

# Chi Alpha: Christian Fellowship Full-stack Mobile Application

*MVP Project Report*

**Chi Alpha WSU**



**Web Ready**

Vincent Yen, Justin Lee

*Table of  
Contents*

*Page*

<i>I.</i>	Introduction	3
<i>II.</i>	Team Members – Bios and Project Roles	3
<i>III.</i>	Project Requirements Specification	4
<i>IV.</i>	Software Design	5
<i>V.</i>	Test Specification and Results	10
<i>VI.</i>	Projects and Tools	22
<i>VII.</i>	Final Prototype Description	22
<i>VIII.</i>	Product Delivery Status	23
<i>IX.</i>	Conclusions and Future Work	23
<i>X.</i>	Acknowledgements	24
<i>XI.</i>	Glossary	24
<i>XII.</i>	References	25
<i>XIII.</i>	Appendices	26

## **I. Introduction**

The intent of this project is to create a resource in the form of a mobile application for Chi Alpha WSU. The mobile app will allow the organization to make announcements, advertise events, a medium for individuals to make donations, and communicate with the organization. The mobile app is intended for the organization to increase their presence and access to and for current members of the fellowship. This document serves as an overview of the intended work for this project.

## **II. Team Members – Bios and Related Roles**

**Name:** Vincent Yen

**Degree Plan:** Computer Science B.S.

**Project Role:** Team Lead, Software Architect

### **Skills and Areas of Expertise:**

- Computer Languages: C/C++, Java, Python, JavaScript/TypeScript, HTML, CSS
- Database: SQL, MongoDB
- Web Tools: Angular, React, Flask, NodeJS, Django, Karma, Jasmine

### **Description:**

Vincent is a computer science student that has experience working with full-stack web applications which carry into this project since mobile applications follow a similar framework. His responsibilities for this team is to: frame the project so that it meets the needs of the client as well as the abilities of the team, work documentation, back-end, database, and aid in whatever areas of the project that cannot be fulfilled by other team members.

**Name:** Justin Lee

**Degree:** Computer Science B.S.

**Project Role:** UI Developer

### **Skills and Areas of Expertise:**

- Computer Languages: C/C++, Java, Python, HTML, CSS
- Database: SQL
- Web Tools: Flask

### **Description:**

Justin is a computer science student that has some prior experience building web applications using Python Flask. Justin sought to work on this project because of his interest in working on web applications. His role in this project is primarily to work on the user interfacing components of the application.

### III. Project Requirements and Specifications

Chi Alpha WSU currently does not have a mobile resource application that they can use to communicate with the fellowship. The client has requested that this application provide an easier way for the web administrator to advertise activities that are happening within the organization.

The project is intended for the organization to increase their presence and access to and for current members of the fellowship. The mobile app will allow the organization to make announcements, advertise events, serve as a medium for individuals to make donations, and communicate with the organization.

#### III.1 Project Stakeholders

Stakeholder	Description
Organization	The organization's image and values are reflected in the design of the application. The components that are delivered through the application should reflect the values of the organization.
Client	Hailey Galletly is the client representing her organization. She will be providing the general overview and design of the application based on the needs of her organization.
Team	The team will require a descriptive requirement description from the client to be able to properly invest time in development to deliver a product that will be useful to the client
General Users	The general users will be the various members in the organization that will be using the application. The users will require the application to be intuitive enough be able to use and serve as a reasonable resource alternative against what is already provided by the organization.

#### III.2 Use Cases

**Story:** Hailey would like to add an announcement to the resource application. Hailey opens the app and logs into the application using an administrator login. The application now shows the ability to add new templated elements to the app, specifically a form to add a new announcement to the application dashboard. Hailey enters the new announcement information into the form and submits for publication. The new announcement is reflected on the announcement dashboard of the application.

**Source:** Vincent Yen

**Story:** A member of the fellowship would like to donate to the organization. The member selects “Giving” from the menu and is taken to the “Giving” page. The member enters their payment information into the third-party payment API and submits [4]. The organization receives the payment that was given by the member.

**Source:** Vincent Yen

### **III.3. Functional Requirements**

- The application should allow adding and displaying announcements from the organization.
- The application should allow adding and displaying events for signing up and viewing.
- The application should allow a medium for donations from any individual who wishes to give.
- The application should show information about the organization and its staff members.
- The application should allow the user to login.
- The application should allow other users to communicate with each other that are using the application.
- An API should provide a medium between the application and the database which stores user information and media [4].
- The application should differentiate general users from admin users and limit certain access permissions to only the admin.

### **III.4 Non-Functional Requirements**

- The mobile application should adhere to federal standards of accessibility.
- User experience should be intuitive and easy to use.
- The application should be intuitive for both general users and the administrators.
- The application should be easily maintained by administrators who have little to no technical experience with mobile applications.

## **IV. Software Design – Solution Approach**

### **IV.1.1 Architecture Design**

This project follows most closely with a MVC model- model-view-controller, of system architecture. The application closely resembles most website MVC models in that it serves content to pages for viewing, information about users and media are modeled for the database schema, and a controller service controls routes to the database and authentication service based on various requests from the user.

The diagram given by Appendix A figure 1 serves to show how the system is decomposed. The system proposed for the project is decomposed into 4 major parts: pages, user menu, server API, and database [4].

Each page is composed with an application page that a user can view to obtain various content provided by the content creator. The announcement page allows the content creator to serve announcements that the organization would like the fellowship community to be aware of.

The events page allows the content creator to publish events that are happening in the community that users can sign up to attend. The giving page allows members of the community to donate to the organization through a third-party payment API service [4]. The contact us page allows users to contact the organization via email service. The about us page gives information about the organization and its staff members. The media page allows the content creator to publish different forms of media such as videos or pictures for users to browse and consume.

The user menu and its subcomponents are necessary specifically for all components that require user differentiation during their session. Currently the only component that requires a user login is the chatroom so that each user can be distinguished from other others in a simultaneous view session.

The server API serves as middleware between the mobile application and the database to aid in serving different media stored in the database as well as user authentication [4, 10].

The diagram in Appendix B figure 1 serves to show the different dependencies and interactions between each package in the system. The diagram given in Appendix B figure 2 serves as an overview of how the application will interact with the various package components.

The pages require the user interface to serve information for viewing. The user may interact with various pages for donations or to submit forms for things like event sign ups. As such the forms require pages and an API server to serve the form for viewing and to save the information into the database for the content creator to view later [4]. Donations also require pages as a way for users to interact with and submit their payments. The donations also require a payment service which will be provided by a third-part payment API [4].

The chat package requires the user interface in order for other users to interact with and submit their messages. The user may or may not login in order to identify themselves or remain anonymous.

The server serves as the middleware between the forms and pages as well as the user authentication service [10]. Forms being submit from a page will be sent to the server for validation and saving into the database. Users logging into the application will need to be authenticated with the user authentication service which will be validated against the database using the server.

The database package serves to save all information about users or store different forms of media based on the models provided in the server.

## **IV.1.2 Subsystem Decomposition**

### **IV.1.2.1 User Interface**

#### **a) Description**

The user interface is as its name implies. It serves as a graphical interface for the user to interact with the various components within the application.

#### **b) Concepts and Algorithms Generated**

The user interface is built using the React-Native framework which uses components derived from ReactJS [3, 11]. This framework allows the mobile application to be developed for both of the popular mobile operating systems- iOS and Android, simultaneously without having to work specifically with the native code for those operating systems [1, 6].

The drawback for using React-Native is having to use the specific syntax used in that framework as well as not being able to perform more specific functions that are only available in the native language for those operating systems [3].

### **IV.1.2.2 Pages**

#### **a) Description**

The pages serve as the view component for users to view different content provided by a content creator.

#### **b) Concepts and Algorithms Generated**

Since each page is likely not change in its form each page in the page component will hard coded based on the requirements provided by the client. A basic template will allow serving of various content from the database in the media page, announcement page, or the events page.

A drawback of this would be that it would limit the flexibility of the content creator to form the view of the given pages if in the future it is decided that the view of the page is not to the liking of the content provider.

#### **c) Interface Description**

##### Services Required

1. Service Name: RESTful service [8]  
Service Provided to: Server  
Description: The server will provide all content that is requested from each page that is consumed by the user.

### **IV.1.2.3 Forms**

#### **a) Description**

Each form will provide various ways for users to submit information that the content creator is asking for.

### **b) Concepts and Algorithms Generated**

Each form will be created from a template which should allow some flexibility for the content creator to enter their own specific information or questions for user submission.

If the content creator requires a more specific model of how the forms should be represented it may require more tailored forms that would deviate from templating and more towards hardcoding.

### **c) Interface Description**

#### Services Provided

1. Service Name: Form template  
Service Provided to: Forms  
Description: Provides a template for all forms to be used for the content creator.

## **IV.1.2.3 Chat**

### **a) Description**

The chat subsystem allows various users to chat with other users in the app.

### **b) Concepts and Algorithms Generated**

The chat subsystem will be a socket interface using session tokens for each user that logs into the chat system during their view session. A user that spends a certain amount of time idle in the chat will be removed from the system in order to alleviate resource usage. Users may login with prior created login credentials to identify themselves in the chat or enter chat without logging in as an anonymous guest.

A particular draw-back of allowing anonymous guests is the possibility of traffic flooding or a form of DDoS attack. Furthermore, by remaining anonymous various individuals may be able to act in other malicious ways in the chat service without repercussions because the individual cannot be easily traced and stopped.

### **c) Interface Description**

#### Services Required

1. Service Name: Server  
Service Provided to: Chat  
Description: The chat system requires user information for each session user in the chat. The server also serves as the middleware for all information exchanged in the chat so that a single user may broadcast to other session users or receive information that was broadcast from other users.

## **IV.1.2.4 Server**



### **a) Description**

The server serves as a middleware between the database and the resource application. The server will route various requests to various functions in the server such as user authentication, saving information submitted from forms into the database, or serving various media requests from the Media page.

### **b) Concepts and Algorithms Generated**

Given that the server is middleware between the database and the mobile application, this system serves as the controller component in the MVC architecture. The server will contain routes for which various requests will be routed to functions. The database schema will be built based on models provided in the server. Any forms generated for user submission will be built from models provided in this server as well.

A draw-back of this design is that if changes need to be made in database schema it may be difficult to change without affecting the information already being stored in database. Possible ways to overcome this is to make a new schema model to meet the new needs of the content creator while maintaining the old model for previously store information.

### **c) Interface Description**

#### Services Provided

1. Service Name: User Authentication  
Service Provided to: Chat  
Description: The user authentication service verifies the credentials of a user that is logging into the system. This allows the user to be identified distinctly in the chat system.
2. Service Name: Database Model  
Service Provided to: Database  
Description: The database model provides the database with the template needed to form the database schemas in which each storage item will be follow.
3. Service Name: Routes  
Service Provided to: Pages/Forms  
Description: Any page that requires particular content will require requests from the server which the server will determine which function it should be routed to based on the type of request. Requests may include: get, put, and post.

#### Services Required

1. Service Name: Object Storage

Service Provided to: Database

Description: The server requires storage for all the information that it receives from the front-end.

#### **IV.1.2.5 Database**

##### **a) Description**

The database serves a single point of storage for all information that is needed for serving various content or saving content. Some of this information includes: media storage, user credentials and identification, organization and staff bios, and information submitted from forms.

##### **b) Concepts and Algorithms Generated**

The chosen database structure will be MongoDB which will be paired with the Mongoose JavaScript library. MongoDB uses a non-relational type of storage which provides more flexibility and quicker data retrieval when compared to SQL type databases. Currently there are no forms of data being saved that requires indexing therefore this lends itself well to this form storage.

##### **c) Interface Description**

###### Services Provided

1. Service Name: Data Storage

Service Provided to: Server

Description: Data storage allows the server to obtain pertinent information that was previously saved for various purposes such as user validation, form data storage, and media storage.

###### Services Required

1. Service Name: Models

Service Provided to: Server

Description: The database requires models that are specified by the server in order to determine how it should form its schema templates.

#### **IV.2 Data Design**

The database will be using a non-relation type of database called MongoDB [5]. MongoDB represents data in the form of JSON as opposed to tables as seen in SQL type databases [5, 9, 5]. This form of database does not enforce the use of schema's and gives it more flexibility in how objects can be stored. Furthermore, this flexibility improves the performance read/write as it does not require information to be inserted into a rigid table schema [5].

#### **IV.3 User Interface**

The picture provided in Appendix C figure 1 serves as a general image of how the graphical user interface should appear. A menu with clear descriptions serves as a means for

the user to switch between the different page views for consumption. Future renditions should include a user menu, not shown in the figure, which provides options to login and perform various functions that require user credentials such as the chat system.

User should be able to obtain the application via the Google Play store for Android devices or the Apple App store for iOS devices [1, 6, 15]. In order to obtain the application from these stores the user must already have an account or otherwise register to create an account with the relative store in order to gain access to the store which allows the user to download the application onto their device.

## **V. Test Case Specifications and Results**

### **V.1. Test Overview**

The approach to content delivery in will be based on a Continuous Integration paradigm. Some components such as the pages can be tested without integration by simply determining if it will render the proper elements on the page. Other components will require some minimal integration in order to properly test functionality. The testing will primarily be done manually as the test requirements are relatively simple and would be more tasking to create automated tests for.

#### **V.1.2 Objectives**

The objectives for this application's testing are outlined as follows:

- Each unit test will be representative of a use case for that unit.
- Each end-to-end test will be representative of a start to finish use case of a session of use.
- Each unit test shall account for reasonable edge cases that are representative of a user interaction with the app component.

Tools that will be used for testing include the following:

- Postman API Platform - An API platform used for building and using APIs.
- Android Studio - An Android Emulator used to emulate various Android devices and versions of their operating systems.
- 

#### **V.1.2.3 Unit Testing**

Unit testing will likely be conducted manually by running through use case scenarios using Android Studio. If time permits unit testing may be streamlined using a test framework called Jest. Jest allows for a testing methodology called snapshot testing which allows an image of a UI component to be rendered and a static "snapshot" to be taken of that rendering. The snapshot is used as a reference to compare against another snapshot that will be taken to see if they match. This helps to identify if the unit is operating consistently and correctly against an

intended output. This portion of the testing is primarily for testing the UI portion of the application only.

The other form of unit testing that will take place will be through the use of an API platform called Postman. Postman allows interaction with the server component of the application without having to integrate with a UI. This will allow for manually testing different forms of RESTful requests to the server to check for proper responses [8].

#### **V.1.2.4 Integration Testing**

The integration testing will likely be done manually through the use of general use case scenarios which will involve running the database API and the user interface. If time permits the integration testing involved in this process may make use of the Jest testing framework. By testing including information pulled from the database and how it will be represented or rendered in the UI component this will require more substantial use of the snapshot testing methodology provided by Jest.

#### **V.1.2.5 System Testing**

The end-to-end testing will likely be done manually through the use of general use case scenarios. This will most likely require deployment of the application before proper end-to-end testing can occur, as it will be difficult to determine proper system functionality without testing it on the actual platform for which it is designed. If time permits we may use Detox because it is specifically designed for React-native. This will allow for designing an automated test that is representative of the application lifecycle through a user's use session [3].

#### **V.1.2.6 Functional Testing**

The functional testing for this application will likely be done manually through the use of general use case scenarios. If time permits the Jest framework may be used for more automated testing. As described previously in unit testing, the intended output of rendering of a UI component can be observed with the snapshot tests and the intended input and output of the middleware API and database can be tested manually using Postman.

#### **V.1.2.7 Performance Testing**

There isn't much involvement for the sake of performance testing for the application especially since the performance is relative to the application and the user's device and does not primarily involve user traffic. However, one component that may hinder performance may be traffic from database requests through the middleware API. This would require virtualizing multiple users making simultaneous requests to the middleware; however, this performance test may be considered excessive considering that the application is meant to service a relatively small audience.

#### **V.1.2.8 User Acceptance Testing**

A portion of the acceptance testing will be demonstrating the application to the client to see that it meets their needs. Furthermore, some individuals may be asked to sample the

application in order to determine whether there has been oversight on user interaction or that all requirements have been met to provide proper user experience.

## V.2 Environment Requirements

All tests will be done primarily through manual testing guided by use case scenarios. Automation testing if time permits will be conducted through the test framework Jest while end-to-end tests will be done using the Detox framework. Preliminary testing will be done through a virtual Android platform delivered via Android Studio [1]. Testing solely through Android initially is because of hardware limitations as the development team uses a pc platform and testing for a iOS platform would require a mac platform to virtualize a iOS device [6]. Once the majority of the basic features have been developed and tested on the virtualized Android platform the prototype should then also be tested on a virtualized iOS platform by testing remotely using a macOS server to host our testing needs [1, 6].

Finally, in final acceptance testing and demonstration the application shall be run on at least two different mobile devices, one containing the Android OS and the other containing the iOS [1, 6]. This will be tested against the requirements of the client and sampling of users to determine the value of the user experience that the application provides.

## V.3 Test Cases and Results

The following is list of all the current test cases and its most recent results.

### Page Components

Home (Announcement) Page
Test ID: P 1.1 Aspect to be tested: Announcement (Home) Page Expected Result: <ul style="list-style-type: none"><li>• Should use GET http request to obtain information to display.</li><li>• Should list all obtained title item to display.</li><li>• Should display "No announcements available" if unable to obtain correct http response of database is empty.</li></ul> Observed Result: <ul style="list-style-type: none"><li>• Announcement page was visited with server api running. Announcement JSON objects were obtained from http response and rendered properly.</li><li>• Announcement page was visited without server api running. Announcement page rendered "No announcement available" due to inability to obtain correct http response.</li></ul> Last Tested: 12/7/2022 Tested By: Vincent Yen Test Case Requirement: NodeJS, Android Emulator or Device, Expo Server.

Announcement Detail Page
Test ID: P 1.2 Aspect to be tested: Announcement Detail Page Expected Result: <ul style="list-style-type: none"><li>• Should obtain information from the parameter routes.</li><li>• Should display information from the routes.</li></ul>

**Observed Result:**

- Announcement detail page was toggled from announcement page. Information was displayed properly.

Last Tested: 12/7/2022

Tested By: Vincent Yen

Test Case Requirement: NodeJS, Android Emulator or Device, Expo Server.

**Event Page**

Test ID: P 2.1

Aspect to be tested: Event Page

**Expected Result:**

- Should use GET http request to obtain information to display.
- Should list all obtained title item to display.
- Should display "No events available" if unable to obtain correct http response or database is empty.

**Observed Result:**

- Event page was visited without server api running. "No events available" message was rendered due to inability to obtain http response.
- Event page was visited with server api running. Event JSON objects were obtained from database and displayed onto page correctly.

Last Tested: 12/7/2022

Tested By: Vincent Yen

Test Case Requirement: NodeJS, Android Emulator or Device, Expo Server.

**Event Detail Page**

Test ID: P 2.2

Aspect to be tested: Event Detail Page

**Expected Result:**

- Should obtain information from the parameter routes.
- Should display information from the routes.

**Observed Result:**

- Event detail page was toggled from event page. Information was displayed properly.

Last Tested: 12/7/2022

Tested By: Vincent Yen

Test Case Requirement: NodeJS, Android Emulator or Device, Expo Server.

**About Us Page**

Test ID: P 3.1

Aspect to be tested: About Us Page

**Expected Result:**

- Should use GET http request to obtain information to display.
- Should display all items obtained from request.
- Should display "No profiles available" if unable to obtain correct http response or database is empty.

**Observed Result:**

- About us page was visited without server api running. "No profiles available" was rendered due to inability to retrieve proper http response.

Last Tested: 12/7/2022 Tested By: Vincent Yen Test Case Requirement: NodeJS, Android Emulator or Device, Expo Server.
---

## **Server API**

### **Route Functions**

<b>Home Function</b>
Test ID: SR 1.1 Aspect to be tested: app.get("/api") Expected Result: <ul style="list-style-type: none"><li>• Should call function main.index(req, res)</li><li>• Should return a JSON response.</li></ul> Observed Result: <ul style="list-style-type: none"><li>• index function is called. Response was returned from function.</li></ul> Last Tested: 12/2/2022 Tested By: Vincent Yen Test Case Requirement: NodeJS

<b>User Functions</b>
Test ID: SR 2.1 Aspect to be tested: app.post("/api/user/login") Expected Result: <ul style="list-style-type: none"><li>• Should call function main.loginUser(req, res)</li><li>• Should return a JSON response.</li></ul> Observed Result: <ul style="list-style-type: none"><li>• loginUser function is called. Response was returned from function.</li></ul> Last Tested: 12/2/2022 Tested By: Vincent Yen Test Case Requirement: NodeJS
Test ID: SR 2.2 Aspect to be tested: app.post("/api/user/new") Expected Result: <ul style="list-style-type: none"><li>• Should call function main.newUser(req, res)</li><li>• Should return a JSON response.</li></ul> Observed Result: <ul style="list-style-type: none"><li>• newUser function is called. Response was returned from function.</li></ul> Last Tested: 12/2/2022 Tested By: Vincent Yen Test Case Requirement: NodeJS

<b>Announcement Functions</b>
Test ID: SR 3.1 Aspect to be tested: app.get("/api/announcement/:announcementID") Expected Result:

<ul style="list-style-type: none"> <li>• Should call function main.getOneAnnouncement(req, res)</li> <li>• Should return a JSON response.</li> </ul> <p>Observed Result:</p> <ul style="list-style-type: none"> <li>• getOneAnnouncement function is called. Response was returned from function.</li> </ul> <p>Last Tested: 12/2/2022  Tested By: Vincent Yen  Test Case Requirement: NodeJS</p>
<p>Test ID: SR 3.2  Aspect to be tested: app.get("/api/announcements/all")  Expected Result:</p> <ul style="list-style-type: none"> <li>• Should call function main.getAllAnnouncements(req, res)</li> <li>• Should return a JSON response.</li> </ul> <p>Observed Result:</p> <ul style="list-style-type: none"> <li>• getAllAnnouncements function is called. Response was returned from function.</li> </ul> <p>Last Tested: 12/2/2022  Tested By: Vincent Yen  Test Case Requirement: NodeJS</p>
<p>Test ID: SR 3.3  Aspect to be tested: app.post("/api/announcements/new")  Expected Result:</p> <ul style="list-style-type: none"> <li>• Should call function main.newAnnouncement(req, res)</li> <li>• Should return a JSON response.</li> </ul> <p>Observed Result:</p> <ul style="list-style-type: none"> <li>• newAnnouncement function is called. Response was returned from function.</li> </ul> <p>Last Tested: 12/2/2022  Tested By: Vincent Yen  Test Case Requirement: NodeJS</p>
<p>Test ID: SR 3.4  Aspect to be tested: app.put("/api/announcements/edit/:announcementID")  Expected Result:</p> <ul style="list-style-type: none"> <li>• Should call function main.editAnnouncement(req, res)</li> <li>• Should return a JSON response.</li> </ul> <p>Observed Result:</p> <ul style="list-style-type: none"> <li>• editAnnouncement function is called. Response was returned from function.</li> </ul> <p>Last Tested: 12/2/2022  Tested By: Vincent Yen  Test Case Requirement: NodeJS</p>

<b>Event Functions</b>
<p>Test ID: SR 4.1  Aspect to be tested: app.get("/api/event/:eventID")  Expected Result:</p> <ul style="list-style-type: none"> <li>• Should call function main.getOneEvent(req, res)</li> <li>• Should return a JSON response.</li> </ul> <p>Observed Result:</p> <ul style="list-style-type: none"> <li>• getOneEvent function is called. Response was returned from function.</li> </ul> <p>Last Tested: 12/2/2022  Tested By: Vincent Yen  Test Case Requirement: NodeJS</p>



<p>Test ID: SR 4.2</p> <p>Aspect to be tested: app.get("/api/events/all")</p> <p>Expected Result:</p> <ul style="list-style-type: none"> <li>• Should call function main.getAllEvents(req, res)</li> <li>• Should return a JSON response.</li> </ul> <p>Observed Result:</p> <ul style="list-style-type: none"> <li>• getAllEvents function was called. Response was returned from function.</li> </ul> <p>Last Tested: 12/2/2022</p> <p>Tested By: Vincent Yen</p> <p>Test Case Requirement: NodeJS</p>
<p>Test ID: SR 4.3</p> <p>Aspect to be tested: app.post("/api/events/new")</p> <p>Expected Result:</p> <ul style="list-style-type: none"> <li>• Should call main.newEvent(req, res)</li> <li>• Should return a JSON response.</li> </ul> <p>Observed Result:</p> <ul style="list-style-type: none"> <li>• newEvent function was called. Response was returned from function.</li> </ul> <p>Last Tested: 12/2/2022</p> <p>Tested By: Vincent Yen</p> <p>Test Case Requirement: NodeJS</p>
<p>Test ID: SR 4.4</p> <p>Aspect to be tested: app.put("/api/events/edit/:eventID")</p> <p>Expected Result:</p> <ul style="list-style-type: none"> <li>• Should call main.editEvent(req, res)</li> <li>• Should return a JSON response.</li> </ul> <p>Observed Result:</p> <ul style="list-style-type: none"> <li>• editEvent function was called. Response was returned from function.</li> </ul> <p>Last Tested: 12/2/2022</p> <p>Test Case Requirement: NodeJS</p>

<p><b>Response Functions</b></p>
<p>Test ID: SR 5.1</p> <p>Aspect to be tested: app.put("/api/events/response/new/:eventID")</p> <p>Expected Result:</p> <ul style="list-style-type: none"> <li>• Should call main.addResponse(req, res)</li> <li>• Should return a JSON response.</li> </ul> <p>Observed Result:</p> <ul style="list-style-type: none"> <li>• addResponse function was called. Response was returned from function.</li> </ul> <p>Last Tested: 12/2/2022</p> <p>Tested By: Vincent Yen</p> <p>Test Case Requirement: NodeJS</p>

## Main Functions

<p><b>Home Function</b></p>
<p>Test ID: SM 1.1</p> <p>Aspect to be tested: index</p>

Expected Result:

- Should return message: "Error 505 – Internal Server Error"

Observed Result:

- Proper message was returned

Last Tested: 12/2/2022

Tested By: Vincent Yen

Test Case Requirement: NodeJS, MongooseJS, BcryptJS, MongoDB

## User Functions

Test ID: SM 2.1

Aspect to be tested: loginUser

Expected Result:

- If username from request is correct, database should return a user object from the database
- If username from request is not correct, it should return an error or a null object
- If a user object is returned from the database, if the password is correct it should return the proper json object stored in the "check" variable.
- If a user object is returned from the database, if the password is incorrect it should return a json object stored in the "err" variable containing the error message

Observed Result:

- Function called without request body. Response returned with error message.
- Function called with only legitimate username in request body. Response returned with error message.
- Function called with legitimate username and incorrect password. Response returned with error message.
- Function called with legitimate username and password. Response returned with success message.

Last Tested: 12/3/2022

Tested By: Vincent Yen

Test Case Requirement: NodeJS, MongooseJS, BcryptJS, MongoDB

Test ID: SM 2.2

Aspect to be tested: newUser

Expected Result:

- If the username already exists in the database, it should return an error that says the user exists
- If the username does not exist, if the password length is less than 8 characters it should return an error that says, "the password length must be at least 8 characters".
- If the username does not exist and the given password is 8 characters or greater, it should create a user object, hash the given password, and save all the information into the database. It should return the user object that was created.
- If request body is not given, should return an error.

Observed Result:

- Function called with required parameters. User object was created successfully. Response was returned with success message and user object.
- Function called with username already existing in database. Response was returned with error message noting the user already existing.
- Function called without request body. Response was returned with error message.

- Function called with password not meeting requirements. Response was returned with error message.

Last Tested: 12/3/2022

Tested By: Vincent Yen

Test Case Requirement: NodeJS, MongooseJS, BcryptJS, MongoDB

### Announcement Functions

Test ID: SM 3.1

Aspect to be tested: getOneAnnouncement

Expected Result:

- Should find an announcement based on the given announcement id from the URL parameter, return the announcement object, and a success message.
- If the announcement id is incorrect it should return an error

Observed Result:

- Function called with non-existent announcement id. Response returned with error message.
- Function called with correct announcement id as URL parameter. Response returned with success message and announcement object.

Last Tested: 12/3/2022

Tested By: Vincent Yen

Test Case Requirement: NodeJS, MongooseJS, MongoDB

Test ID: SM 3.2

Aspect to be tested: getAllAnnouncements

Expected Result:

- It should return all announcements and a success message.
- If unable to obtain announcements or no announcements are stored, it should return an error

Observed Result:

- Function called. Response returned with success message and all announcements stored in database.

Last Tested: 12/3/2022

Tested By: Vincent Yen

Test Case Requirement: NodeJS, MongooseJS, MongoDB

Test ID: SM 3.3

Aspect to be tested: newAnnouncement

Expected Result:

- It should create a new announcement object if the correct parameters are given and store it in the database.
- If it is unable to create and store the announcement, it should return an error
- If request sent without request body, it should return an error.

Observed Result:

- Function called without request body. Response returned with error message.
- Function called without all needed parameters in request body. Response returned with error message.
- Function called with all needed parameters in request body. Response returned with success message.

Last Tested: 12/3/2022

Tested By: Vincent Yen

Test Case Requirement: NodeJS, MongooseJS, MongoDB

Test ID: SM 3.4

Aspect to be tested: editAnnouncement

Expected Result:

- If the correct announcement id is given from the URL parameter, it should obtain the announcement object and modify it with the proper parameters from the request body.
- If the incorrect announcement id is given from the URL parameter, it should return an error.
- If the announcement object cannot be saved into database, it should return an error.
- If the announcement object was modified and saved, it should return a success message.
- If request body is not provided, it should return an error.

Observed Result:

- Called function without request body and announcement id. Response returned with error message.
- Called function with proper announcement id in URL parameter, but no request body. Response returned with error message. Changes were not saved to database because it did not meet save requirements.
- Called function with proper announcement id in URL parameter, but request body did not meet requirements for saving. Changes were not saved to database. Response returned with error message.
- Called function with proper announcement id in URL parameter and request body meeting save requirements. Changes were saved and reflected in the database. Response returned with success message.

Last Tested: 12/3/2022

Tested By: Vincent Yen

Test Case Requirement: NodeJS, MongooseJS, MongoDB

### Event Functions

Test ID: SM 4.1

Aspect to be tested: getOneEvent

Expected Result:

- If the correct event id is given, it should be able to return the event object and a success message.
- If the incorrect event id is given, it should return an error message.

Observed Result:

- Called function with correct event id in URL parameter. Function returned with success message and event object.
- Called function with incorrect event id in URL parameter. Function returned with error message.

Last Tested: 12/3/2022

Tested By: Vincent Yen

Test Case Requirement: NodeJS, MongooseJS, MongoDB

Aspect to be tested: getAllEvents

Expected Result:

- If there are events saved in the database, it should return all of the events that are stored and a success message.
- If there are no events saved in the database or an error occurs in retrieving, it should return an error message.

Observed Result:

<ul style="list-style-type: none"> <li>Function called. Response returned with success message and all events stored in database.</li> </ul> <p>Last Tested: 12/3/2022  Tested By: Vincent Yen  Test Case Requirement: NodeJS, MongooseJS, MongoDB</p>
<p>Test ID: SM 4.2  Aspect to be tested: newEvent  Expected Result:</p> <ul style="list-style-type: none"> <li>An event object should be created. If the correct parameters are given from the request body, it should save the information in the database.</li> <li>If parameters are missing from the request body, it should return an error because those parameters are required.</li> <li>If an error occurs when saving to database, it should return an error.</li> <li>If the object is saved in the database, it should return a success message.</li> </ul> <p>Observed Result:</p> <ul style="list-style-type: none"> <li>Function called without request body. Response returned with error message.</li> <li>Function called with request body, but without all needed parameters. Response returned with error message.</li> <li>Function called with request body and all proper parameters. Response returned with success message and announcement object.</li> </ul> <p>Last Tested: 12/3/2022  Tested By: Vincent Yen  Test Case Requirement: NodeJS, MongooseJS, MongoDB</p>
<p>Test ID: SM 4.3  Aspect to be tested: editEvent  Expected Result:</p> <ul style="list-style-type: none"> <li>If the proper event id is given, it should be able to retrieve the event object.</li> <li>If the event object cannot be retrieved or a null object is retrieved, an error message should be returned.</li> <li>If the event object is retrieved it should save the information provided by the request body.</li> <li>If an error occurred when saving to database, an error should be returned.</li> <li>If the event object was saved properly, it should return a success message.</li> </ul> <p>Observed Result:</p> <ul style="list-style-type: none"> <li>Function called with incorrect event id. Response returned with error message.</li> <li>Function called with correct event id and without request body. Response returned with error message. No changes made to event object in database.</li> <li>Function called with correct event id and request body, but without all proper parameters. Response returned with error message.</li> <li>Function called with correct event id and request body parameters. Response returned with success message and changes are reflected in database.</li> </ul> <p>Last Tested: 12/3/2022  Tested By: Vincent Yen  Test Case Requirement: NodeJS, MongooseJS, MongoDB</p>

<b>Response Functions</b>
<p>Test ID: SM 5.1  Aspect to be tested: addResponse  Expected Result:</p>

- If the proper event id is given, it should create a response object and save the list of responses to each event question.
- If the list of responses are not equal to the list of questions, an error message should be returned
- If it was unable to retrieve the proper event object, it should return an error message.
- If the event object was unable to save properly, it should return an error message.

**Observed Result:**

- Function called with proper event id and without request body. Response returned with error message.
- Function called with incorrect event id. Function returned with error message.
- Function called with correct event id and incorrect parameter. Response returned with error message.
- Function called with correct event id and correct parameter without preexisting questions. Response returned with error message.
- Function called with correct event id and correct parameters, but without equal number of responses to questions. Response returned with error message.
- Function called with correct event id and correct parameters, but response was not provided in array object. Response returned with error message.
- Function called with correct event id and correct parameters. Response returned with success message and response was added to event object stored in database.

Last Tested: 12/3/2022

Tested By: Vincent Yen

Test Case Requirement: NodeJS, MongooseJS, MongoDB

## **Server**

### **Server File**

Test ID: SS 1.1

Aspect to be tested: server.js

**Expected Result:**

- Should be able to parse URL encoded request bodies using Body Parser library.
- Should be able to connect to the correct database.
- Should be able to listen on the correct port.

**Observed Result:**

- Various URLs pinged at proper port address. Request is able to be sent. Request body is properly parsed.
- Various URLs pinged at improper port address. Request could not be sent.
- Process determines correct port is used when process is running.

Last Tested: 12/3/2022

Tested By: Vincent Yen

Test Case Requirement: NodeJS

## **VI. Projects and Tools Used**

Tools/Libraries/Frameworks	Function
React-Native	The primary platform used to code the application and compile into the Native languages for IOS and Android.

Expo	An assistive wrapper library to react-native that will allow integration into IOS and Android platforms.
Android Studio	The emulator which is used for testing and developing.
Postman	An API platform used for testing and building APIs.
NodeJS	The primary library used to develop the database API.
MongoDB	The database used to store information for the app.
Mongoose	The middleware library that queries the database from the database API.

Language Used
JavaScript

## VII. Description of Final Prototype

The final prototype will include the bare essentials for what the application should be. The features that will be included in the MVP are:

- Announcement page – A page which displays various current events that the content creator would like to advertise about the organization.
- Events page – A page for members of the organization view and to sign for.
- Database API – An api that will serve as the middleware between the UI and the database to serve and receive information to and from the database through RESTful queries.
- Database – A database to store and receive information from the content creator and the general users.
- An administrative website for the content creator to add new announcements and events to the database for the mobile app to pull for viewing later.

A general visual overview of the application user interface can be found in the appendix.

## VIII. Product Delivery Status

The product has no expected delivery date; however, it is planned to be delivered within the timeframe of the given project timeline. The product will be delivered with assistance of the client in determining a proper platform to host the application and the costs associated with it. Any further work on the product will likely be done by Vincent, depending on future circumstances.

All pertinent documentation and source materials can be found here:

<https://github.com/WSUCptSCapstone-Fall2022Spring2023/cacf-fullstackapp/tree/master>

## IX. Conclusion

### XIX.1 Limitations and Recommendations

There have been a few setbacks throughout the development of the project. The storing of images to MongoDB is not as straightforward as initially conceived and requires more research to find a proper solution. Some pages receive information from that database that have multiple relationships with other schemas that require some more intricate effort to be able to obtain, render, and manipulate to save in the database. The admin pages, particularly, are an example of this since these pages require that they only be rendered given the condition that the administrator credentials are present.

Given that there is a limited amount of time remaining to complete the project some revision to the project plan had to be made. After consulting with Hailey, our client, it has been decided that several features be placed on lower priority to allow more concentration on more higher value features for the minimum value product. Some of these features include removing pages that involve rendering images from the database. The original administrative pages have been moved from the mobile application transferred to a pure web user interface. The separation of the administrative functions from the general user functions provides for a smoother user experience and provides for a more secure context of operation for the administrator given that the administrative functions have been placed onto a separate platform.

### **XIX.2 Future Work**

For the remainder of the project, the plan is to focus on the user interfacing components and the major back-end components of the project such as the database, the database server interface, and the administrative website. The plan for minimum value product will be composed of the pages that the users will consume, the website that the admin will use to create content, and the database to store all the information that will be used for consumption. The hope for the project is that the team will be able to move beyond the expected plan for the minimum value product and deliver more features for a better user experience.

## **X. Acknowledgements**

Special thank you to Professor Ananth Jillepalli for mentoring the team and assistance in communications with the client and our client Hailey Galletly and Chi Alpha for the opportunity to learn and develop this application for the organization's particular needs.

## **XI. Glossary**

**Android** – The world's most popular mobile operating system [1].

**API** – Stands for application programming interface. A third-party application that may be used in conjunction with another application for its specific services [4].

**Apple** - "Apple is a multinational information technology company based in California, USA." The company make various electronic devices, computers, laptops, as well as software [15].

**iOS** – A mobile operating system for Apple mobile devices [6].



**Java** - A programming language and computing platform developed by Sun Microsystems in 1995 and currently owned by Oracle [10].

**JavaScript** - A scripting language used in web development to implement complex features and logical algorithms [7].

**JSON** – Stands for JavaScript Object Notation. It is used as a format for data transfer on the web [9].

**Middleware** – Software that allows two or more applications or application components to communicate with each other [10].

**MongoDB** – A non-relational database that stores information in the form of objects similar to JSON structure [5].

**Mongoose** – An Object Data Modeling JavaScript library used in conjunction with MongoDB and a JavaScript library called NodeJS. Mongoose allows relational database management and schema validations while still maintaining some flexibility in the non-relational structure of MongoDB [2].

**React-Native** – React Native is an open-source mobile framework using JavaScript to design applications for IOS and Android. React Native is derived from ReactJS, a JavaScript framework used for web application development [3].

**ReactJS** – A JavaScript library which is used to build reusable UI components in Web development [11].

**SQL** – A relational database which stores data using tables and rows and enforces “referential integrity” [5].

**Swift** - A programming language for iOS and other Apple devices [9].

**RESTful** – REST stands for Representational State Transfer. An API that uses this architectural design is said to be RESTful. REST is used to across various forms of network protocols to transfer information [8].

**Xcode** - An integrated development environment created by Apple which supports the following languages: Swift, Objective-C, C++, etc [2].

## **XII. References**

- [1] Brown, C. Scott. *What is Android? Here’s everything you need to know*. Feb. 19, 2022. Accessed: Sep. 29, 2022. [Online]. Available: <https://www.androidauthority.com/what-is-android-328076/>
- [2] Karnik, Nick. *Introduction to Mongoose for MongoDB*. Feb. 11, 2018. Accessed: Oct. 4, 2022. [Online]. Available: <https://www.freecodecamp.org/news/introduction-to-mongoose-for-mongodb-d2a7aa593c57/>
- [3] Shilpa S. *What is React Native?*. Jul. 01, 2021. Accessed: Sep, 29, 2022. [Online]. Available: <https://www.tutorialspoint.com/what-is-react-native>

- [4] *Application Programming Interface (API)*. Accessed: Sep 18, 2022. [Online]. Available: <https://www.ibm.com/cloud/learn/api>
- [5] *MongoDB vs. MySQL Differences*. Accessed: Oct. 4, 2022. [Online]. Available: <https://www.mongodb.com/compare/mongodb-mysql>
- [6] *iOS*. Aug. 28, 2012. Accessed: Sep, 29, 2022. [Online]. Available: <https://www.techopedia.com/definition/25206/ios>
- [7] *What is JavaScript?*. Sep. 14, 2022. Accessed: Oct. 4, 2022. [Online]. Available: [https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First\\_steps/What\\_is\\_JavaScript](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript)
- [8] *What is RESTful API?*. Accessed: Oct. 4, 2022. [Online]. Available: <https://www.mulesoft.com/resources/api/restful-api>
- [9] *Swift: The powerful programming language that is also easy to learn*. Accessed: Sep. 29, 2022. [Online]. Available: <https://developer.apple.com/swift/>
- [10] *What is Java technology and why do I need it?*. Accessed: Sep. 29, 2022. [Online]. Available: [https://www.java.com/en/download/help/whatis\\_java.html](https://www.java.com/en/download/help/whatis_java.html)
- [11] *JSON Defined*. Accessed: Oct. 4, 2022. [Online]. Available: <https://www.oracle.com/database/what-is-json/>
- [12] *Middleware*. Mar. 5, 2021. Accessed: Oct. 4, 2022. [Online]. Available: <https://www.ibm.com/cloud/learn/middleware>
- [13] *ReactJS – Overview*. Accessed: Oct. 4, 2022. [Online]. Available: [https://www.tutorialspoint.com/reactjs/reactjs\\_overview.htm](https://www.tutorialspoint.com/reactjs/reactjs_overview.htm)
- [14] Cox, Daniel. *What is Xcode, and What can You Do With it?*. Apr. 7, 2022. Accessed: Sep. 29, 2022. [Online]. Available: <https://www.corecommerce.com/blog/what-is-xcode-and-what-can-you-do-with-it/>
- [15] Joe, Alexander. *Apple Inc. – Company Information*. Mar. 7, 2014. Accessed: Sep. 29, 2022. [Online]. Available: <https://marketbusinessnews.com/apple-inc-company-information/14919/>

## XIII. Appendices

### XIII.1. Appendix A – System Decomposition

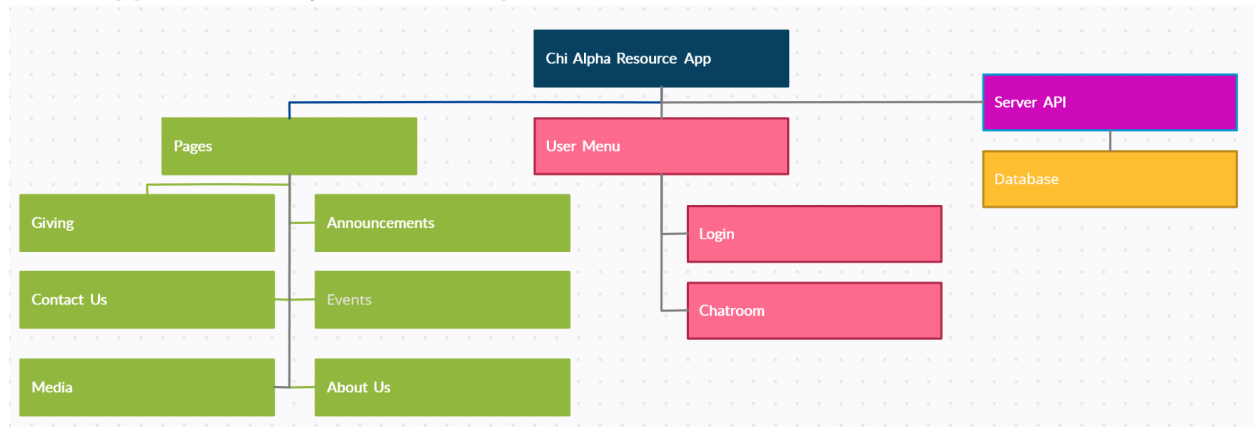


Figure 1

## XIII.2. Appendix B – System Diagrams and Dependencies

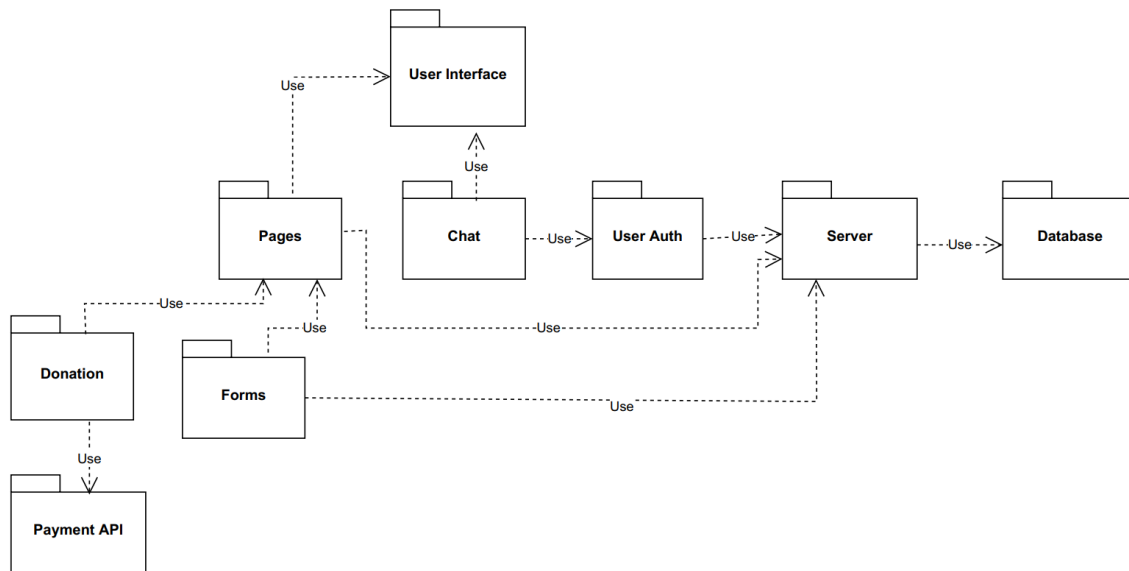


Figure 1

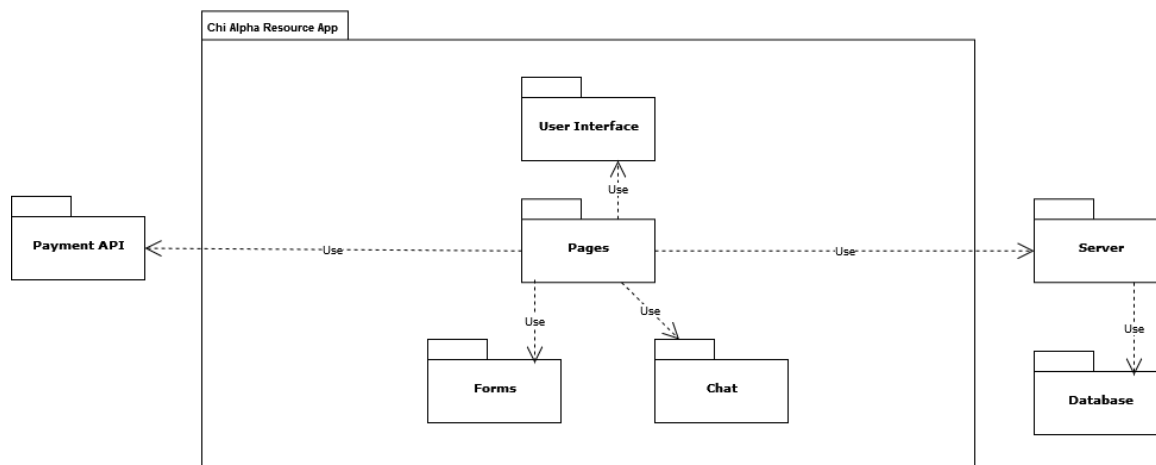


Figure 2

### XIII.3. Appendix C – User Interface and Design



**Figure 1**

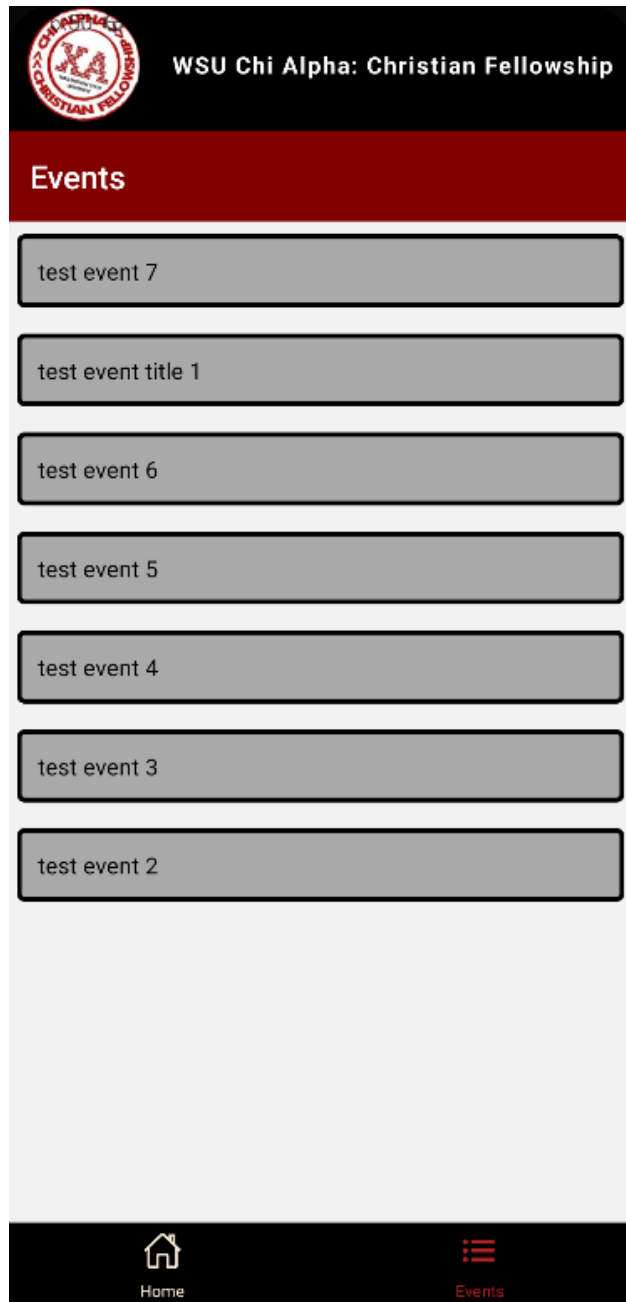



Figure 2



Chi Alpha: Christian Fellowship

[Home](#) [Announcements](#) [Events](#)

### Login


Username

Password

Don't have an account? [Register](#)

Submit

Figure 3



Chi Alpha: Christian Fellowship

[Home](#) [Announcements](#) [Events](#)

### Events

Title	Description	Questions	Last Updated
<a href="#">test event 8</a>	test description 8	<div>Add Question</div>	3/1/2023, 7:33:12 PM
<a href="#">test event 7</a>	test description 7	<div><a href="#">test question 1</a><div>Add Question</div></div>	3/1/2023, 6:09:58 PM
<a href="#">test event title 1</a>	test event description 1	<div><div><a href="#">test question 1</a><a href="#">test question 2</a><a href="#">test question 3</a></div><div>Add Question</div></div>	3/1/2023, 6:09:46 PM
<a href="#">test event 6</a>	test description 6	<div>Add Question</div>	1/31/2023, 10:22:19 PM
<a href="#">test event 5</a>	test description 5	<div>Add Question</div>	1/31/2023, 10:21:48 PM
<a href="#">test event 4</a>	test description 4	<div>Add Question</div>	1/31/2023, 10:21:44 PM
<a href="#">test event 3</a>	test description 3	<div>Add Question</div>	1/31/2023, 10:21:39 PM
<a href="#">test event 2</a>	test description 2	<div>Add Question</div>	1/31/2023, 10:21:34 PM




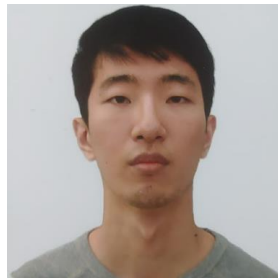
Figure 4

#### **XIII.4 Appendix D – Team Members**



**Vincent Yen**

**vincent.yen@wsu.edu**



**Justin Lee**

**dongjoon.lee@wsu.edu**

#### **XIII.5 Appendix E – Example Testing Strategy Reporting**

Test case requirements for the unit tests can be found in section V. Test Specifications and Results. The test case requirements for the remaining tests will be updated here as those requirements have been determined.

#### **XIII.6 Appendix F – Project Management**

General project details are discussed over discord, while on occasion members of the team will meet to discuss finer details about project design and expectations in person. Meetings with Professor Jillepalli occur monthly for mentoring and advice on how to proceed properly to meet the academic expectations and guidelines.

Project requirements are defined through the use of issues on GitHub. A Kanban board was initially used to describe the priorities of the project and components completed; however, given the size of the team it became more of a burden to update the Kanban board instead of just communicating changes and needs of the project directly.