

Florida International University
School of Computing and Information Sciences

Software Engineering Focus

Final Deliverable

CREATivators: Web and Mobile Development for Advertisement 1.0
Positive Pathways Mobile Application

Team Members: Mikaila Daniel, Michael Quiros, Alejandro Thornton

Product Owner(s): Dr. Steven Rios

Mentor(s): Dr. Francisco Ortega, Dr. Margo Berman

Instructor: Dr. Masoud Sadjadi

The MIT License (MIT)
Copyright (c) 2017 *Florida International University*

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

ABSTRACT

This document presents the information necessary to gain a good understanding of the Positive Pathways Mobile Application. The Positive Pathways Program has members and partners spread across the state. As these members of the program have monthly meetings, the need for an application where they can communicate in one place arose. The mobile application developed in this project addresses these needs. Users can create and edit their member and program profiles, as well as create and edit meeting entries. Additionally, the application serves as a central repository for information regarding the Positive Pathways Program.

TABLE OF CONTENTS

Introduction	5
User Characteristics	5
Current System	5
Purpose of New System	5
User Stories	6
Implemented User Stories	6
Sprint 2	7
Sprint 3	8
Sprint 4	16
Sprint 5	39
Sprint 6	48
Project Plan	52
Software Resources	52
Sprint Planning	52
System Design	58
Architectural Patterns	58
System and Subsystem Decomposition	58
Deployment Diagram	60
Design Patterns	60
System Validation	61
Glossary	80
Appendix	81
Appendix A: UML Diagrams	81
Appendix B: User Interface Design	91
Appendix C: Sprint Review Reports	97
Appendix D: Video Playlist	105

I. INTRODUCTION

The Positive Pathways Program (PPP) was created by the Florida Department of Children and Families (DCF) for the purpose of assisting former foster youths in attending and completing an education within Florida's public post-secondary educational system. These youths are assisted through dedicated campus-based programs. There are programs instituted in several schools throughout the state, creating a web of support. This web only continues to grow as more schools opt to participate in the Positive Pathways Program. With the growth of this program and the increasing need of cooperation among the support staff, a solution is needed to facilitate communication among them. The purpose of this project is to provide that solution in the form of a mobile application.

USER CHARACTERISTICS

The primary users of the application are the 'members' of the Positive Pathways Program. Members are employees of the program, spanning from administrative staff to the foster youth liaisons themselves. Characteristics such as age and gender may vary among members. Members may have technical knowledge, but not necessarily.

CURRENT SYSTEM

Currently, the members of PPP communicate through e-mail and phone conversations. There is no unified method for the transmission of important information and coordination of monthly meetings. Additionally, there is no dedicated location where information about individual programs and members can be found.

PURPOSE OF NEW SYSTEM

With the Positive Pathways Mobile Application, members can quickly find information about the monthly meetings: when they are, the requisite meeting link, the meeting agenda, and the minutes of meetings which have already occurred. Members may also find contact details for other members as well as general information about the programs and partners of the Positive Pathways Program. Members with registered accounts can update the information on their individual profiles as well as the programs they are involved in. Furthermore, links to training materials for members will also be included within the application.

II. USER STORIES

The user stories written through the course of the project are used as a guide for the implementation of the requirements laid out by the product owner. User stories are assigned to team members during sprint planning meetings, as laid out in section 3 of this report. The user stories contain the corresponding use cases. Additionally, visuals are provided such as use case diagrams, sequence diagrams, and visual user guides in the form of screenshots of the features.

IMPLEMENTED USER STORIES

The following are the user stories implemented throughout the course of the project:

Sprints	Completed User Stories
2	#666: Server Setup
3	#670: Create Profile Model #671: Create Profile API Routes #676: Create Main Menu Page
4	#678: Testing Calls to the Database #681: Create Meeting Model #683: Create Meeting API Routes #684: Create User Authentication Structure #688: User Authentication #690: Token Verification
5	#682: View Meetings #685: Create Program Model #686: Create Program API Routes
6	#680: Create New Meeting #691: View Member Profile #692: Log onto Server from Application #693: Search by Name or University #694: Edit Meeting #695: Edit Profile

SPRINT 2

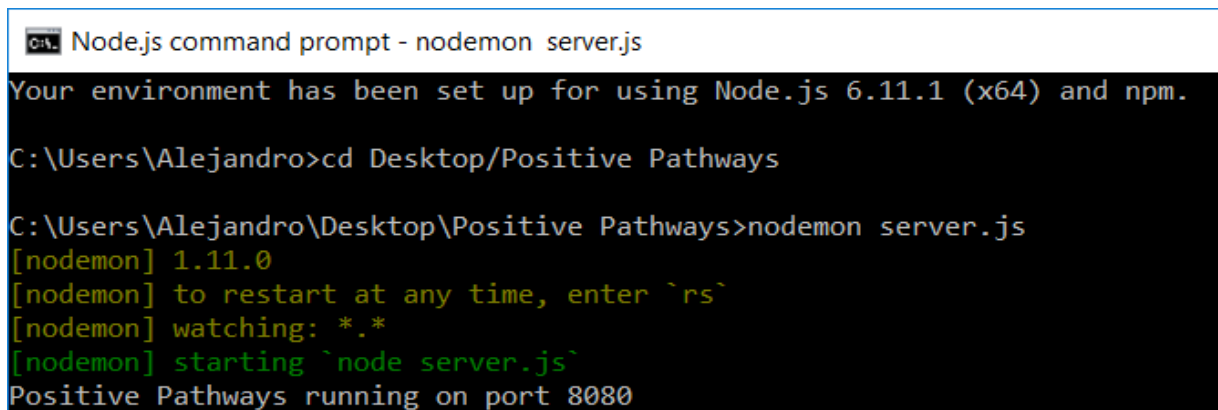
USER STORY #666: SERVER SETUP

Description: As a Developer **I would like** to create the files necessary for setting up the server operations for the application and the file structure for the project **so that** the team is able to continue development of the application.

Acceptance Criteria

- Produced server file with dependencies defined.
- Ability to run server without error.
- Directory structure in place for future builds.

Visual User Guide



```
Node.js command prompt - nodemon server.js
Your environment has been set up for using Node.js 6.11.1 (x64) and npm.
C:\Users\Alejandro>cd Desktop/Positive Pathways
C:\Users\Alejandro\Desktop\Positive Pathways>nodemon server.js
[nodemon] 1.11.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node server.js`
Positive Pathways running on port 8080
```

SPRINT 3

USER STORY #670: CREATE PROFILE MODEL

Description: **As a Developer I would like** to create the models for the objects being stored in the database: specifically the member and partner profiles, **so that** the relevant data is stored in the database.

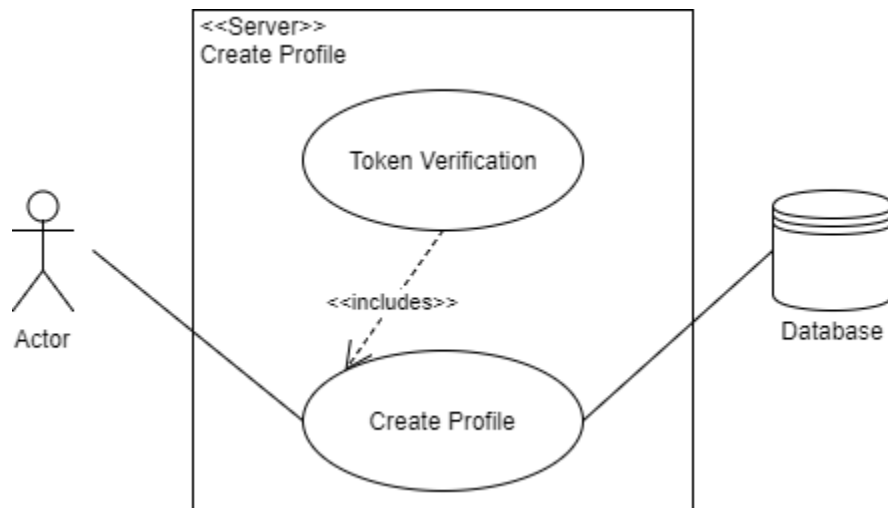
Acceptance Criteria

- Created JSON file for profile model.

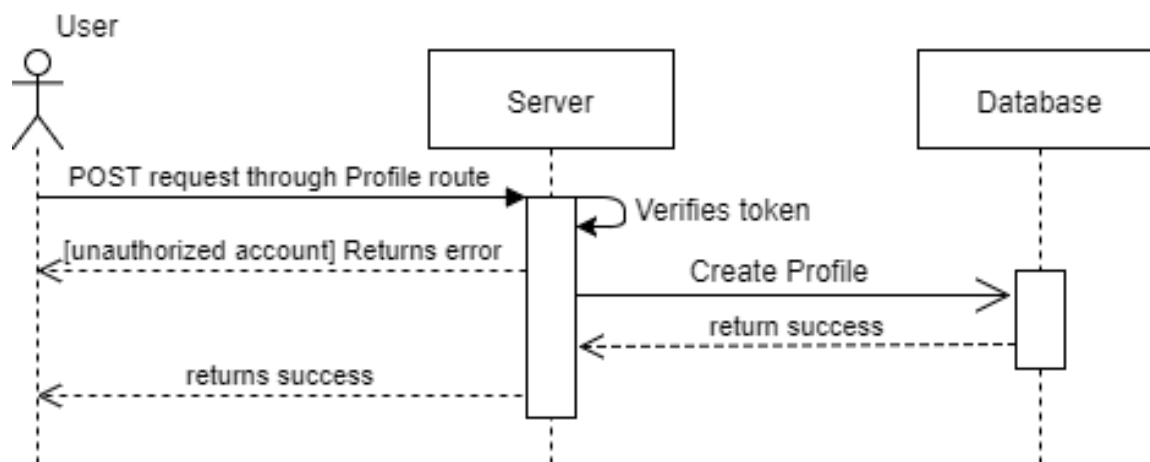
Use Case

Use Case Name	Create Profile
Participating Actors	User, Server, Database
Entry Condition	1. User makes a POST request through the Profile route.
Flow of Events	2. Server verifies the user is authorized to create a profile. If the user is not authorized: 2.1: Go to step 4.2. If the user is authorized: 2.2: Go to step 3. 3. Database creates new Profile entry. For each field in Profile: If there is a matching entry in the request body: 3.1: Field value is set equal to value given in request body. If there is not a matching entry in the request body: 3.2: Field value is set to a default value.
Exit Condition	4. Server returns response to User. If Profile was successfully created: 4.1. Server returns success message to User. If Profile was not successfully created: 4.2. Server returns error message to User.

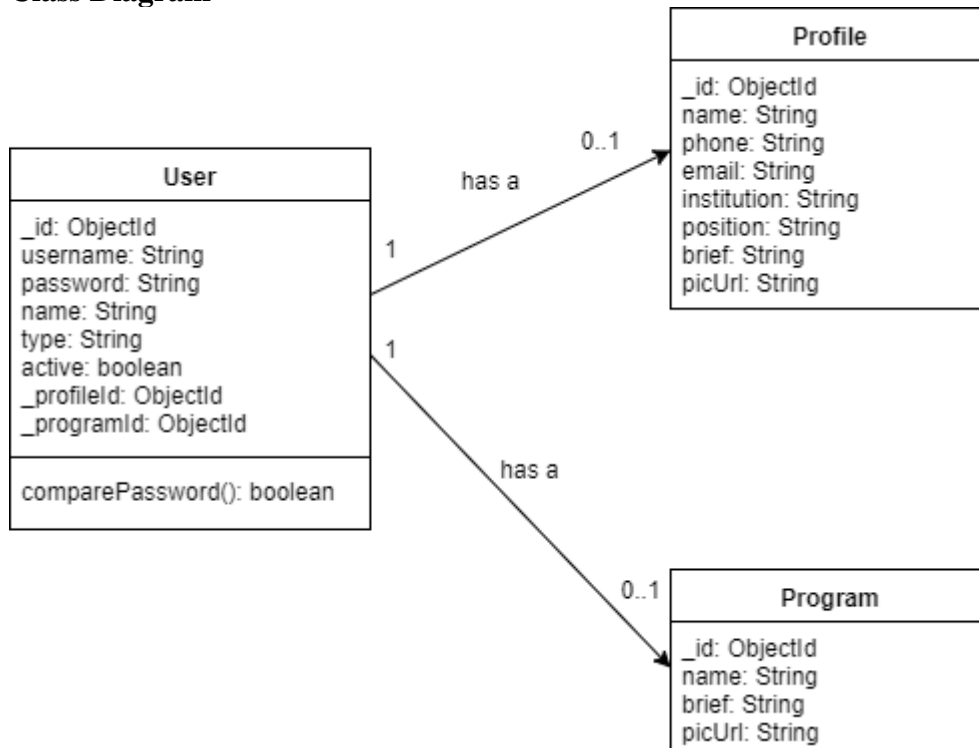
Use Case Diagram



Sequence Diagram



Class Diagram



Visual User Guide

POST localhost:8080/api/profiles Params Send

Authorization Headers (2) Body Pre-request Script Tests

Key	Value	Description
<input checked="" type="checkbox"/> Content-Type	application/x-www-form-urlencoded	
<input checked="" type="checkbox"/> x-access-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0eXBlljoiQWRtaW5pc3R...	
New key	Value	Description

Body Cookies Headers (9) Test Results Status: 200 OK

Pretty Raw Preview JSON

```
1 {
2   "_id": "5a1db9f76c5aa3b888ea46a",
3   "message": "Profile created!"
4 }
```

USER STORY #671: CREATE PROFILE API ROUTES

Description: As a Developer I would like create the routing for Profile requests so that the profile data within the database may be accessed and manipulated.

Acceptance Criteria

- Created routes for creating profile entries in the database.
- Created routes for accessing profile data in the database.
- Created routes for manipulating profile data in the database.

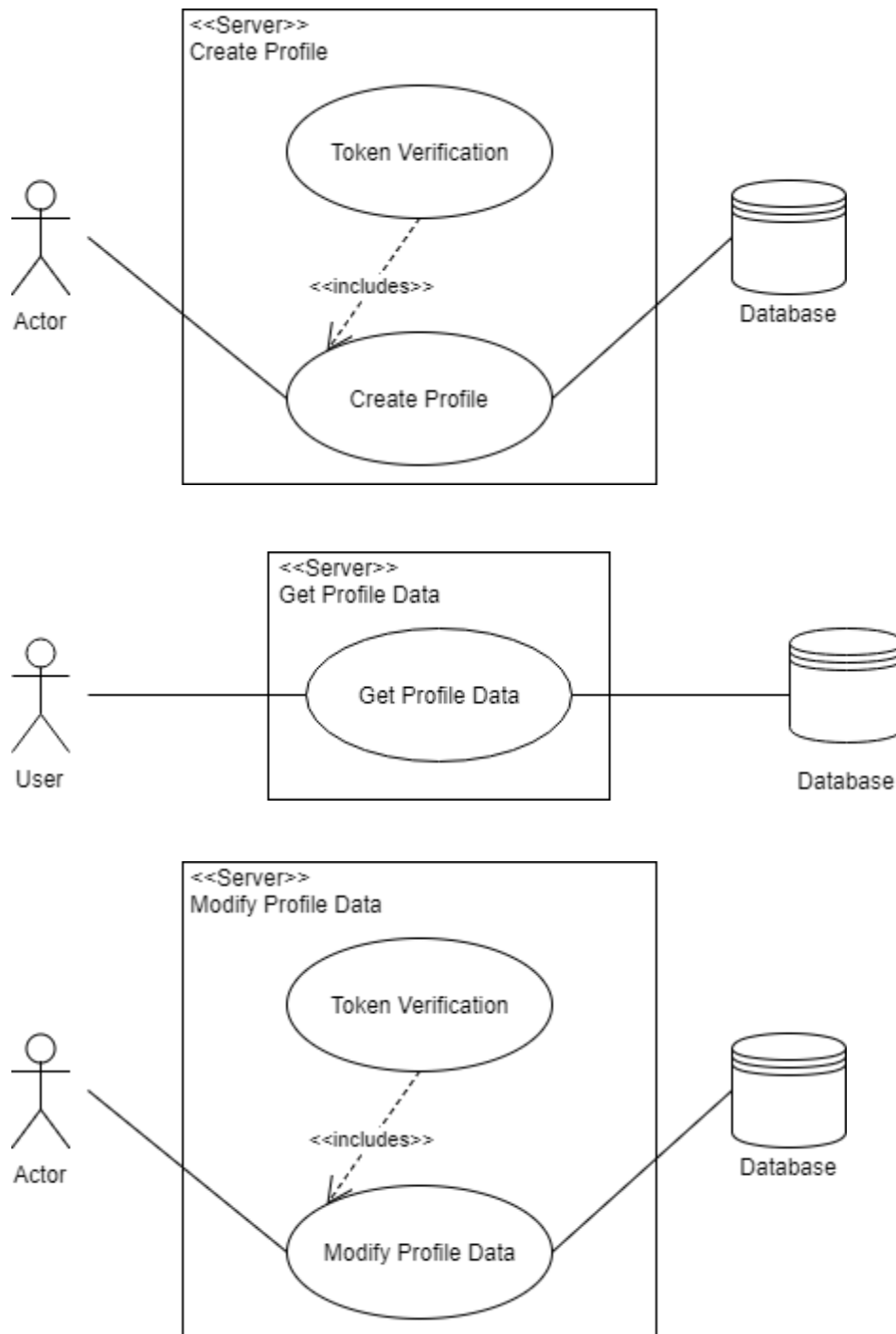
Use Cases

Use Case Name	Create Profile
Participating Actors	User, Server, Database
Entry Condition	1. User makes a POST request through the Profile route.
Flow of Events	<p>2. Server verifies the user is authorized to create a profile.</p> <p> If the user is not authorized:</p> <p> 2.1: Go to step 4.2.</p> <p> If the user is authorized:</p> <p> 2.2: Go to step 3.</p> <p>3. Database creates new Profile entry.</p> <p> For each field in Profile:</p> <p> If there is a matching entry in the request body:</p> <p> 3.1: Field value is set equal to value given in request body.</p> <p> If there is not a matching entry in the request body:</p> <p> 3.2: Field value is set to a default value.</p>
Exit Condition	<p>4. Server returns response to User.</p> <p> If Profile was successfully created:</p> <p> 4.1. Server returns success message to User.</p> <p> If Profile was not successfully created:</p> <p> 4.2. Server returns error message to User.</p>

Use Case Name	Get Profile Data
Participating Actors	User, Server, Database
Entry Condition	1. User makes a GET request through the Profile route.
Flow of Events	<p>2. Server retrieves Profile data from Database.</p> <p>If the request was made for a single Profile:</p> <p>2.1: Server verifies given ID matches a Profile in Database.</p> <p>If the ID doesn't match any Profile in Database:</p> <p>2.1.1: Go to step 3.2.</p> <p>If the ID does match a Profile in Database:</p> <p>2.1.2: Go to step 3.1.</p> <p>If the request was made for multiple Profiles:</p> <p>2.2: Go to step 3.1.</p>
Exit Condition	<p>3. Server returns response to User.</p> <p>If Profile data was successfully retrieved:</p> <p>3.1. Server returns success message and Profile data to User.</p> <p>If Profile data was not successfully retrieved:</p> <p>3.2. Server returns error message to User.</p>

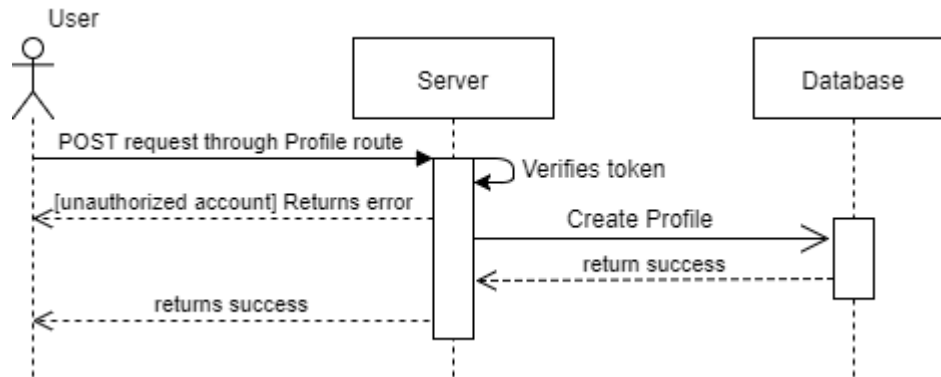
Use Case Name	Modifying Profile Data
Participating Actors	User, Server, Database
Entry Condition	<p>1. User makes a request through the Profile route with a Profile ID.</p> <p>1.1: User makes a PUT request.</p> <p>1.2: User makes a DELETE request.</p>
Flow of Events	<p>2. Server verifies the user is authorized to modify a profile.</p> <p>If the user is not authorized:</p> <p>2.1: Go to step 4.2.</p> <p>If the user is authorized:</p> <p>2.2: Go to step 3.</p> <p>3. Server verifies given ID matches a Profile in Database.</p> <p>If the ID doesn't match any Profile in Database:</p> <p>3.1: Go to step 4.2.</p> <p>If the ID does match a Profile in Database:</p> <p>3.2: Requested modifications are made in Database.</p>
Exit Condition	<p>4. Server returns response to User.</p> <p>If Profile was successfully modified:</p> <p>4.1. Server returns success message to User.</p> <p>If Profile was not successfully modified:</p> <p>4.2. Server returns error message to User.</p>

Use Case Diagrams

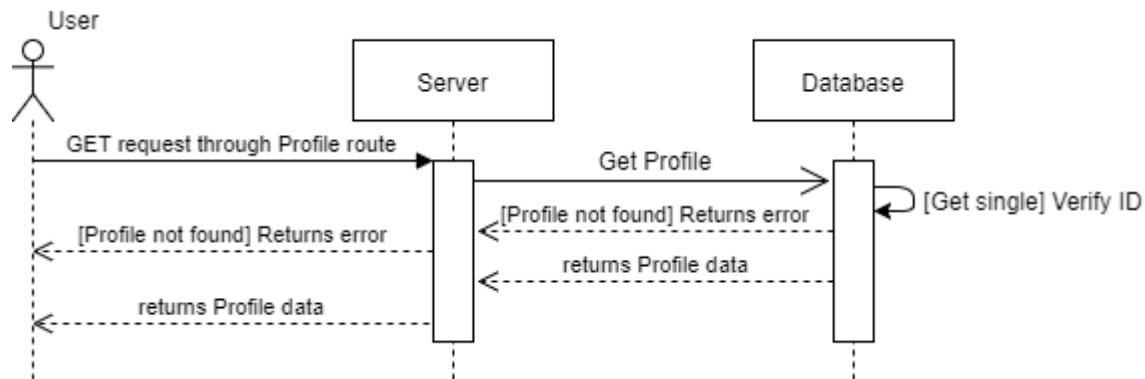


Sequence Diagrams

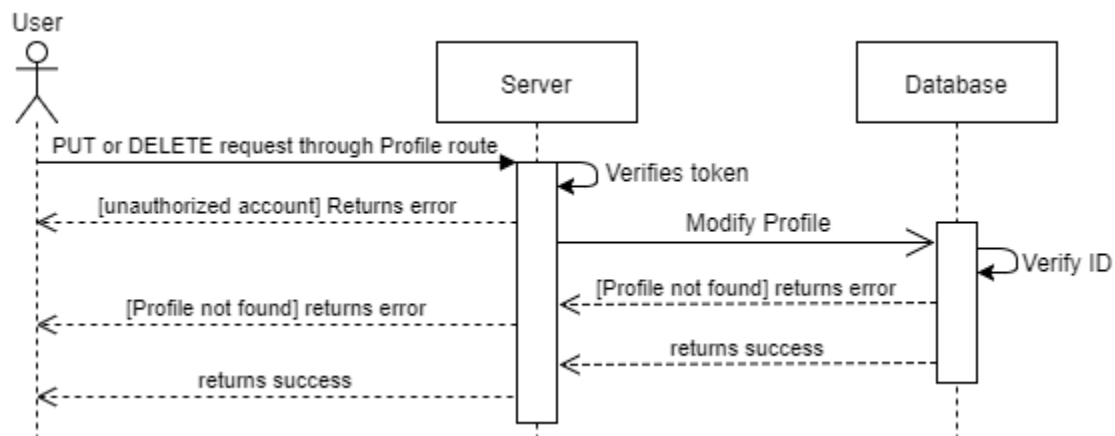
Create Profile



Get Profile Data



Modify Profile Data



Visual User Guide

GETlocalhost:8080/api/profilesParamsSend

AuthorizationHeaders (2)BodyPre-request ScriptTests

Key	Value	Description	Bulk Edit
<input checked="" type="checkbox"/> Content-Type	application/x-www-form-urlencoded		
<input checked="" type="checkbox"/> x-access-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0eXBlljoiQWRtaW5pc3R...		
New key	Value	Description	

BodyCookiesHeaders (9)Test ResultsStatus: 200 OK

PrettyRawPreviewJSON

```
1 [
2   {
3     "_id": "59f091a8b3118036f0c5ffaa",
4     "picUrl": "(Picture URL here)",
5     "brief": "(Profile Brief)",
6     "position": "(Position)",
7     "institution": "(Institution)",
8     "email": "(Email)",
9     "phone": "(Phone Number)",
10    "name": "Mikaila Daniel",
11    "_v": 0
12  },
13  {
14    "_id": "59f09441afd2cf44c065b0ff",
15    "picUrl": "(Picture URL here)",
16    "brief": "(Program Brief)",
17    "name": "(Name)",
18    "_v": 0,
19    "institution": "(Institution)",
20    "phone": "(Phone)"
21  },
22  {
23    "_id": "59f09464404ced42b07f2486",
24    "picUrl": "(Picture URL here)",
25    "brief": "(Program Brief)",
```

DELETElocalhost:8080/api/profiles/5a1dbf9f76c5aa3b808ea46aParamsSend

AuthorizationHeaders (2)BodyPre-request ScriptTests

Key	Value	Description	Bulk Edit
<input checked="" type="checkbox"/> Content-Type	application/x-www-form-urlencoded		
<input checked="" type="checkbox"/> x-access-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0eXBlljoiQWRtaW5pc3R...		
New key	Value	Description	

BodyCookiesHeaders (9)Test ResultsStatus: 200 OK

PrettyRawPreviewJSON

```
1 {
2   "message": "Successfully deleted profile."
3 }
```

SPRINT 4**USER STORY #678: TESTING CALLS TO THE DATABASE**

Description: **As a Developer I would like** to ensure function calls to the database return the requested data, **so that** the application data is displayed properly.

Acceptance Criteria

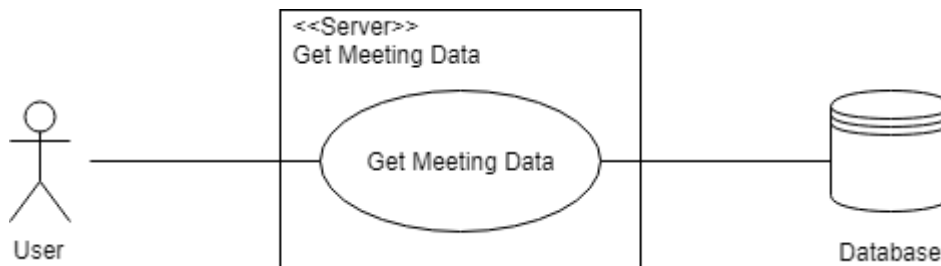
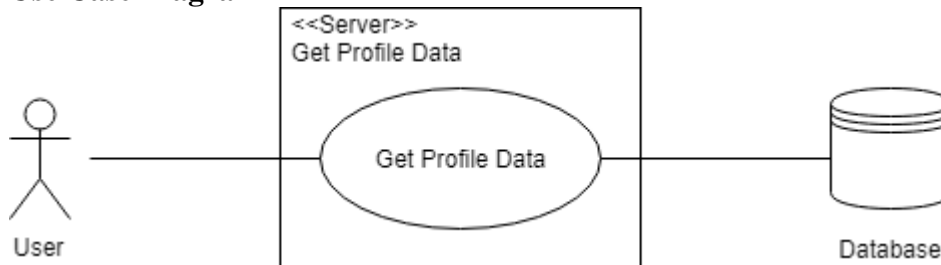
- Functions successfully fetch singular profiles.
- Functions successfully fetch meetings from database.

Use Cases

Use Case Name	Get Profile Data
Participating Actors	User, Server, Database
Entry Condition	1. User makes a GET request through the Profile route.
Flow of Events	2. Server retrieves Profile data from Database. If the request was made for a single Profile: 2.1: Server verifies given ID matches a Profile in Database. If the ID doesn't match any Profile in Database: 2.1.1: Go to step 3.2. If the ID does match a Profile in Database: 2.1.2: Go to step 3.1. If the request was made for multiple Profiles: 2.2: Go to step 3.1.
Exit Condition	3. Server returns response to User. If Profile data was successfully retrieved: 3.1. Server returns success message and Profile data to User. If Profile data was not successfully retrieved: 3.2. Server returns error message to User.

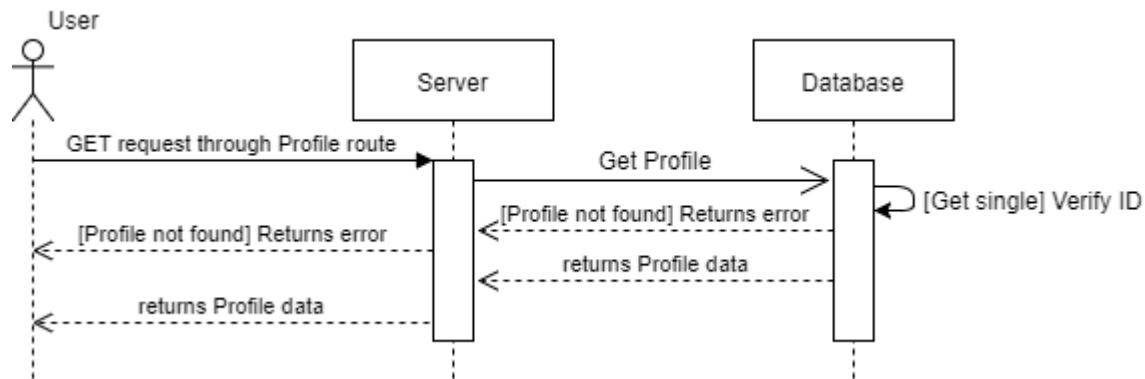
Use Case Name	Get Meeting Data
Participating Actors	User, Server, Database
Entry Condition	1. User makes a GET request through the Meeting route.
Flow of Events	<p>2. Server retrieves Meeting data from Database.</p> <p>If the request was made for a single Meeting:</p> <p>2.1: Server verifies given ID matches a Meeting in Database.</p> <p>If the ID doesn't match any Meeting in Database:</p> <p>2.1.1: Go to step 3.2.</p> <p>If the ID does match a Meeting in Database:</p> <p>2.1.2: Go to step 3.1.</p> <p>If the request was made for multiple Meetings:</p> <p>2.2: Go to step 3.1.</p>
Exit Condition	<p>3. Server returns response to User.</p> <p>If Meeting data was successfully retrieved:</p> <p>3.1. Server returns success message and Meeting data to User.</p> <p>If Meeting data was not successfully retrieved:</p> <p>3.2. Server returns error message to User.</p>

Use Case Diagram

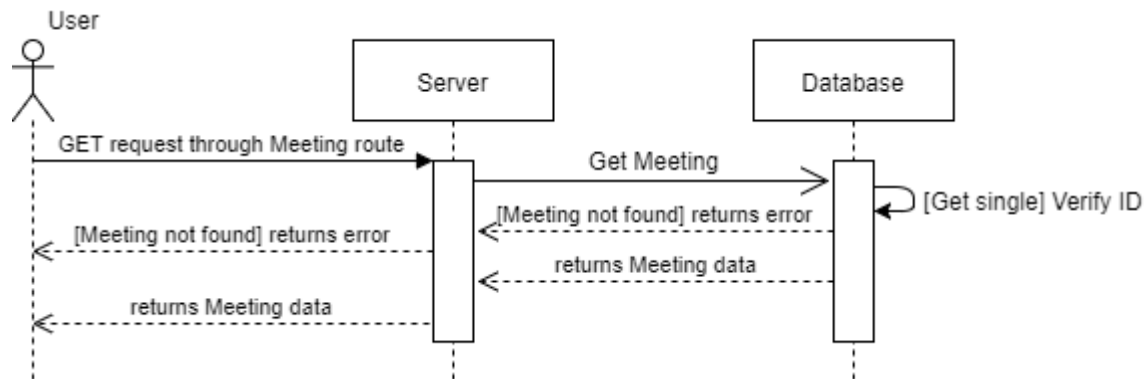


Sequence Diagrams

Get Profile Data



Get Meeting Data



Visual User Guide

GET ▼
localhost:8080/api/profiles/59f091a8b3118036fc5ffaa

GET ▼
localhost:8080/api/meetings/59f8be9540feba0b0af021c4

Authorization
Headers (2)
Body
Pre-request Script
Tests

Key	Value
<input checked="" type="checkbox"/> Content-Type	application
<input checked="" type="checkbox"/> x-access-token	eyJhbGciOiJIUzI1NiIsInR5cGE6ICJ1Iiwi
New key	Value

Body
Cookies
Headers (9)
Test Results

Pretty
Raw
Preview
JSON ▼

```

1 {
2   "id": "59f091a8b3118036fc5ffaa",
3   "picUrl": "(Picture URL here)",
4   "brief": "(Profile Brief)",
5   "position": "(Position)",
6   "institution": "(Institution)",
7   "email": "(Email)",
8   "phone": "(Phone Number)",
9   "name": "Mikaila Daniel",
10  "_v": 0
11 }

```

Body
Cookies
Headers (9)
Test Results

Pretty
Raw
Preview
JSON ▼

```

1 {
2   "id": "59f8be9540feba0b0af021c4",
3   "date": "Fri Nov 24 2017 14:19:47 GMT-0500 (EST)",
4   "minutes": "We are expecting 100 people to attend.",
5   "agenda": "We need to get a list of what everyone is going to bring.",
6   "title": "November Potluck",
7   "_v": 0,
8   "past": true
9 }

```

USER STORY #681: CREATE MEETING MODEL

Description: As a Developer **I would like** to create the model for the meeting object. **so that** the relevant data may be stored in the database.

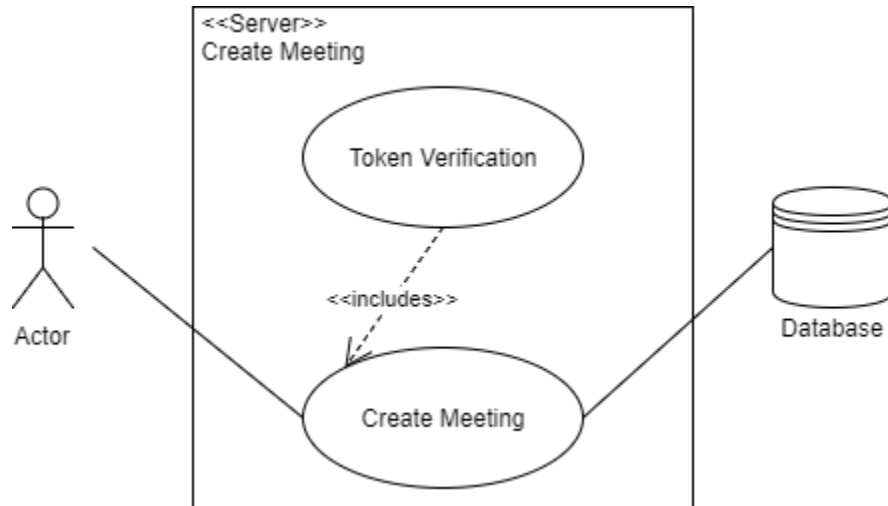
Acceptance Criteria

- Created JSON file for meeting model.

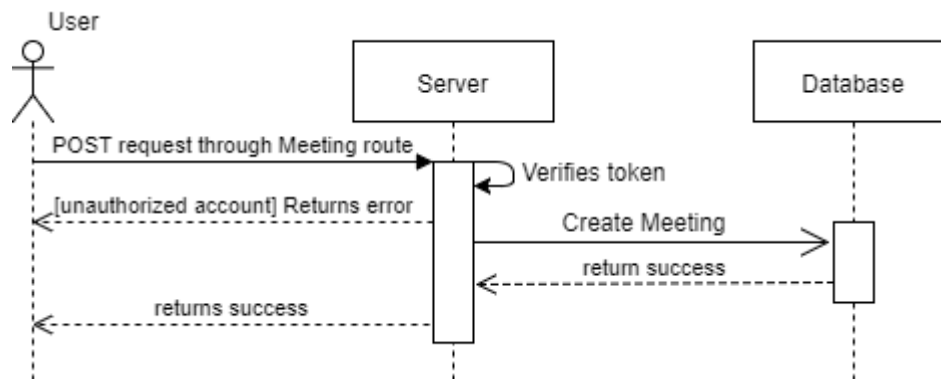
Use Case

Use Case Name	Create Meeting
Participating Actors	User, Server, Database
Entry Condition	1. User makes a POST request through the Meeting route.
Flow of Events	<p>2. Server verifies the user is authorized to create a meeting. If the user is not authorized: 2.1: Go to step 4.2. If the user is authorized: 2.2: Go to step 3.</p> <p>3. Database creates new Meeting entry. For each field in Meeting: If there is a matching entry in the request body: 3.1: Field value is set equal to value given in request body. If there is not a matching entry in the request body: 3.2: Field value is set to a default value.</p>
Exit Condition	<p>4. Server returns response to User. If Meeting was successfully created: 4.1. Server returns success message to User. If Meeting was not successfully created: 4.2. Server returns error message to User.</p>

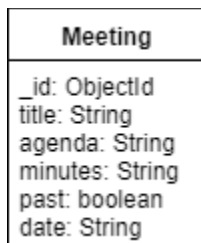
Use Case Diagram



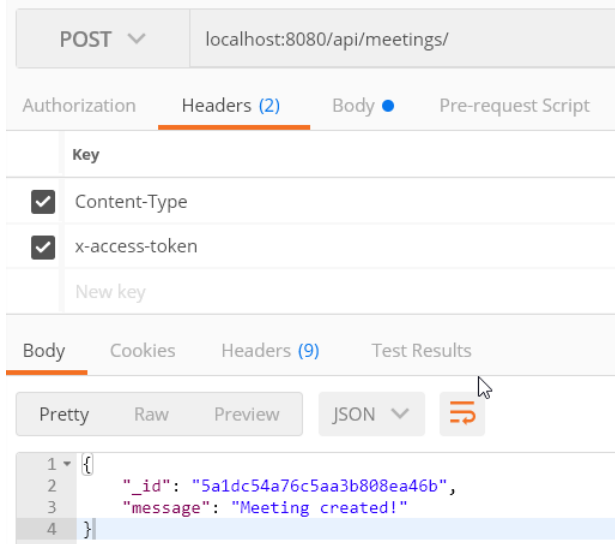
Sequence Diagram



Class Diagram



Visual User Guide



USER STORY #683: CREATE MEETING API ROUTES

Description: As a Developer I would like to create the routing for Meeting requests so that the meeting data may be accessed and manipulated within the database.

Acceptance Criteria

- Created routes for creating meeting entries in the database.
- Created routes for accessing meeting data in the database.
- Created routes for manipulating meeting data in the database.

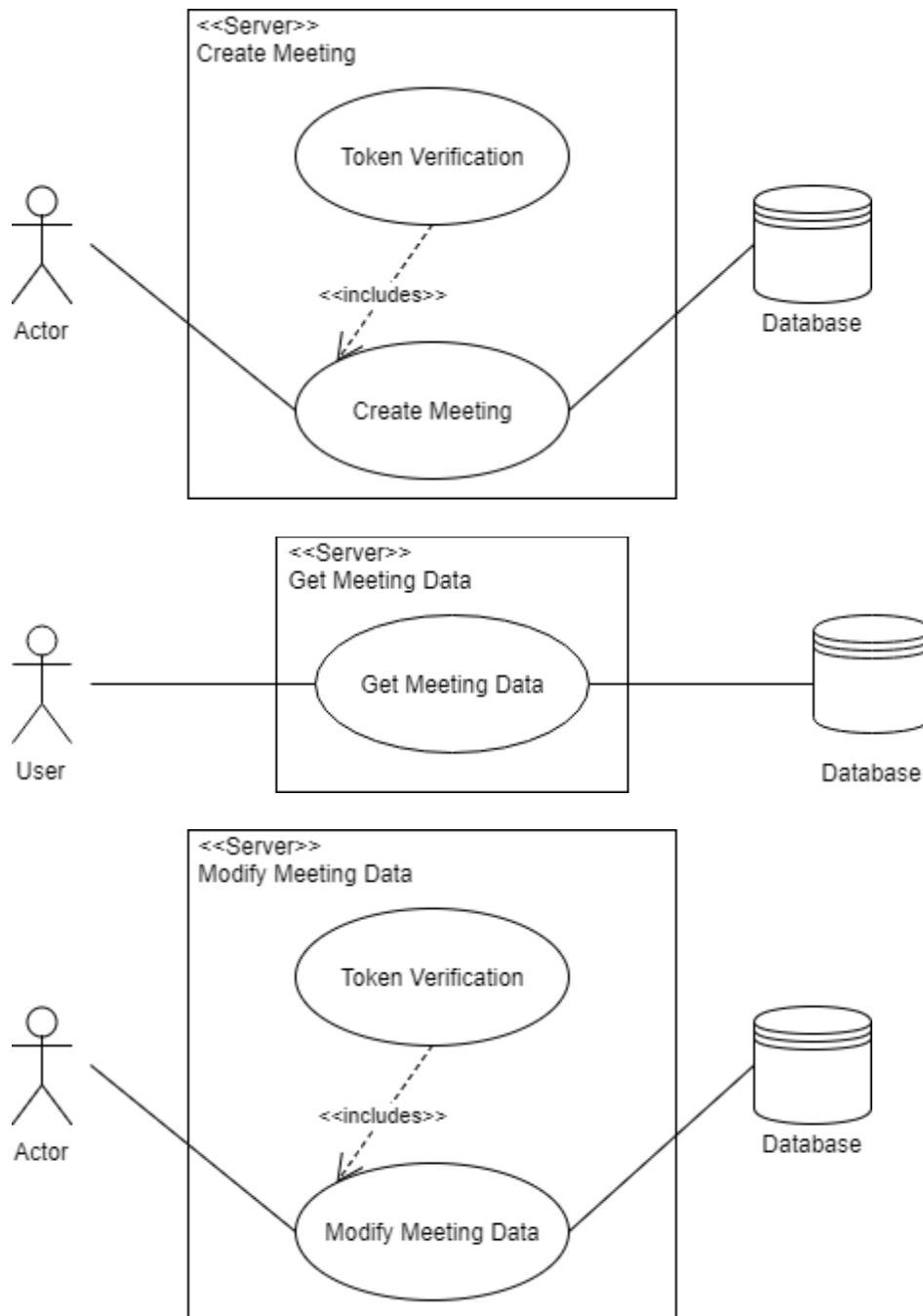
Use Cases

Use Case Name	Create Meeting
Participating Actors	User, Server, Database
Entry Condition	1. User makes a POST request through the Meeting route.
Flow of Events	<p>2. Server verifies the user is authorized to create a meeting.</p> <p> If the user is not authorized:</p> <p> 2.1: Go to step 4.2.</p> <p> If the user is authorized:</p> <p> 2.2: Go to step 3.</p> <p>3. Database creates new Meeting entry.</p> <p> For each field in Meeting:</p> <p> If there is a matching entry in the request body:</p> <p> 3.1: Field value is set equal to value given in request body.</p> <p> If there is not a matching entry in the request body:</p> <p> 3.2: Field value is set to a default value.</p>
Exit Condition	<p>4. Server returns response to User.</p> <p> If Meeting was successfully created:</p> <p> 4.1. Server returns success message to User.</p> <p> If Meeting was not successfully created:</p> <p> 4.2. Server returns error message to User.</p>

Use Case Name	Get Meeting Data
Participating Actors	User, Server, Database
Entry Condition	1. User makes a GET request through the Meeting route.
Flow of Events	<p>2. Server retrieves Meeting data from Database.</p> <p> If the request was made for a single Meeting:</p> <p> 2.1: Server verifies given ID matches a Meeting in Database.</p> <p> If the ID doesn't match any Meeting in Database:</p> <p> 2.1.1: Go to step 3.2.</p> <p> If the ID does match a Meeting in Database:</p> <p> 2.1.2: Go to step 3.1.</p> <p> If the request was made for multiple Meetings:</p> <p> 2.2: Go to step 3.1.</p>
Exit Condition	<p>3. Server returns response to User.</p> <p> If Meeting data was successfully retrieved:</p> <p> 3.1. Server returns success message and Meeting data to User.</p> <p> If Meeting data was not successfully retrieved:</p> <p> 3.2. Server returns error message to User.</p>

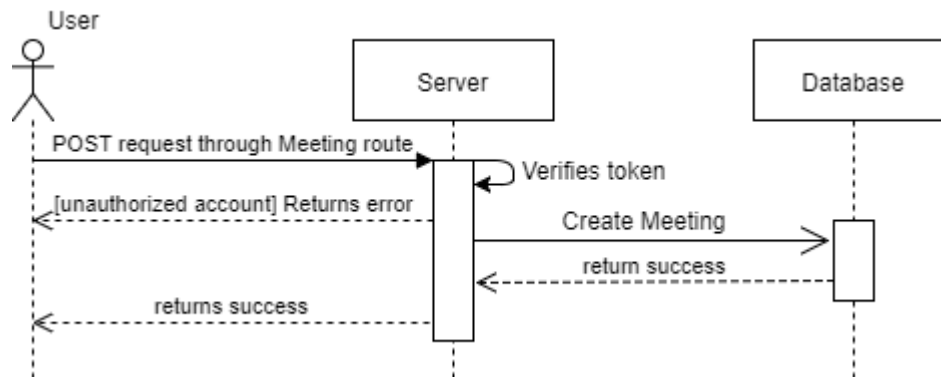
Use Case Name	Modifying Meeting Data
Participating Actors	User, Server, Database
Entry Condition	<p>1. User makes a request through the Meeting route with a Meeting ID.</p> <p> 1.1: User makes a PUT request.</p> <p> 1.2: User makes a DELETE request.</p>
Flow of Events	<p>2. Server verifies the user is authorized to modify a meeting.</p> <p> If the user is not authorized:</p> <p> 2.1: Go to step 4.2.</p> <p> If the user is authorized:</p> <p> 2.2: Go to step 3.</p> <p>3. Server verifies given ID matches a Meeting in Database.</p> <p> If the ID doesn't match any Meeting in Database:</p> <p> 3.1: Go to step 4.2.</p> <p> If the ID does match a Meeting in Database:</p> <p> 3.2: Requested modifications are made in Database.</p>
Exit Condition	<p>4. Server returns response to User.</p> <p> If Meeting was successfully modified:</p> <p> 4.1. Server returns success message to User.</p> <p> If Meeting was not successfully modified:</p> <p> 4.2. Server returns error message to User.</p>

Use Case Diagrams

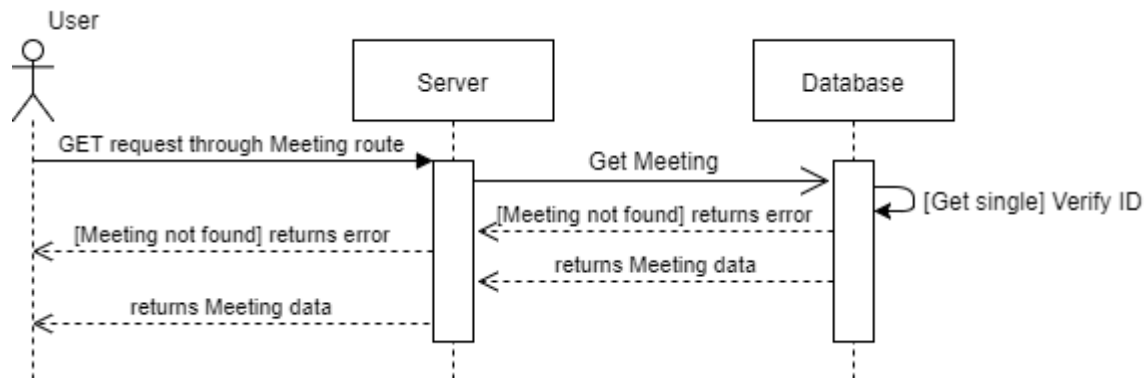


Sequence Diagram

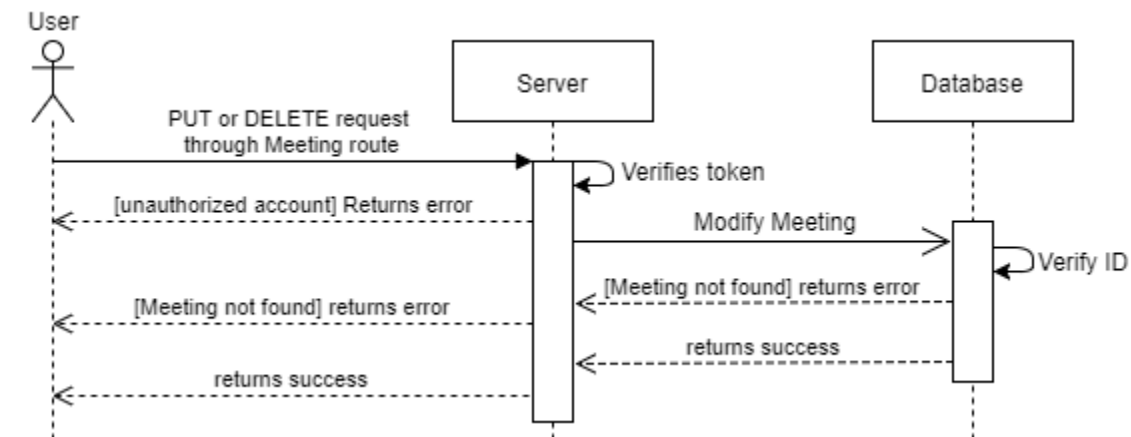
Create Meeting



Get Meeting Data



Modify Meeting Data



Visual User Guide

GET localhost:8080/api/meetings/

Authorization Headers (2) Body Pre-request Script Tests

Key	Value
<input checked="" type="checkbox"/> Content-Type	application/x-www-form-urlencoded
<input checked="" type="checkbox"/> x-access-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0eXBlljoiQWRtaW5pc3R
New key	Value

Body Cookies Headers (9) Test Results

Pretty Raw Preview JSON

```

1 [
2   {
3     "_id": "59f12310b7903d344409e9cf",
4     "date": "Wed Oct 25 2017 19:49:36 GMT-0400 (Eastern Daylight Time)",
5     "minutes": "(Post Minutes Body)",
6     "agenda": "(Post Agenda Body)",
7     "title": "(Meeting Title)",
8     "_v": 0,
9     "past": false
10  },
11  {
12    "_id": "59f8be9540feba0b0af021c4",
13    "date": "Fri Nov 24 2017 14:19:47 GMT-0500 (EST)",
14    "minutes": "We are expecting 100 people to attend.",
15    "agenda": "We need to get a list of what everyone is going to bring.",
16    "title": "November Potluck",
17    "_v": 0,
18    "past": true
19  },
20  {
21    "_id": "5a18a0906498e60b3546ad85",
22    "date": "Fri Nov 24 2017 17:49:18 GMT-0500 (EST)",
23    "minutes": "(Post Minutes Body)",
24    "agenda": "This is the first meeting used for the application. Created for testing purposes.",
25    "title": "First Meeting",
  
```

DELETE localhost:8080/api/meetings/5a1dc54a76c5aa3b808ea46b

Authorization Headers (2) Body ● Pre-request Script Tests

Key	Value
<input checked="" type="checkbox"/> Content-Type	application/x-www-
<input checked="" type="checkbox"/> x-access-token	eyJhbGciOiJIUzI1NiI
New key	Value

Body Cookies Headers (9) Test Results

Pretty Raw Preview JSON

```

1 {
2   "message": "Successfully deleted meeting."
3 }
  
```

USER STORY #684: CREATE USER MODEL AND ROUTES

Description: **As a Developer I would like** to create the infrastructure necessary to support user accounts and authentication for the application **so that** certain functions which require privileged access to the database may be enabled.

Acceptance Criteria

- Created JSON file for user database model.
- Created API routes for manipulating and accessing user data.

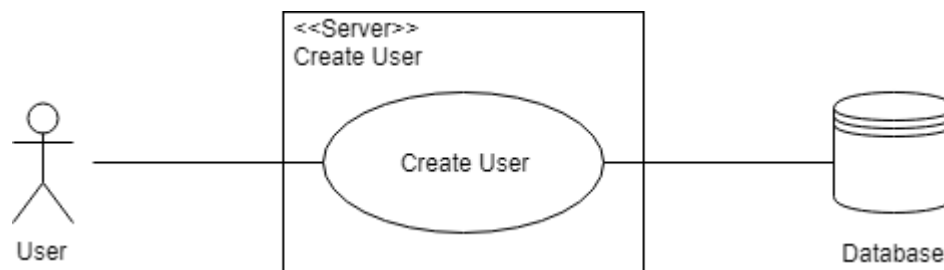
Use Cases

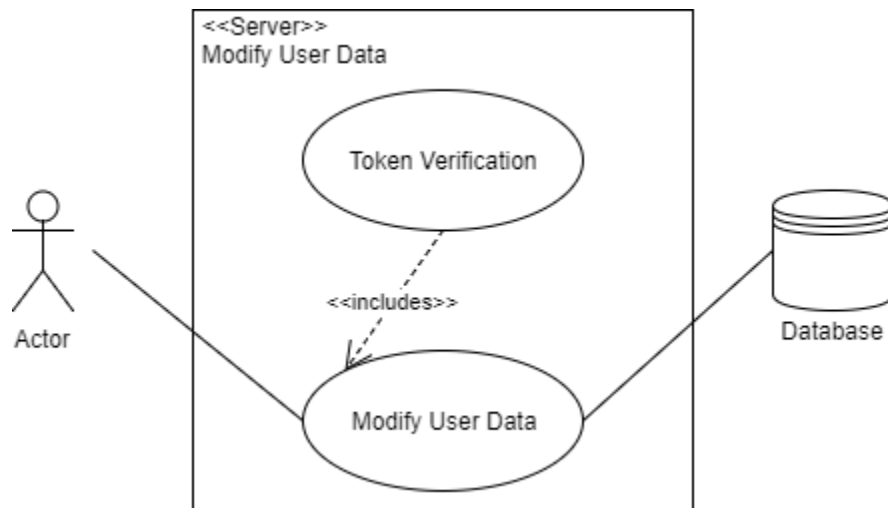
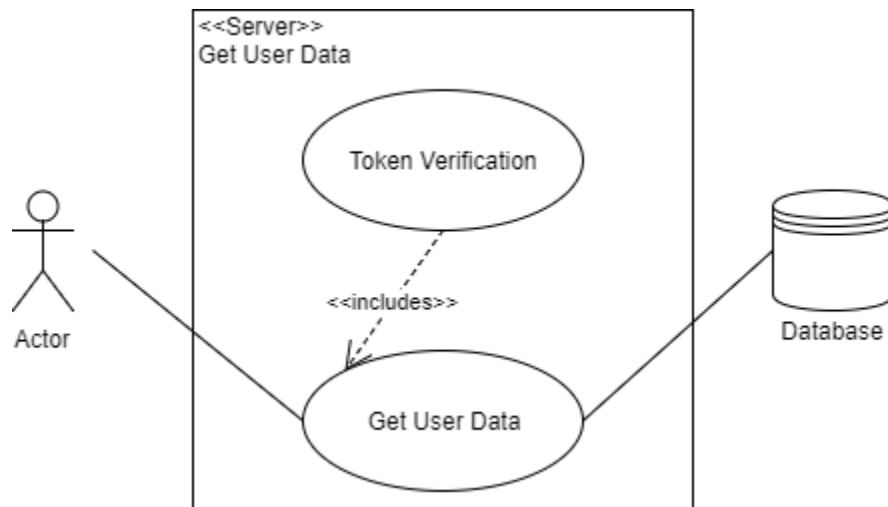
Use Case Name	Create User
Participating Actors	User, Server, Database
Entry Condition	1. User makes a POST request through the User route.
Flow of Events	2. Database creates new User account with given field values.
Exit Condition	3. Server returns response to User. If account was successfully created: 3.1. Server returns success message to User. If account was not successfully created: 3.2. Server returns error message to User.

Use Case Name	Get User Data
Participating Actors	User, Server, Database
Entry Condition	1. User makes a GET request through the User route.
Flow of Events	<p>2. Server verifies the User is authorized to get User data. If the user is not authorized: 2.1: Go to step 4.2. If the user is authorized: 2.2: Go to step 3.</p> <p>3. Server retrieves User data from Database. If the request was made for a single User: 3.1: Server verifies given ID matches a User in Database. If the ID doesn't match any User in Database: 3.1.1: Go to step 4.2. If the ID does match a User in Database: 3.1.2: Go to step 4.1. If the request was made for multiple Users: 3.2: Go to step 4.1.</p>
Exit Condition	<p>4. Server returns response to User. If User data was successfully retrieved: 4.1. Server returns success message and User data to User. If User data was not successfully retrieved: 4.2. Server returns error message to User.</p>

Use Case Name	Modifying User Data
Participating Actors	User, Server, Database
Entry Condition	1. User makes a request through the User route with a User ID. 1.1: User makes a PUT request. 1.2: User makes a DELETE request.
Flow of Events	2. Server verifies the user is authorized to modify a User. If the user is not authorized: 2.1: Go to step 4.2. If the user is authorized: 2.2: Go to step 3. 3. Server verifies given ID matches a User in Database. If the ID doesn't match any User in Database: 3.1: Go to step 4.2. If the ID does match a User in Database: 3.2: Requested modifications are made in Database.
Exit Condition	4. Server returns response to user. If User was successfully modified: 4.1. Server returns success message to user. If User was not successfully modified: 4.2. Server returns error message to user.

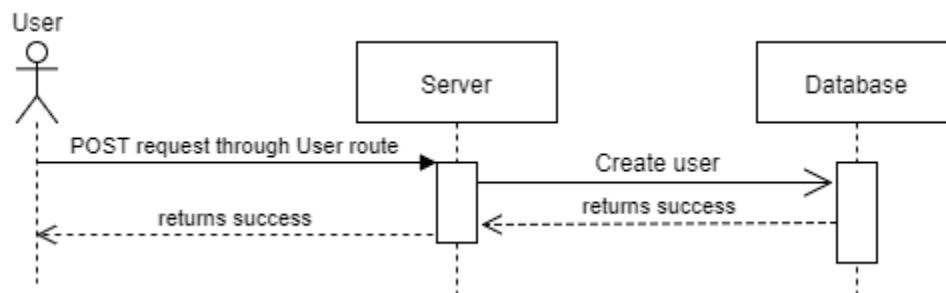
Use Case Diagrams



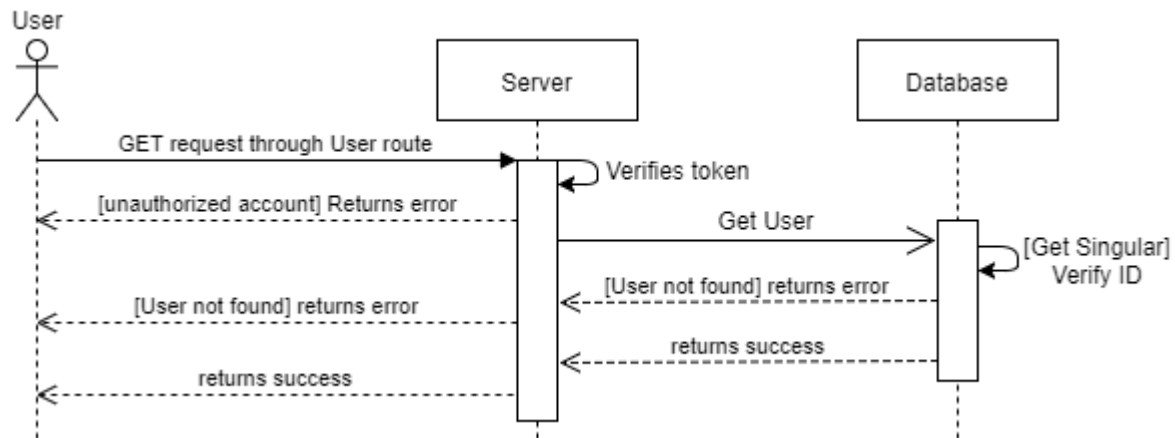


Sequence Diagram

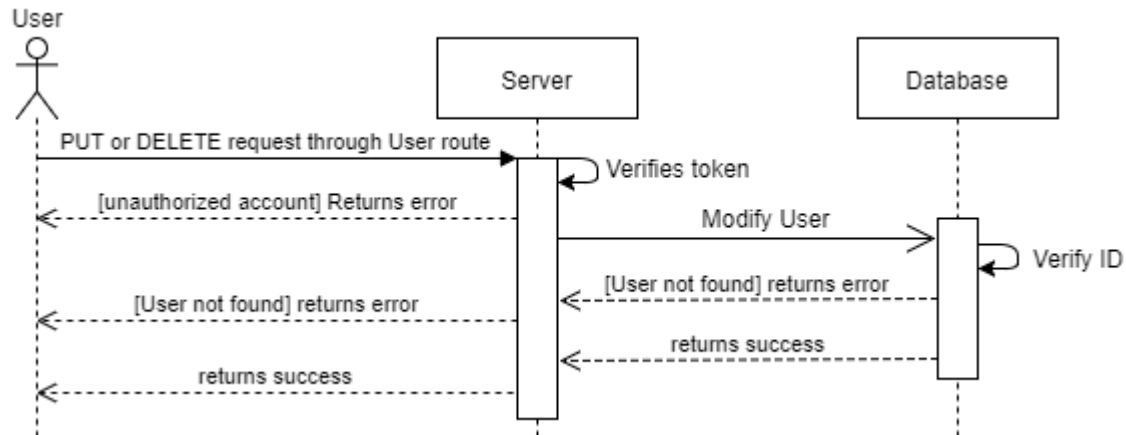
Create User



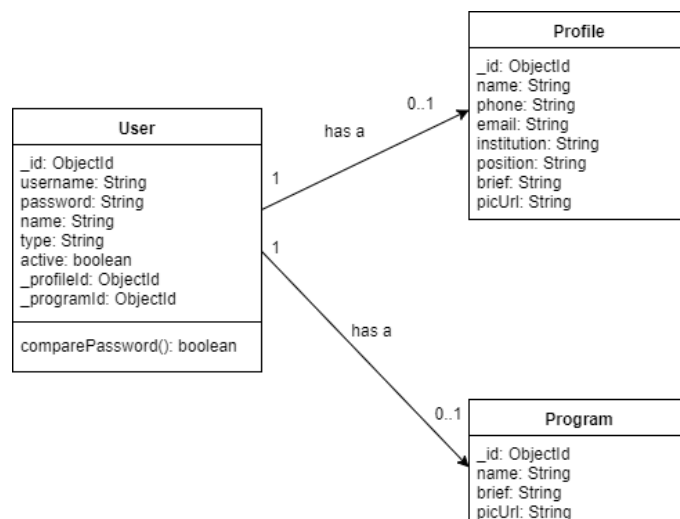
Get User Data



Modify User Data



Class Diagram



Visual User Guide

POST localhost:8080/api/users

Authorization Headers (2) Body Pre-request Script Tests

☐ form-data ☒ x-www-form-urlencoded ☐ raw ☐ binary

Key	Value
<input checked="" type="checkbox"/> username	testing
<input checked="" type="checkbox"/> password	test
<input checked="" type="checkbox"/> type	Member
New key	Value

Body Cookies Headers (9) Test Results

Pretty Raw Preview JSON

```
1 {  
2   "_id": "5a1dc86376c5aa3b808ea46d",  
3   "message": "User account created!"  
4 }
```

localhost:8080/api/us localhost:8080/api/authenti + ...

DELETE localhost:8080/api/users/5a1dc86376c5aa3b808ea46d

Authorization Headers (2) Body Pre-request Script Tests

Key	Value
<input checked="" type="checkbox"/> Content-Type	applicat
<input checked="" type="checkbox"/> x-access-token	eyJhbGc
New key	Value

Body Cookies Headers (9) Test Results

Pretty Raw Preview JSON

```
1 {  
2   "message": "Successfully deleted user account."  
3 }
```


USER STORY #688: USER AUTHENTICATION

Description: As a Product Owner **I would like** users logging into an account on the application to be authenticated **so that** privileged actions within the application are secure.

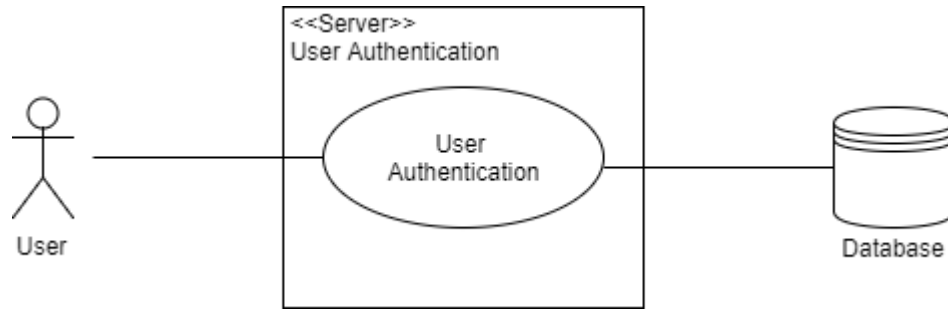
Acceptance Criteria

- System accepts username/password combinations that correspond to user entries in the database, given the account is active (approved by moderator or administrator).
- Authorized users are provided a token to allow access to privileged actions.
- System rejects invalid username/password combinations and accounts which are not active.

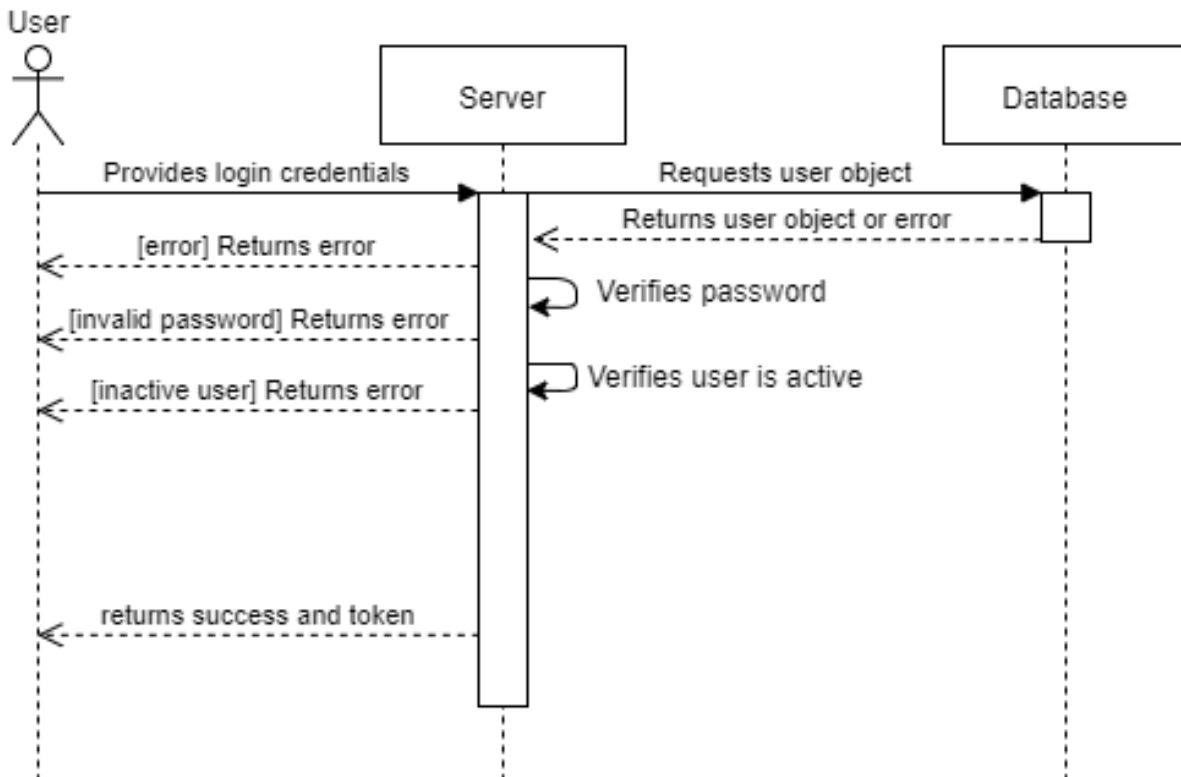
Use Case

Use Case Name	User Authentication
Participating Actors	User, Server, Database
Entry Condition	1. User provides credentials to Server.
Flow of Events	<p>2. Server verifies that the username exists in the Database. If username does not exist in the Database: 3.1. Go to step 5.2. If username does exist in the Database: 3.2. Go to step 3.</p> <p>3. Server verifies the provided password matches the password of the account associated with the username in the Database. If the passwords do not match: 4.1: Go to step 5.2. If the passwords do match: 4.2: Go to step 4.</p> <p>4. Server verifies that the account is currently active. If the account is not active: 5.1: Go to step 5.2. If the account is active: 5.2: Go to step 5.1.</p>
Exit Condition	<p>5. Server returns response to User. If credentials were successfully authenticated: 6.1. Server returns success message and token to User. If credentials were not successfully authenticated: 6.2. Server returns error message to User.</p>

Use Case Diagram



Sequence Diagram



Visual User Guide

POST

localhost:8080/api/authenticate

Params

Send

Authorization

Headers (2)

BodyPre-request ScriptTests

form-data

x-www-form-urlencoded

raw

binary

	Key	Value	Description
<input checked="" type="checkbox"/>	username	SampleAdmin	
<input checked="" type="checkbox"/>	password	administrator	
	New key	Value	Description

BodyCookiesHeaders (9)Test Results

Status: 200 OK

Pretty

Raw

Preview

JSON

1

{

2

"success": true,

3

"message": "Authentication success, token passed.",

4

"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0eXB1IjoiaWRTaW5pc3RyYXRvcisiInVzZXJuYW11IjoieU2FtcGx1QWRtaW4iLCJpYXQiOiE1MTE4OTg4OTgsImV4cCI6MTUxMTk4NTI5OH0.YvBGNqp3y6Y1TmUJny1Bhs0rVGRgzXk0_qCsBwguLS0"

5

}

USER STORY #690: TOKEN VERIFICATION

Description: **As a Developer I would like** the application to verify the user's token **so that** their level of privilege is appropriate for the current API call to ensure the application is secure.

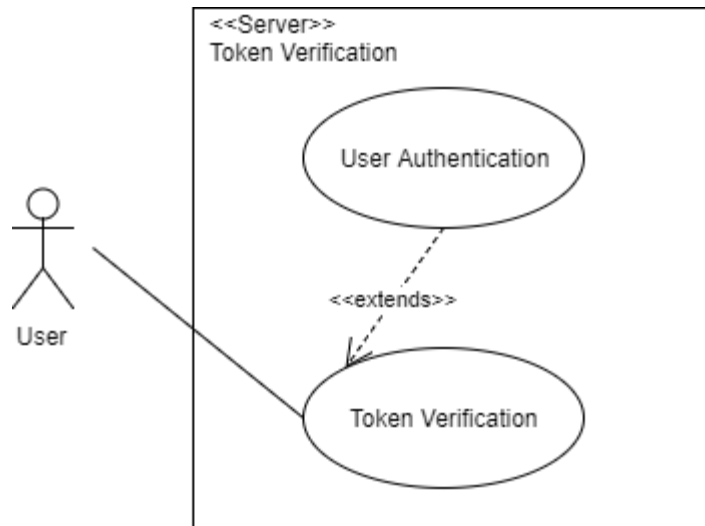
Acceptance Criteria

- Application allows access to users which have sufficient privileges for the current API call.
- Application denies access to users which do not have sufficient privileges for the current API call.

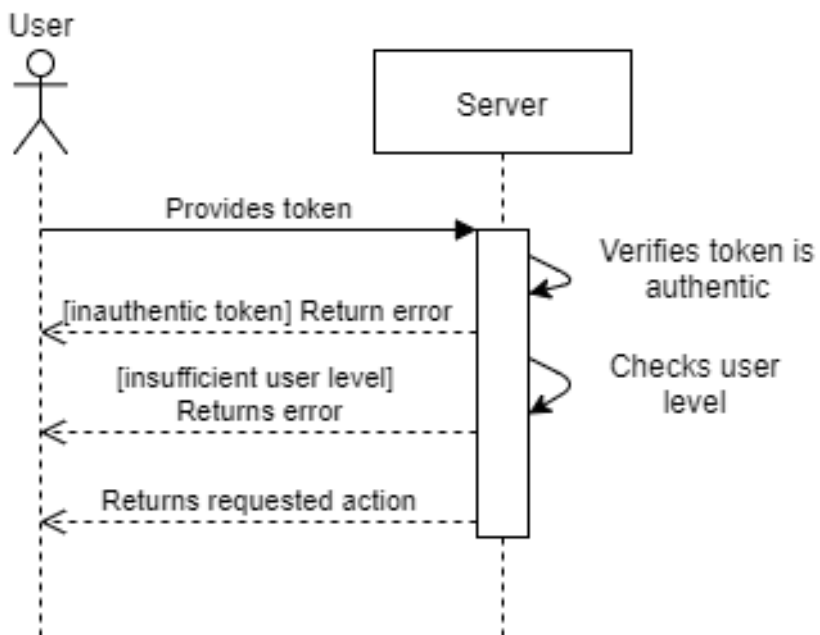
Use Case

Use Case Name	Token Verification
Participating Actors	User, Server
Entry Condition	1. User provides token to Server.
Flow of Events	2. Server verifies token authenticity using predefined secret phrase. If token is not authentic: 2.1. Go to step X. If token is authentic: 2.2. Go to step 3. 3. Server checks the user level contained within the token. If the user level is insufficient for the requested action: 3.1. Go to step X. If the user level is sufficient for the requested action: 3.2. Go to step 4.
Exit Condition	4. Server either allows or prevents access. If token does not meet requirements for access: 4.1. Server returns error message. If token does meet requirements for access: 4.2. Server continues with requested action.

Use Case Diagram



Sequence Diagram



Visual User Guide

POST ▾


localhost:8080/api/profiles

Key	Value
New key	Value

Authorization Headers (2) Body ● Pre-request Script Tests

Key	Value
<input checked="" type="checkbox"/> Content-Type	application/x-www-form-urlencoded
<input checked="" type="checkbox"/> x-access-token	sfsdafd
New key	Value

Body Cookies Headers (9) Test Results

Pretty Raw Preview JSON ▾ 

1 ▾ {

2 "success": false,

3 "message": "Failed to authenticate token."

4 }

SPRINT 5

USER STORY #682: VIEW MEETINGS

Description: As a general user, I would like to view the past announcements of Positive Pathways so that I can remain up-to-date with the current events.

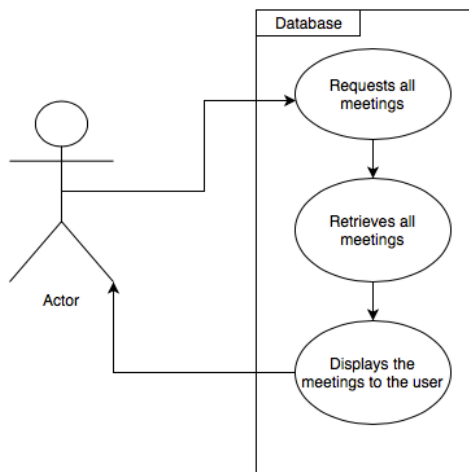
Acceptance Criteria

- The announcements page must return all the previous announcements in reverse chronological order.

Use Case

- Name: View Meetings
- Actor: User, Database
- Preconditions: The user must clicked “View Meetings/Future Events”
- Description:
 1. The application connects to the database.
 2. The application requests all meetings.
 3. The application displays the date of each meeting and includes links to the meeting minutes.

Use Case Diagram



USER STORY #685: CREATE PROGRAM MODEL

Description: As a Developer **I would like** to create the model for the program object **so that** the relevant data may be stored in the database.

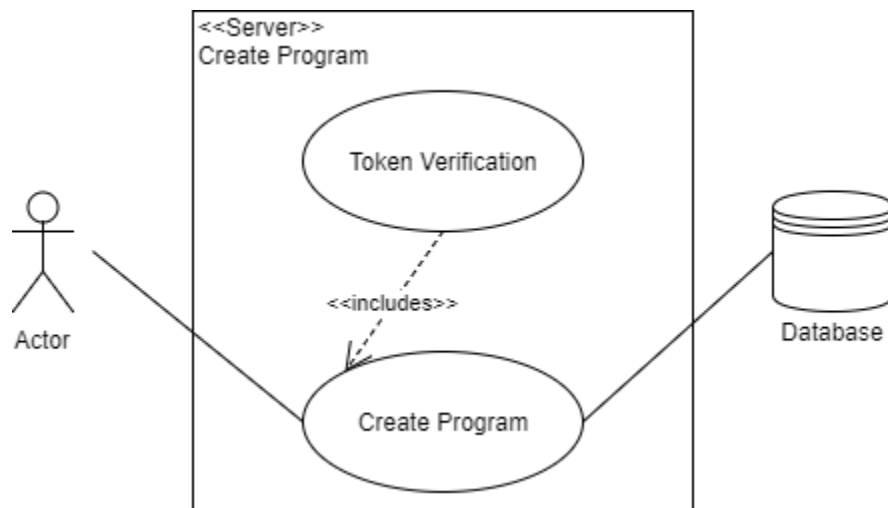
Acceptance Criteria

- Created JSON file for program model.

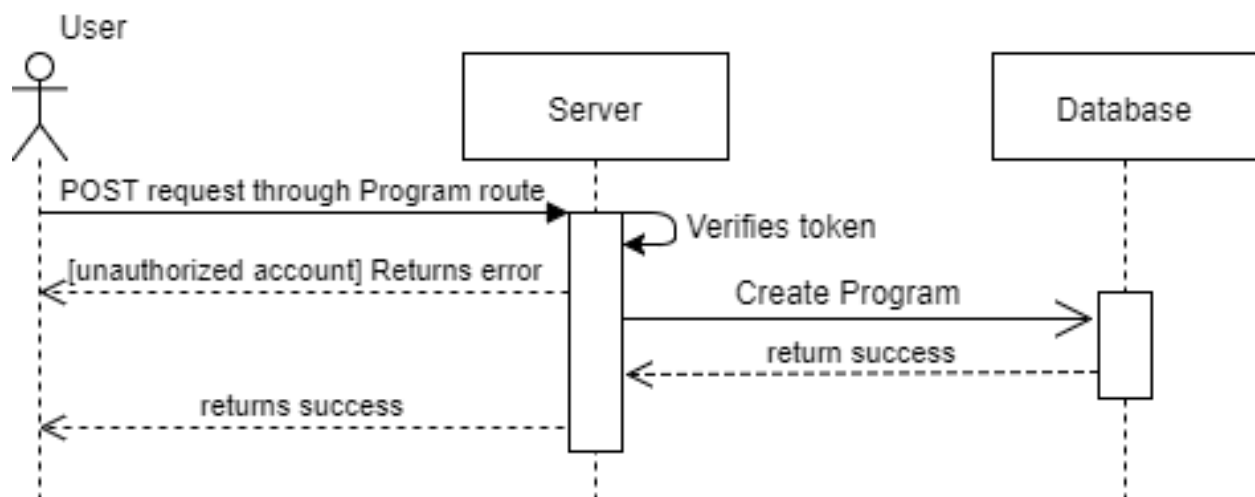
Use Case

Use Case Name	Create Program
Participating Actors	User, Server, Database
Entry Condition	1. User makes a POST request through the Program route.
Flow of Events	<p>2. Server verifies the user is authorized to create a program. If the user is not authorized: 2.1: Go to step 4.2. If the user is authorized: 2.2: Go to step 3.</p> <p>3. Database creates new Program entry. For each field in Program: If there is a matching entry in the request body: 3.1: Field value is set equal to value given in request body. If there is not a matching entry in the request body: 3.2: Field value is set to a default value.</p>
Exit Condition	<p>4. Server returns response to User. If Program was successfully created: 4.1. Server returns success message to User. If Program was not successfully created: 4.2. Server returns error message to User.</p>

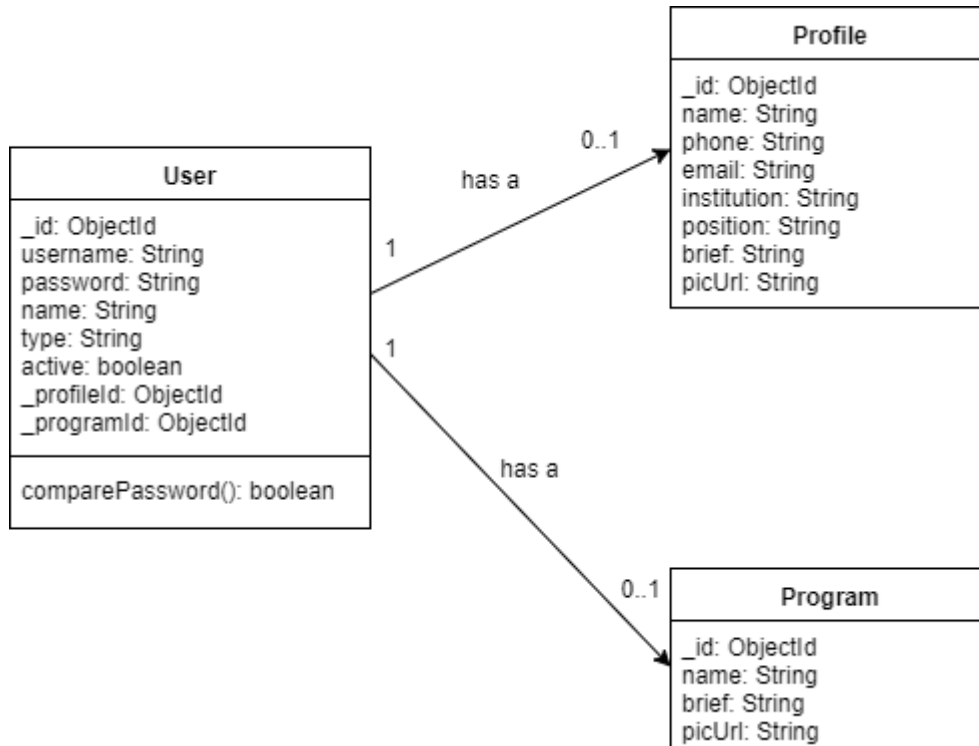
Use Case Diagram



Sequence Diagram



Class Diagram



Visual User Guide

The screenshot shows the Postman interface for a POST request to `localhost:8080/api/programs`. The **Headers** tab is selected, displaying two headers: `Content-Type` and `x-access-token`, both of which are checked. The **Body** tab is also visible, showing a JSON response with the following structure:

```

1 {
2   "_id": "5a1dcdf176c5aa3b808ea46e",
3   "message": "Program created!"
4 }

```

USER STORY #686: CREATE PROGRAM API ROUTES

Description: As a Developer I would like to create the routing for Program requests so that the program data may be accessed and manipulated within the database.

Acceptance Criteria

- Created routes for creating program entries in the database.
- Created routes for accessing program data in the database.
- Created routes for manipulating program data in the database.

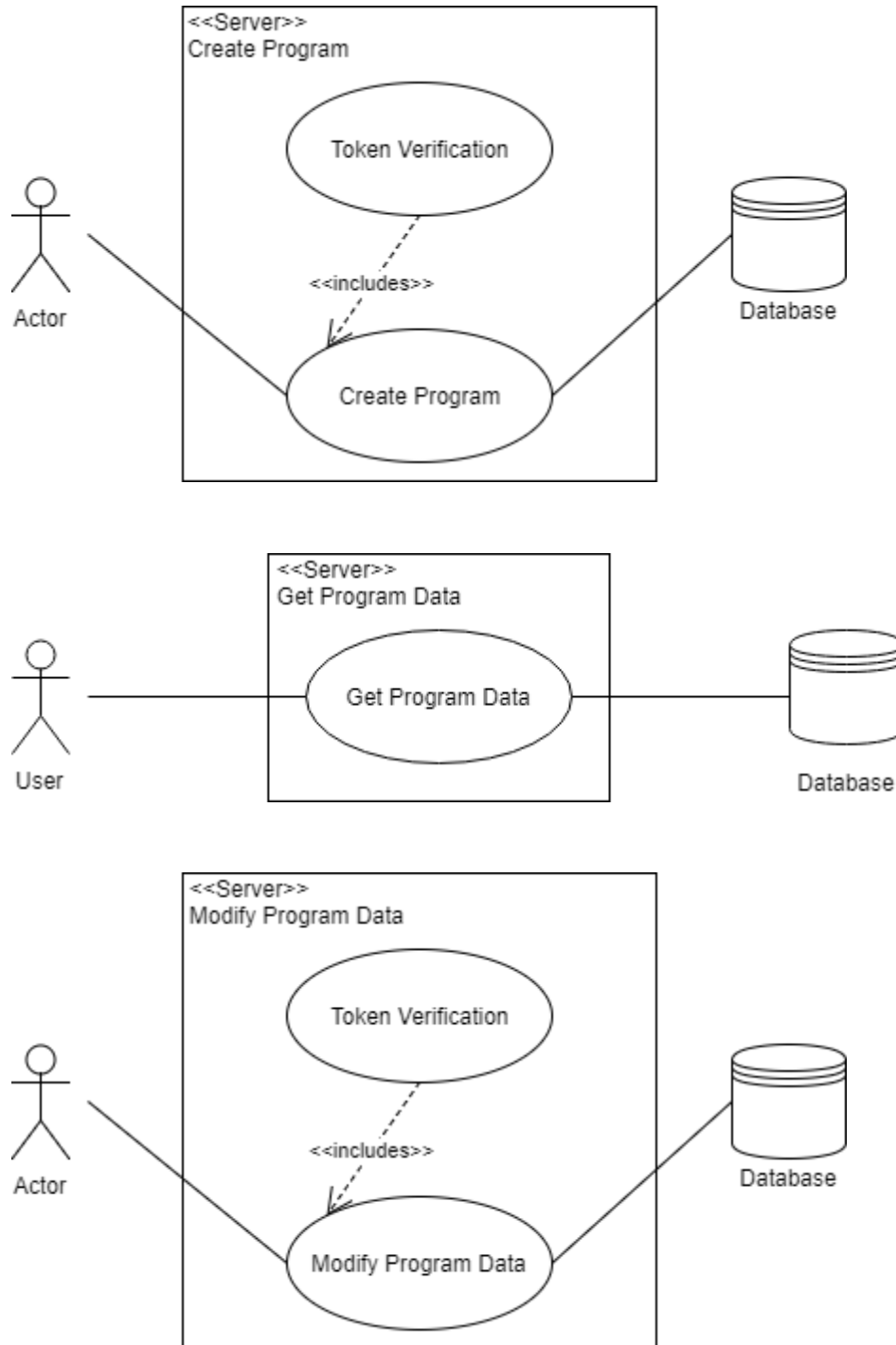
Use Case

Use Case Name	Create Program
Participating Actors	User, Server, Database
Entry Condition	1. User makes a POST request through the Program route.
Flow of Events	<p>2. Server verifies the user is authorized to create a program.</p> <p> If the user is not authorized:</p> <p> 2.1: Go to step 4.2.</p> <p> If the user is authorized:</p> <p> 2.2: Go to step 3.</p> <p>3. Database creates new Program entry.</p> <p> For each field in Program:</p> <p> If there is a matching entry in the request body:</p> <p> 3.1: Field value is set equal to value given in request body.</p> <p> If there is not a matching entry in the request body:</p> <p> 3.2: Field value is set to a default value.</p>
Exit Condition	<p>4. Server returns response to User.</p> <p> If Program was successfully created:</p> <p> 4.1. Server returns success message to User.</p> <p> If Program was not successfully created:</p> <p> 4.2. Server returns error message to User.</p>

Use Case Name	Get Program Data
Participating Actors	User, Server, Database
Entry Condition	1. User makes a GET request through the Program route.
Flow of Events	<p>2. Server retrieves Program data from Database.</p> <p> If the request was made for a single Program:</p> <p> 2.1: Server verifies given ID matches a Program in Database.</p> <p> If the ID doesn't match any Program in Database:</p> <p> 2.1.1: Go to step 3.2.</p> <p> If the ID does match a Program in Database:</p> <p> 2.1.2: Go to step 3.1.</p> <p> If the request was made for multiple Programs:</p> <p> 2.2: Go to step 3.1.</p>
Exit Condition	<p>3. Server returns response to User.</p> <p> If Program data was successfully retrieved:</p> <p> 3.1. Server returns success message and Program data to User.</p> <p> If Program data was not successfully retrieved:</p> <p> 3.2. Server returns error message to User.</p>

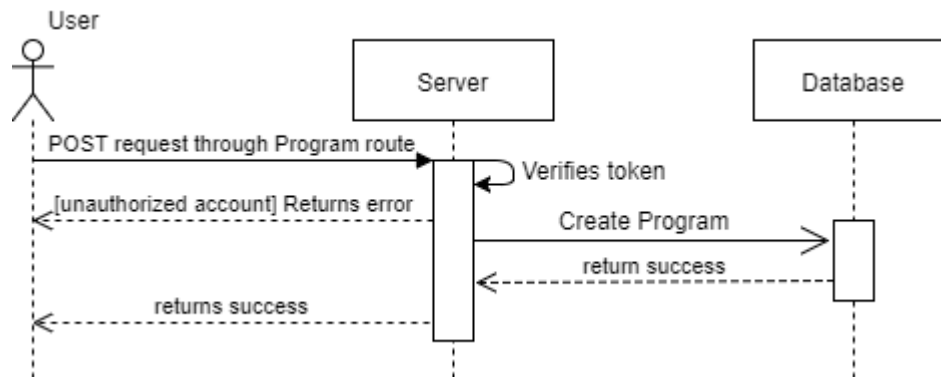
Use Case Name	Modifying Program Data
Participating Actors	User, Server, Database
Entry Condition	<p>1. User makes a request through the Program route with a Program ID.</p> <p> 1.1: User makes a PUT request.</p> <p> 1.2: User makes a DELETE request.</p>
Flow of Events	<p>2. Server verifies the user is authorized to modify a program.</p> <p> If the user is not authorized:</p> <p> 2.1: Go to step 4.2.</p> <p> If the user is authorized:</p> <p> 2.2: Go to step 3.</p> <p>3. Server verifies given ID matches a Program in Database.</p> <p> If the ID doesn't match any Program in Database:</p> <p> 3.1: Go to step 4.2.</p> <p> If the ID does match a Program in Database:</p> <p> 3.2: Requested modifications are made in Database.</p>
Exit Condition	<p>4. Server returns response to User.</p> <p> If Program was successfully modified:</p> <p> 4.1. Server returns success message to User.</p> <p> If Program was not successfully modified:</p> <p> 4.2. Server returns error message to User.</p>

Use Case Diagrams

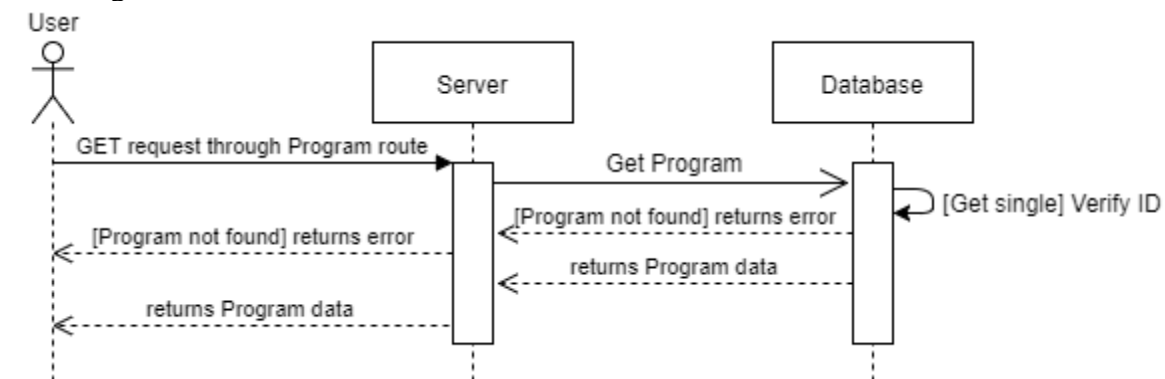


Sequence Diagram

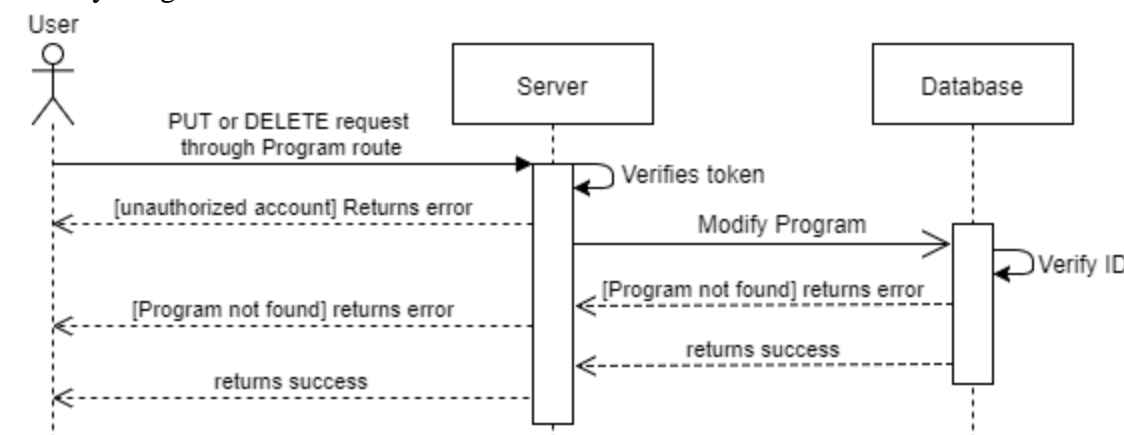
Create Program



Get Program Data



Modify Program Data



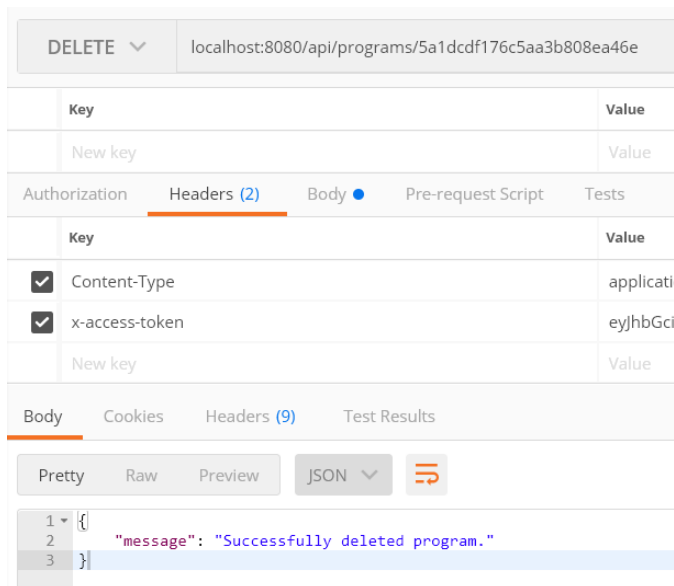
Visual User Guide



```
GET localhost:8080/api/programs

Pretty Raw Preview JSON

[
  {
    "_id": "59f09441afd2cf44c065b0ff",
    "picUrl": "(Picture URL here)",
    "brief": "(Program Brief)",
    "name": "(Name)",
    "__v": 0,
    "institution": "(Institution)",
    "phone": "(Phone)"
  },
  {
    "_id": "59f09464404ced42b07f2486",
    "picUrl": "(Picture URL here)",
    "brief": "(Program Brief)",
    "name": "(Name)",
    "__v": 0,
    "institution": "(Institution)"
  },
  {
    "_id": "5a1bc29e06ca460198c695d8",
    "picUrl": "(Picture URL here)",
    "brief": "(Profile Brief)",
    "position": "(Position)",
    "institution": "(Institution)",
    "email": "(Email)",
    "phone": "(Phone Number)",
    "name": "(Name)",
    "__v": 0
  },
  {
    "_id": "5a1dcdf176c5aa3b808ea46e",
    "picUrl": "(Picture URL here)",
    "brief": "(Program Brief)",
    "name": "(Name)",
    "__v": 0
  }
]
```



```
DELETE localhost:8080/api/programs/5a1dcdf176c5aa3b808ea46e

Key Value
New key Value

Authorization Headers (2) Body Pre-request Script Tests
Key Value
[X] Content-Type applicati
[X] x-access-token eyJhbGciOi
New key Value

Body Cookies Headers (9) Test Results
Pretty Raw Preview JSON

1 {
2   "message": "Successfully deleted program."
3 }
```

SPRINT 6

USER STORY #680: CREATE NEW MEETING

- Description: As a Liaison, I want to create announcements to notify the group of upcoming events.

Acceptance Criteria

- Display a form to fill out, and does not accept partially completed meetings.
- Check if the user has authorization to create a new meeting
- Automatically refreshes the list to display the new meeting (if on the Future Meetings page.)

Use Case

- Name: Create New Meeting
- Actor: User (administrator or liaison), Database
- Preconditions: User must be logged into an approved account and on the “Future Events” page.
- Description:
 1. When the user clicks on the button “Create New Meeting,” the application checks to see if the user is logged in.
 - a. If the user is logged in, we go to step 2.
 - b. If the user is not logged in, we display a message stating the user needs to log in and STOP.
 2. The application calls the fragment with the empty fields to create an announcement.
 3. When the user clicks “Create New Meeting,” the application creates a JSON object based on the input and sends it to the database.
 4. The page refreshes and the newest meeting is displayed to the user.

USER STORY #691: VIEW MEMBER PROFILE

- Description: As a user, I would want to use the application to find the liaison contact information for the liaison from my university, as well as find out the home university of each member in the application.

Acceptance Criteria

- Load a list of current programs in the application and have each image be a link to the more detailed page.
- Load a list of current members who are part of Positive Pathways and have each image be a link to the more detailed page.

Use Case

- Name: View Member Profile
- Actor: User, Database
- Preconditions: User must be on either the “Programs”, “Partners” or “Members” pages.
- Description:
 1. The application GETs the JSONArray corresponding to the page’s api route from the database.
 2. The application inflates a listview with the simple view of the profiles. Each profile is a List_item
 3. If the user clicks on a List_item, then the entire profile is sent to a more detailed layout page.
 4. The elements of the page are filled in according to the profile’s information. All empty elements are not shown to the user.
 5. If the user selects “Edit” then Edit Profile is run.
 6. Otherwise, when the user selects “Back/Up” then the application returns to the previous list.

USER STORY #692: LOG ONTO SERVER FROM APPLICATION

- Description: As a user I would like to modify information relevant to me and my program and not allow other users to modify it.

Acceptance Criteria

- Log onto the server and maintain a token for the purpose of creating new meetings and modifying profiles.

Use Case

- Name: Log onto Server from Application
- Actor: User, Database
- Preconditions: None. This page can be accessed at any point in the application.
- Description:
 1. When the user clicks on the “Login” button, a screen is displayed with two input boxes for the username and password.
 2. When the user clicks “Login”, the application attempts to log in by sending a JSON with the username and password to the authentication route.
 3. If the login is successful, the token is saved for use in other parts of the program, and the application returns to the previous screen.
 4. If the login is unsuccessful, then an error message s displayed to the user.

USER STORY #693: SEARCH BY NAME OR UNIVERSITY

- Description: As a user, I would like to search for a member name or university so that I can access their program information.

Acceptance Criteria

- Load the profiles partially matching on either name or institution.
- Be able to click on the profile and view the detailed page.

Use Case

- Name: Search by Name or University
- Actor: User, Database
- Preconditions: User must be on the main page of the application.
- Description:
 1. User types value into the search bar and clicks search.
 2. The application makes a request to receive a JSONArray containing all the profiles.
 3. The application iterates through each profile and keeps those that match on either name or institution.
 4. Clicking on a profile runs the use case View Member Profile (#691)

USER STORY #694: EDIT MEETING

- Description: As a liaison I would like to edit meetings so that I can alert the other members of updates to the meeting minutes, as well as set a meeting as past.

Acceptance Criteria

- Display the meeting elements in editable boxes, as well as a checkbox for marking a meeting as “passed.”
- Check if the user has authorization to edit or delete a meeting
- Automatically refreshes the list to display the updated meetings list.

Use Case

- Name: Edit Meeting
 - Actor: User (administrator or liaison), Database
 - Preconditions: User must be on either the “Future Meetings” or “Past Meetings” page.
 - Description:
 1. When the user clicks on a meeting, the application loads the meeting into EditText boxes.
 2. When the user is done making changes, they can either:
 - a. Click on the button labeled “Submit Meeting” and go to step 3.
 - b. Click on the button labeled “Delete Meeting” and go to step 4.
 - c. Click on the “Back/Up” button in the toolbar and go to step 7.
 3. The application checks to see if the user is logged in
- . If the user is logged in, we go to step 5.

USER STORY #695: EDIT PROFILE

- Description: As an administrative user, I need to edit the profiles of the liaisons, as well as allow them to edit their own profiles so that the application information can remain up to date.

Acceptance Criteria

- Load the profile in an editable textbox and be able to make in-line edits.
- Changes are all reflected in the application as long as the user has proper authorization to make these changes.

Use Case

- Name: Edit Profile
- Actor: User, Database
- Preconditions: User must be viewing a full profile and have clicked on “Edit Profile”
- Description:
 1. The application switches the set of textviews with editable boxes filled with the profile values.
 2. The user is free to edit any of the boxes. Previously hidden boxes will be visible to edit.
 3. When the user is done making changes, they can either
 - a. Click on the button labeled “Finish Editing” and go to step 4
 - b. Click on the button labeled “Delete Profile” and go to step 5
 - c. Click on the “Back/Up” button in the toolbar and go to step 8
 4. The application checks to see if the user is logged in
 - a. If the user is logged in, we go to step 6.
 - b. If the user is not logged in, we display a message stating the user needs to log in and go to step 2.
 5. The application checks to see if the user has sufficient authorization
 - a. If the user does, we go to step 7.
 - b. If not, we display this information to the user and go to step 2.
 6. The application sends a PUT command into the database based on the id of the meeting being edited. Go to step 8.
 7. The application displays a message requesting secondary confirmation that the user wants to delete the profile.
 - a. If the user says “YES”, send the DELETE command to the database based on the id of the profile being edited. Go to step 8.
 - b. If the user selects “NO”, go to step 1.
 8. Refresh the list displayed to the user before they clicked on the meeting.

III. PROJECT PLAN

The project was developed with the Agile (SCRUM) development process. The sprints were in two-week segments with a planning meeting at the beginning of the sprint with review and retrospective meetings conducted at the end of the sprints.

SOFTWARE RESOURCES

The following software was used in the creation of the application is listed here along with its primary purpose in development:

- Node.js - Server Operations
- Express - Request and Response Routing
- Mongoose - Database Management
- Body-Parser - Parsing Body of POST Requests
- Morgan – Debug Messages to Console
- Postman - API Testing
- mLab – Remote Database Hosting
- Android Studio – Mobile Application Development

SPRINT PLANNING

Sprint 1

Attendees: Alejandro Thornton, Mikaila Daniel, Michael Quiros

Start time: 8/28/17, 3:00pm

End time: 8/28/17, 5:00pm

After discussion, the velocity of the team was estimated to be 60 hours.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

- #666: Server Setup
- #667: Simplify HTML
- #668: MEAN Stack Familiarization

The team members indicated their willingness to work on the following user stories.

- Alejandro Thornton
 - #666: Server Setup
- Mikaila Daniel
 - #667: Simplify HTML
- Michael Quiros
 - #668: MEAN Stack Familiarization

Sprint 2

Attendees: Alejandro Thornton, Mikaila Daniel, Michael Quiros

Start time: 9/23/17, 5:30pm

End time: 9/23/17, 7:00pm

After discussion, the velocity of the team was estimated to be 90 hours.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

- #670 – Create Profile Model
- #671 – Create Profile API Routes
- #669 - Creation of Temporary Server Testing Space
- #675 - Familiarize Ourselves with Android Studio
- #674 - Create Basic Mock Up

The team members indicated their willingness to work on the following user stories.

- Alejandro Thornton
 - #670 – Create Profile Models
 - #671 – Create Profile API Routes
- Mikaila Daniel
 - #675 - Familiarize Ourselves with Android Studio
 - #674 - Create Basic Mock Up
- Michael Quiros
 - #669 - Creation of Temporary Server Testing Space
 - #675 - Familiarize Ourselves with Android Studio

Sprint 3

Attendees: Alejandro Thornton, Mikaila Daniel, Michael Quiros

Start time: 10/3/17, 5:15pm

End time: 10/3/17, 6:15pm

After discussion, the velocity of the team was estimated to be ~86 hours.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

- #670 – Create Profile Model
- #671 – Create Profile API Routes
- #669 - Creation of Temporary Server Testing Space
- #674 - Create Basic App Mockup
- #676 - Main Menu Page
- #672 - Profile Page
- #678 – Testing Calls to the Database
- #679 - Testing Function Calls to the Application

The team members indicated their willingness to work on the following user stories.

- Alejandro Thornton
 - #670 – Create Profile Model
 - #671 – Create Profile API Routes
 - #678 – Testing Calls to the Database
- Mikaila Daniel
 - #674 - Create Basic App Mockup
 - #676 - Main Menu Page
- Michael Quiros
 - #669 - Creation of Temporary Server Testing Space
 - #672 - Profile Page
 - #679 - Testing Function Calls to the Application

Sprint 4

Attendees: Alejandro Thornton, Mikaila Daniel, Michael Quiros

Start time: 10/17/17, 4:30pm

End time: 10/17/17, 6:00pm

After discussion, the velocity of the team was estimated to be ~100 hours.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

- #678 - Testing Calls to the Database
- #681 - Create Meeting Model
- #683 - Create Meeting API Routes
- #679 - Test Function Calls to the Application
- #680 - Create New Announcements
- #682 - View Announcements
- #672 - Profile Page
- #684 - Create User Authentication Structure
- #688 - User Authentication
- #690 - Token Verification

The team members indicated their willingness to work on the following user stories.

- Alejandro Thornton
 - #678 - Testing Calls to the Database
 - #681 - Create Meeting Model
 - #683 - Create Meeting API Routes
 - #684 - Create User Authentication Structure
 - #688 - User Authentication
 - #690 - Token Verification
- Mikaila Daniel
 - #680 - Create New Announcements
 - #682 - View Announcements
- Michael Quiros
 - #672 - Profile Page
 - #679 - Test Function Calls to the Application

Sprint 5

Attendees: Alejandro Thornton, Mikaila Daniel, Michael Quiros

Start time: 10/30/17, 3:00pm

End time: 10/30/17, 3:30pm

After discussion, the velocity of the team was estimated to be 78 hours.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

- #685: Create Program Model
- #686: Create Program API Routes
- #687: Test Meeting Calls from Application
- #680: Create New Meeting
- #682: View Meetings
- #689: Create Local Database on Remote Server

The team members indicated their willingness to work on the following user stories.

- Alejandro Thornton
 - #685 - Create Program Model
 - #686 - Create Program API Routes
- Mikaila Daniel
 - #687: Test Meeting Calls from Application
 - #680: Create New Meeting
 - #682: View Meetings
- Michael Quiros
 - #689: Create Local Database on Remote Server

Sprint 6

Attendees: Alejandro Thornton, Mikaila Daniel, Michael Quiros

Start time: 11/15/17, 3:00pm

End time: 11/15/17, 3:30pm

After discussion, the velocity of the team was estimated to be 80 hours.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

- #693: Search by Name or University
- #692: Log onto Server from Application
- #695: Edit Profile
- #680: Create New Meeting
- #691: View Member Profile
- #694: Edit Meeting
- #689: Create Local Database on Remote Server

The team members indicated their willingness to work on the following user stories.

- Alejandro Thornton
 - None for this sprint.
- Mikaila Daniel
 - #692: Log onto Server from Application
 - #680: Create New Meeting
 - #691: View Member Profile
 - #695: Edit Profile
 - #694: Edit Meeting
 - #693: Search by Name or University
- Michael Quiros
 - #689: Create Local Database on Remote Server

IV. SYSTEM DESIGN

The following section discusses the design of the project. An overall view of the system is provided through discussion of architectural patterns, system and subsystem decomposition, a deployment diagram, and design patterns.

ARCHITECTURAL PATTERNS

There are two main components of the project: the front-end and the back-end. The back-end is implemented with a REST architectural pattern for the API using Express for routing requests and responses and MongoDB for database organization, managed by Mongoose. The front-end is the mobile application, implemented in the Java programming language using Android Studio. The mobile application is made up of views which are populated by information retrieved from the database. These two sides of the application are connected by making HTTP requests from the

SYSTEM AND SUBSYSTEM DECOMPOSITION

The front-end of the system is divided into pages which are for displaying and modifying the data in the database. Java methods built into the pages are used to make HTTP requests to the server and interpret the JSON documents returned by these requests. Figure S1 on the following page represents the flow of the application and the pages contained therein.

The back-end of the system is comprised of models and routes. Models are the templates with which the database creates and maintains the data for the four objects in the application: Users, Profiles, Programs, and Meetings. The routes are the functions used to manipulate this data. There are four types of requests which can be made to the server for each object: GET, POST, PUT, DELETE. GET requests retrieve data for one or multiple entities in the database. POST requests are for the creation of new entities in the database. PUT requests are for modifying existing entries in the database. DELETE requests are for removing entities in the database.

Some of these routes are gated by authorization, where only certain types of accounts may perform certain actions. DELETE requests can only be performed by Administrator type accounts, for example. There are three types of users: Members, Moderators, and Administrators. Authorization is progressive: Administrators can perform actions which Moderators can perform and Moderators can perform actions which Members can perform. Additionally, new accounts created must be approved by a Moderator or Administrator before they can be authorized to use the system. This structure protects routes from tampering and safeguards the integrity of the database. Figure S2 on the following page represents the design of the authentication and token verification scheme of the application.

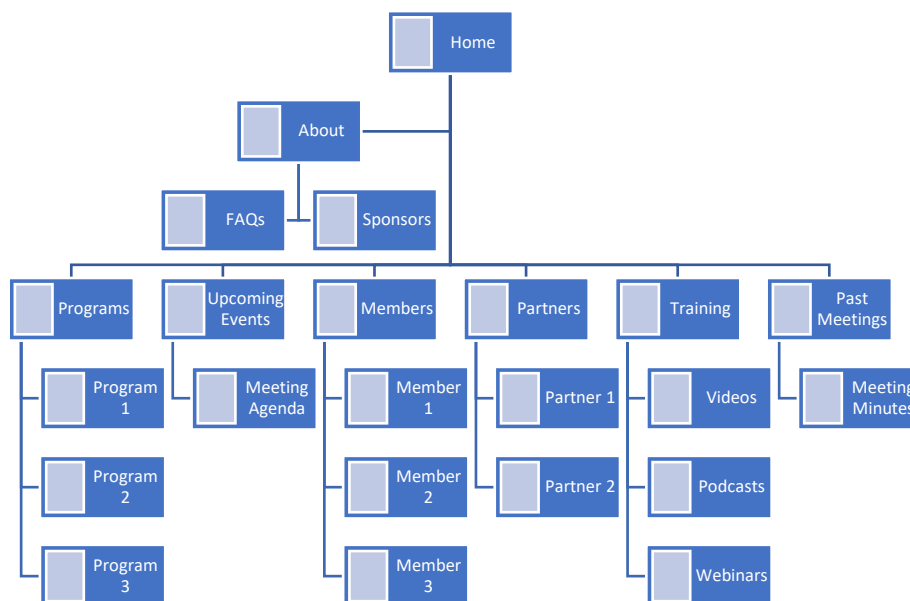


Figure S1: Application Page Map

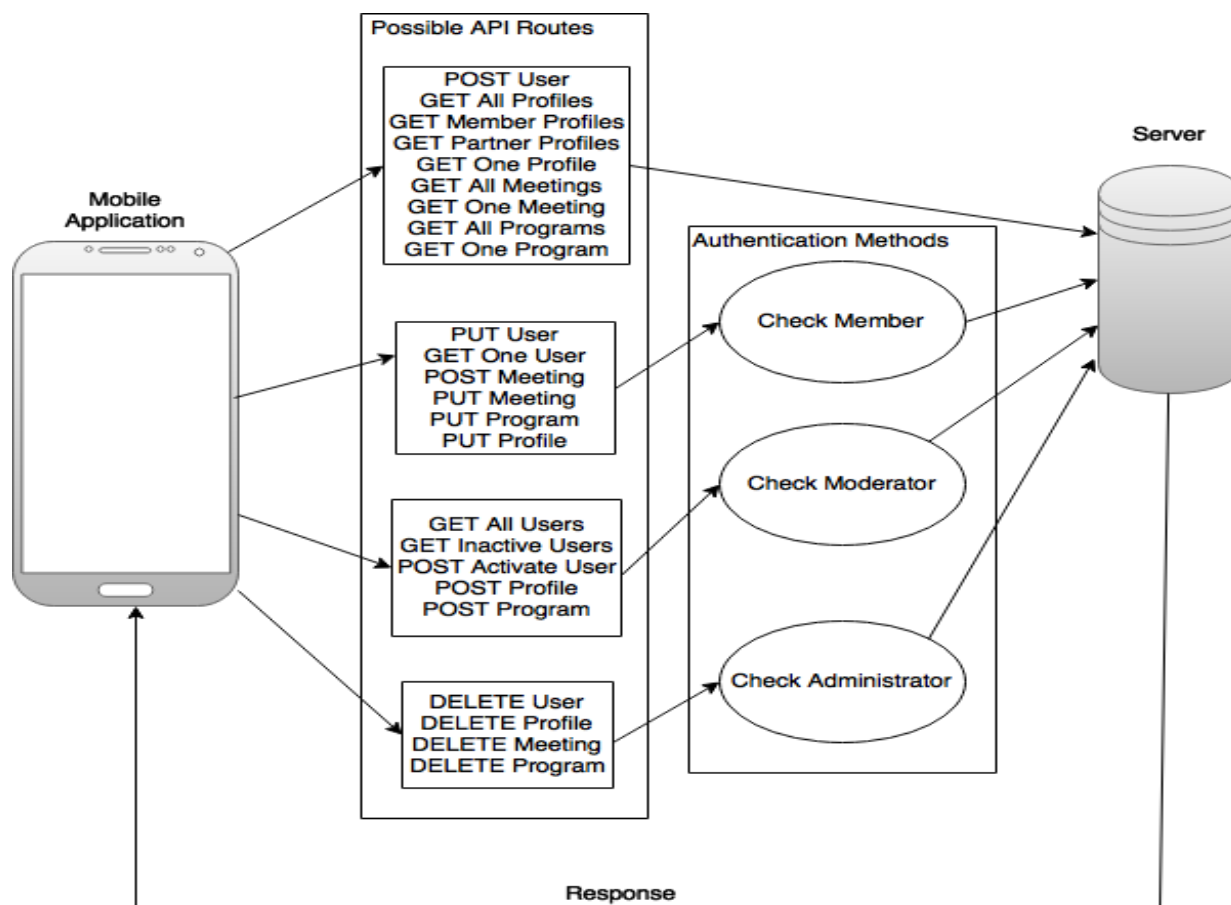


Figure S2: Authentication Design

DEPLOYMENT DIAGRAM

The following diagram represents how the subsystems relate to each other in the application.

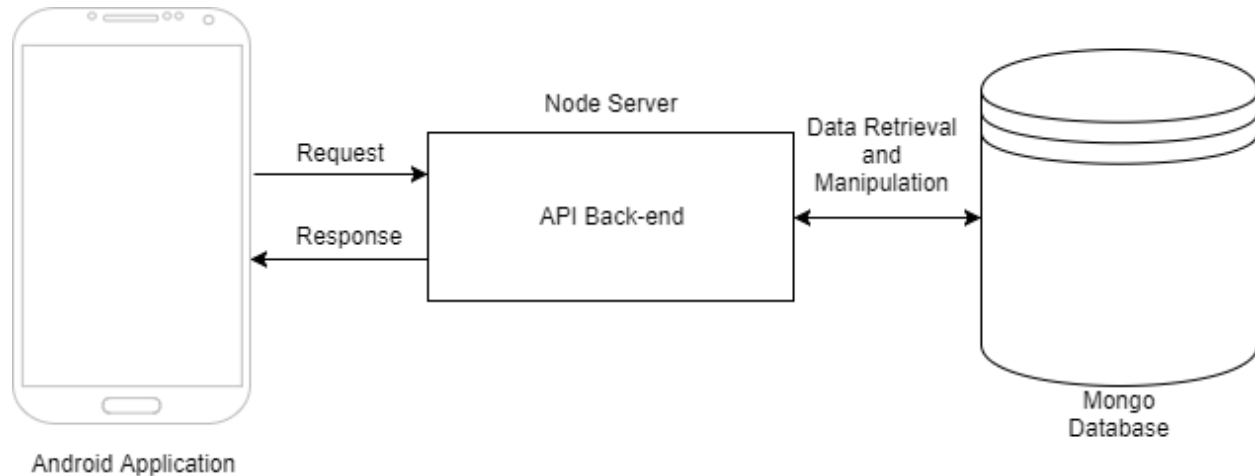


Figure S3: Deployment Diagram

DESIGN PATTERNS

The visual design of the mobile application was provided by the VIP (Vertically Integrated Projects) Marketing and Public Relations team which was cooperating with our team throughout the course of the project. The paradigm of the design was a clean, minimal interface with recognizable layouts and functions for ease-of-use. Users familiar with mobile applications will find that the application functions as they expect in terms of navigation and data input.

The system design was derived from the requirements laid out by the product owner in conjunction with the application design provided by the PR team. These elements informed the eventual API architecture and the functions used to retrieve and manipulate the database as discussed above.

V. SYSTEM VALIDATION

The following are the unit tests used to ensure the features of the system meet the requirements:

UNIT TESTS

Test Case ID	local_server_hosting
Description / Summary of Test	Given the system hosting the server file is connected to a local network, when the file is run with the Node Command Prompt, then the server should be locally hosted without error.
Pre-Condition	Server file is hosted on a system connected to a local network.
Expected Result	The server should run without error and a message should appear in the developer console.
Actual Result	Server file ran successfully, without error, and a message appeared in the developer console.
Status (Pass/Fail)	Pass

Test Case ID	create_profile_authorized
Description / Summary of Test	Given the user account is authorized, when the user makes a POST request through the Profile route, then the database should create a new Profile entry.
Pre-Condition	N/A
Expected Result	The database creates a new Profile entry and server returns a success message.
Actual Result	New Profile entry created and success message returned.
Status (Pass/Fail)	Pass

Test Case ID	create_profile_unauthorized
Description / Summary of Test	Given the user account is not authorized, when the user makes a POST request through the Profile route, then the server should return an error message.
Pre-Condition	N/A
Expected Result	Server returns an error message.
Actual Result	Error message returned.
Status (Pass/Fail)	Pass

Test Case ID	create_profile_authorized
Description / Summary of Test	Given the user account is authorized, when the user makes a POST request through the Profile route, then the database should create a new Profile entry.
Pre-Condition	N/A
Expected Result	The database creates a new Profile entry and server returns a success message.
Actual Result	New Profile entry created and success message returned.
Status (Pass/Fail)	Pass

Test Case ID	create_profile_unauthorized
Description / Summary of Test	Given the user account is not authorized, when the user makes a POST request through the Profile route, then the server should return an error message.
Pre-Condition	N/A
Expected Result	Server returns an error message.
Actual Result	Error message returned.
Status (Pass/Fail)	Pass

Test Case ID	get_profile_singular_matching_ID
Description / Summary of Test	Given there is a matching Profile ID in the database, when the user makes a GET request through the Profile route with an ID, then the server should return the Profile data.
Pre-Condition	User navigates to a Profile page.
Expected Result	Server returns Profile data.
Actual Result	Profile data returned.
Status (Pass/Fail)	Pass

Test Case ID	get_profile_singular_no_matching_ID
Description / Summary of Test	Given there is a no matching Profile ID in the database, when the user makes a GET request through the Profile route with an ID, then the server should return an error message.
Pre-Condition	N/A
Expected Result	Server returns an error message.
Actual Result	Error message returned.
Status (Pass/Fail)	Pass

Test Case ID	get_profile_all
Description / Summary of Test	Given there are Profile entries in the database, when the user makes a GET request through the Profile route, then the server should return data for each Profile entry.
Pre-Condition	User navigates to a Profile list page.
Expected Result	Server returns Profile data.
Actual Result	Profile data returned.
Status (Pass/Fail)	Pass

Test Case ID	modify_profile_matching_ID
Description / Summary of Test	Given the user is authorized to modify the given Profile, when the user makes a PUT request through the Profile route with an ID, then the server should modify the data of the matching entry and return a success message.
Pre-Condition	N/A
Expected Result	Profile data modified in database and server returned success message.
Actual Result	Profile data modified and success message returned.
Status (Pass/Fail)	Pass

Test Case ID	modify_profile_no_matching_ID
Description / Summary of Test	Given there is a no matching Profile ID in the database, when the user makes a PUT request through the Profile route with an ID, then the server should return an error message.
Pre-Condition	N/A
Expected Result	Server returns an error message.
Actual Result	Error message returned.
Status (Pass/Fail)	Pass

Test Case ID	delete_profile_matching_ID
Description / Summary of Test	Given the user is authorized to delete Profiles, when the user makes a DELETE request through the Profile route with an ID, then the server should return a success message.
Pre-Condition	N/A
Expected Result	Profile is deleted from the database and server returns success message.
Actual Result	Profile deleted and success message returned.
Status (Pass/Fail)	Pass

Test Case ID	delete_profile_no_matching_ID
Description / Summary of Test	Given there is a no matching Profile ID in the database, when the user makes a DELETE request through the Profile route with an ID, then the server should return an error message.
Pre-Condition	N/A
Expected Result	Server returns an error message.
Actual Result	Error message returned.
Status (Pass/Fail)	Pass

Test Case ID	get_profile_singular_matching_ID
Description / Summary of Test	Given there is a matching Profile ID in the database, when the user makes a GET request through the Profile route with an ID, then the server should return the Profile data.
Pre-Condition	User navigates to a Profile page.
Expected Result	Server returns Profile data.
Actual Result	Profile data returned.
Status (Pass/Fail)	Pass

Test Case ID	get_profile_singular_no_matching_ID
Description / Summary of Test	Given there is a no matching Profile ID in the database, when the user makes a GET request through the Profile route with an ID, then the server should return an error message.
Pre-Condition	N/A
Expected Result	Server returns an error message.
Actual Result	Error message returned.
Status (Pass/Fail)	Pass

Test Case ID	get_profile_all
Description / Summary of Test	Given there are Profile entries in the database, when the user makes a GET request through the Profile route, then the server should return data for each Profile entry.
Pre-Condition	User navigates to a Profile list page.
Expected Result	Server returns Profile data.
Actual Result	Profile data returned.
Status (Pass/Fail)	Pass

Test Case ID	get_meeting_singular_matching_ID
Description / Summary of Test	Given there is a matching Meeting ID in the database, when the user makes a GET request through the Meeting route with an ID, then the server should return the Meeting data.
Pre-Condition	User navigates to a Meeting page.
Expected Result	Server returns Meeting data.
Actual Result	Meeting data returned.
Status (Pass/Fail)	Pass

Test Case ID	get_meeting_singular_no_matching_ID
Description / Summary of Test	Given there is a no matching Meeting ID in the database, when the user makes a GET request through the Meeting route with an ID, then the server should return an error message.
Pre-Condition	N/A
Expected Result	Server returns an error message.
Actual Result	Error message returned.
Status (Pass/Fail)	Pass

Test Case ID	get_meeting_all
Description / Summary of Test	Given there are Meeting entries in the database, when the user makes a GET request through the Meeting route, then the server should return data for each Meeting entry.
Pre-Condition	User navigates to a Meeting list page.
Expected Result	Server returns Meeting data.
Actual Result	Meeting data returned.
Status (Pass/Fail)	Pass

Test Case ID	create_meeting_authorized
Description / Summary of Test	Given the user account is authorized, when the user makes a POST request through the Meeting route, then the database should create a new Meeting entry.
Pre-Condition	N/A
Expected Result	The database creates a new Meeting entry and server returns a success message.
Actual Result	New Meeting entry created and success message returned.
Status (Pass/Fail)	Pass

Test Case ID	create_meeting_unauthorized
Description / Summary of Test	Given the user account is not authorized, when the user makes a POST request through the Meeting route, then the server should return an error message.
Pre-Condition	N/A
Expected Result	Server returns an error message.
Actual Result	Error message returned.
Status (Pass/Fail)	Pass

Test Case ID	create_meeting_authorized
Description / Summary of Test	Given the user account is authorized, when the user makes a POST request through the Meeting route, then the database should create a new Meeting entry.
Pre-Condition	N/A
Expected Result	The database creates a new Meeting entry and server returns a success message.
Actual Result	New Meeting entry created and success message returned.
Status (Pass/Fail)	Pass

Test Case ID	create_meeting_unauthorized
Description / Summary of Test	Given the user account is not authorized, when the user makes a POST request through the Meeting route, then the server should return an error message.
Pre-Condition	N/A
Expected Result	Server returns an error message.
Actual Result	Error message returned.
Status (Pass/Fail)	Pass

Test Case ID	get_meeting_singular_matching_ID
Description / Summary of Test	Given there is a matching Meeting ID in the database, when the user makes a GET request through the Meeting route with an ID, then the server should return the Meeting data.
Pre-Condition	User navigates to a Meeting page.
Expected Result	Server returns Meeting data.
Actual Result	Meeting data returned.
Status (Pass/Fail)	Pass

Test Case ID	get_meeting_singular_no_matching_ID
Description / Summary of Test	Given there is a no matching Meeting ID in the database, when the user makes a GET request through the Meeting route with an ID, then the server should return an error message.
Pre-Condition	N/A
Expected Result	Server returns an error message.
Actual Result	Error message returned.
Status (Pass/Fail)	Pass

Test Case ID	get_meeting_all
Description / Summary of Test	Given there are Meeting entries in the database, when the user makes a GET request through the Meeting route, then the server should return data for each Meeting entry.
Pre-Condition	User navigates to a Meeting list page.
Expected Result	Server returns Meeting data.
Actual Result	Meeting data returned.
Status (Pass/Fail)	Pass

Test Case ID	modify_meeting_matching_ID
Description / Summary of Test	Given the user is authorized to modify the given Meeting, when the user makes a PUT request through the Meeting route with an ID, then the server should modify the data of the matching entry and return a success message.
Pre-Condition	N/A
Expected Result	Meeting data modified in database and server returned success message.
Actual Result	Meeting data modified and success message returned.
Status (Pass/Fail)	Pass

Test Case ID	modify_meeting_no_matching_ID
Description / Summary of Test	Given there is a no matching Meeting ID in the database, when the user makes a PUT request through the Meeting route with an ID, then the server should return an error message.
Pre-Condition	N/A
Expected Result	Server returns an error message.
Actual Result	Error message returned.
Status (Pass/Fail)	Pass

Test Case ID	delete_meeting_matching_ID
Description / Summary of Test	Given the user is authorized to delete Meetings, when the user makes a DELETE request through the Meeting route with an ID, then the server should return a success message.
Pre-Condition	N/A
Expected Result	Meeting is deleted from the database and server returns success message.
Actual Result	Meeting deleted and success message returned.
Status (Pass/Fail)	Pass

Test Case ID	delete_meeting_no_matching_ID
Description / Summary of Test	Given there is a no matching Meeting ID in the database, when the user makes a DELETE request through the Meeting route with an ID, then the server should return an error message.
Pre-Condition	N/A
Expected Result	Server returns an error message.
Actual Result	Error message returned.
Status (Pass/Fail)	Pass

Test Case ID	create_user
Description / Summary of Test	Given the user is not logged in, when the user makes a POST request through the User route, then the database should create a new User entry.
Pre-Condition	N/A
Expected Result	The database creates a new User entry and server returns a success message.
Actual Result	New User entry created and success message returned.
Status (Pass/Fail)	Pass

Test Case ID	get_user_singular_matching_ID
Description / Summary of Test	Given there is a matching User ID in the database and the user is authorized, when the user makes a GET request through the User route with an ID, then the server should return the User data.
Pre-Condition	User navigates to a User page.
Expected Result	Server returns User data.
Actual Result	User data returned.
Status (Pass/Fail)	Pass

Test Case ID	get_user_singular_no_matching_ID
Description / Summary of Test	Given there is a no matching User ID in the database, when the user makes a GET request through the User route with an ID, then the server should return an error message.
Pre-Condition	N/A
Expected Result	Server returns an error message.
Actual Result	Error message returned.
Status (Pass/Fail)	Pass

Test Case ID	get_user_all
Description / Summary of Test	Given the user account is authorized, when the user makes a GET request through the User route, then the server should return data for each User entry.
Pre-Condition	User navigates to a User list page.
Expected Result	Server returns User data.
Actual Result	User data returned.
Status (Pass/Fail)	Pass

Test Case ID	modify_user_matching_ID
Description / Summary of Test	Given the user is authorized to modify the given User, when the user makes a PUT request through the User route with an ID, then the server should modify the data of the matching entry and return a success message.
Pre-Condition	N/A
Expected Result	User data modified in database and server returned success message.
Actual Result	User data modified and success message returned.
Status (Pass/Fail)	Pass

Test Case ID	modify_user_no_matching_ID
Description / Summary of Test	Given there is a no matching User ID in the database, when the user makes a PUT request through the User route with an ID, then the server should return an error message.
Pre-Condition	N/A
Expected Result	Server returns an error message.
Actual Result	Error message returned.
Status (Pass/Fail)	Pass

Test Case ID	delete_user_matching_ID
Description / Summary of Test	Given the user is authorized to delete Users, when the user makes a DELETE request through the User route with an ID, then the server should return a success message.
Pre-Condition	N/A
Expected Result	User is deleted from the database and server returns success message.
Actual Result	User deleted and success message returned.
Status (Pass/Fail)	Pass

Test Case ID	delete_user_no_matching_ID
Description / Summary of Test	Given there is a no matching User ID in the database, when the user makes a DELETE request through the User route with an ID, then the server should return an error message.
Pre-Condition	N/A
Expected Result	Server returns an error message.
Actual Result	Error message returned.
Status (Pass/Fail)	Pass

Test Case ID	authentication_valid_username_password
Description / Summary of Test	Given the user account is present and activated, when the user provides a valid username and password combination, then the server should return a token along with a success message.
Pre-Condition	User goes to the login page in the application.
Expected Result	The server should return a token along with a success message.
Actual Result	Success message and token are provided to user.
Status (Pass/Fail)	Pass

Test Case ID	authentication_invalid_username
Description / Summary of Test	Given there are user accounts present in the database, when the user provides a username which does not match any in the database, then the server should return an error message.
Pre-Condition	User goes to the login page in the application.
Expected Result	The server should return an error message.
Actual Result	Error message provided to user.
Status (Pass/Fail)	Pass

Test Case ID	authentication_inactive_account
Description / Summary of Test	Given the user account is present and inactive, when the user provides a valid username and password combination, then the server should return an error message.
Pre-Condition	User goes to the login page in the application.
Expected Result	The server should return an error message.
Actual Result	Error message provided to user.
Status (Pass/Fail)	Pass

Test Case ID	authentication_invalid_password
Description / Summary of Test	Given the user account is present and inactive, when the user provides a valid username but an invalid password, then the server should return an error message.
Pre-Condition	User goes to the login page in the application.
Expected Result	The server should return an error message.
Actual Result	Error message provided to user.
Status (Pass/Fail)	Pass

Test Case ID	create_program_authorized
Description / Summary of Test	Given the user account is authorized, when the user makes a POST request through the Program route, then the database should create a new Program entry.
Pre-Condition	N/A
Expected Result	The database creates a new Program entry and server returns a success message.
Actual Result	New Program entry created and success message returned.
Status (Pass/Fail)	Pass

Test Case ID	create_program_unauthorized
Description / Summary of Test	Given the user account is not authorized, when the user makes a POST request through the Program route, then the server should return an error message.
Pre-Condition	N/A
Expected Result	Server returns an error message.
Actual Result	Error message returned.
Status (Pass/Fail)	Pass

Test Case ID	get_program_singular_matching_ID
Description / Summary of Test	Given there is a matching Program ID in the database, when the user makes a GET request through the Program route with an ID, then the server should return the Program data.
Pre-Condition	User navigates to a Program page.
Expected Result	Server returns Program data.
Actual Result	Program data returned.
Status (Pass/Fail)	Pass

Test Case ID	get_program_singular_no_matching_ID
Description / Summary of Test	Given there is a no matching Program ID in the database, when the user makes a GET request through the Program route with an ID, then the server should return an error message.
Pre-Condition	N/A
Expected Result	Server returns an error message.
Actual Result	Error message returned.
Status (Pass/Fail)	Pass

Test Case ID	get_program_all
Description / Summary of Test	Given there are Program entries in the database, when the user makes a GET request through the Program route, then the server should return data for each Program entry.
Pre-Condition	User navigates to a Program list page.
Expected Result	Server returns Program data.
Actual Result	Program data returned.
Status (Pass/Fail)	Pass

Test Case ID	modify_program_matching_ID
Description / Summary of Test	Given the user is authorized to modify the given Program, when the user makes a PUT request through the Program route with an ID, then the server should modify the data of the matching entry and return a success message.
Pre-Condition	N/A
Expected Result	Program data modified in database and server returned success message.
Actual Result	Program data modified and success message returned.
Status (Pass/Fail)	Pass

Test Case ID	modify_program_no_matching_ID
Description / Summary of Test	Given there is a no matching Program ID in the database, when the user makes a PUT request through the Program route with an ID, then the server should return an error message.
Pre-Condition	N/A
Expected Result	Server returns an error message.
Actual Result	Error message returned.
Status (Pass/Fail)	Pass

Test Case ID	delete_program_matching_ID
Description / Summary of Test	Given the user is authorized to delete Programs, when the user makes a DELETE request through the Program route with an ID, then the server should return a success message.
Pre-Condition	N/A
Expected Result	Program is deleted from the database and server returns success message.
Actual Result	Program deleted and success message returned.
Status (Pass/Fail)	Pass

Test Case ID	delete_program_no_matching_ID
Description / Summary of Test	Given there is a no matching Program ID in the database, when the user makes a DELETE request through the Program route with an ID, then the server should return an error message.
Pre-Condition	N/A
Expected Result	Server returns an error message.
Actual Result	Error message returned.
Status (Pass/Fail)	Pass

Test Case ID	token_verification_member_route
Description / Summary of Test	Given the user is a Member or above, when the user attempts to access a route which requires an account which is Member level, then the server should return the requested action.
Pre-Condition	User provides a token to the server.
Expected Result	The server should return the requested action.
Actual Result	Requested action is provided to the user.
Status (Pass/Fail)	Pass

Test Case ID	token_verification_moderator_route
Description / Summary of Test	Given the user is a Moderator or above, when the user attempts to access a route which requires an account which is Moderator level or below, then the server should return the requested action.
Pre-Condition	User provides a token to the server.
Expected Result	The server should return the requested action.
Actual Result	Requested action is provided to the user.
Status (Pass/Fail)	Pass

Test Case ID	token_verification_administrator_route
Description / Summary of Test	Given the user is an Administrator, when the user attempts to access a route which requires an account which is Administrator level or below, then the server should return the requested action.
Pre-Condition	User provides a token to the server.
Expected Result	The server should return the requested action.
Actual Result	Requested action is provided to the user.
Status (Pass/Fail)	Pass

Test Case ID	no_token_protected_route
Description / Summary of Test	Given the user is not logged in, when the user attempts to access a route which requires an account, then the server should return an error message.
Pre-Condition	N/A
Expected Result	The server should return an error message.
Actual Result	Requested action is provided to the user.
Status (Pass/Fail)	Pass

Test Case ID	invalid_token_moderator_route
Description / Summary of Test	Given the user is a Member, when the user attempts to access a route which requires an account which is Moderator level or above, then the server should return an error message.
Pre-Condition	User provides a token to the server.
Expected Result	The server should return an error message.
Actual Result	Requested action is provided to the user.
Status (Pass/Fail)	Pass

Test Case ID	invalid_token_administrator_route
Description / Summary of Test	Given the user is a Member or Moderator, when the user attempts to access a route which requires an Administrator account, then the server should return an error message.
Pre-Condition	User provides a token to the server.
Expected Result	The server should return an error message.
Actual Result	Requested action is provided to the user.
Status (Pass/Fail)	Pass

Test Case ID	create_program_authorized
Description / Summary of Test	Given the user account is authorized, when the user makes a POST request through the Program route, then the database should create a new Program entry.
Pre-Condition	N/A
Expected Result	The database creates a new Program entry and server returns a success message.
Actual Result	New Program entry created and success message returned.
Status (Pass/Fail)	Pass

Test Case ID	create_program_unauthorized
Description / Summary of Test	Given the user account is not authorized, when the user makes a POST request through the Program route, then the server should return an error message.
Pre-Condition	N/A
Expected Result	Server returns an error message.
Actual Result	Error message returned.
Status (Pass/Fail)	Pass

VI. GLOSSARY

API – Application Programming Interface; the set of functions called by the front-end (user side) of an application to the back-end (server/database side).

JSON – JavaScript Object Notation; a document format used by certain database systems and networked applications.

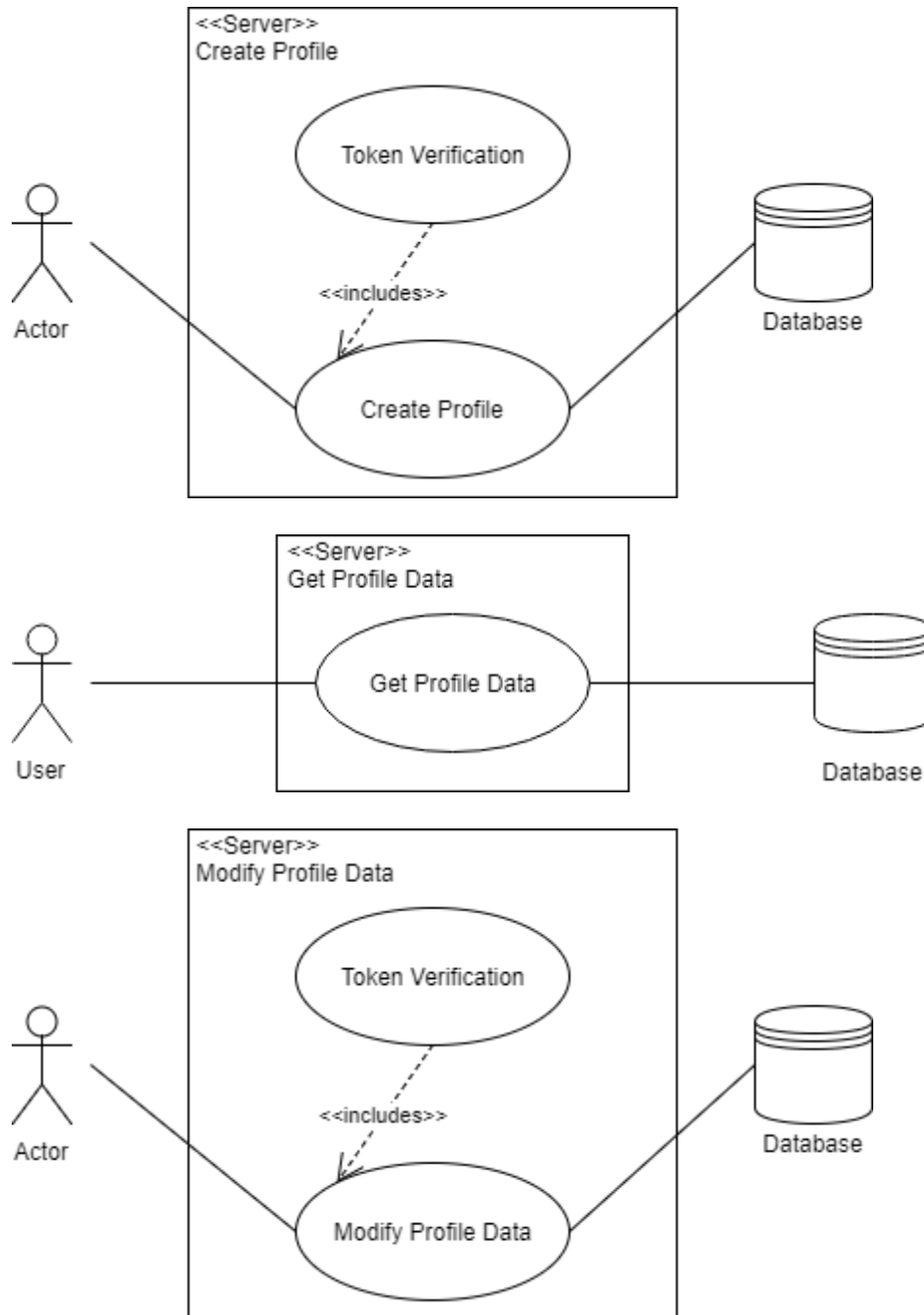
HTTP – Hypertext Transfer Protocol; the basis of network communications, used to send data across networks.

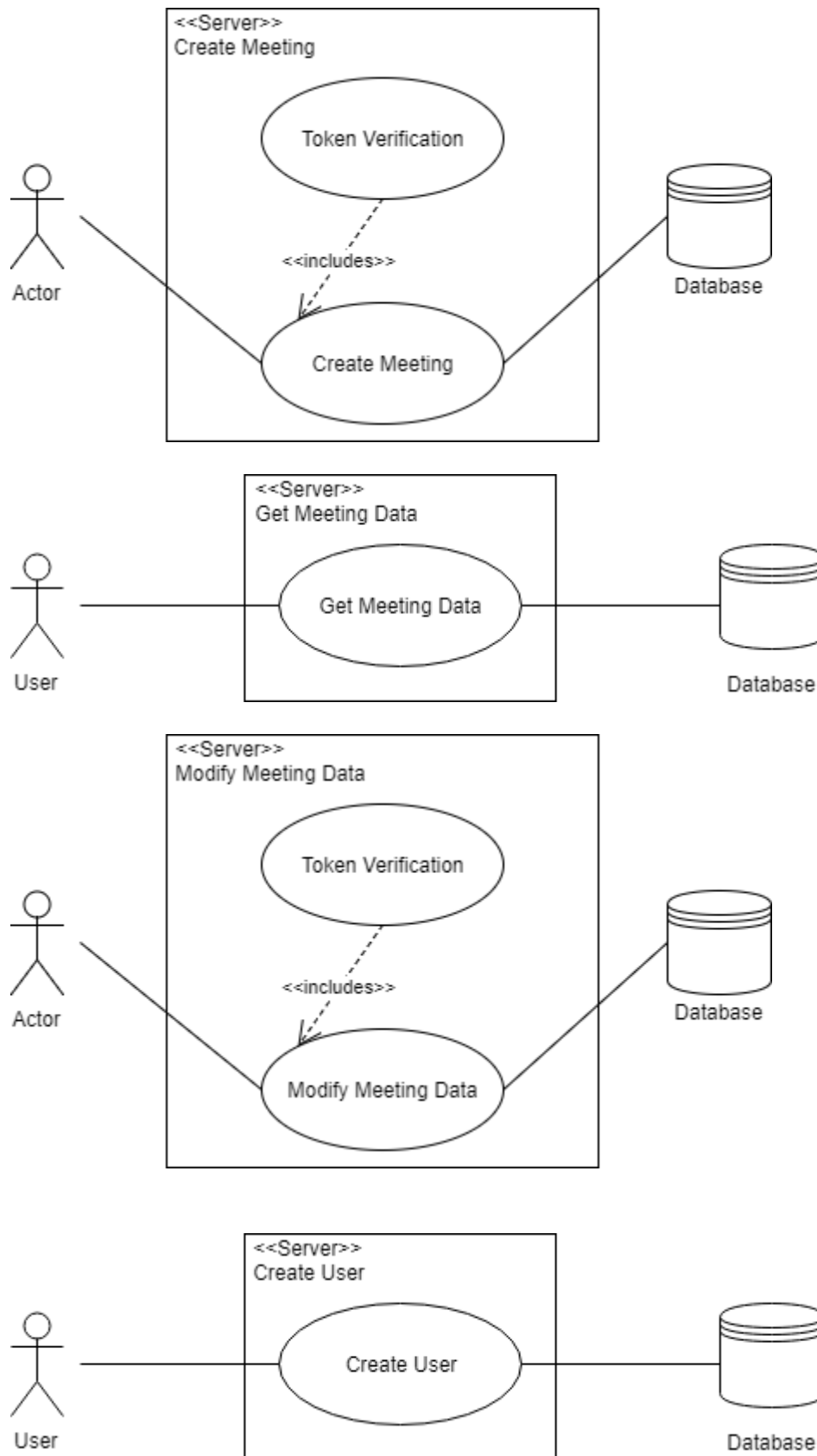
REST – Representational State Transfer; an architectural style used in networked applications.

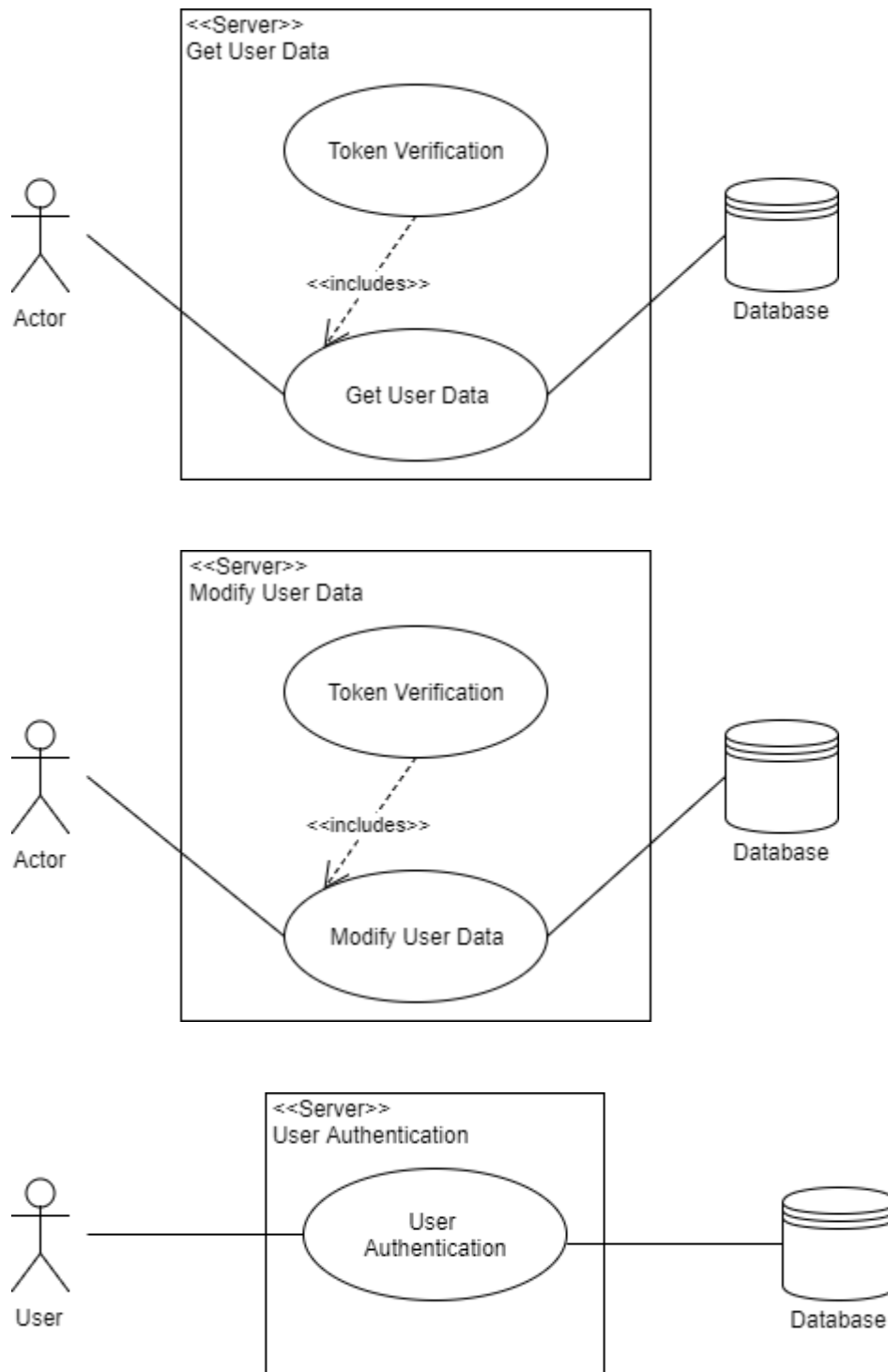
VII. APPENDIX

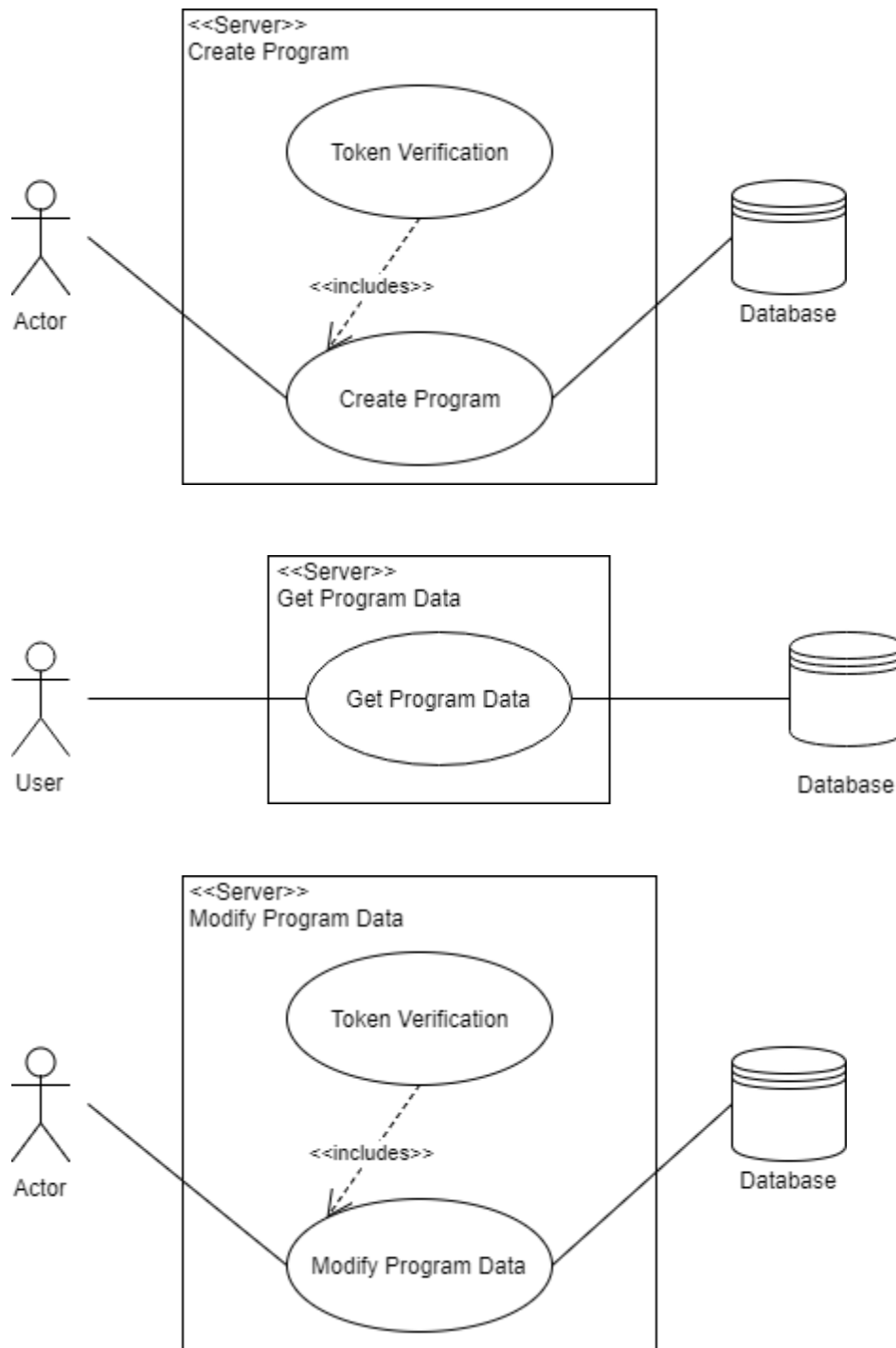
APPENDIX A: UML DIAGRAMS

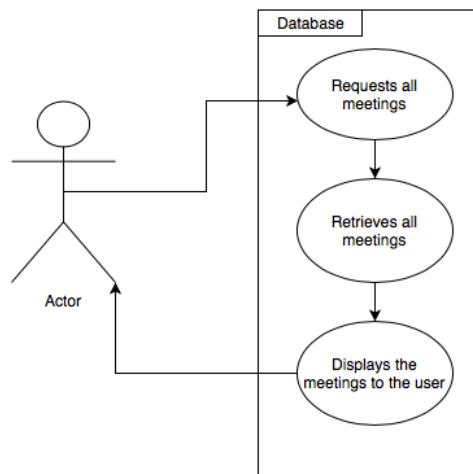
Use Case Diagrams





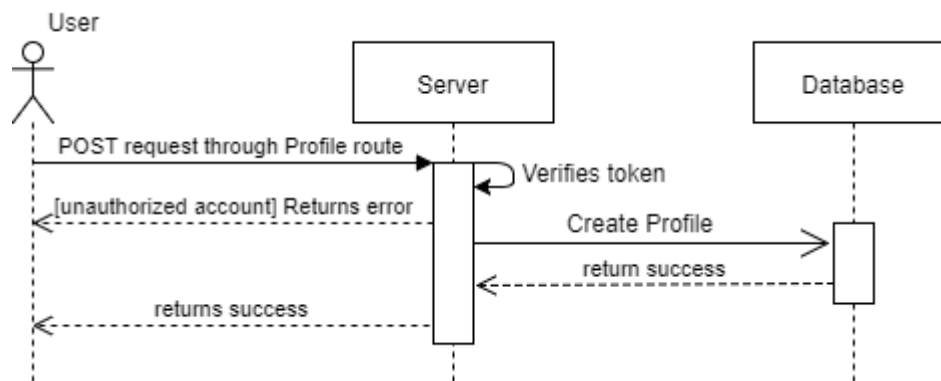




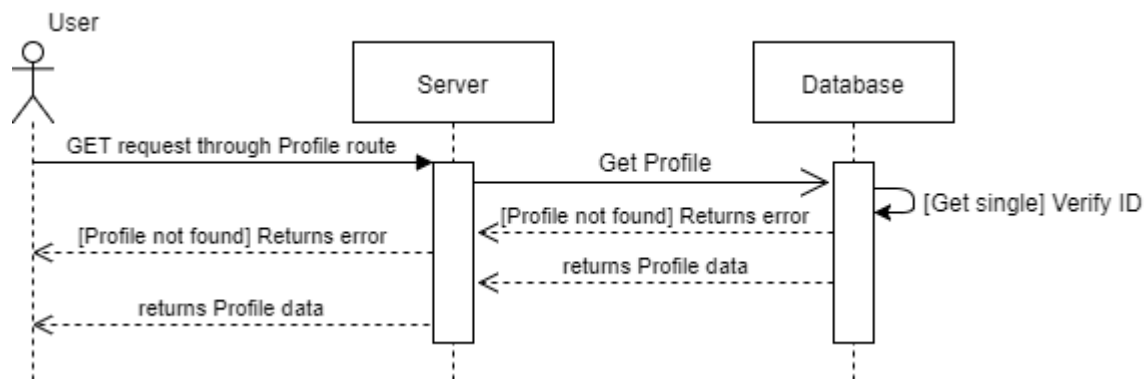


Sequence Diagrams

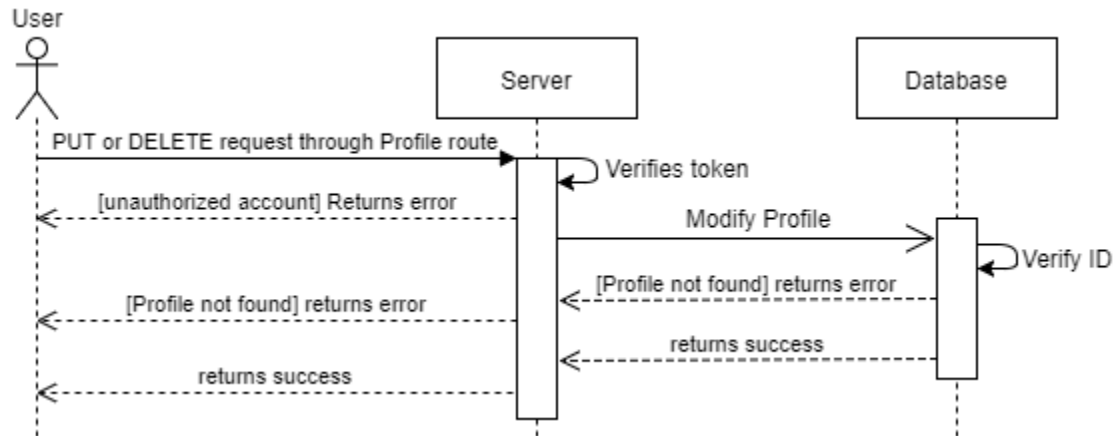
Create Profile



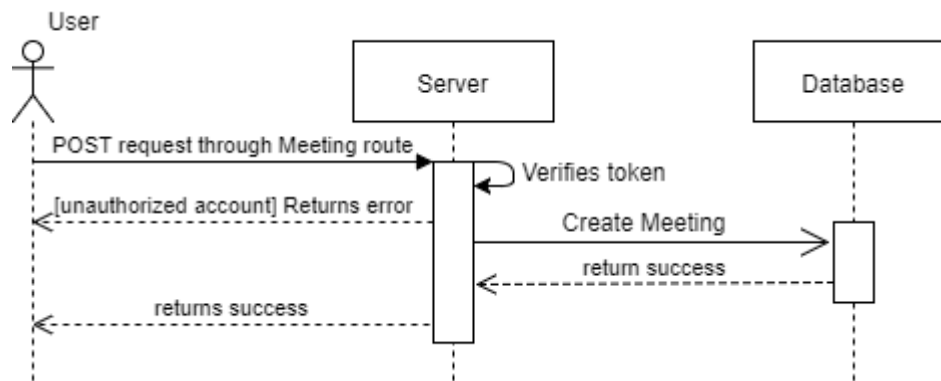
Get Profile Data



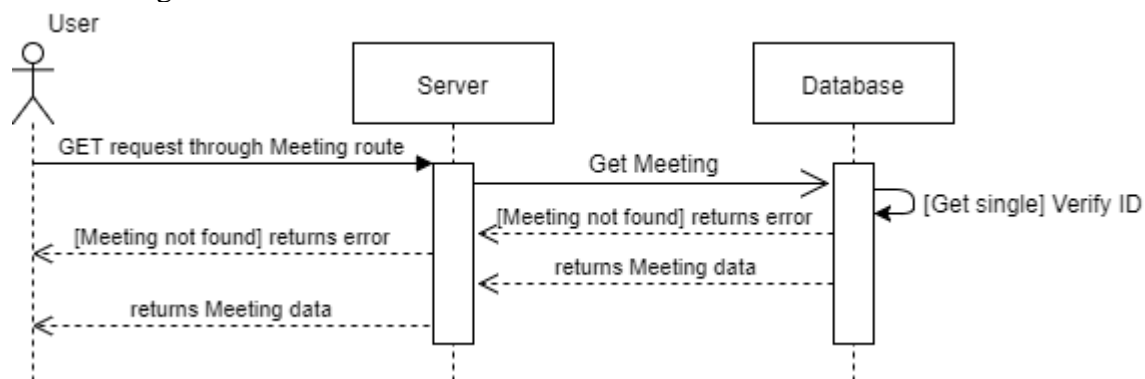
Modify Profile Data



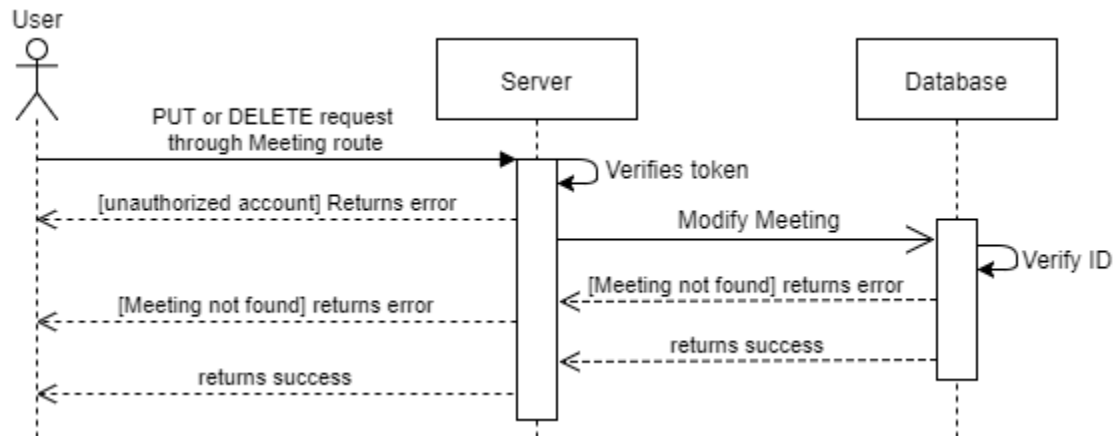
Create Meeting



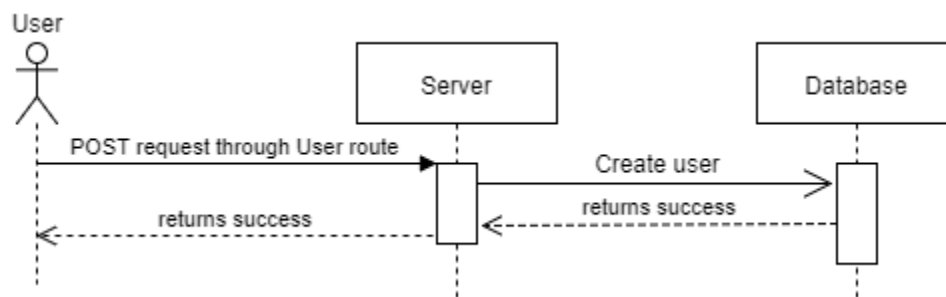
Get Meeting Data



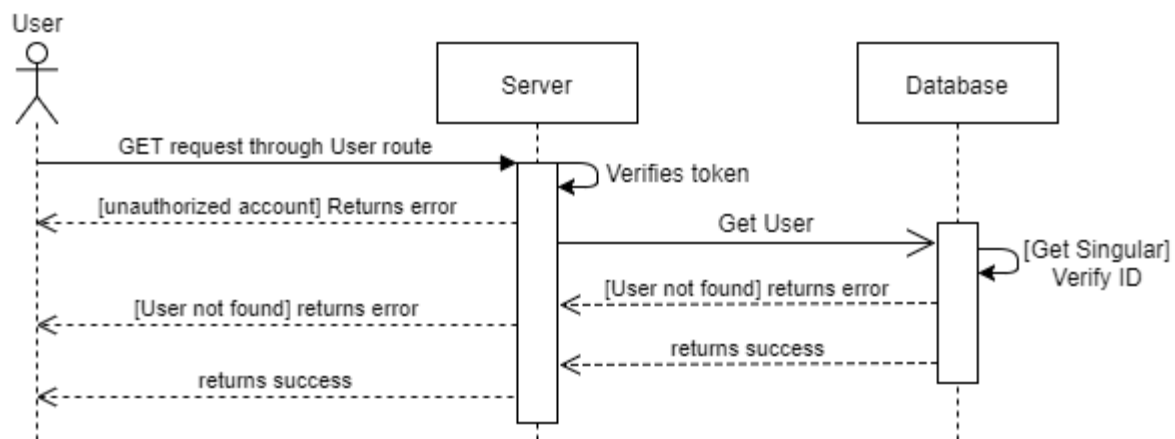
Modify Meeting Data



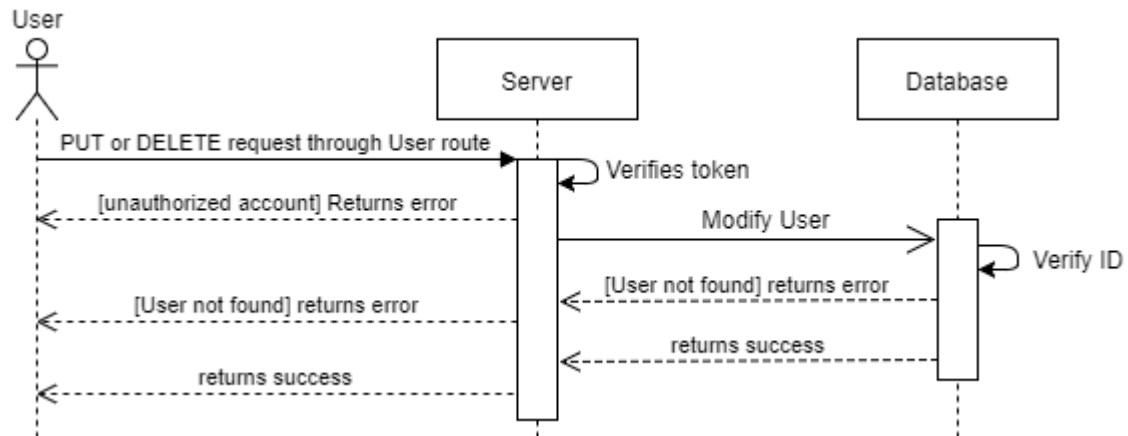
Create User



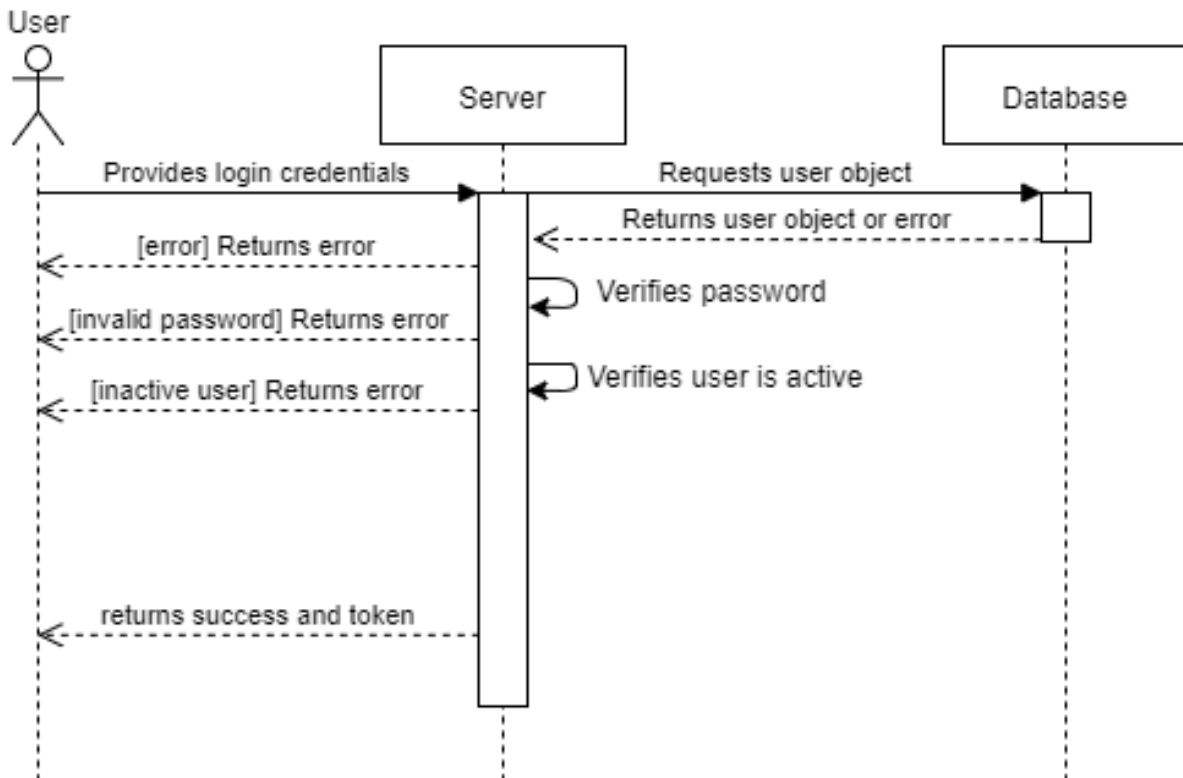
Get User Data



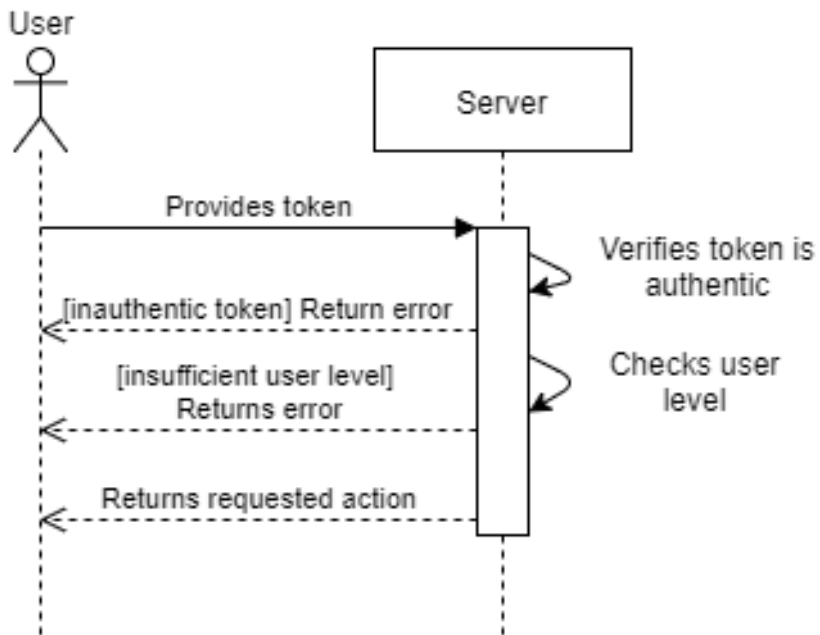
Modify User Data



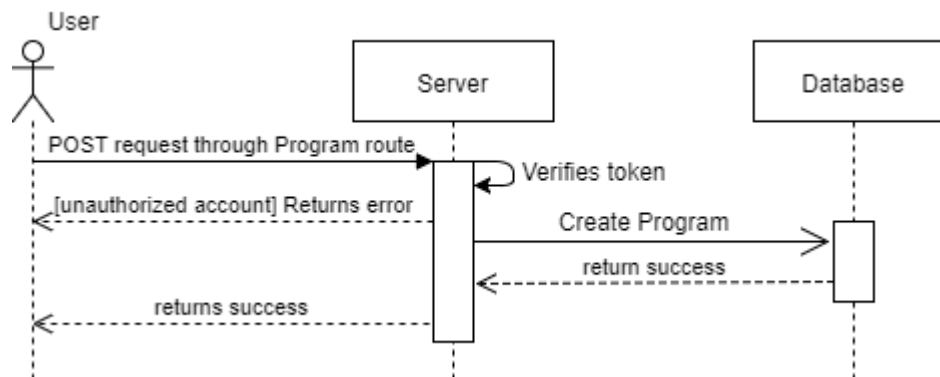
User Authentication



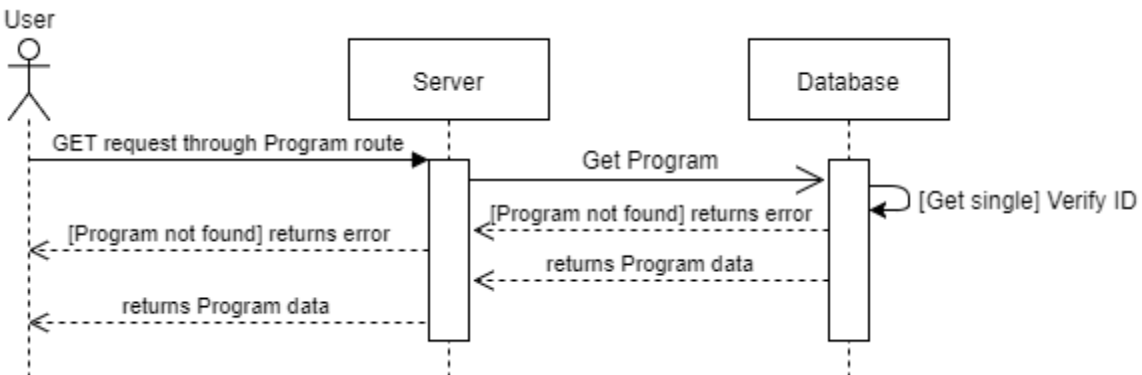
Token Verification



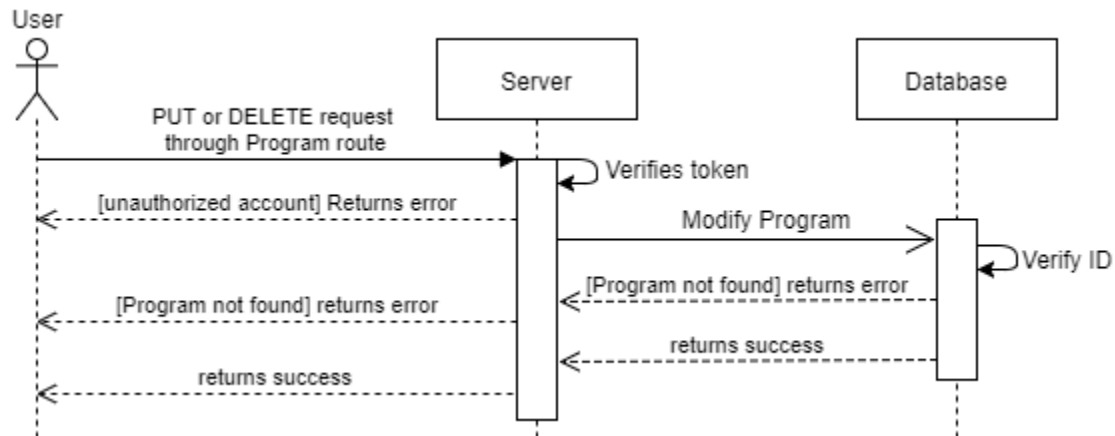
Create Program



Get Program Data

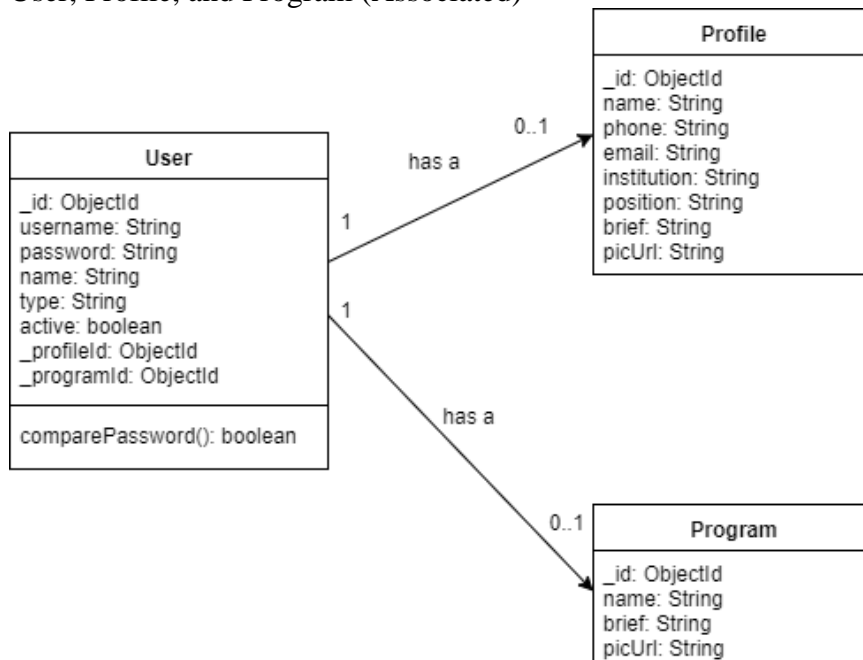


Modify Program Data

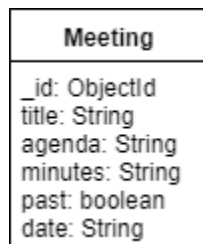


Class Diagrams

User, Profile, and Program (Associated)

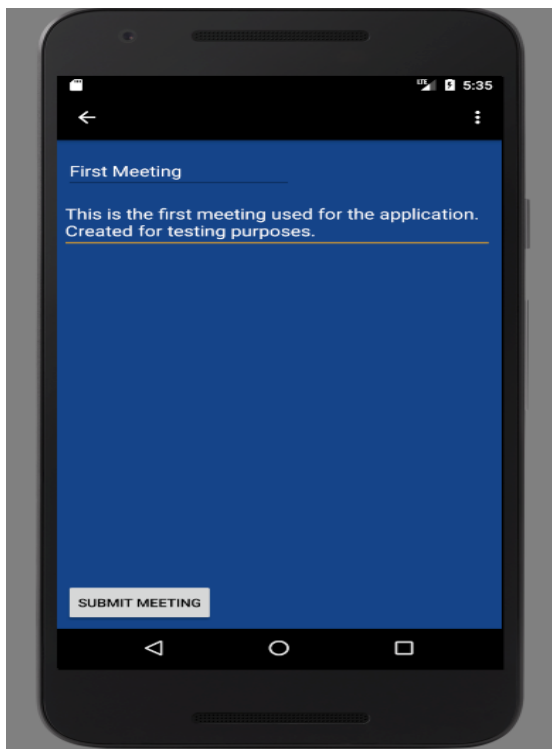
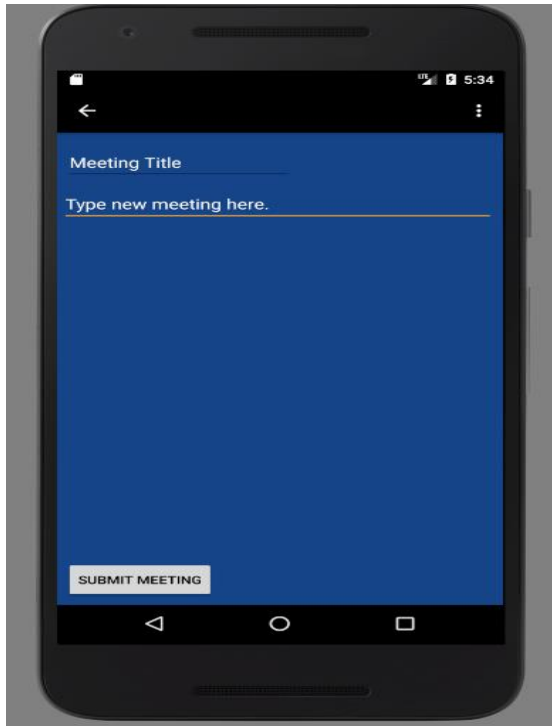


Meeting

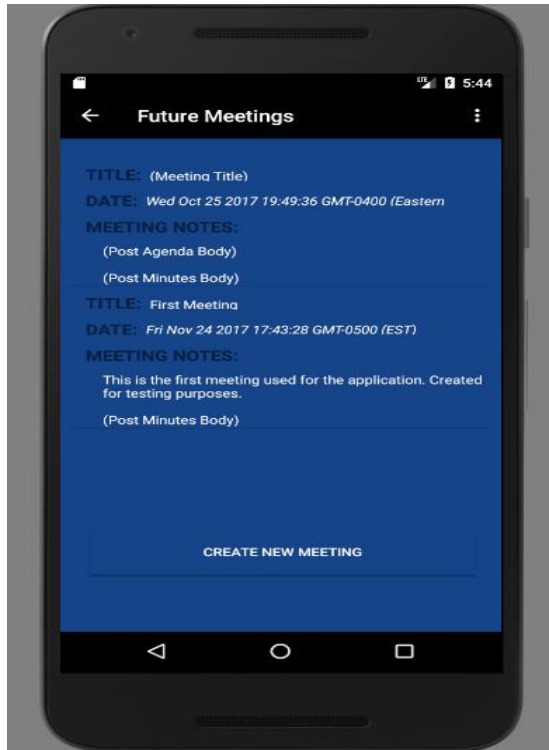


APPENDIX B: USER INTERFACE DESIGN

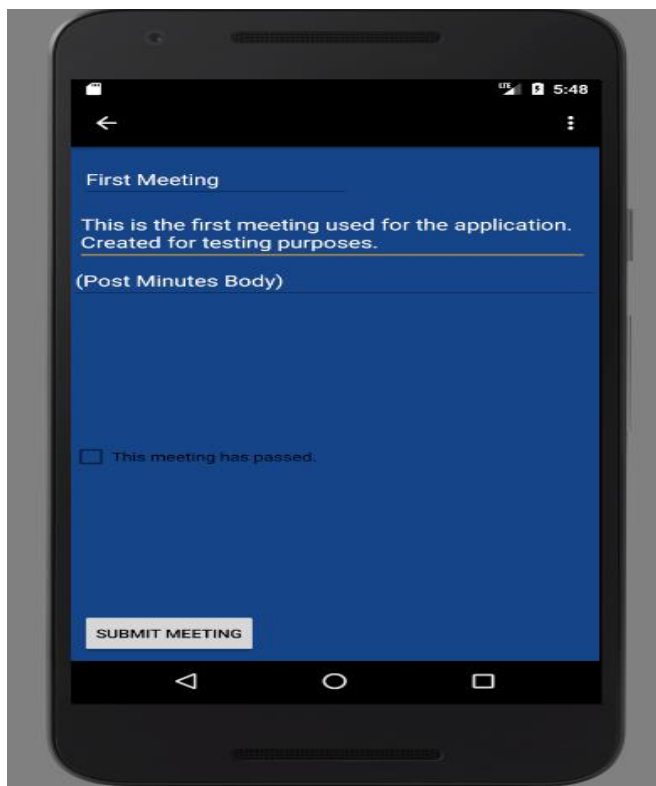
Create Meeting



Future Meeting List

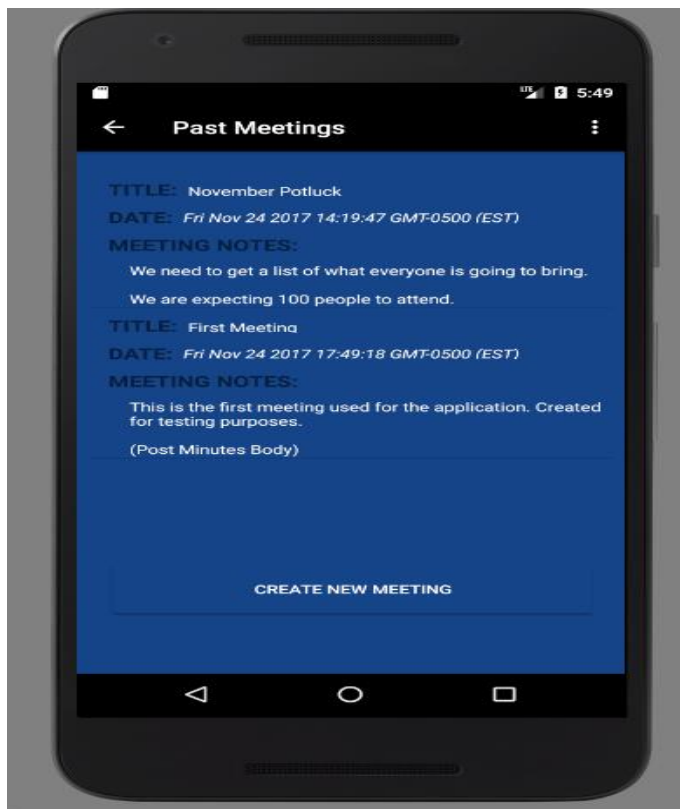


Editing Meeting





Past Meetings List



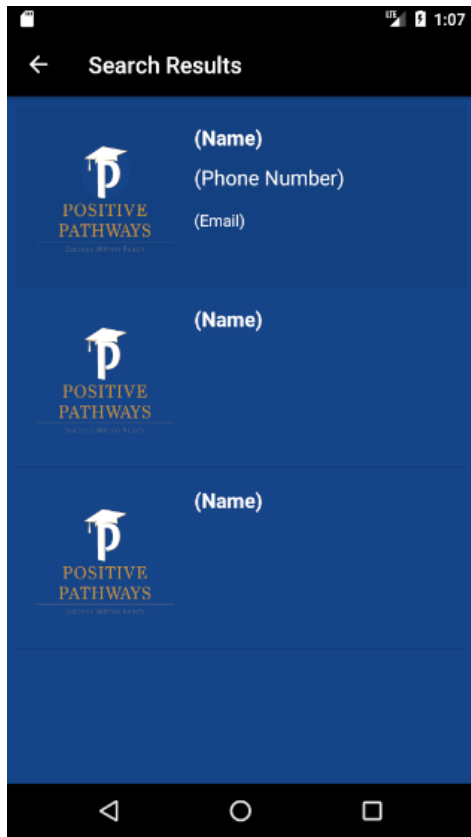
Home Page



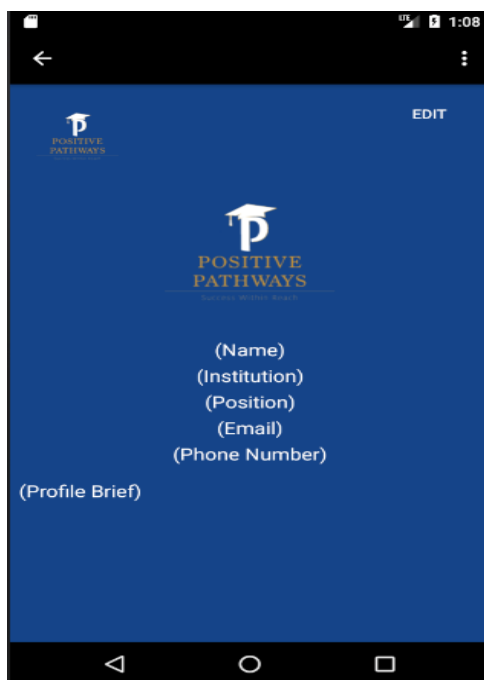
Login Page



Search Results Page



Profile Page



Editing Profile

← FINISH EDITING

Mikaila Daniel

(Institution)

(Position)

(Email)

(Phone Number)

(Profile Brief)

Daniels Daniel Danielle

q w e r t y u i o p


a s d f g h j k l

↑ z x c v b n m ↵

?123 , . ↵

Result of Edited Profile

← Search Results

 **Mikaila Daniel**
(Phone Number)
(Email)

APPENDIX C: SPRINT REVIEW REPORTS

Sprint 1 Review Meeting Minutes

Attendees: Alejandro Thornton, Mikaila Daniel, Michael Quiros

Start time: 9/18/17, 5:30pm

End time: 9/18/17, 6:00pm

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.

- User Story #666: Server Setup

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Spring Planning meeting.

- Not applicable to any of the user stories for this sprint.

Sprint 1 Retrospective Meeting Minutes

Attendees: Alejandro Thornton, Mikaila Daniel, Michael Quiros

Start time: 9/18/17, 6:00pm

End time: 9/18/17, 6:30pm

What went wrong?

- Did we do a good job estimating our team's velocity?
 - The time and resources available for this sprint were deeply impacted by Irma, which makes it difficult to assess the accuracy of the team velocity.
- Did we do a good job estimating the points (time required) for each user story?
 - User Story #666 was completed within the allotted points.
- Did each team member work as scheduled?
 - The team members worked as scheduled before the advent of Hurricane Irma.

What went right?

- User Story #666 - User Setup was completed

How to address the issues in the next sprint?

- How to improve the process?
 - Improved communication with product owner and non-major professor.
- How to improve the product?
 - Adding features once the goals of the product owner become known.

Sprint 2 Review Meeting Minutes

Attendees: Alejandro Thornton, Mikaila Daniel, Michael Quiros

Start time: 10/2/17, 11:30am

End time: 10/2/17, 12:00pm

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.

- User Story #674: Create Basic App Mock Up
- User Story #676: Create Main Menu Page

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Spring Planning meeting.

- User Story #670: Create Profile Model
- User Story #671: Create Profile API Routes
- User Story #669: Creation of Temporary Server Testing Space

Sprint 2 Retrospective Meeting Minutes

Attendees: Alejandro Thornton, Mikaila Daniel, Michael Quiros

Start time: 10/2/17, 12:00pm

End time: 10/2/17, 12:30pm

What went wrong?

- Did we do a good job estimating our team's velocity?
 - Less allotment of points for the user stories in this sprint which came after the meeting with the product owner.
- Did we do a good job estimating the points (time required) for each user story?
 - User Story #669: Creation of Temporary Server Testing Space -
 - User Story #670: Database Models - Slightly underestimated, one model is completed but one more is needed.
 - User Story #671: API Routes - Underestimated the point allotment for this user story.
 - User Story #674: Create Basic App Mockup - Overestimated the point allotment for this user story. Android Studio's tutorial for a simple "Hello World" application took less than four hours to complete.
- Did each team member work as scheduled?
 - Outside coursework and work duties impacted overall team velocity.

What went right?

- Continued familiarization with Android Studio. The model for the Liaison profiles was completed and the corresponding API routes are mostly complete.

How to address the issues in the next sprint?

- How to improve the process?
 - Continued communication across team members. Improving accuracy of point allotments for current and future user stories.
- How to improve the product?
 - Continued work on the user stories generated thus far.

Sprint 3 Review Meeting Minutes

Attendees: Alejandro Thornton, Mikaila Daniel, Michael Quiros

Start time: 10/15/17, 5:00pm

End time: 10/15/17, 6:00pm

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners:

- User Story #670: Create Profile Model
- User Story #671: Create Profile API Routes
- User Story #669: Create Temporary Server Testing Space
- User Story #674: Create Basic App Mockup
- User Story #676: Create Main Menu Page

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Spring Planning meeting.

- User Story # 679: Test Function Calls to the Application
- User Story # 672: Create Profile Page

Sprint 3 Retrospective Meeting Minutes

Attendees: Alejandro Thornton, Mikaila Daniel, Michael Quiros

Start time: 10/15/2017, 6:00pm

End time: 10/15/2017, 7:00pm

What went wrong?

- Did we do a good job estimating our team's velocity?
 - Some of the user stories for this sprint were completed within the allotted points. However, others were not completed in this sprint within the allotted points. Ultimately, the predicted team velocity can and should be improved for future sprints.
- Did we do a good job estimating the points (time required) for each user story?
 - For the user stories in future sprints we will have to allot more points for the user stories than we have been for past sprints.
- Did each team member work as scheduled?
 - Each team member has worked as scheduled.

What went right?

- User stories related to the database operations have been completed on time. Much of the foundational work for the application has been done in this sprint which should facilitate the work done in future sprints.

How to address the issues in the next sprint?

- How to improve the process?
 - With the groundwork laid in this sprint, further iterations of the project should be easier to implement.
- How to improve the product?
 - Work with the marketing arm of the project to coordinate ideas on the design of the mobile application.

Sprint 4 Review Meeting Minutes

Attendees: Alejandro Thornton, Mikaila Daniel, Michael Quiros

Start time: 10/29/17, 6:00pm

End time: 10/29/17, 7:00pm

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners:

- User Story #678: Testing Calls From the Database
- User Story #681: Create Meeting Model
- User Story #683: Create Meeting API Routes
- User Story #684: Create User Model and Routes
- User Story #688: User Authentication
- User Story #690: Token Verification

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Spring Planning meeting.

- User Story #680: Create New Meeting
- User Story #682: View Meetings
- User Story #669: Create Temporary Server Testing Space
- User Story #672: Profile Page
- User Story #679: Testing Function Calls to the Application

Sprint 4 Retrospective Meeting Minutes

Attendees: Alejandro Thornton, Mikaila Daniel, Michael Quiros

Start time: 10/29/2017, 7:00pm

End time: 10/29/2017, 7:30pm

What went wrong?

- Did we do a good job estimating our team's velocity?
 - The velocity was mostly accurate, however some user stories are being pushed to the next sprint.
- Did we do a good job estimating the points (time required) for each user story?
 - Most of the user stories had accurate point estimation. The exception being the user stories which are being passed on to future sprints.
- Did each team member work as scheduled?
 - For the most part.

What went right?

- Authentication is all but completed for the system. The database operations are essentially fully implemented, pending some updates from the front-end and design teams.

How to address the issues in the next sprint?

- How to improve the process?
 - Better communication between team members when complications in implementation arise.
- How to improve the product?
 - Further iteration on the front-end to realize the work done by the design team.

Sprint 5 Review Meeting Minutes

Attendees: Alejandro Thornton, Mikaila Daniel, Michael Quiros

Start time: 11/14/17, 5:30pm

End time: 11/14/17, 6:00pm

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners:

- User Story #685: Create Program Model
- User Story #686: Create Program API Routes
- User Story #687: Test Meeting Calls From Application
- User Story #682: View Meetings

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Spring Planning meeting.

- User Story #680: Create New Meeting
- User Story #689: Create Local Database on Remote Server

Sprint 5 Retrospective Meeting Minutes

Attendees: Alejandro Thornton, Mikaila Daniel, Michael Quiros

Start time: 11/14/2017, 6:00pm

End time: 11/14/2017, 6:30pm

What went wrong?

- Did we do a good job estimating our team's velocity?
 - Overall, it was accurate except for a few user stories which were close to completion, but not yet done.
- Did we do a good job estimating the points (time required) for each user story?
 - A couple of the user stories had accurate point totals but the rest were underestimated.
- Did each team member work as scheduled?
 - Yes, each team member worked as scheduled.

What went right?

- The back-end of the application is completed unless some corrections come up as the work continues.

How to address the issues in the next sprint?

- How to improve the process?
 - Continue putting in time for the final push to complete the project.
- How to improve the product?
 - Continue coordinating with the marketing/design team to realize their vision for the project.

Sprint 6 Review Meeting Minutes

Attendees: Alejandro Thornton, Mikaila Daniel, Michael Quiros

Start time: 11/29/17, 2:20pm

End time: 11/29/17, 2:50pm

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners:

- #693: Search by Name or University
- #692: Log onto Server from Application
- #695: Edit Profile
- #680: Create New Meeting
- #691: View Member Profile
- #694: Edit Meeting
- #689: Create Local Database on Remote Server

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Spring Planning meeting.

- None

Sprint 6 Retrospective Meeting Minutes

Attendees: Alejandro Thornton, Mikaila Daniel, Michael Quiros

Start time: 11/29/2017 2:20pm

End time: 11/29/2017 2:50pm

What went wrong?

- Did we do a good job estimating our team's velocity?
 - Overall the actual team velocity was close to the velocity estimated at the start of the sprint.
- Did we do a good job estimating the points (time required) for each user story?
 - Some of the user stories for this sprint were overestimated while others were underestimated. The front-end work went more quickly than anticipated while the back-end was slower.
- Did each team member work as scheduled?
 - For the most part, team members worked as scheduled in this sprint.

What went right?

- The mobile application is essentially fully functional and the database is populated. Much of the requirements specified at the beginning of the project have been accomplished.

How to address the issues in the next sprint?

- How to improve the process?
 - Continue with the groundwork laid out in previous sprints.
- How to improve the product?
 - Focus on providing administrative pages for managing the application content. Review the product with the product owner to ensure it is up specification.

Sprint 7 Review Meeting Minutes

Attendees: Alejandro Thornton, Mikaila Daniel, Michael Quiros

Start time: 12/14/17, 6:30pm

End time: 12/14/17, 7:00pm

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners:

- #696: Display Action Messages

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Spring Planning meeting.

- #697: Sign up for Account
- #698: Formatting Clean-Up
- #699: Set up Administrative Data Manipulation Pages

Sprint 7 Retrospective Meeting Minutes

Attendees: Alejandro Thornton, Mikaila Daniel, Michael Quiros

Start time: 12/14/17, 7:00pm

End time: 12/14/17, 7:30pm

What went wrong?

- Did we do a good job estimating our team's velocity?
 - No, the time available for this project was overestimated against preparing for finals and completing the documentation for the project.
- Did we do a good job estimating the points (time required) for each user story?
 - Only one user story was completed on time
- Did each team member work as scheduled?
 - The work aside from the user stories in this sprint was completed as scheduled.

What went right?

- This project presents a good foundation for a follow-up in an upcoming semester, many goals of the project were reached. The final deliverable will contain a wishlight and the project shortcomings.

APPENDIX D: VIDEO PLAYLIST

This link contains the playlist for the introductory, tutorial, installation, and wishlist/shortcomings videos:

https://www.youtube.com/watch?v=_MivOQcF7LA&list=PLG3lylzOg8FhALtQkCCWswZPIJiGAjwxW