NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Design Report on Software Maintainability for TimeWise

Team 0
Lab group : SSP7

Group members	Roles
Mantri Raghav	Project Manager
Dwivedee Lakshyajeet	Development Lead
Harding James	Back-End Developer
Alex Bernini	Front-End Developer
Mittal Madhav	Release Manager
Xue Xueting	QA Engineer
Koh Hui Ling	QA Engineer
Lek Jie Ling	QA Manager

Version History

Version #	Implemented By	Revision Date	Approved By	Approval Date	Reason
1.0	Xue Xueting	8 April 2020	Lakshyajeet	10 April 2020	First Version
1.1	Koh Hui Ling	8 April 2020	Lakshyajeet	10 April 2020	Chapter 1 updates
1.2	Harding James Sebastian	8 April 2020	Lakshyajeet	10 April 2020	Completed document

Table of Contents

Introduction

Purpose

Software maintenance is to modify and update the software application after delivery to correct faults and to improve performance. It is necessary as it is required in order to fix bugs and errors that can occur in the software system, to improve the performance of the software, to improve functionality of the software so that it is also more compatible and to remove functions that are outdated in the software. On average, 80% of the effort will go to maintenance, and the remaining 20% of the effort will go to the development of a new software.

Scope

There will be four types of software maintenance that will be used to design our software maintenance plan are:

- **Correction Maintenance**
  - **Corrective Maintenance**
    - Addresses to the defects in the software that occurs due to the errors and faults in a design, logic and code of the software.
  - **Preventive Maintenance**
    - Software change made to prevent the occurrence of future errors
- **Enhancement Maintenance**
  - **Adaptive Maintenance**
    - Aims at updating and modifying the software to keep it up-to-date and usable
  - **Perfective Maintenance**
    - When updating the software system to improve its value according to the user demands

Software Evolution (Lehman's Law)

Law	**Description**
Continuing change	A program that is used in a real-world environment must change or become progressively less useful in that environment.
Increasing complexity	As an evolving program changes, its structure tends to become more complex. Extra resources must be devoted to preserving and simplifying the structure.
Declining quality	The quality of systems will appear to be declining unless they adapt to changes in their operational environment.
Organisational	Over a program's lifetime, its rate of development is approximately constant

stability	and independent of the resources devoted to system development.

Maintainability Analysis

1. **Corrective Maintenance**
   In **TimeWise**, NPM is used to update the packages when the library is changed. By using this, it will be easier to maintain the software as packages in the library will be standardised, which means there will be lesser errors.

2. **Preventive Maintenance**
   After deployment of the application, regular testing and revision are required to be conducted in order to detect any potential errors that may occur in the future and prevent it. In **TimeWise**, React Native is used which allows the easy change of components, as it caters to both Android and IOS.

3. **Adaptive Maintenance**
   In **TimeWise**, Heroku is used for easy adaptation to changing users, since Heroku makes it easy for applications to scale. This allows the application to run smoothly and efficiently despite the amount of users on both Android and IOS.

4. **Perfective Maintenance**
   In **TimeWise**, by using Redux, custom task tags in the future are made easy, as Redux is an enhanced version of the Model-View-Controller (MVC) architecture. Since Redux has separation of concerns, it separates concerns better than MVC as the controller is split into two parts, that is Actions and Reducers. Redux also ensures modularity and ease of adding new features.

Design Strategies for software maintainability

The design strategy adopted by the TimeWise time involves keeping the software modular and maintaining separate components of the system. For example instead of having the web scraper built into the application and running on the users device, it was decided to have it as a server and worked as a RESTful API. It should also be noted that the database helps to link the data from these individual components (for example, user events and timetable events are formed from separate components of the system).

Design Philosophy

The team is using Heroku in order to facilitate the deployment of the web scraper and the database. Heroku servers scale automatically depending on its usage at the current time, meaning the cost and upkeep in terms of maintainability is lower than it otherwise would be.

The team also aimed to make the code easy to understand and easily maintainable. It is common knowledge that to do this often requires that subsystems have low coupling but high cohesion as this helps to maintain high modularity within a code base. This also helps to make it easy to define the functionality of individual subsystems and how they interface with each other. Service and components of the system were also designed in such a way as to make components easily reusable and therefore components could potentially be used by other systems or subsystems later on down the line.

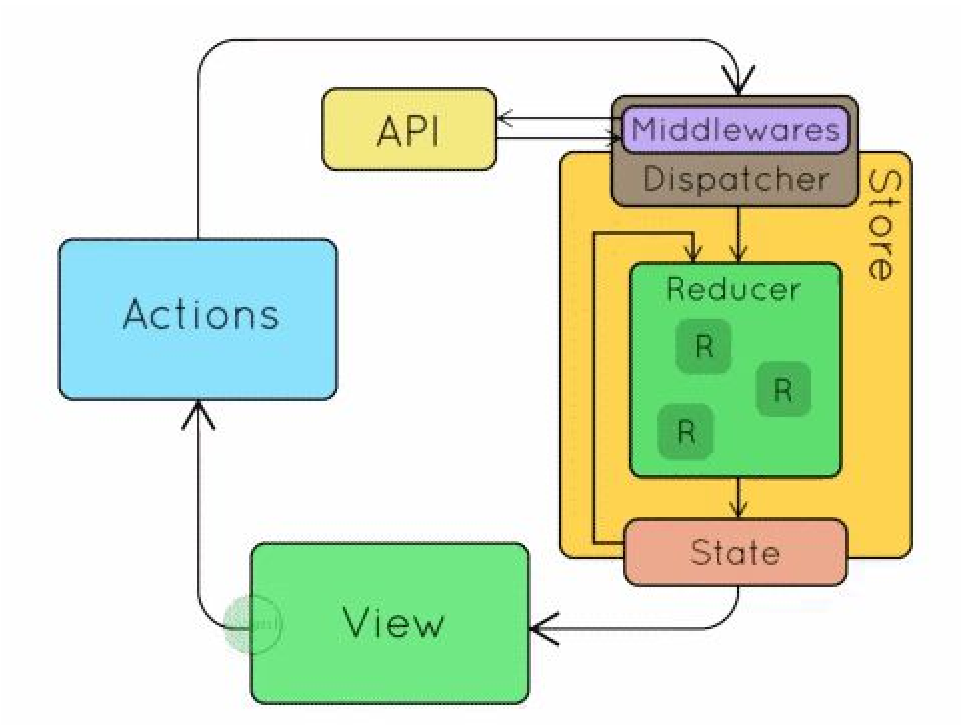
Our overall philosophy should help make the system easier to understand, edit and update. In turn this should help to improve the maintainability of the code base.

Architectural Design Pattern

The redux architecture is used for our project and is composed of 4 main components actions, reducer, store and view.

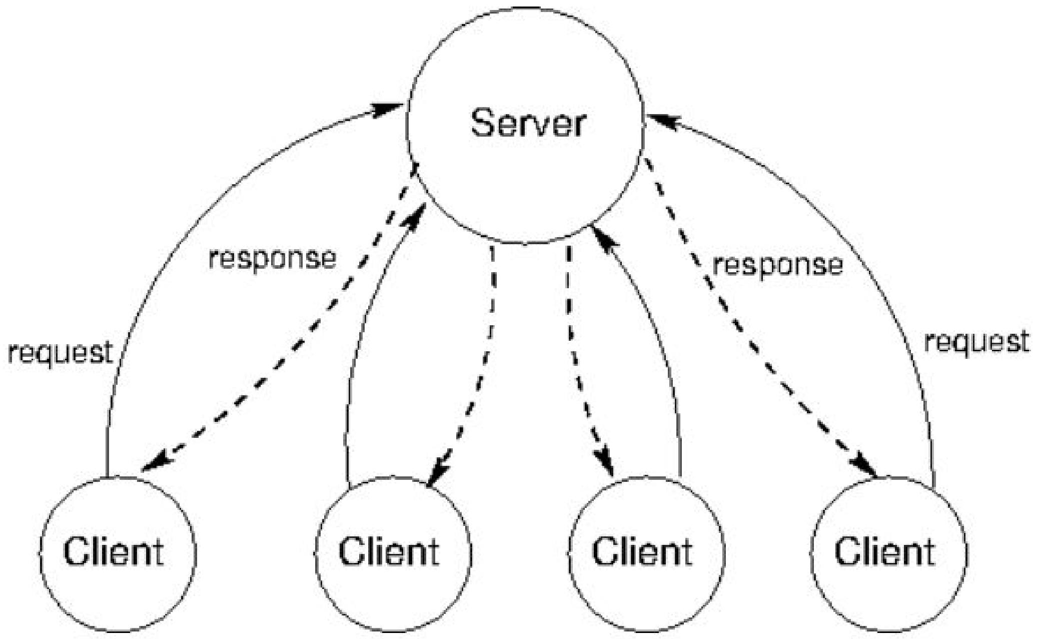
1. Store: The store contains all the data for the entire client side application. The data is broken down into different collections depending on the concerns. E.g. User data collection, task data collection, etc.
2. View: The view is the UI which shows the data from the store and can be interacted with to launch actions.

3. Actions: Actions are functions which are called upon interaction with the View and dispatch requests to the backend API. It can be noticed that **Redux is the link between the front-end and the back-end**. The data received after an API call is sent to a reducer.
4. Reducer: Reducers are functions which take in data and modify the Store using this data.



Redux is an enhanced version of the model view controller architecture and effectively separates individual components of a system better than MVC as the controller is in two parts (actions which call the backend and reducers which change the data in the store). Separating the components helps to enforce low coupling and high cohesion. Therefore, code is easier to edit and understand since it encourages interfaces to be well defined. Not only this but the Store is not coded manually but is generated dynamically so it does not change as more code is added. As such, this helps to manage maintainability as bugs are easier to find and fix, as well as features being easier to implement.

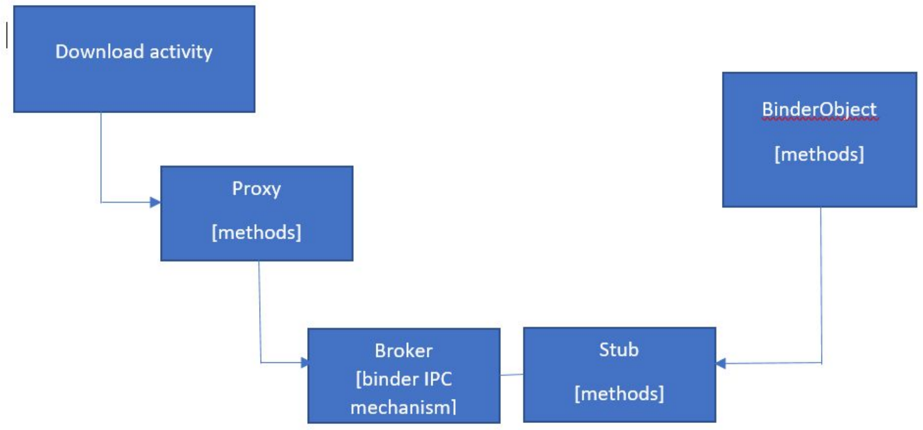
However, the app itself is in fact a component of another design pattern, the client and server model. The idea is that the server performs some kind of task for the client which either performs better on a server or requires a third party to control and retain data. In our implementation the database and web scraper are the servers and the application the client. The application uses HTTP POST requests to get data from the web scraper and inserts data into the database or queries the database. These are both servers which perform tasks or maintain data so the application can function. Using such a model once again separates components and elements of a system, in this case it can be used to improve performance since the team has control over the hardware or platform that the servers are run on. Also, fixes and updates can be done more easily as components of the overall system are isolated so become less complex.



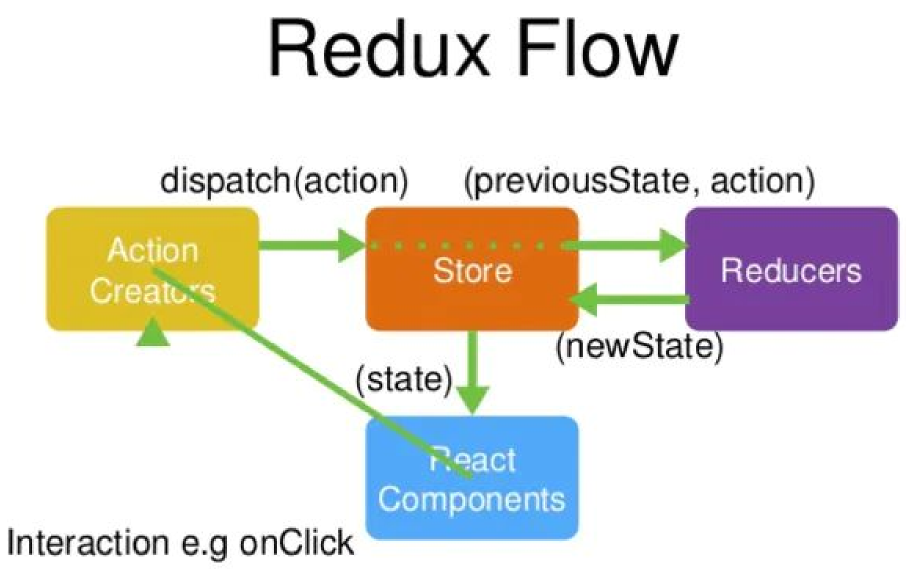
Mid-Level Design Patterns

Mid-Level Design Patterns are templates which provide a reusable solution to a known software development problem and can be adapted to different problems as necessary within a given context. The following design patterns have been included to enhance the Timewise system.

1. Broker Design Pattern



2. Redux Design Pattern



Testing and Code Revision

The TimeWise team implemented 4 methods of testing the quality of the system produced, which are described in this section. These help retain the system's maintainability and identify any necessary changes to the system.

User Acceptance Testing
Real users are used to test the final release to make sure it both accomplishes the task required for users within a real world scenario and also checks the design of the application to make sure it is liked by users as well as useful for users.

System Testing
Testing the whole system together to check that it works properly and functions adequately for the task it was made for. This can be validately by checking it against functional and non-functional requirements.

Integration Testing
Combining elements of the system to check that they work together as needed and as expected. It is also necessary in determining whether the elements run efficiently together or cause the application to perhaps “hang”.

Unit Testing
Individual components and methods of the system are checked to make sure its components behave as expected and provide the correct outputs given specific inputs.

Configuration Management Tools

Below are all the tools used by us for Configuration Management:

Tools	Reason
MediaWiki	**MediaWiki** is a wiki software that collects and organizes knowledge to make it available to the Team. Team members will update the content in MediaWiki for all of our project's documentation. It serves as a documentation repository so that each member can access and collaborate.
TortoiseSVN	**TortoiseSVN** is a Subversion client, implemented as a Microsoft Windows shell extension, that helps programmers manage different versions of the source code for their programs. This SVN will be used for code submissions as requested by the client.
GitHub	**GitHub Inc.** is a web-based hosting service for version control using Git. It is mostly used for computer code. It offers all of the distributed version control and source code management functionality of Git as well as adding its own features. We will be using GitHub as our code repository and easier for team members to share codes.
Google Drive	**Google Drive** is a file storage and synchronization service developed by Google. It will be used as a storage space for all of our documents as well as to allow collaborative work to be done. We will first use Google drive to work on the documents among team members. After we finish all the documents, it will then be moved to MediaWiki for assessment.

Node Package Manager	**NPM** is an online repository for the publishing of open-source Node.js projects, it is also a command-line utility for interacting with said repository that aids in package installation, version management, and dependency management. It is used in our project for managing libraries. The package.json file ensures all developers use the same version of the libraries