The Team

Design Report on Software Maintainability

E-Tendance - Facial-recognition based attendance taking system

The Team

Li Shanlan
Zeng Jinpo
Akshaya Muthu
Simon El Nahas Christensen
MN Shaanmugam
Cao Ngoc Thai

Submitted to:

NANYANG TECHNOLOGICAL UNIVERSITY

VERSION HISTORY

Version #	**Implemented By**	**Revision Date**	**Approve By**	**Approval Date**	**Reason**
1.0	*Zeng Jinpo*	*11/07/2019*	*Li Shanlan*	*11/07/2019*	Initial Draft
1.1	*Zeng Jinpo*	*11/09/2019*	*Li Shanlan*	*11/09/2019*	Add Design considerations
1.2	Li Shanlan	*11/09/2019*	*Zeng Jinpo*	*11/09/2019*	Evaluation metrics

TABLE OF CONTENTS

1. INTRODUCTION

This report documents the design considerations during the development of E-Tendance, the Facial-recognition based attendance taking system. The report will analyze and discuss the design strategies, use of architectural design patterns, and application frameworks of E-Tendance. Moreover, the maintainability of E-Tendance will be assessed based on a set of Web Application metrics at both system and component levels. The report thereby drives the development of E-Tendance towards better maintainability.

2. PROBLEMS

Due to the limited time to develop E-Tendance, and the initial emphasis on making the product work in a reliable and usable manner, less emphasis is placed on the maintainability of the product. According to Lehman's laws of software evolution, we highlight some of the key difficulty in maintaining the web applications:

(a) Continuing change

The software requirements keep changing according to the environment; in this case, students or tutors may demand different product features to enhance the experience of attendance taking.

(b) Declining quality

The student population continues growing, results in more users of the E-Tendance system. This may result in slower attendance taking process, as the facial recognition has to be compared against much more user images. Thus, the quality of E-Tendance will appear to be declining.

Upon in-depth analysis, we have predicted:

(a) Maintainability

The facial recognition component will require occasional maintenance to ensure faster and more accurate facial recognition, given more user requests.

(b) Maintenance costs

The maintenance costs over the next year is a part-time staff's annual salary, SGD15/hr * 4 hours/weeks * 52 weeks = SGD3,120

(c) System changes

The front-end designs are most likely to be affected according to the user requests, which will consequently result in the way how back-end shall respond to the new API calls. However, such change requests are expected to be infrequent, and will not result in much hassle in responding to the new changes.

3. GOALS

Software maintainability is defined as the degree to which an application is understood, repaired, or enhanced. Hence, the design considerations shall aim to facilitate the process of maintenance, and make it easier to enhance performance or fix bugs in E-Tendance. Consequently, there will be less maintenance cost, and better system efficiency.

With application of law of maintenance to design considerations, we shall achieve:

(a) Separate module to ensure a good modularity, with low coupling and high cohesion
(b) Abstraction with encapsulation and design patterns

The report will entail how we achieve the two goals through the design considerations.

4. DESIGN CONSIDERATIONS

4.1 CLIENT-SERVER ARCHITECTURE

We utilized the client-server architecture: a client makes a request for a service and receives a reply to that request; a server receives and processes a request, and sends back the required response. In this case, we used ReactJs for front-end, and ExpressJS for API server, whereby ReactJS contains data driven components, and is a single source of truth, while ExpressJS is an asynchronous backend.

By separating client and server, the encapsulation of services is guaranteed. Maintenance, such as server upgrades, can be done without affecting clients, as long as the published message interface is unchanged.

Moreover, such modular and extensible design, with single responsibility components, help achieve low coupling and a flexible infrastructure.

4.2 REACT FOR FRONT-END

Through following Reactive design patterns, we achieved a high-order component, whereby a component wraps another component by adding extra functionality or extra properties. The high level of abstraction from the commonly used logic helps to ensure there is minimal repetition of the codes.

Moreover, through having a composition of components, it helped ensure the functionality is added to a component without causing rippling changes throughout the codebase.

Furthermore, React helps achieve a resilient, responsive and elastic front-end.

4.3 EXPRESS FOR BACK-END

Express is a minimal and flexible Node.js web application framework with the provision of a robust set of features for web application.

Through Express, we used a variety of design patterns to ensure good modularity. For example, factory pattern is used to abstract the object creation implementation. Moreover, middleware pattern is used to avoid coupling and ensure flexibility; Both receiver and sender have no explicit knowledge of each other.

5. EVALUATION OF MAINTAINABILITY

To evaluate the maintainability of the application, we utilized metrics defined in "Maintainability assessment of web based application".

5.1 REVIEW OF CONFORMITY TO SERVICE-ORIENTATION DESIGN PRINCIPLES

Metric Name	**Description**	**Conformity Level**	**Software Architect's Review**
Well-define d interfaces	A well specified description of the service that permits consumers to invoke it in a standard way	Medium	Our subsystems communicate with each other through well-defined HTTP APIs.
Loose coupling	Service consumer is independent of the implementation specifics of service provider	High	Our subsystems communicate with each other through well-defined HTTP APIs while keeping the implementation logic only to itself.
Logical and physical separation of business logic from presentation logic	Service functionality is independent of user interface aspects	Medium	The data that our backend system processes on does not depend on user interface's implementation, though there's a pre-agreement that the front-end is taking photos in a certain format.
Highly reusable services	Services are designed in such a way that they are consumable by multiple applications	High	Our backend provides RESTful APIs such that it is able to take requests from multiple applications.

Coarse-grained granularity	Services are business-centric, i.e., reflect a meaningful business service not implementation internals	High	Our backend provides RESTful APIs which have separate functionalities and concerns.

5.2 QUANTITATIVE MAINTAINABILITY ASSESSMENT METRICS

Metric Name	**Description**	**Remark**	**Value**
TotalWebPage#	TotalServerPage# + TotalStaticClientPage#	Small	3
TotalLOC#	TotalServerPageLOC# + TotalStaticClientPageLOC#	Small	~1800
ServerScript#	Total number of server scripts	Very Small	less than 10
ClientScript#	Total number of client scripts	Very Small	less than 10
InterfaceObject#	Total number of interface objects	Small	6
TotalData#	Total number of different data identifiers	Small	21
I/OField#	Total number of form fields + number of I/O data from/to mass storage devices	Small to Medium	3 form fields + 2 webcam video input
TotalConnectivity#	Total number of relationships among web pages = Link# + Redirect# + Submit# + Build# + Include#	Very Small	2
TotalLanguages#	Total number of programming/scripting languages used to implement the system	Very Small	1

These two tables demonstrate that the E-Tendance system is a generally small scale system with simple system architecture. With medium to high conformity to service-orientation design principles and client-server architecture design considerations, this system has good maintainability/easy to maintain.

6. BIBLIOGRAPHY

1. "Software Maintainability - CAST Software." https://www.castsoftware.com/glossary/software-maintainability. Accessed 7 Nov. 2019.
2. "Anatomy of the Client/Server Model - Oracle Help Center." https://docs.oracle.com/cd/E13203_01/tuxedo/tux80/atmi/intbas3.htm. Accessed 9 Nov. 2019.
3. "2019 ReactJS Best Practices & Design Patterns - Medium." 8 Mar. 2019, https://medium.com/@konstankino/2019-reactjs-best-practices-design-patterns-516e1c3ca06a. Accessed 9 Nov. 2019.
4. "Design Principles – React." https://reactjs.org/docs/design-principles.html. Accessed 9 Nov. 2019.
5. "Design Patterns in Express.js - DZone Web Dev." 23 Jun. 2016, https://dzone.com/articles/design-patterns-in-expressjs. Accessed 9 Nov. 2019.
6. "Understanding the Middleware Pattern in Express.js - DZone ...." 13 Jun. 2016, https://dzone.com/articles/understanding-middleware-pattern-in-expressjs. Accessed 9 Nov. 2019.
7. "maintainability assessment of web based application." https://pdfs.semanticscholar.org/cdec/f7e2931e5837142e1ea84d7be3165bdac39d.pdf. Accessed 9 Nov. 2019.