SeniorConnect System
Design and Maintainability Analysis_v2

GruFamily Project Team
Nanyang Technological University
50 Nanyang Avenue
Singapore 639798
22/10/2015

Revision History

Name	Date	A*M D	Reason For Changes	Version
Ma Xiaoxue	14/10/2015	A, M	Draft design analysis	V1.1
Ma Xiaoxue	15/10/2015	A,M	Draft maintainability analysis	V1.2
Ma Xiaoxue	18/10/2015	A	Add in design pattern diagrams	V1.3
Ma Xiaoxue	22/10/2015	M	Finalize report	V2

*A - Added M - Modified D – Deleted

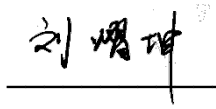
SeniorConnect System
Design and Maintainability Analysis

22/10/2015

Design and Maintainability Analysis Approvals:

22/10/2015

SQA Manager **Date**



22/10/2015

Project Manager **Date**

Contents

1. Introduction

1.1. Purpose

In software, maintenance is defined as the work done to change the product after it is launched for operation. Maintenance is necessary in order to preserve the value of software products over the time. In general, maintenance work is carried in the purpose of meeting new requirements in the changing environment or correcting software defects. A software project with appropriate maintenance will have longer useful life and gain higher users' satisfactory.

For most software projects, maintenance will cost more efforts and time than initial development. Therefore, a design with good maintainability should be adopted during development phase for the ease of maintenance in the future. This report provides a maintainability analysis regarding on the design of SeniorConnect project.

1.2. Scope

The report will analyze various design patterns adopted by SeniorConnect system, including client-server architecture, service-oriented design, peer-to-peer communication, layered architecture, MVC pattern. These design patterns provide a general picture of the technical design of SeniorConnect system.

The analysis will be carried on regarding the maintainability of the system design from four perspectives, i.e. adaptive maintainability, perfective maintainability, corrective maintainability and preventive maintainability. Adaptive maintainability will be the focus as SeniorConnect is a user-centered mobile application and is supposed to be updated over time to meet changing user requirements.

2. System Design

2.1. Technology Adoption

The SeniorConnect application adopts HTML5 hybrid development, web socket, and web services as its technical approach.

2.1.1. Ionic Framework

At client side, Ionic Framework is used for fast development and wide adoption across platform. Ionic Framework is a framework based on Cordova, which allows using HTML5 to build mobile application. Basically, Cordova will run the HTML5 code in a native web view of the mobile device. It also handles system APIs such as camera, microphone, and file system. Ionic Framework provides additional sets of libraries so that building mobile-feel user interface and interaction becomes simple and intuitive.

Using Ionic Framework has the following advantages:

- It is platform independent. The framework supports all major mobile platforms such as iOS, Android, Windows Phone, and Blackberry. Therefore, one piece of code will be deployed to multiple platforms with some minor customization.
- It allows quick development. The development is easy to learn with built-in libraries. For example, constructing a simple application interface with four tabs and contents only takes a few minutes.
- It uses web technologies. The usage of web technologies makes it easier for web developers to pick up mobile development. Also, it reduces learning curve for most programmers.
- It has direct access to native APIs. The framework does not stop developers from using system native functionalities. Instead, all functions are nicely packaged in libraries for easy access. Sometimes the access to native APIs is even easier than native code.

2.1.2. Flask Framework

At server side, we use Flask, which is a Python web framework to construct our web services, host web socket server, and handles data storage and persistence. Flask is a simple yet powerful web framework which has nice integration with major web technologies. Also, it has many extensions that handles web socket, database connection, and security easily.

Using Flask has the following advantage:

- Light and Simple. The framework is the most lightweight framework on Python platform, or even the most lightweight framework in the market. It binds each web service API to a method, which handles the web request. Also, the simplicity makes most programmers pick up the technology pretty fast, enabling fast development and prototype.
- Extensible and Flexible. The framework does not sacrifice extensibility and flexibility despite being simple. In fact, many extensions enables Flask for integration with other web technologies and enhance security. For example, Flask-SocketIO integrates nicely with web sockets. Flask-SQLAlchemy integrates the application with database through object-relational mapping. Flask also has extensions for login, security context, and OAuth.
- Easy testing. Testing web API in Flask can be as simple as hitting the web URL. Also, by integrating with other test frameworks in python, we can simply test functions behind each web services. Testing tools such as SoapUI can also be used.

- RESTful. The way Flask constructs API supports RESTful API structure natively.

2.2. Client-Server Architecture

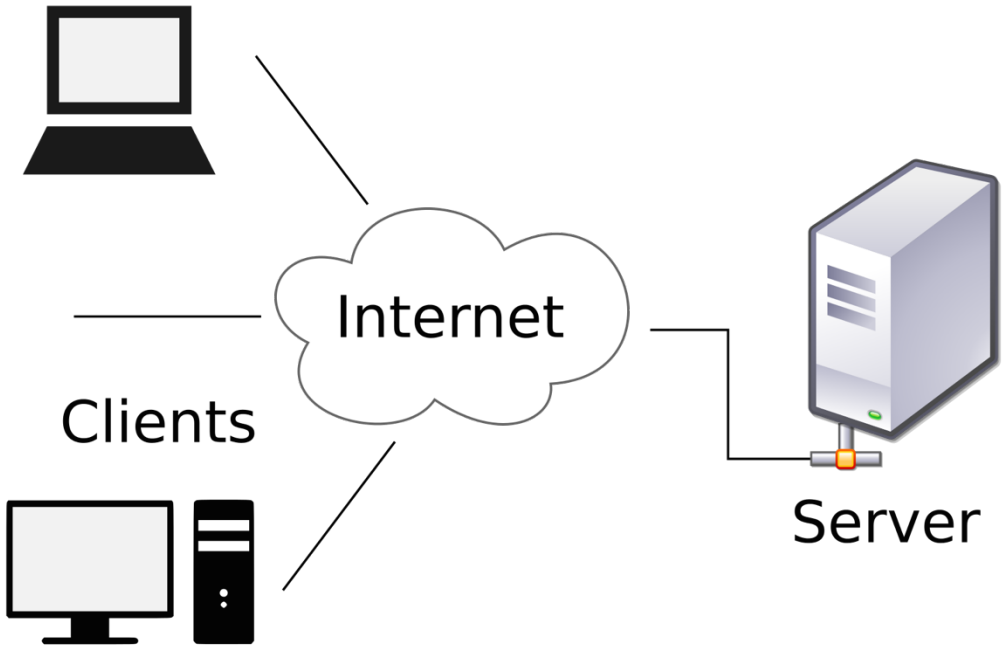


Figure 2.1 – Client-Server System Architecture

The application works in client-server architecture. Basically, it consists of a client running HTML5, either on a mobile device or on a desktop PC, and a web server running in Nginx. All features, except for real-time voice and video call, follow this architecture. However, there are two implementations of such architecture, namely Socket-IO communication and web service calls.

2.2.1. Socket-IO Communication

For messaging function where users send voice message across the network, client and server communicate using Socket-IO. Socket-IO is an event-driven real-time communication technology. At client side, the communication is achieved through Socket-IO library in JavaScript. At server side, the communication is achieved through Flask-SocketIO integration.

Although socket communication is generally considered as bi-directional communication, we still distinguish client and server. Server side is more heavy-weight because it not only handles socket communication with a client in one-to-one messaging, but also acts as a relay for group messaging. Also, server side handles client channel registration and de-registration. The details will be explained below.

2.2.1.1. Channel Establishment

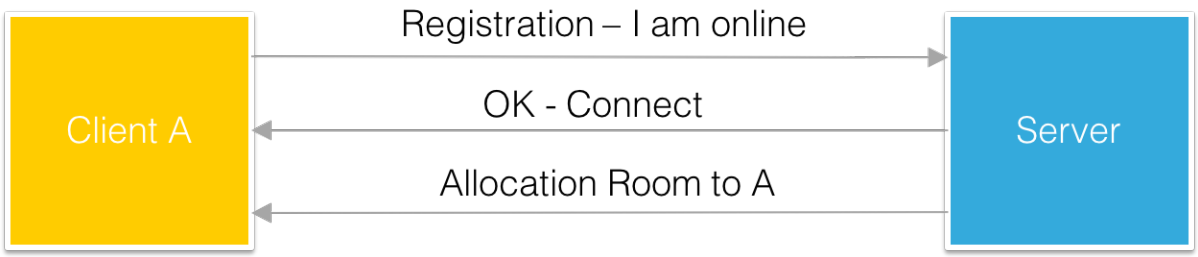


Figure 2.2 – Socket-IO Channel Establishment

Figure 2.2 illustrates channel establishment in Socket-IO communication. The server is always online waiting for client communication. When a client goes online, it will firstly send a connection message to the server. Upon receiving client's request, the server will send acknowledge message to client. Additionally, the server will allocation a room, which is a dedicated channel for communicating with this client, to the client application. Client application will only receive message sent to this room. By doing so, the application separate the communication channels among different clients.

2.2.1.2. One-to-one Messaging

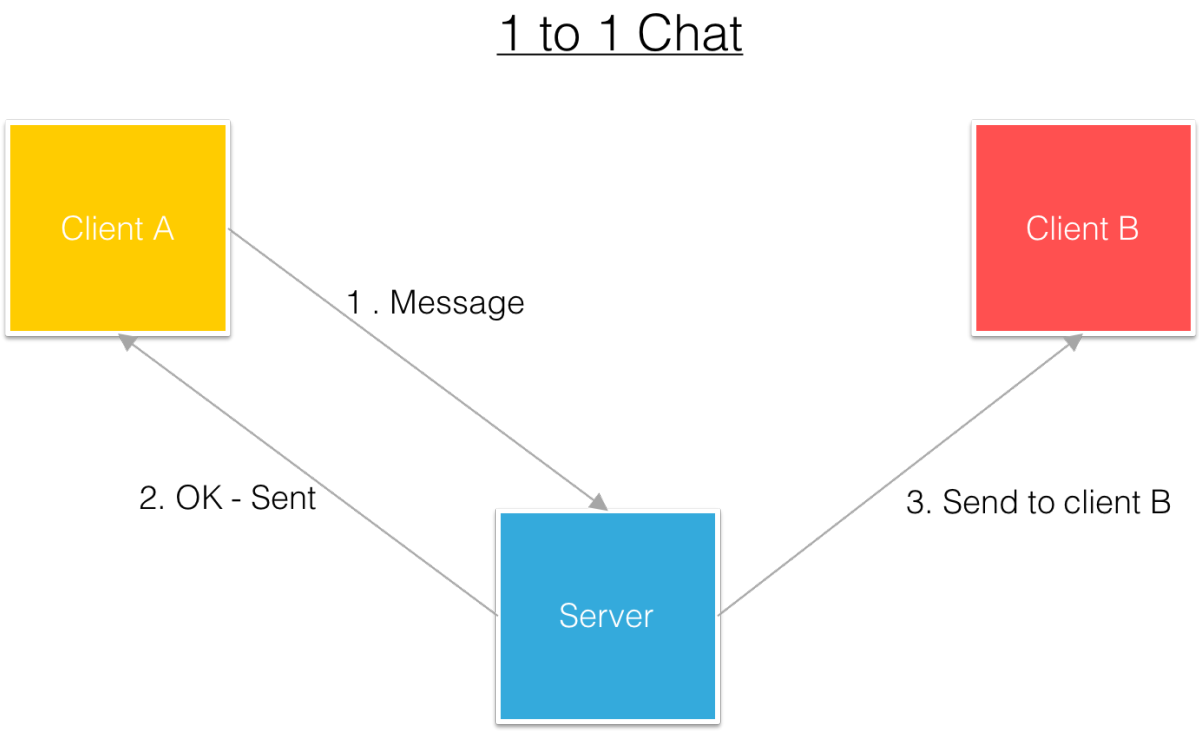


Figure 2.3 – 1-to-1 Messaging

Figure 2.3 illustrates one-to-one messaging between two clients and illustrates the role of server. When client A sends a voice message to client B, client A will send an event to server, together with the message content. Also, it will upload the voice message to server, which is explained later in the next section. Upon receiving the event from client A, the server will store the message content to database, and send an event to the room where client B is in. If client B is online, it will receive the event and also the message content. In the case that client B is offline, the information will be ready for client B's retrieval when client B comes online.

2.2.1.3. Group Messaging

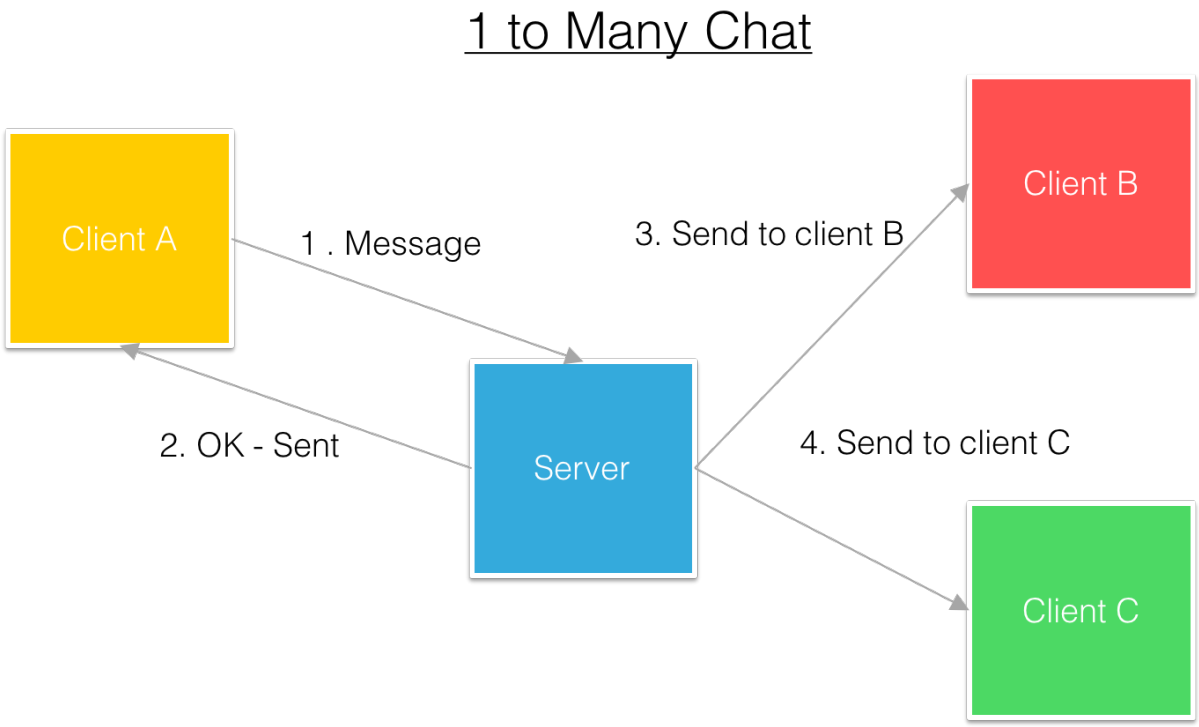


Figure 2.4 – Group Messaging

Figure 2.4 illustrates group messaging in a group. Assuming client A, B, and C are in the same group. When client A sends a voice message to the group, client A will send an event to server, together with the message content. When receiving this message, the server will store the message content to database. Additionally, the server will act as a relay for this group message. The server will discover the members in the group, and send an event, together with the message content, to the room of each member in the group. In other words, group message is achieved by sending the message to all members in the group iteratively, and hence consumes much more server resources.

2.2.2. Web Service Calls

Apart from Socket-IO based communication, all other communication in client-server architecture is achieved through web service calls. At server side, Flask web framework opens up web service endpoints for client application to consume. The web service endpoints follow RESTful API convention, and using JSON as common content format. When client needs to retrieve data, the client simply sends a HTTP GET request to retrieve data from server. When client needs to create/update data, it simply sends HTTP POST/PUT requests to server.

The server also handles resources uploads for photo and voice message using web service calls, and therefore, does not provide resuming from breakpoint functionality. A dedicated web service address is provided for uploading the resources to server using POST method, and the service address is also used to retrieve data using GET method.

2.3 Peer-to-peer Architecture

Real-time voice and video communication follows peer-to-peer pattern, in which server support is necessary for initiation of communication but not necessary in subsequent

communication. The real-time communication in this application is achieved through PhoneRTC, a JavaScript library extended based on WebRTC technology.

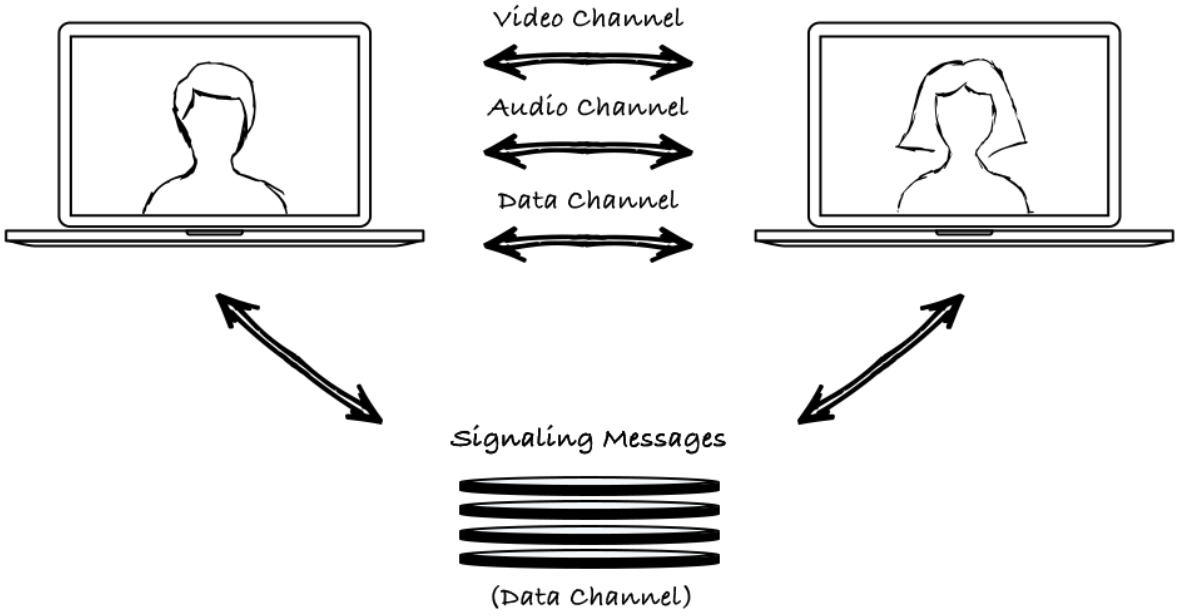


Figure 2.5 – Peer-to-peer Real-time Communication

Figure 2.5 is the architecture of the infrastructure that supports real-time voice call. In a typical flow of voice call, the peer-to-peer connection is established with the help of server, using the SocketIO channel as described above. However, the SocketIO channel used for real-time communication is separated from the one used for messaging. After the channel is established, the SocketIO server becomes transparent, and will not be necessary for the call.

To establish a call, the caller on the left sends a message to callee on the right through SocketIO server at the bottom. The callee accepts the call by sending a message back through the SocketIO server, and in the meantime, initiate the call channel from his/her side. Upon receiving confirmation from the callee, the caller will also establish a call channel. The call channel of the caller and callee will then match to each other, starting the peer-to-peer call. To stop a call, any participating party can send a termination signal.

3. Component Design

3.1. Layered Client Architecture

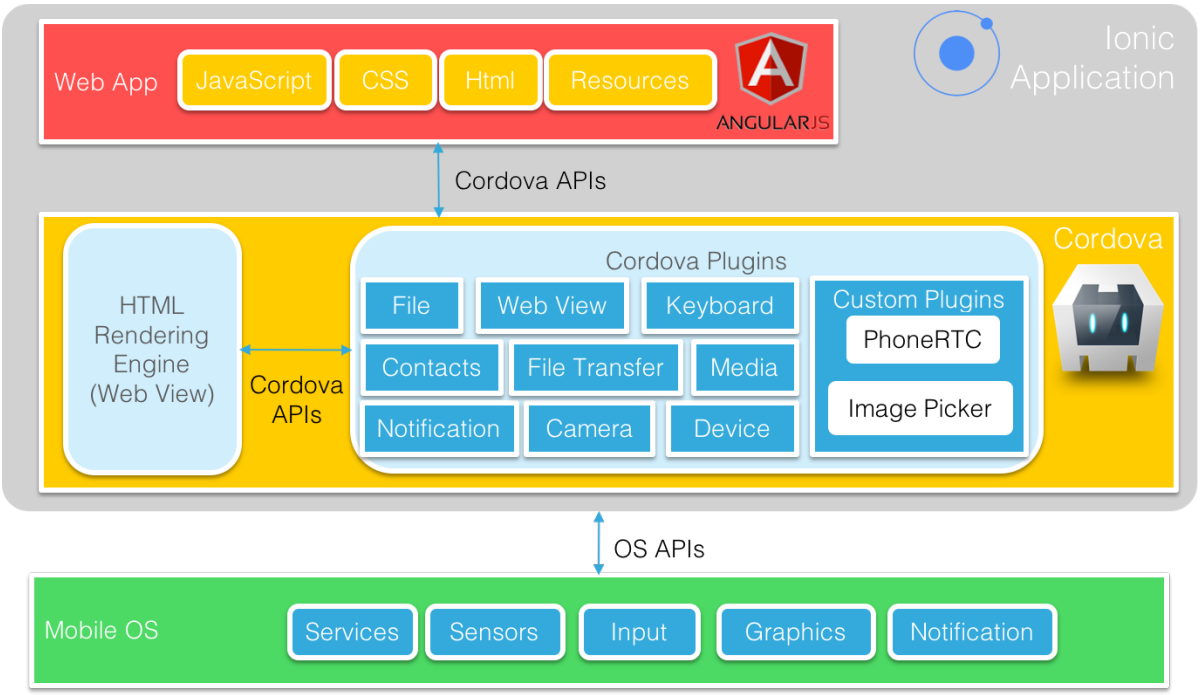


Figure 3.1 – Layered Client Architecture

The client as a whole follows layered architecture as illustrated in Figure 3.1 above. At the bottom, the application is the native OS APIs. The middle layer is the Cordova integration layer which communicates with native OS APIs through various plugins. Also, it connects to the Web App layer which runs in a Web View. In the Web App layer, the application runs in the Web View as a HTML5 web application. The communication with Cordova integration layer is through a series of defined APIs in JavaScript. Ionic Framework also provides libraries in Web App layer so that the development of application can be fast and painless.

3.1.1. Model-View-Controller Web App Pattern

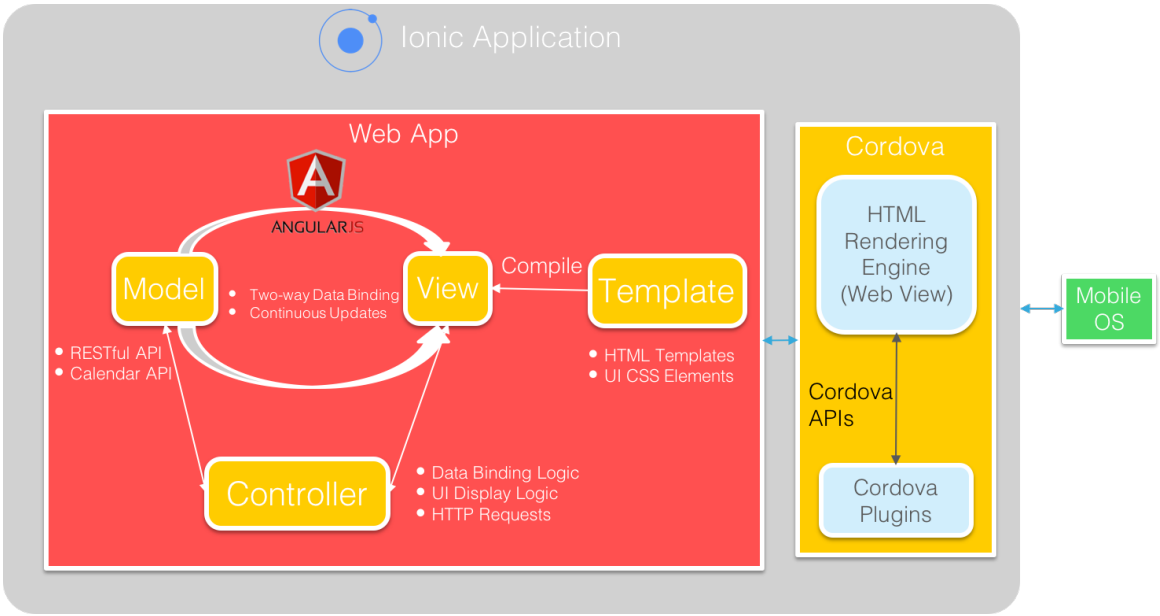


Figure 3.2 – Model-View-Controller Web App Pattern

The Web App layer is the place where major development work is done. In this layer, popular Model-View-Controller pattern is adopted.

- The model in Web App layer is the communication layer which handles data input and output with server side. In the model, data are retrieved through RESTful API end points defined in server. After getting the data from server, the model also maintains a cache for fetched data.
- The view is the web pages shown as the front end of the application. The view has two-way data binding with model, and supports continuous updates through AJAX without refresh. The view is compiled from a template defined in HTML5 and CSS.
- The controllers controls the data binding logics and display logics. It also determines the access to models and refreshes the models when necessary.

3.2. Layered Server Architecture

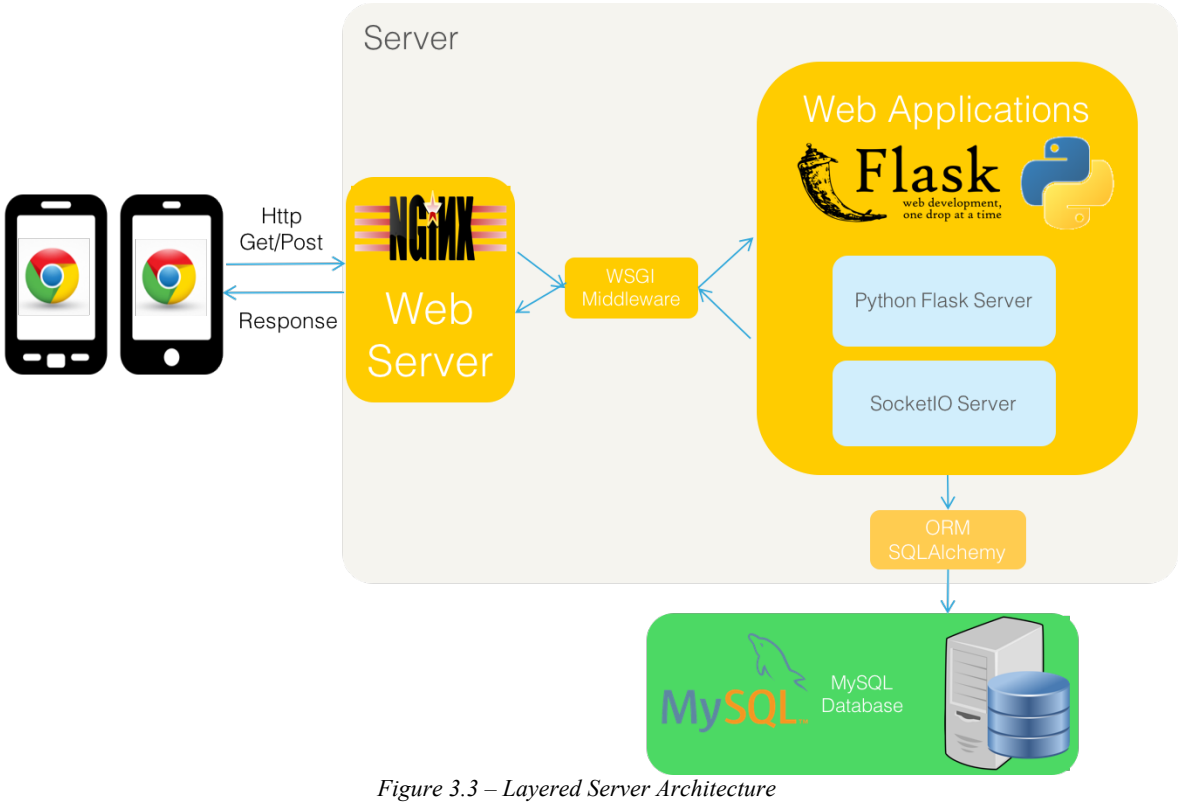


Figure 3.3 – Layered Server Architecture

Figure 3.3 above illustrates the detailed design within server side of the application. The web server is a container, containing the web application written in Flask. The web server, nginx, handles all incoming and outgoing requests. Flask communicates with the web server container through standard WSGI interface. The interface will propagate all requests to Flask application, which contains the logic in handling request.

Flask application contains two parts, namely the web service server and Socket-IO server. The two servers, however, are bundled nicely together in Flask. They always start and terminate together.

To allow Flask server to connect to database, a layer of Object-Relational Mapping is done in SQLAlchemy. This layer of ORM abstracts the model of MySQL database to objects, easing the handling of database.

The server itself is stateless as the state of the application is handled in database. Also, there is a separate space at server machine which stores large files like photo and audio. However, they are not part of the server and can be easily moved to other locations.

The design above has a few advantages:

- Easy to maintain and understand
- Extensible through Flask extensions
- Scalable through multiple servers under common interface

4. Maintainability Analysis

In software, there are mainly four types of maintenance.[1]

- Adaptive: In order to cope with changing software environment, some modifications need to be adopted to the system to meet new or changed user requirements.
- Perfective: After the system is used for operation, the development team may implement new functions or enhance existing function to achieve better software performance.
- Corrective: During the use of the system, users or developers may diagnose some errors. Appropriate maintenance is required to fix the errors.
- Preventive: Some design adjustments will be implemented to increase the software maintainability and reliability.

4.1. Adaptive Maintenance

In adaptive maintenance, the system is modified to cope with changes in the software environment. [1] We have listed the following scenarios that may occur in the following release of our updates with its changed business requirements.

4.1.1. Add more community group in Community page

Chess, Tai Chi and Social Dance are three core communities in current version of design. The elderly can join events organized by the community administrator to achieve social connectedness. In the future, other communities may be required to facilitate different needs of the elderly. In this case, another community different from existing ones is required to be implemented.

In current software design, client and server is loosely coupled. Also, Model-View-Controller pattern is adopted in front-end client web app. That is, the view is always updated when the model is modified. As the “communities” model (in Figure 4.1) is defined through HTTP request to the server, no modification is required on client side mobile applications.

```
<ion-item class="item-remove-animate item-thumbnail-left item-icon-right" type="item-
text-wrap" ng-repeat="community in communitys" ng-click="read(community.id)" style="min-
height: 100pt">
    <img ng-src="../www/img/{{ community.img }}">
    <h2 style="font-size: 18pt">{{ community.name }}</h2>
    <p>    </p>
    <p style="font-size: 18pt" ng-show="status(community.id)">{{'community.joined'|
    translate}}</p>
    <i class="button button-large button-clear button-positive icon ion-person-add icon-
    accessory" style="font-size: 18pt; margin-top:4%" ng-hide="joined(community.id)" ng-
    click="joinCom(community.id)">{{'community.join'| translate}}</i>
    <span class="badge badge-assertive" style="margin-top: 2%" ng-hide="(hasRead(
    community.id) || !joined(community.id))" ng-click="read(community.id)">1</span>
    <i class="icon ion-chevron-right icon-accessory custom-icon" ng-show="(hasRead(
    community.id) && joined(community.id))"></i>
</ion-item>
```

Figure 4.1 – Communities Model-View-Controller Design

The only part to change is to add the new community and associated information to database communities table. Similar action is required to create more events of a specific community.

4.1.2. Add more language options in Profile page

In current software design, translation function is specified in translations variable and wrapped by Angular-translate library, which provides the *translations.js* file as the dictionary for representation of the same word in different languages.

```
var translations = {
    "English": {
        "pull_to_refresh": "Pull down to refresh...",

        "tab.chat": "Chats",
        "tab.friends": "Friends",
        "tab.community": "Community",
        "tab.social": "Moments",
        "tab.me": "Profile",

        "accept": "Accept",
        "ignore": "Ignore",
        "delete": "Delete",
        "cancel": "Cancel",
        "add": "Add",
        "save": "Save",
        "select_all": "Select All",
        "deselect_all": "Deselect All",
        "back":"Back",
```

```
"简体中文": {
        "pull_to_refresh": "下拉刷新...",
        "tab.chat": "聊天",
        "tab.friends": "好友",
        "tab.community": "兴趣小组",
        "tab.social": "朋友圈",
        "tab.me": "我",

        "accept": "同意",
        "ignore": "忽略",
        "delete": "删除",
        "cancel": "取消",
        "add": "添加",
        "save": "保存",
        "select_all": "全选",
        "deselect_all": "全不选",
        "back":"返回",
```

Figure 4.2 – Sample translations in SeniorConnect System

Currently, only English and simplified Chinese are implemented considering these are the two most widely used languages among the elderly in Singapore. If, in the future, we want to add more language options, such as Malay or other international languages, to launch the application worldwide, we just need to add the specific languages to the translations file.

4.1.3. Add new platform support besides Android

The nature of SeniorConnect System is a hybrid app which runs in the browser of native mobile device as an HTML5 web application. Powered by Ionic framework, SeniorConnect application is essentially platform independent. The same piece of code can be deployed to multiple platforms with some minor customization. In current design, only Android version is developed. If in the future, more OS support is required, the only action required is to build the developed system again in different platforms. Some sample commands are shown below for iOS build.

```
// build the iOS version SeniorConnect system
ionic platform add ios
ionic build ios

// build the windows phone version SeniorConnect system
ionic platform add wp8
ionic build wp8
```

4.2. Perfective Maintainability

In perfective maintenance, software product is modified after delivery to improve performance or maintainability.

In our software design, the server and clients are de-coupled. They are assigned with different tasks and transparent to each other. Communication between them is through HTTP standard protocol.

Since they are transparent to each other, improvement in server side will not affect behavior of the client side. There is no need for extra change or testing on the client side on server's algorithm, storage or any other change as long as the public interface still remains the same.

4.3. Corrective Maintainability

Software defects of the system should be reduced during the development phase with good system design and comprehensive testing. However, some defects may still exist and discovered during operations. Corrective maintenance is required when some errors are detected.

The corrective maintainability can be represented by the ease of correcting software errors after the launch of the system. From design perspective, some strategies can be adopted to increase the correcting ease, i.e. reduce maintenance time and cost.

4.3.1. Accessibility

According to statistic from past software projects, during the corrective maintenance work, a large amount of time and efforts will be input to accessing failed parts in codes before the real problem-solving process. In order to reduce accessing cost and time, accessibility should be taken into consideration during design phase.

As for the design of SeniorConnect, service-oriented architecture is designed for the whole system, which enhances the accessibility. Moreover, a clear layered hierarchy is built for both the client front-end web apps and the backend servers, which also ensures accessibility.

4.3.2. Interchangeability

During corrective maintenance process, failed components, functional or physical, may be required to be removed or replaced from the system. The replacement may be very costly or time-consuming, which is also a part of maintenance cost. In order to reduce the replacement cost and consequently the overall corrective maintenance cost, the initial design should include good interchangeability.

In the design of SeniorConnect, interchangeability is achieved via layered design for both software and hardware components. For every layer implementation, standard interface is adopted and APIs are provided. Therefore, during maintenance process, existing loosely-coupled components can be easily replaced by new or outside components, as only small modification will be required to connect interface.

4.3.3. Fault Recognition

System errors and defects are usually detected by development teams or users. Effective reporting system should be built to realize errors in shortest time and corresponding correction actions should be carried on immediately. In order to fix the error, fault recognition is required in the first place. In practice, fault recognition is actually most costly in the corrective maintenance process. A system with cost-effective fault recognition is much more efficient in maintenance process.

As for SeniorConnect project, fault indicators are well designed in all necessary places in codes to handle error recognition. Besides, unambiguous fault isolation capability is also implemented by the clear boundaries between layers.

4.4. Preventive Maintainability

In sentence, preventive maintenance is some scheduled actions and plans, which are used to maintain the system to work with a specified level of performance. The preventive maintenance process usually consists of regularly scheduled inspection, cleaning, adjustments, calibration, parts replacement and lubrication and repairs.

From design perspective, in the SeniorConnect project, some software quality attributes have been enhanced during initial design phase for the ease of future preventive maintenance.

- Testability: According to the user requirements, all functions are designed and implemented in an atomic manner, which guarantees good testability.
- Traceability: A clear hierarchy is adopted in design and dependencies among layers are also unambiguous. Every operation can be easily traced during inspection and potential errors may be detected.

Moreover, the system may include a separate subsystem to assist preventive maintenance, which is usually named as Planned Maintenance System. The subsystem is in charge of regularly scheduled inspection and generating report logs for maintenance personnel to do preventive maintenance.

5. Reference

- http://en.wikipedia.org/wiki/Software_maintenance
- http://ionicframework.com
- http://flask.pocoo.org
- https://angularjs.org
- https://github.com/alongubkin/phonertc
- http://nginx.org/en/
- https://github.com/wymsee/cordova-imagePicker