance: fix bugs reported by the users during the testing process.

2. Preventive Software Maintenance: Perform fault-tolerance to detect and fix errors and ensure reliability.

1.4 ENHANCEMENT BY NATURE

The Extensibility of the software demands changes to be made in the system by adding new features over time. The different types are as follows:

1. Adaptive Software Maintenance: Based on the focus groups and public response, we will need to tweak the functionality of the application frequently. Flexibility for tweaking is ensured with this approach.

2. Perfective Software Maintenance: This approach is to ensure that the system is maintained after the final release.

1.5 MAINTAINABILITY PRACTICES

Maintainability is defined as the ease of modifying the system software or component to fix problems, improve performance or add features. We have utilized the following practices to ensure maintainability of the system:

1. Readable Code: Comments and OOP practices such as modularity, abstraction are used to make the code easy to follow and readable.

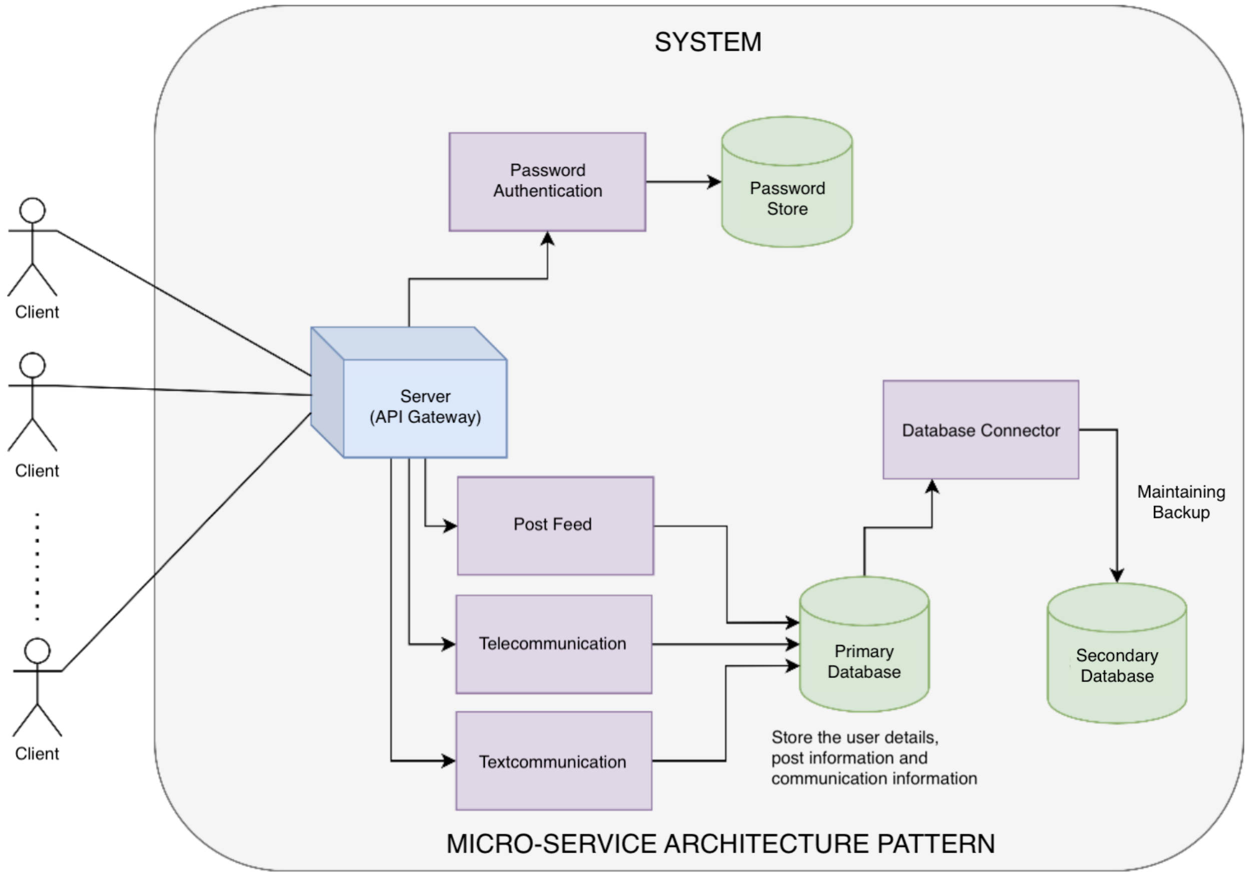
2. Version Control: Git Version control is used to make branches for features, and pull requests are used to integrate features as they are completed. This permits multiple people to work on different parts of the code simultaneously, while keeping track of the history and permitting the reversal to different checkpoints in case of errors. For documentation, SVN is used.

3. Standardized Documentation: Documents are written by following IEEE practices, which permits developers to track changes and understand documentation with ease.

4. Modularity: The micro-service architecture is utilized to implement components in an as decoupled manner as possible. This permits for easy changes in features to be rolled out with updates.

2 DESIGN ARCHITECTURE OF SYSTEM

The design architecture of the system is show in the image below.



Some of the important components in the micro-service architecture can be independently assessed with the use of REST API calls. Some of the components are discussed more in detail in the subsections below:

2.1 DATABASE

The NoSQL Database MongoDB is used as our data-storage platform. It stores the encrypted user-info for authentication, post data and chat history. NoSQL was used in the system to ensure greater flexibility in data-storage. For example, chats with unstructured data might vary significantly depending on the use-case; NoSQL is a great fit for such data.

2.2 TELECOMMUNICATION SERVICE

The two main forms of Telecommunication are chats and video conferencing. Both of these involve simultaneous communication between 2 parties. The hosting for the service is done on the server, and depending on the preference, either of the telecommunication services are offered to the user.

2.3 API SERVER GATEWAY

This gateway is used to pass information to other components. It is primarily used for the purpose of communication in the system architecture. For example, it is able to call the other necessary components of the architecture based on the URL entered.

3 SOFTWARE CONFIGURATION MANAGEMENT TOOLS

The different tools used by the development team for version control and tracking changes is as follows:

3.1 MEDIAWIKI

MediaWiki was utilized because it provides permission-based document editing options to allow authorized developers to edit their respective documents, while others are able to view them. It also permits multiple developers to work on the documentation simultaneously, and provides user-friendly options for viewing and edition documentation on multiple devices.

3.2 GOOGLE DRIVE

Google Drive was used to share documentation and notes with other group members. It was chosen because it is almost universally used and all members were comfortable with it. Additionally, we were able to make quick edits to documents with google docs,

3.3 GITHUB

Github is used for git version control. As mentioned previously, it permits branching, merging and pull requests which permits the developers to work on different features simultaneously and integrate them together.

3.4 SVN

SVN allows developers to track changes in documentation, and revert to previous versions in case the need arises.