

# ***A Band Meetup web application***

## **OO Design Specification Document**

*Group 4*

Rochester Institute of Technology  
Golisano College of Computing and Information Sciences

Date: 11-16-2020

Version 1:

11/16/2020, 7:01 PM

**Version History**

VERSION	AUTHOR(S)	CHANGE DESCRIPTION	DATE	NOTES
1.0	Sultan, Stephon, Devan, Vaibhavi	Create the document	11/16/20	

**Table of Content**

<b>Overview</b>	<b>3</b>
Goals and Objectives	4
Software Context	4
Terminology & Definitions	4
<b>Functional Descriptions</b>	<b>4</b>
<b>Non-Functional Descriptions</b>	<b>5</b>
<b>System Architecture</b>	<b>6</b>
Node Architecture	6
Component Architecture	6
Data Architecture	7
<b>Detailed Architecture</b>	<b>8</b>
High Level Architecture	8
User Accounts Subsystem	10
Profile Subsystem	12
Event Subsystem	13
Search Subsystem	14
DAO Subsystem	15
Interaction Diagram	16
Human interface	20
User interface design	20
<b>Implementation Issues &amp; Challenges</b>	<b>23</b>
Supporting Systems Requirements	<b>23</b>
Design Software Requirements	23
<b>Conclusions &amp; Future Extensions</b>	<b>23</b>
<b>Related Material</b>	<b>23</b>

---

# 1 Overview

This design documentation of the web

## 1.1 Goals and Objectives

---

This is a design documentation of a web application project termed band meetup project. This shows the design decisions where it is made in multiple models transformed from SRS document.

## 1.2 Software Context

---

The band meetup is designed with a friendly user interface to allow different musicians to meet and allows venue search about musicians to invite them in an event.

## 1.3 Terminology & Definitions

---

SRS : Software Requirements Specification

intellij: Java development environment

Them: Tool used to develop in http/html languages

Spring framework: Framework to maintain our software infrastructure.

SQLite: A IDE for database

ERD: An Entity Relationship Diagram

UI: User Interface

---

# 2 Functional Descriptions

### General Profile Requirements:

1. Users shall be able to register new accounts using an email and password.
2. Users shall be able to choose their account type: Musician or Venue
3. Users shall be able to view and edit their profile information.
4. Users shall be able to set their location.

### Musician Profile Requirements:

1. Musicians shall be able to add musical instruments and musical genre to their profile information.
2. Musicians shall be able to add and update a small biography.
3. Musicians shall be able to add and link to external social media profiles.
4. Musicians shall be able to upload audio files to their profile.

11/16/2020, 7:01 PM

5. Musicians shall be able to set profile status.
6. Statuses shall be “Looking to join a band”, “Looking for a band member”, “Looking to start a band”, or “Looking to jam”.
7. Musician Profile displays information: Name, Email, Status, Musical Instrument, Musical Genre, Location and Bio.

#### **Venue Profile Requirements:**

1. Venues shall be able to create and link new events
2. Event details shall consist of Date, Location, and Name.
3. Venue Profile displays its information: Name and Location
4. Venue shall be able to invite musicians to join their events.

#### **Profile and Event Searching Requirements:**

Event should be displayed in Venue’s profile

1. Users shall be able to search Events.
2. Users shall be able to search Venue.
3. Users shall be able to filter their search of musicians based on four categories:
  - 3.1. Area code
  - 3.2. Genre of the music
  - 3.3. Musical instruments
  - 3.4. Profile Status
4. Users shall be able to view user profiles.

#### **Chat Requirements:**

1. Users shall be able to send and receive messages.
2. Users shall be able to invite other users to chat.
3. Chats shall be opened when the invitation is accepted.
4. Users shall be able to view previous chats.

#### **Admin Requirements:**

1. Admin shall be able to create new profiles.
2. Admin shall be able to view profiles.
3. Admin shall be able to modify profiles
4. Admin shall be able to delete profiles.

---

### **3 Non-Functional Descriptions**

- **Security**

User’s data shall be confidential.

The system shall be able to verify the account’s email address.

Each role should have different page

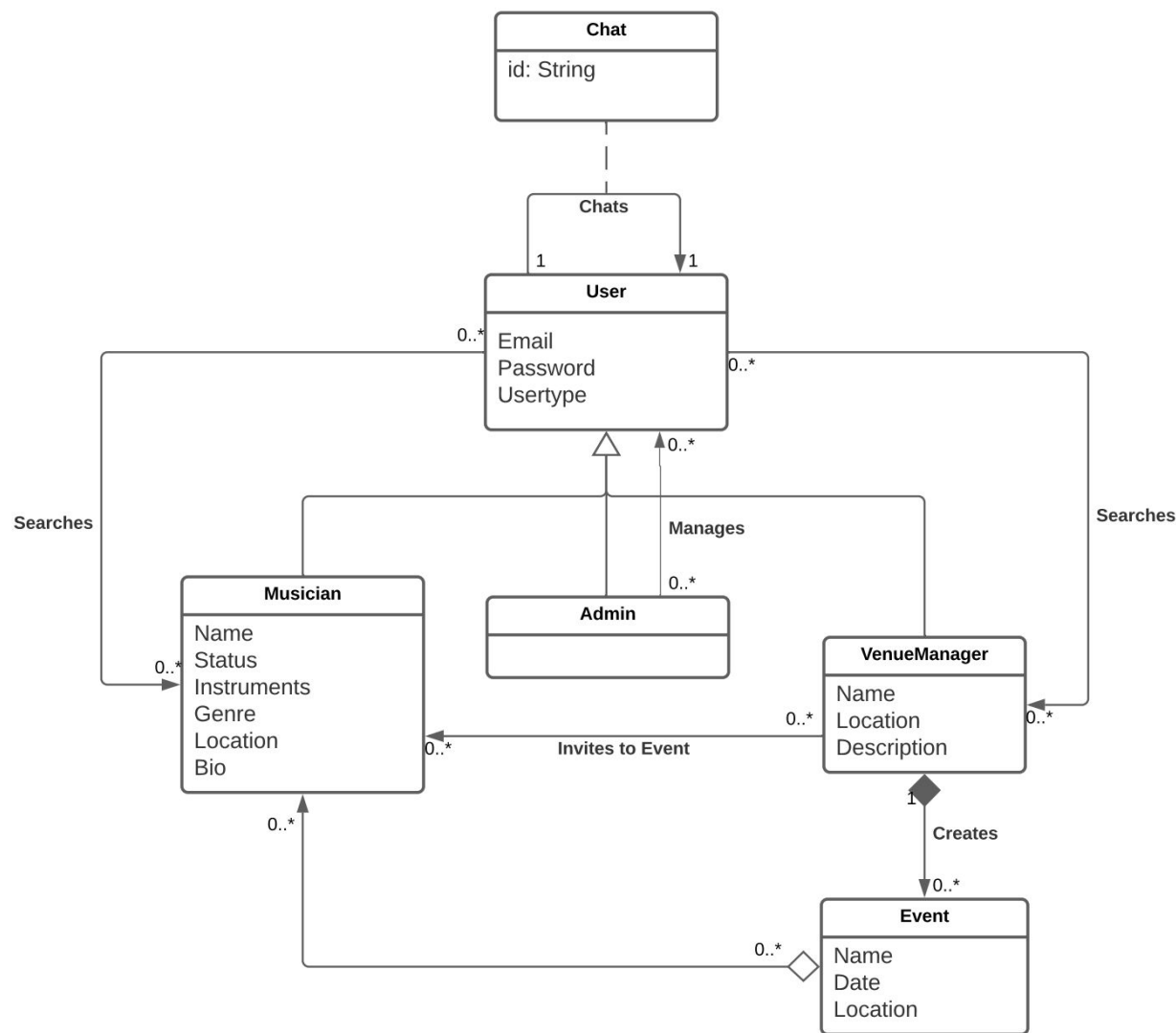
Password shall be encrypted

- **Availability**

The product shall be available only to registered users.

- **Usability**  
User should navigate easily in the system  
System shall support only English language

## 4 System Architecture



### 4.1 Node Architecture

N/A

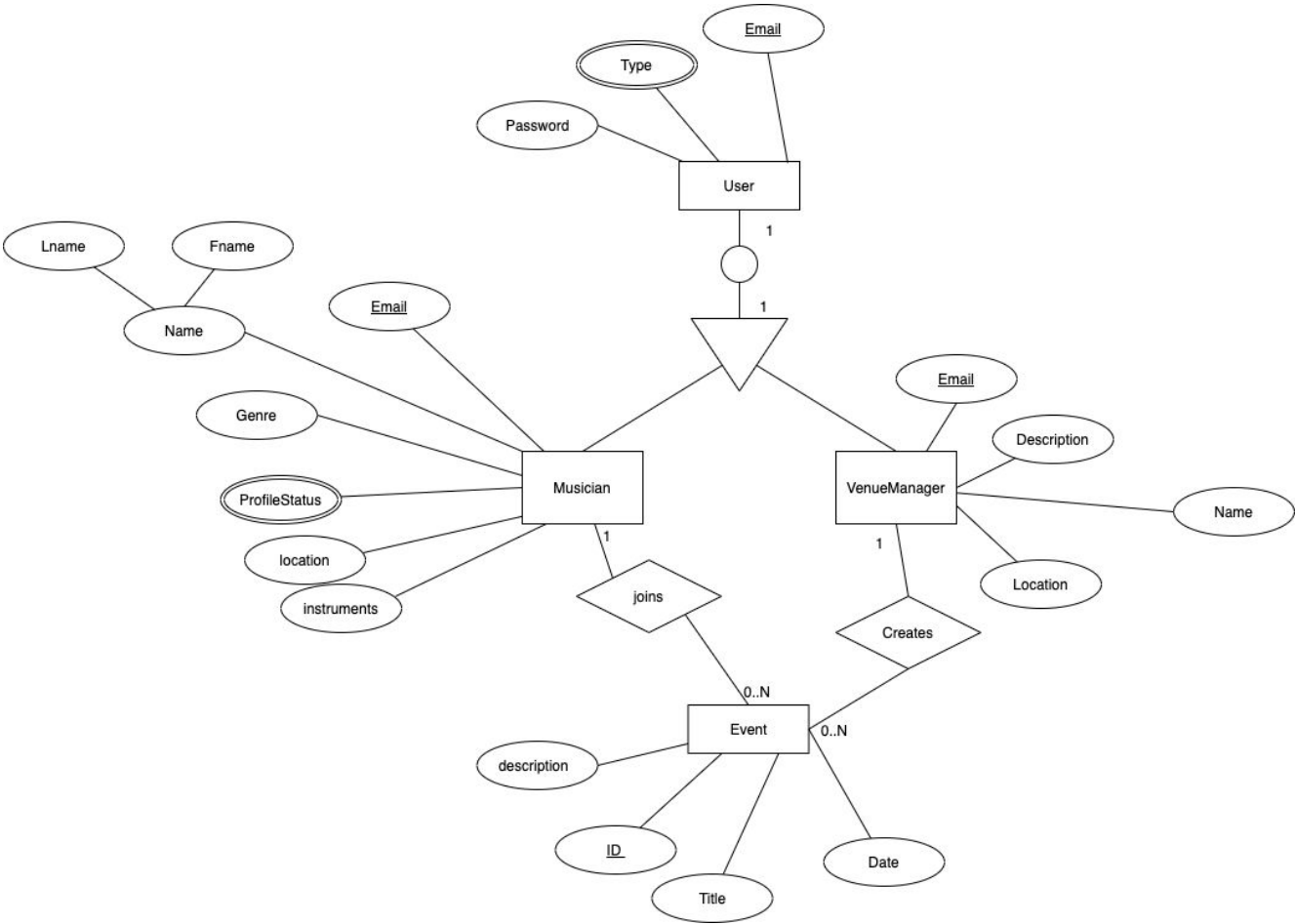
11/16/2020, 7:01 PM

## 4.2 Component Architecture

---

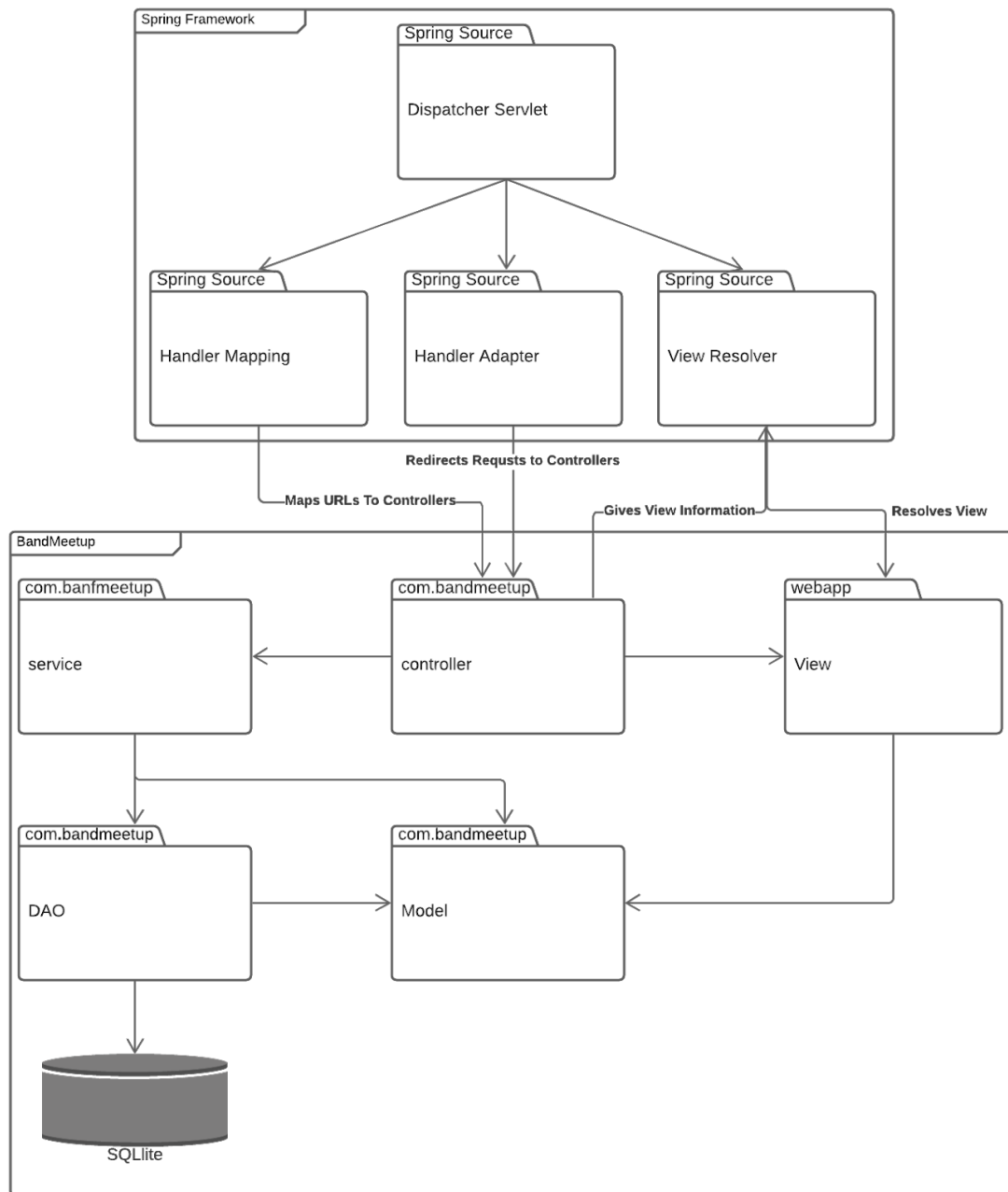
## 4.3 Data Architecture

---



## 5 Detailed Architecture

### 5.1 High Level Architecture

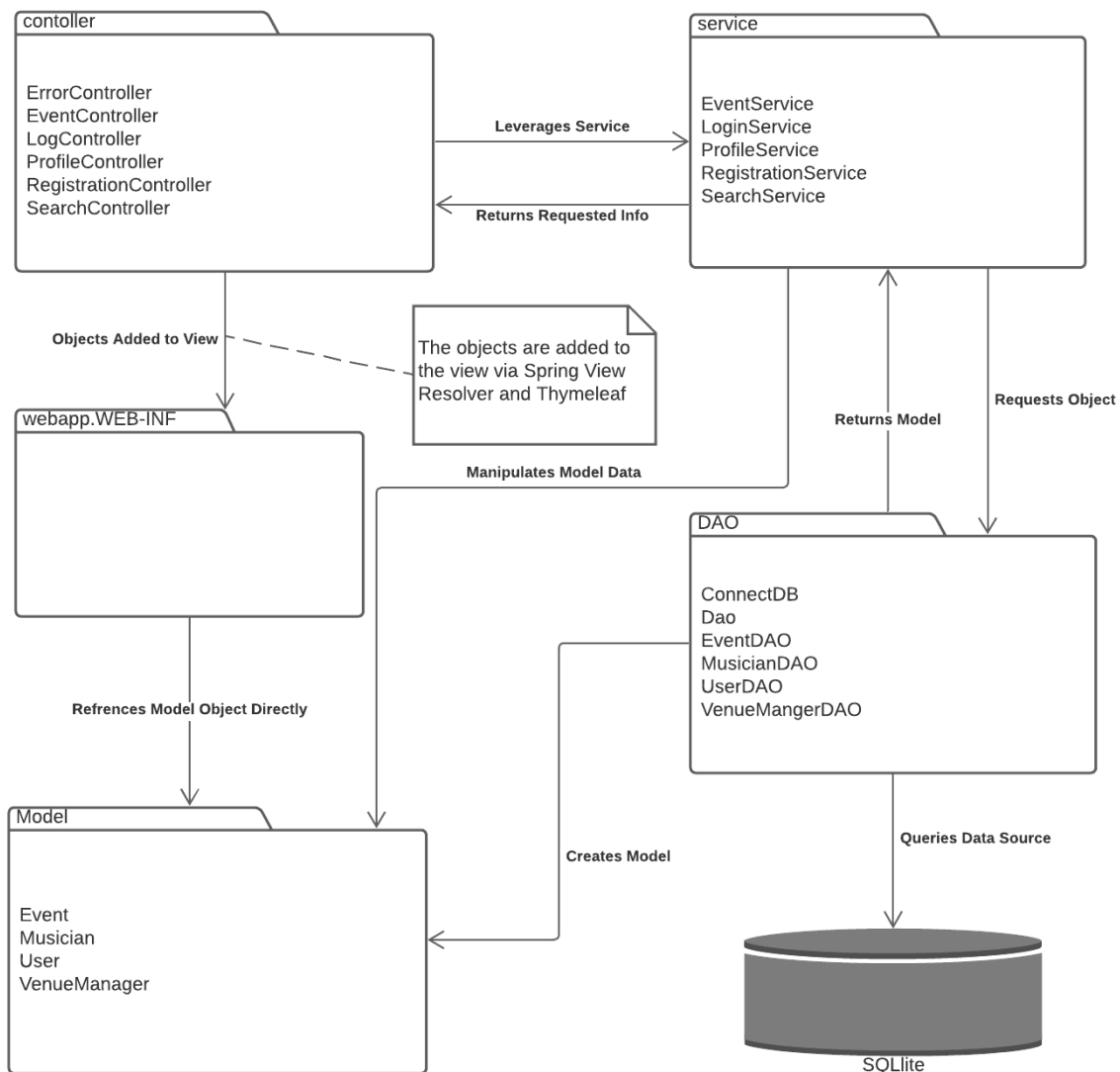


In this Package Diagram, The overall structure including the Spring Framework is roughly outlined. The important modules of the spring framework highlighted here is the Dispatcher Servlet handles and directs incoming HTTP requests. The Dispatcher Servlet leverages three services to achieve this. The first is the Handler Mapping, which is used to map URLs to our controllers. The second is the Handler Adapter, which is used to run the controller code and take in models and redirect information. The

11/16/2020, 7:01 PM



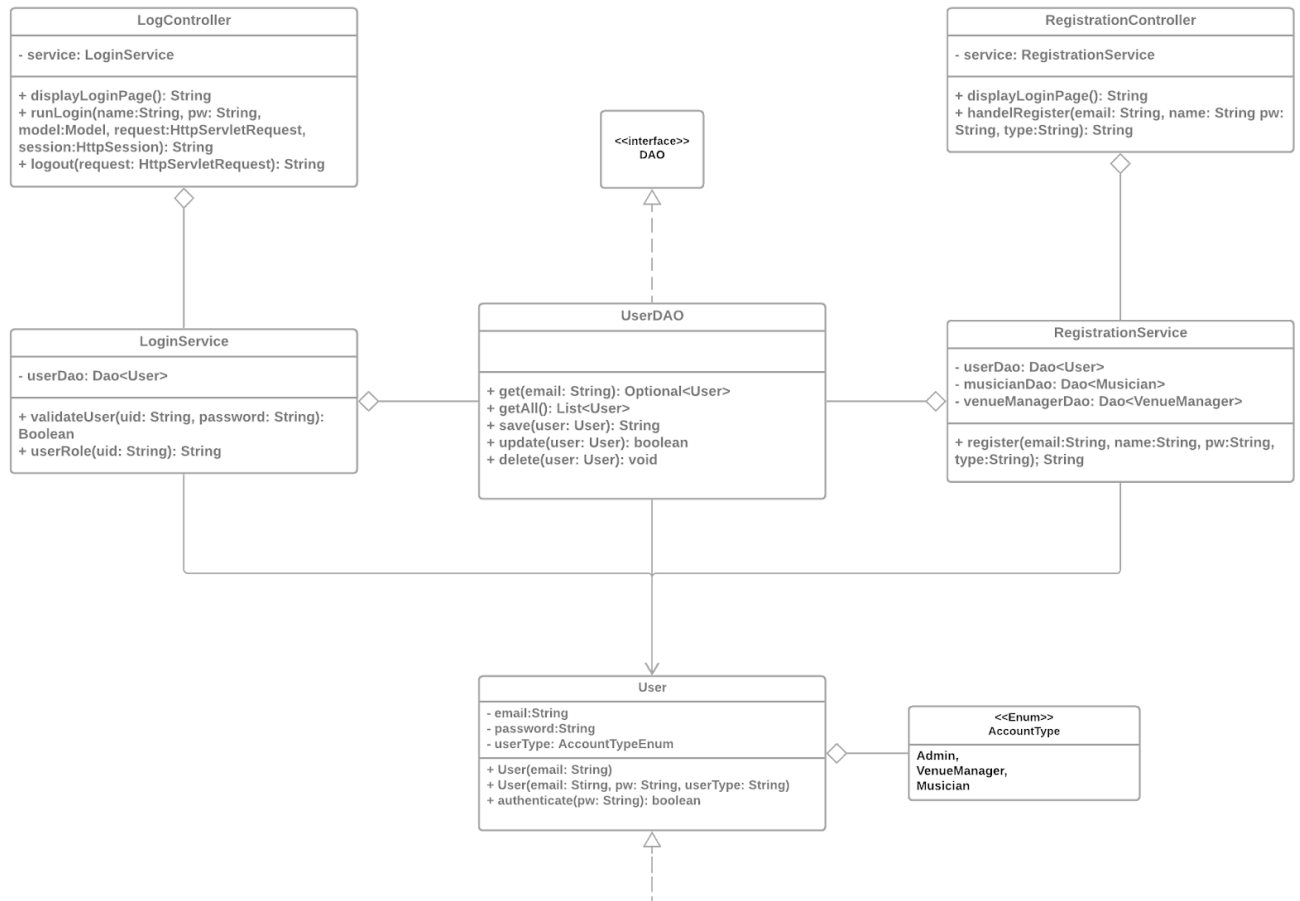
third and final module is the View Resolver. The View Resolver takes the string returned by the controller and the information inputted into the model and displays the mapped view.



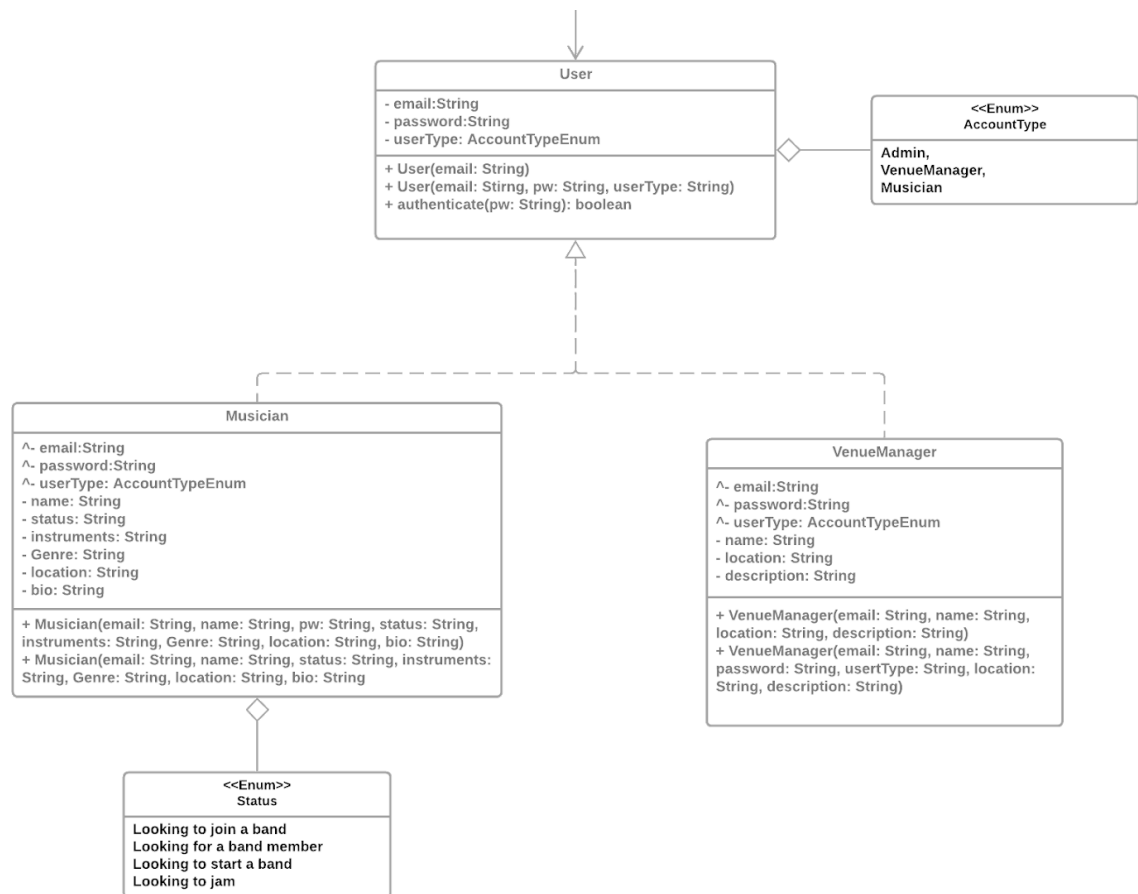
In this diagram, a more in depth demonstration of the Band Meetup architecture is shown. This shows the rough MVC pattern that was used. At the Model Layer, the system uses the Model Package and the Data Access Object (DAO) package. The models are simple objects used to represent domain entities. The current set of models hold little more than database information; however, this structure allows for expanded functionality in the future. The DAOs act as our database interface. The DAOs allow for querying of, saving to, and deletion from our database. At the Controller level we have the Controller and Service. The Controllers are the main access point for the Spring Framework. The main job of the controller is to take information from and pass information to the view. The controller does most of its operations via services. The services are the main data manipulations. The services query the DAOs and manipulate model data. The services are the focus point of our application, as the services are where the domain logic lies. At the view layer, we have the WEB-INF package. This layer holds our html and css and is used to represent the models in a user friendly way. The view leverages the Thymeleaf templating engine and Spring's models to display the model information.

## 5.2 User Accounts Subsystem

The User Accounts Subsystem focuses on the management of users. This includes functionality to create new users as well as login users in.



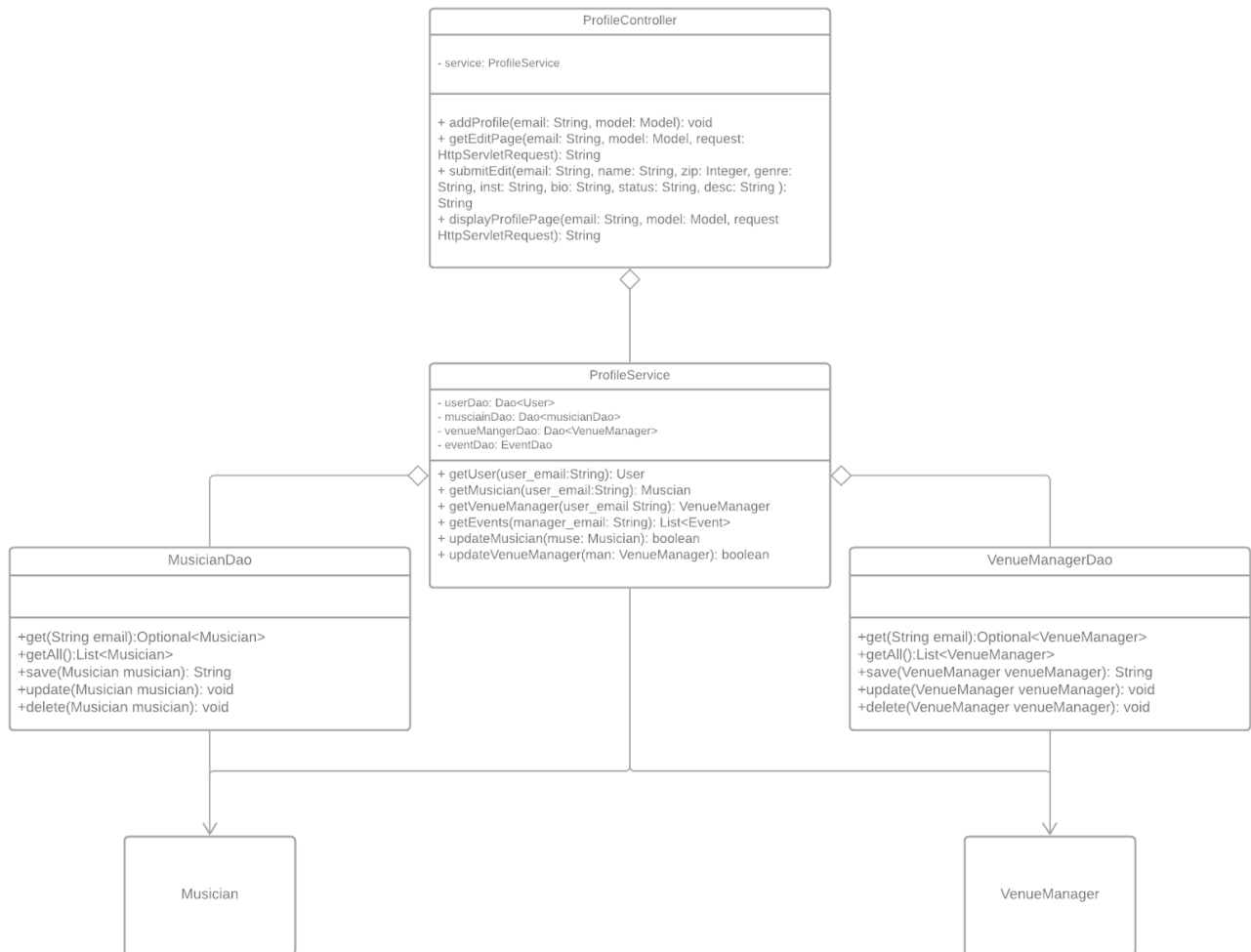
The above diagram shows the functionality for login and registration and their interactions with the user entity. Both the `LogController` and `RegistrationController` leverage their own services. These services use the `UserDAO` to access or save user information. In the `LoginServices` case it simply queries the `UserDAO` for an email address. The `DAO` creates and returns a `User` object if one exists. The `Login Service` then runs the authentication method on the `User` and returns the result. The `RegistrationService` simply uses the `DAO` to save a created user.



This diagram shows the inheritance nature of accounts. Both Musician profile and VenueManager inherit their account information from the user. This design is identified as somewhat inefficient and needing refactoring. Because of the way that the spring framework works, having just user entities representing users and then Musician and VenueManager representing profile information would make more sense. This would allow us to abstract users from profile information, somewhat simplifying user interaction.

### 5.3 Profile Subsystem

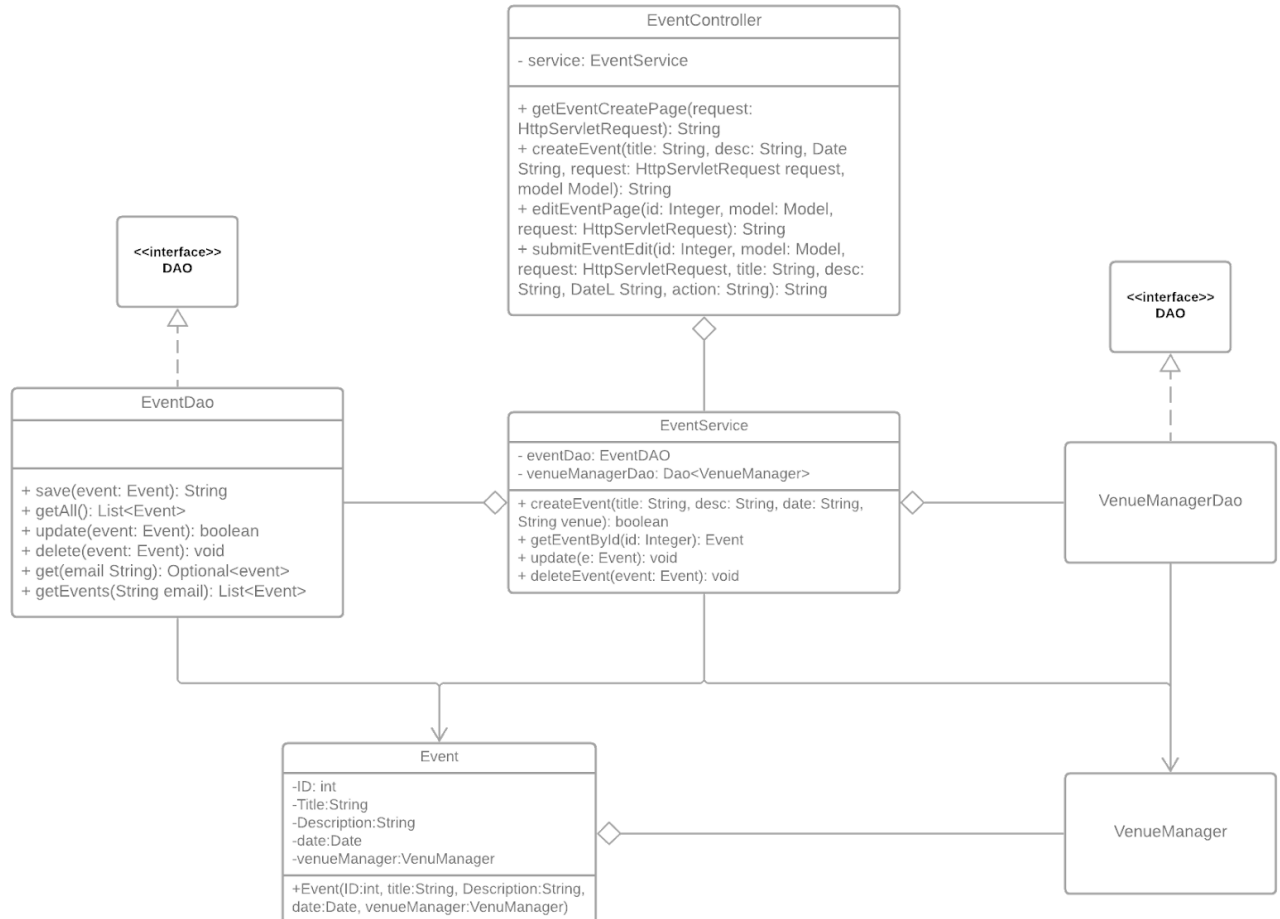
The Profile System handles the getting of profile information for the view as well as the editing of a user profile.



The above diagram shows how the profile controller handles the requests related to profile. Some aspects have been abstracted out as they are explained in other diagrams. The Profile controller takes in frontend redirect requests and form information and passes this on to the service. The service then uses the profile type's DAO to get and create the object from the database. The service then either edits the data and saves the object back to the database or returns it to the controller based on the request.

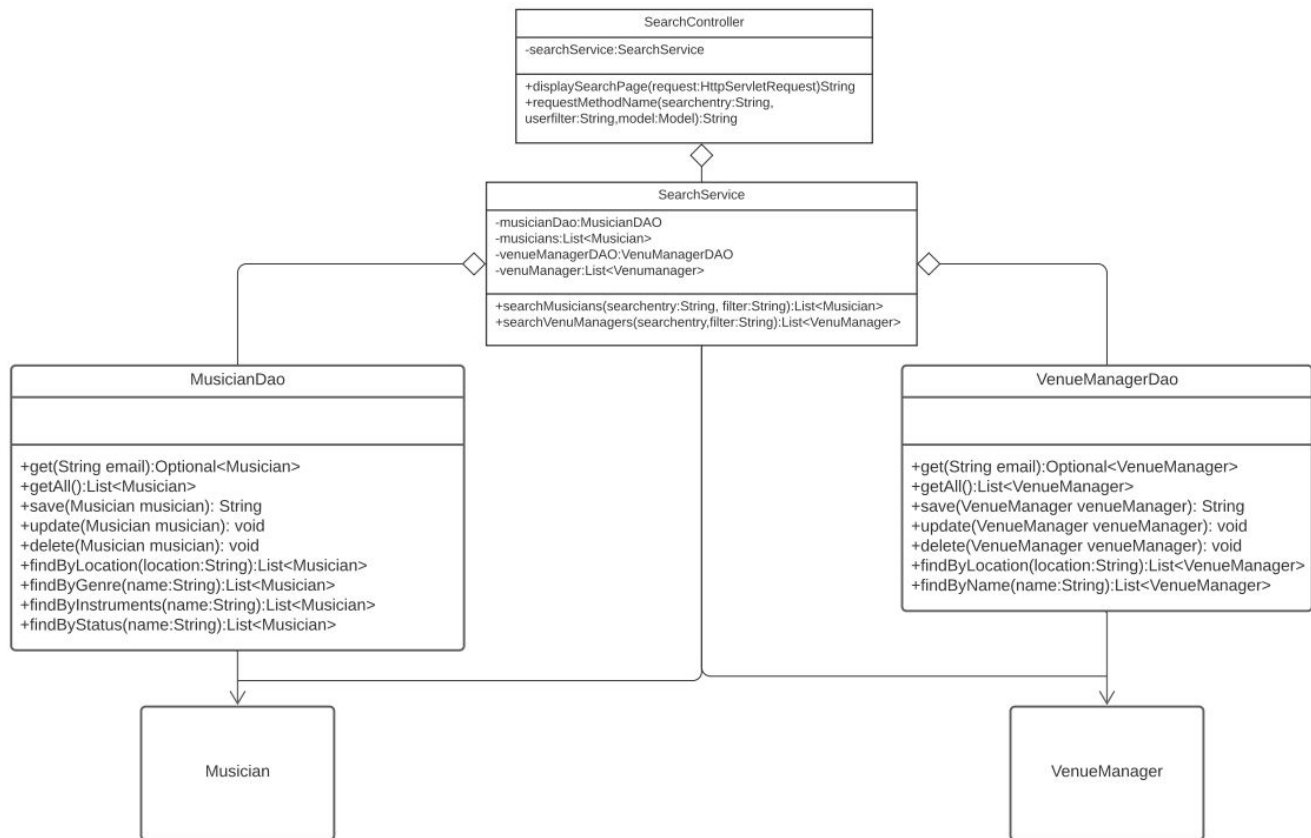
## 5.4 Event Subsystem

The event subsystem focuses on the creation of Venue Events. Venue events are created by Venue Managers. In addition VenueManagers can Edit and Delete their Events.



The above diagram shows the Event Subsystem, with the VenueComponents abstracted out as they are outlined in a previous section. The Event Controller manages the frontend interactions revolving around event creation or editing. The controller leverages the EventService to access and manipulate the event object. On event creation, the service uses the VenueManagerDao to acquire the manager and create a new event object. An event object consists of an auto generated ID, event title, event description, a date, and a reference to the venue manager. Once the service creates the object it uses the EventDao to save it to the DB. In the event of edit or deletion, the event service gets the object from the Dao. The service then authorizes the venue manager and edits or deletes the event. Edits are done via the EventDao's update method while deletes are done via the delete method.

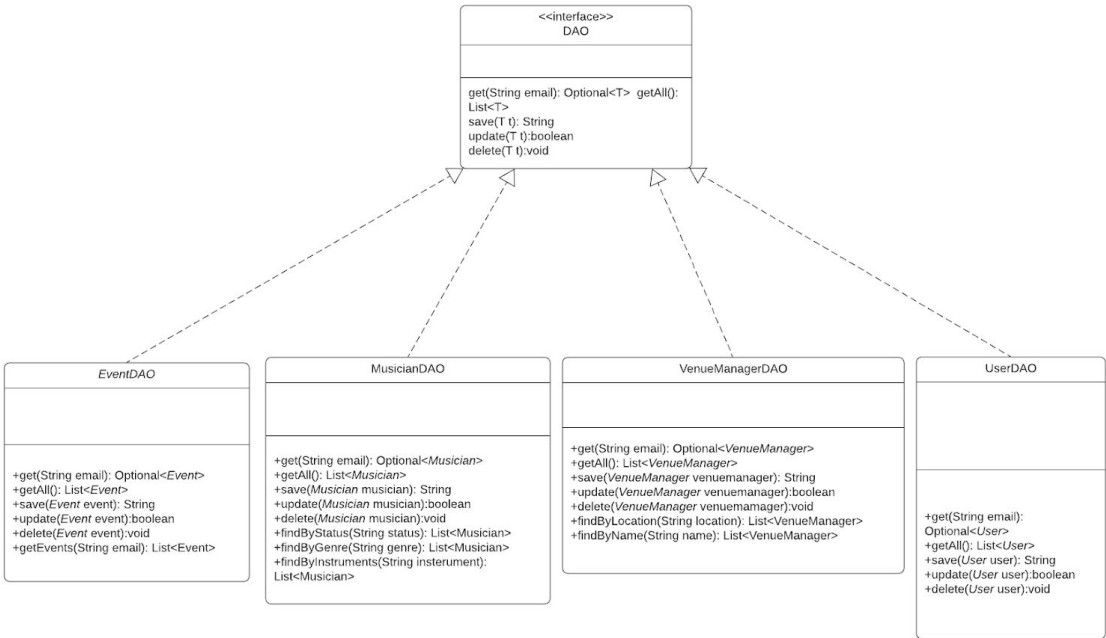
## 5.5 Search Subsystem



The above class diagram is the Search subsystem. The Search controller manages frontend interaction based around search for Musicians or VenuManagers based on specific filters. The controller leverages the SearchService to access and find Musicians or VenuManagers based on a filter. The service uses either MusicianDao or VenuManagerDao calling the findby methods like “findByName.” These would return a list of Musician or VenuManagers which are then passed by the Controller to display. Each object would have a profile button to access the profile page of that user.

## 5.6 DAO Subsystem

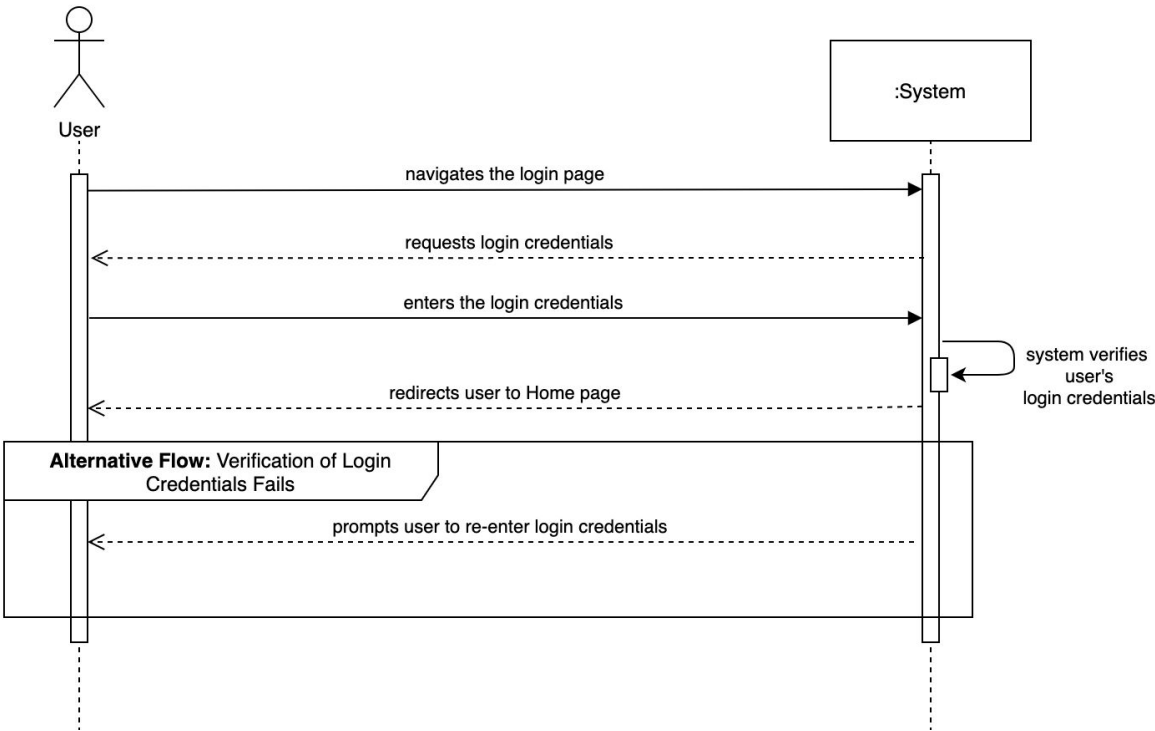
The DAO subsystem focuses on database integration to maintain the sql statements. The functions handle creating, deleting, updating or retrieving objects from the database.



The above class diagram is the DAO subsystem. The DAO class is an interface while UserDAO, VenueManagerDao, MusicianDAO and EventDAO are implementing the DAO class.

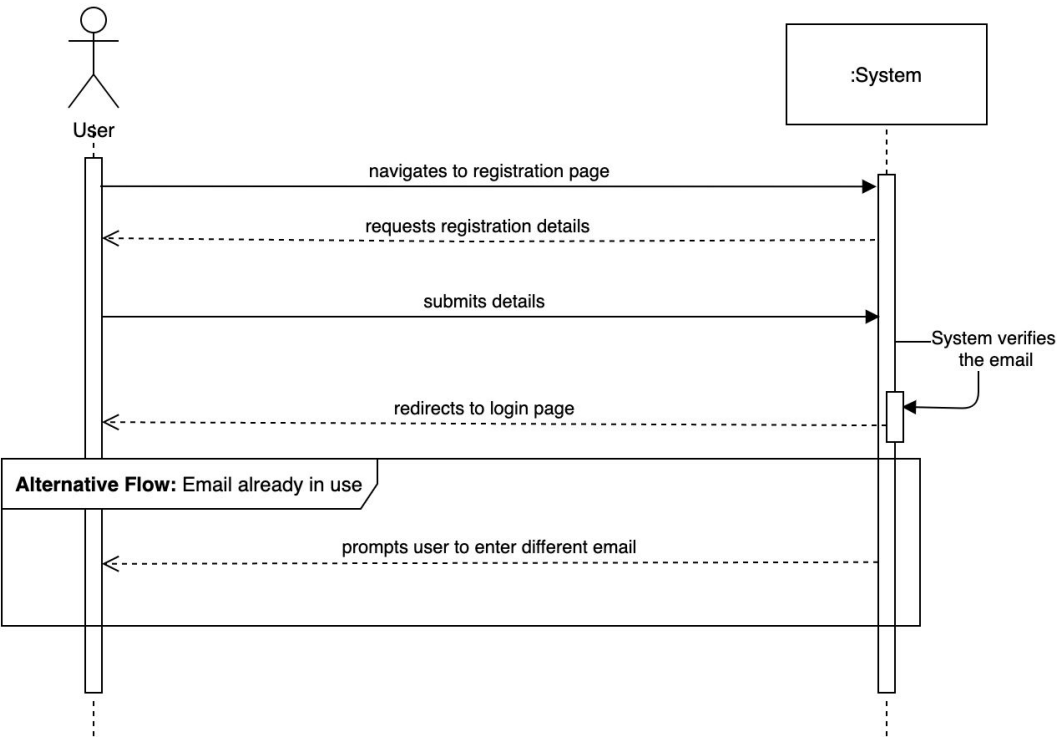
5.7 Interaction Diagram

Login Sequence Diagram

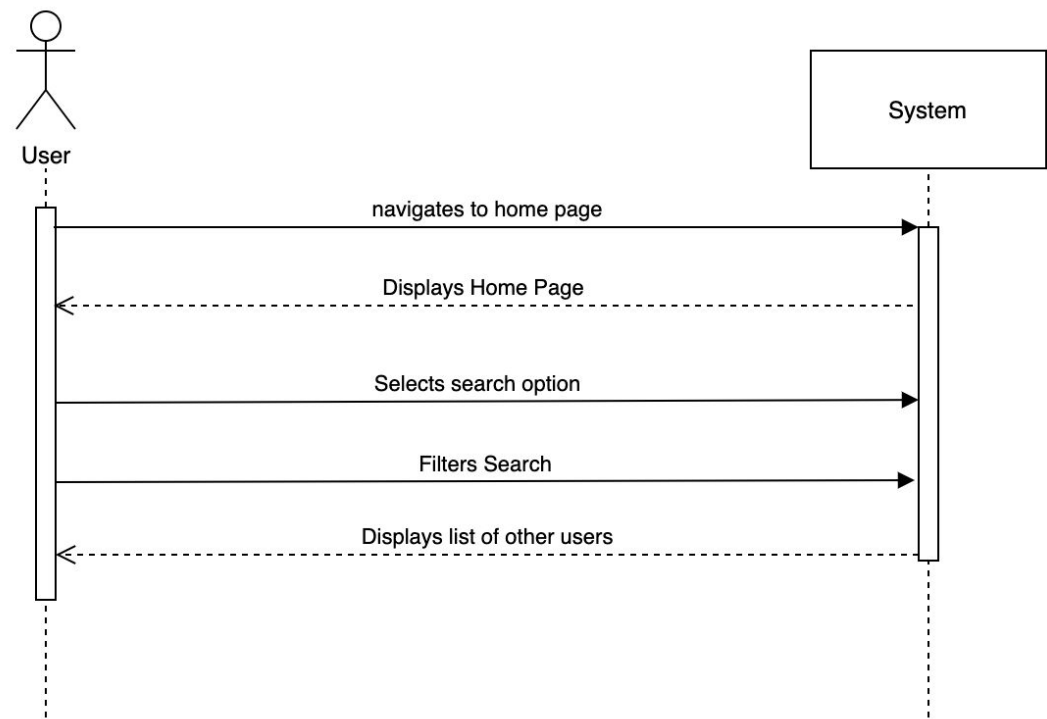




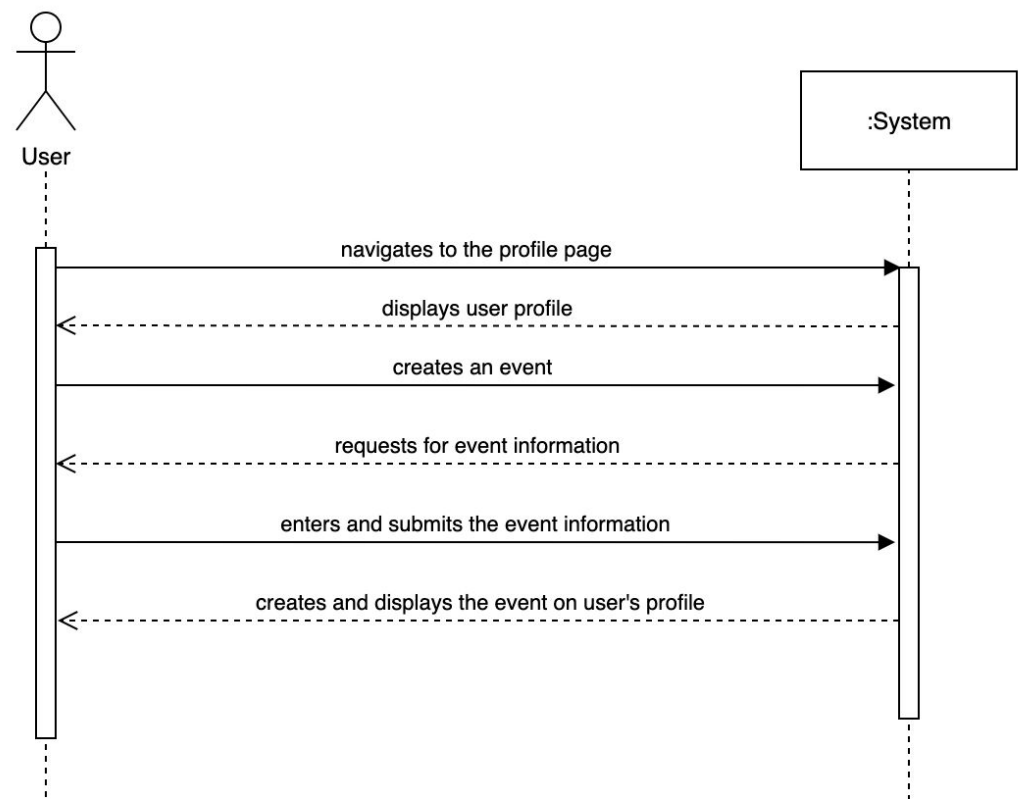
Registration Sequence Diagram



Search Sequence Diagram



# Create Event System Sequence Diagram



## 6 Human interface

---

### 6.1.1 User interface design

#### 6.1.1.1 login

Wireframe:

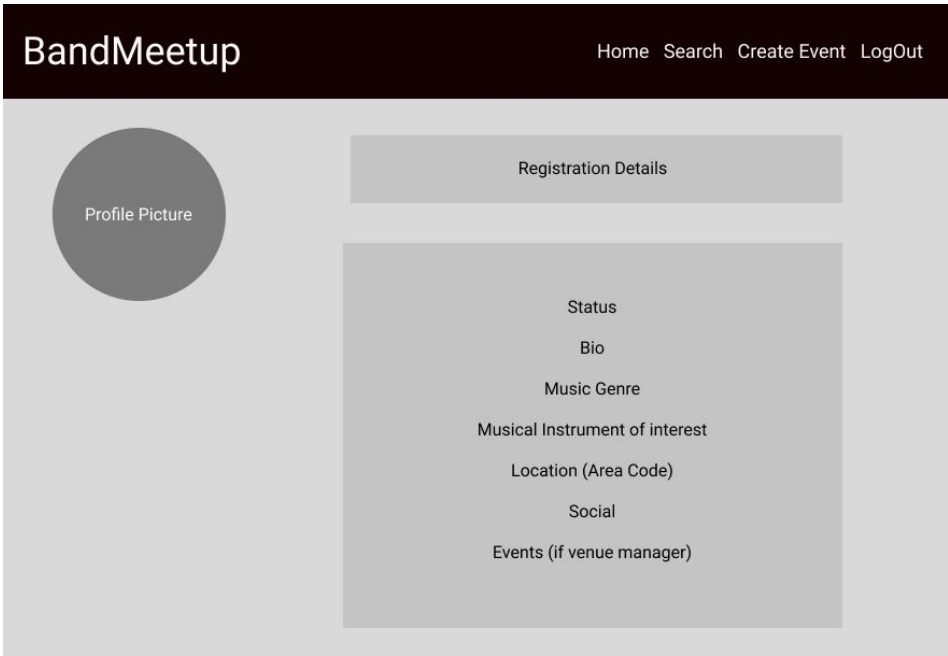
The wireframe shows a login page for 'BandMeetup'. At the top, there is a dark header bar with the site name 'BandMeetup' on the left and 'Sign Up' and 'Login' links on the right. The main content area is light gray and contains a centered white box with a light gray border. Inside this box, there are two input fields: 'Username:' followed by a text input field, and 'Password:' followed by a text input field. Below these fields is a dark gray button with the text 'Login' in white.

#### 6.1.1.2 Register

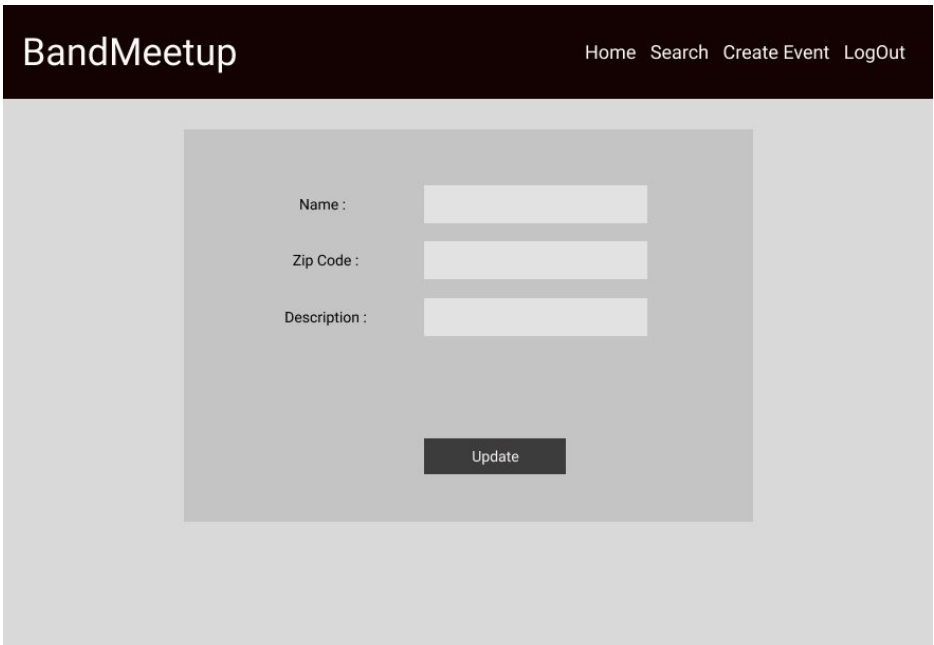
The wireframe shows a register page for 'BandMeetup'. At the top, there is a dark header bar with the site name 'BandMeetup' on the left and 'Sign Up' and 'Login' links on the right. The main content area is light gray and contains a centered white box with a light gray border. Inside this box, there are five input fields: 'Name :', 'Email :', 'Password :', 'Confirm Password :', and 'Account Type :', each followed by a text input field. Below these fields is a dark gray button with the text 'Register' in white.

11/16/2020, 7:01 PM

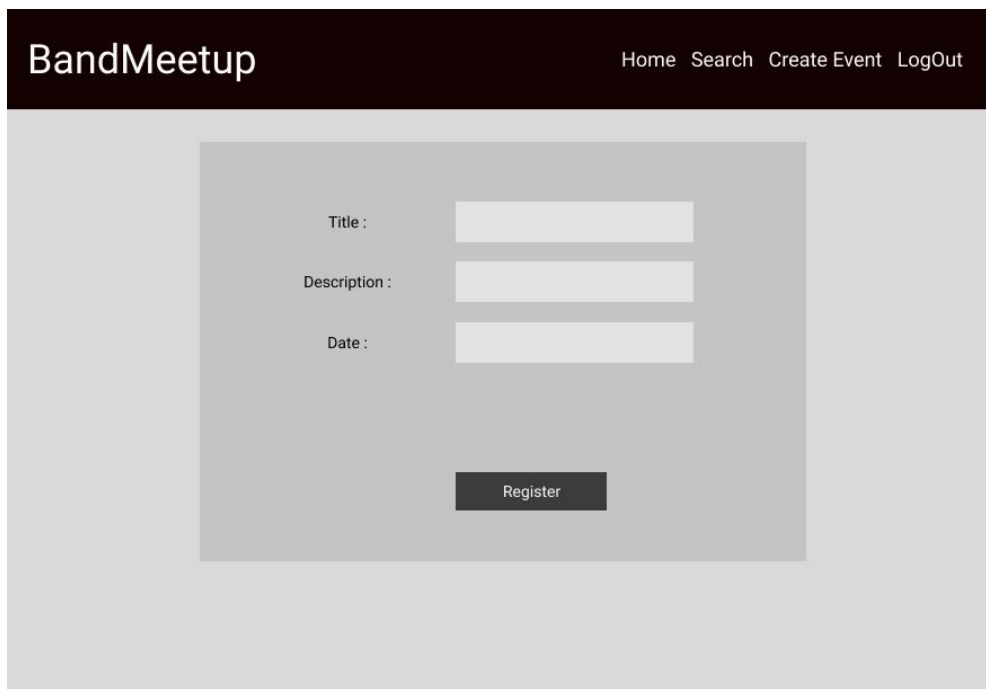
6.1.1.3 Home page



6.1.1.4 Edit Profile

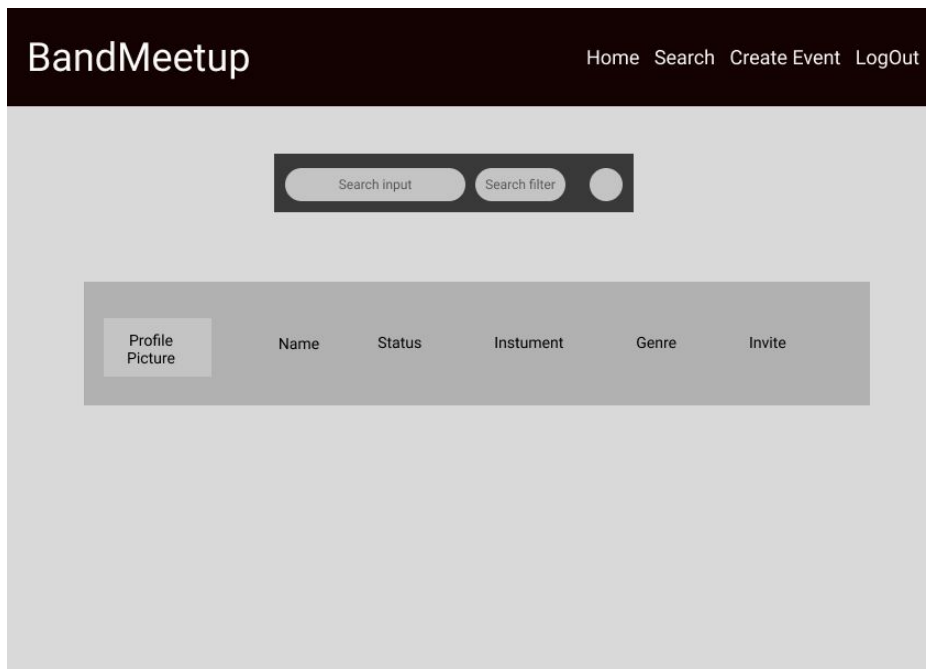


### 6.1.1.5 Create New Event



The screenshot shows the 'Create New Event' page of the BandMeetup application. The header is dark blue with the 'BandMeetup' logo on the left and navigation links 'Home', 'Search', 'Create Event', and 'LogOut' on the right. The main content area is light gray and contains a white form box. Inside the form box, there are three input fields labeled 'Title :', 'Description :', and 'Date :'. Below these fields is a dark blue button labeled 'Register'.

### 6.1.1.6 Search Page



The screenshot shows the 'Search' page of the BandMeetup application. The header is dark blue with the 'BandMeetup' logo on the left and navigation links 'Home', 'Search', 'Create Event', and 'LogOut' on the right. The main content area is light gray. At the top, there is a search bar with a 'Search input' field, a 'Search filter' button, and a circular search icon. Below the search bar is a table with the following columns: 'Profile Picture', 'Name', 'Status', 'Instument', 'Genre', and 'Invite'.

## 7 Implementation Issues & Challenges

---

## 8 Supporting Systems Requirements

---

### 8.1 Design Software Requirements

---

- Back-end: java
  - Front-end: Thymeleaf
  - Spring framework
  - Database: SQLite
- 

## 9 Conclusions & Future Extensions

In this design document, we design our software system “A Band Meetup web application”. This is done by designing: domain model, class diagram, system sequence diagram, ERD, and wireframe UI. In addition, this document shows our functional and none-functional requirements transferred from our SRS document. For future extension, we would improve our software system and develop chat features to allow users to communicate directly to each other. This will allow our system more interaction.

---

## 10 Related Material

[1] SRS document.