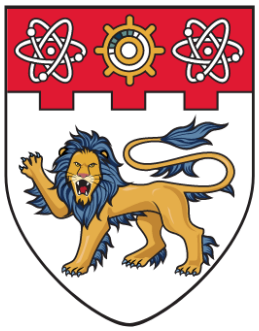


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

Design Report on Software Maintainability for Planit

Team Members	Role
Ryan Tan Jinn En	Project Manager
Neo Yong Tai	Development Lead
Lee Yu Sheng Daniel	Back-End Developer
Mamuduri Paulani	Front-End Developer
Frankie Ye Htet Myat	Release Engineer / Manager
Chen Xueyao	QA Manager
Kundu Koushani	QA Engineer

Name of Design Team: **Team Syan**
School of Computer Science and Engineering, Nanyang Technological University

Version History

Version #	Implemented By	Revision Date	Reason for Revision
1.0	Ryan Tan	22/10/2021	First Version, Template only
1.1	Ryan Tan	23/10/2021	Added Design Pattern, Software Configuration Tools
1.2	Ryan Tan	24/10/2021	Revised Version

TABLE OF CONTENTS

1. Introduction

1.1. 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software, to improve functionality of the software so that it is also more compatible and to remove functions that are outdated in the software.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

1.2. 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

1.3. 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 operation environment.
Organisational stability	Over a program's lifetime, its rate of development is approximately constant and independent of the resources devoted to system development

2. Maintainability Analysis

1. Corrective Maintenance

In **Planit**, 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 fewer errors.

2. Preventative 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. React is used to allow the easy change of components.

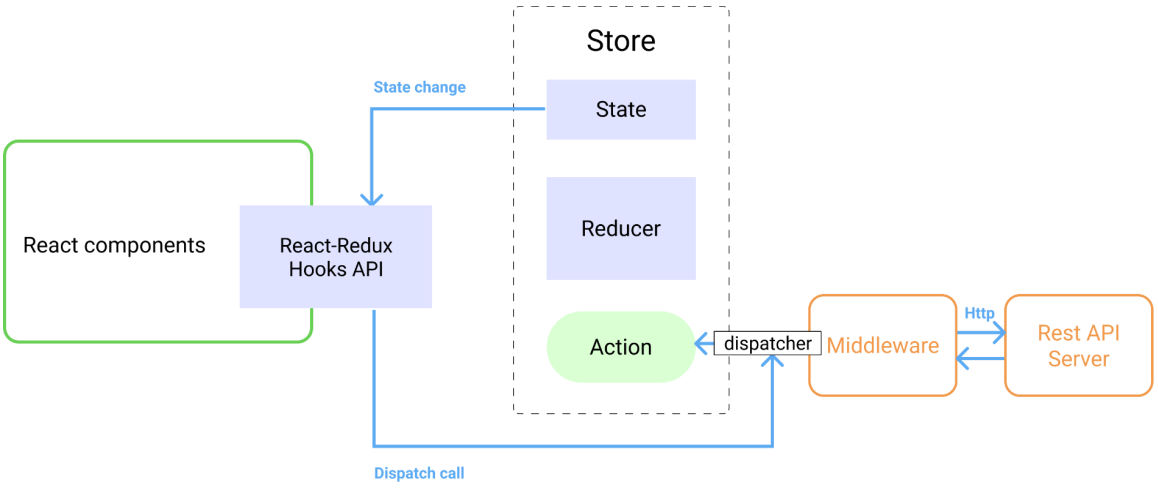
3. Adaptive Maintenance

In **Planit**, AWS is used for its easy adaptation to changing users, since AWS allows applications to scale with ease. This allows the application to run smoothly and efficiently despite the volume of users.

4. Perfective Maintenance

In Planit, by using Redux, custom notifications in the future are easily doable, 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.

Following is the Architectural Design Pattern:



3. Software 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**