Design Report on Software Maintainability

Version 1.0

By: Pranwanth, Anant, Japhet, Ming Da, Weiyi, Jovian, Trung

**Document Change Record**

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Description of Change** | **Approved by** | **Date** |
| 0.10 | Initial Template | Lim Zhe Hong, Jovian | 12/10/2022 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Contents**

[1. Design Strategies 3](#_Toc116486758)

[1.1. Before Development 3](#_Toc116486759)

[1.2. Developing 3](#_Toc116486760)

[1.3. Correction by Nature 3](#_Toc116486761)

[1.4. Enhancement by Nature 3](#_Toc116486762)

[2. Architectural Design Patterns 4](#_Toc116486763)

[3. Software Configuration Management Tools 4](#_Toc116486764)

[3.1. MediaWiki 4](#_Toc116486765)

|  |  |  |
| --- | --- | --- |
| 3.1. | MediaWiki | 4 |
| 3.2. | GitHub | 5 |
| 3.3. | OneDrive | 5 |

# Design Strategies

## Before Development

Before any development, we first began by coming up with a core theme, before deciding on how to expand upon that theme. As such, we also predicted who our target audience would be, and how many average users can we expect.

Next, we figured out the logical components that go into the software, and determined what kind of technical stack we would need to enable a smooth experience for the users, as well as how do we scale up in terms of hardware.

We decided to adopt the Model-View-Controller design pattern when developing TuGucci? This separation of model and view allows us to work on the user interface separately from the core functionality simultaneously.

## Development

At each step of development, we made sure to use the most relevant and cohesive technologies we can to ensure ease of scalability, and maintainability.

By using Firestore, a non-SQL database, we made sure that the data writing, storage, and fetching requirements of our software were met, while not having to keep track of relations while adding more data.

We also used TailwindCSS, allowing for robust and consistent style of our software, regardless of the type of device or browser used to access our software.

Using VueJS, we are able to expand upon ideas by inheriting and creating new components, without disrupting the functionality of previously working logic.

Using Dockerized Flask, we are able to ensure scalability of our API needed to perform sentiment analysis, allowing multiple users to send a request simultaneously without affecting the experience of other users.

## Bugfixes

While working on the software, we would frequently access it, to get a sense of what the user experience would be like. We would look out for the following:

* + 1. **Corrective Maintainability**

We would use Google Chrome to simulate opening our software on various screen sizes, at various internet speeds, and monitoring network access. This allows us to spot any bugs for every possible device configuration, and fix those bugs immediately.

* + 1. **Preventive Maintainability**

Each feature and page will be added one at a time, which prevents new features from potentially disrupting the process of previously established features.

* 1. **Maintainability Practices**

To uphold quality in both process and product, we have implemented the following maintainability practices over the course of our project:

* Meaningful naming of files and variables
* Separation of different components
* Comments in our code
* A wiki for documentation
* Version Control

# Architectural Design Patterns

TuGucci? uses the Model-View-Controller (MVC) design pattern.

The Model layer contains all the logic that goes into running the software, including the list of pages, the transitional logic from one page to the next, the API calls to the database and sentiment analysis.

The View layer controls the which backdrop and components are rendered, as well as how, such as the text animations, or the camera animations.

The Controller layer is where user input is received, and parsed into a json request to be sent to our model layer, where the database is then updated. The model layer then sends a signal to the view layer, updating the page with the latest data from the database.

# Software Configuration Management Tools

These are the tools we used in order to keep track of development progress, version control, as well as documenting every piece of the software.

## MediaWiki

MediaWiki is a free and open-source application, which provides a platform for instructions on how to use the software to be nicely presented. It also holds all the documentation that goes into the software development process, such as the software requirements, meeting minutes, and backlogs. This allows the team to easily keep track of what tasks to do, and who does what.

* 1. **GitHub**

GitHub is a source code hosting platform using the distributed version control and source code management Git. With the help of GitHub, the team can easily see who implemented which components of the software. It is also possible to revert major errors due to Git having a robust memory of the different versions of the software, hence being able to get back to the latest working version. GitHub also has a issue reporting system, allowing for the team to create tasks to work on, such as bugs, as well as assign and check the progress of the issue. Git also has a pull request system, allowing for team members to perform code review before any major changes are merged into the main release, thus increasing accountability and reliability.

* 1. **Google Drive**

Google Drive is used as a file storage and for the backup of documents initially created. This allows the team to share a common folder, which is backed up on the cloud. This also allows the team to work together remotely, and simultaneously. Google Drive also has a version control system, as well as a commenting system, allowing for seamless teamwork.