ITCS 4155
Eventure Design Document

1. Project Overview

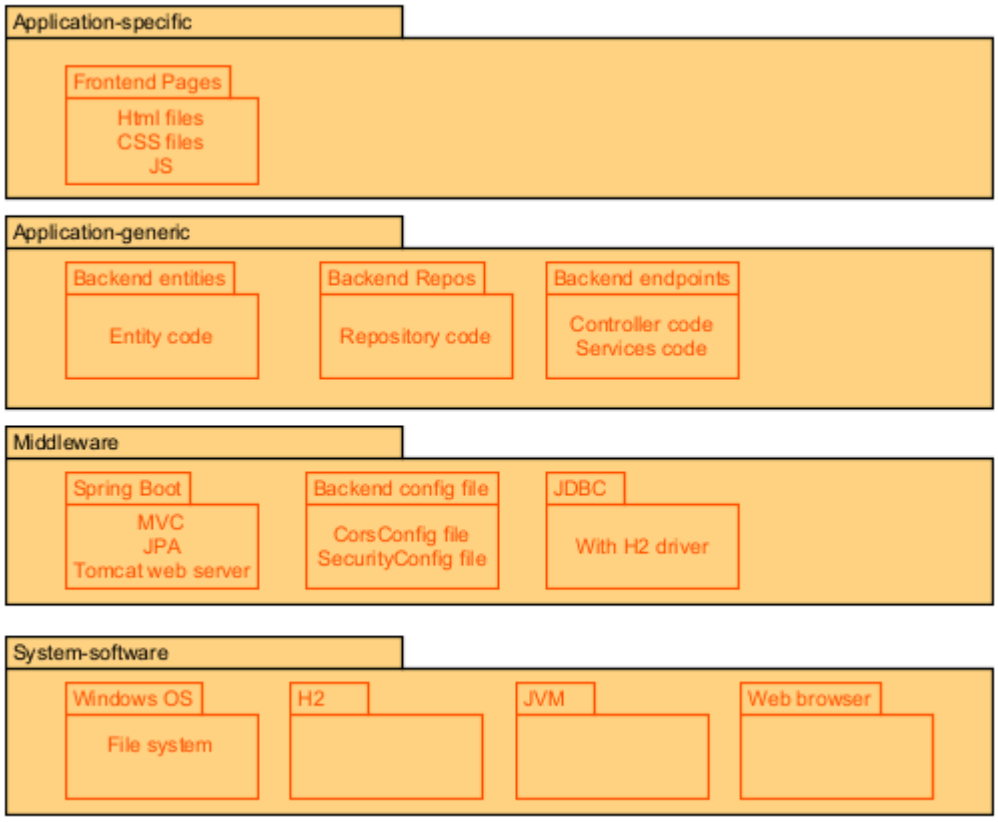
There are multiple events happening at UNC Charlotte at any given time: informal student meetups, university-hosted events, and events organized by student organizations. These events are advertised through a mix of physical flyers on corkboards, yard signs, TV screens on campus, and one-off digital posts. Students often miss these advertisements or discover them too late to adjust their schedules.

Eventure is a web application that centralizes event discovery for students. It allows students to browse flyers in one place, follow organizations, and save flyers they are interested in. Students can use Eventure to quickly publish digital flyers and reach a larger audience without relying only on physical posters. The system stores users, organizations, flyers, and interactions so that students can keep track of events over time instead of relying on discovering physical signs they walked past or having to keep track of a bunch of social media posts.

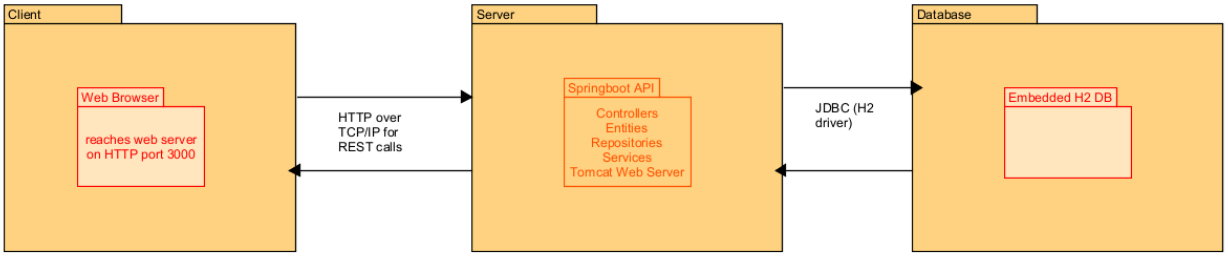
2. Architectural Overview

We chose a client–server architecture with a layered design mostly because it was simple and something we were all familiar with. It also achieved what we needed to accomplish, so we did not seriously consider other alternatives. The application uses static HTML, CSS, and JavaScript files served from a web server as the frontend of the application. The JavaScript running in the browser uses fetch calls to the app's endpoints in a REST API built with the Spring Boot framework. The API then uses a relational database for data persistence. We originally had a MySQL database, but we later switched to an H2 database embedded in Spring Boot so that, while working on it, nobody would have to deploy their own MySQL server, removing an extra build and setup step.

2.1 Subsystem Architecture



- Application specific.
    - Frontend pages: How the user interacts with the application, sends fetch requests to endpoints.
- Application generic
    - Backend Entities: Entity code with classes that establish the tables.
    - Backend Repos: Handles query commands to database.
    - Backend Endpoints Used by frontend to call services that provide data or allow the user to input data to the database.
- Middleware
    - SpringBoot Framework: Comes with tools like JPA, Tomcat, and MVC to enable the REST functionality and provide a web server that allows access to the endpoints.
    - Backend config file: CorsConfig and SecurityConfig provide restrictions on how one can access the endpoints or web server.
    - JDBC: Used to communicate with the H2 database.
- System software
    - Windows OS: Provides process execution and the file system stores the H2 database files and uploaded flyer images.
    - H2 Database: Stores the data from the application to provide data persistence.
    - JVM: Executes the database and the application.
    - Web browser: Allows the user to view the html files and interact with the application.



Our application, when deployed, is connected to by the user's web browser and it reaches the web server over HTTP port 3000 while the backend runs the Tomcat web server on HTTP port 8080. The client while accessing the frontend sends fetch requests over TCP/IP to the API to get and send data to the server. The backend uses JDBC to connect to the embedded H2 database and grabs and sends data to it for the endpoints.

2.3 Persistent Data Storage

We used H2 as our relational database. We needed to store all user account data, as well as any orgs. We also stored comments data and the data for the flyers. We used tables to store flyers that users have saved and orgs that users wanted in their feed. This is our schema:

```
CREATE TABLE users (

  id BIGINT NOT NULL AUTO_INCREMENT,

  username VARCHAR(255) NOT NULL,

  password VARCHAR(255) NOT NULL,

  email VARCHAR(255) NOT NULL,
```

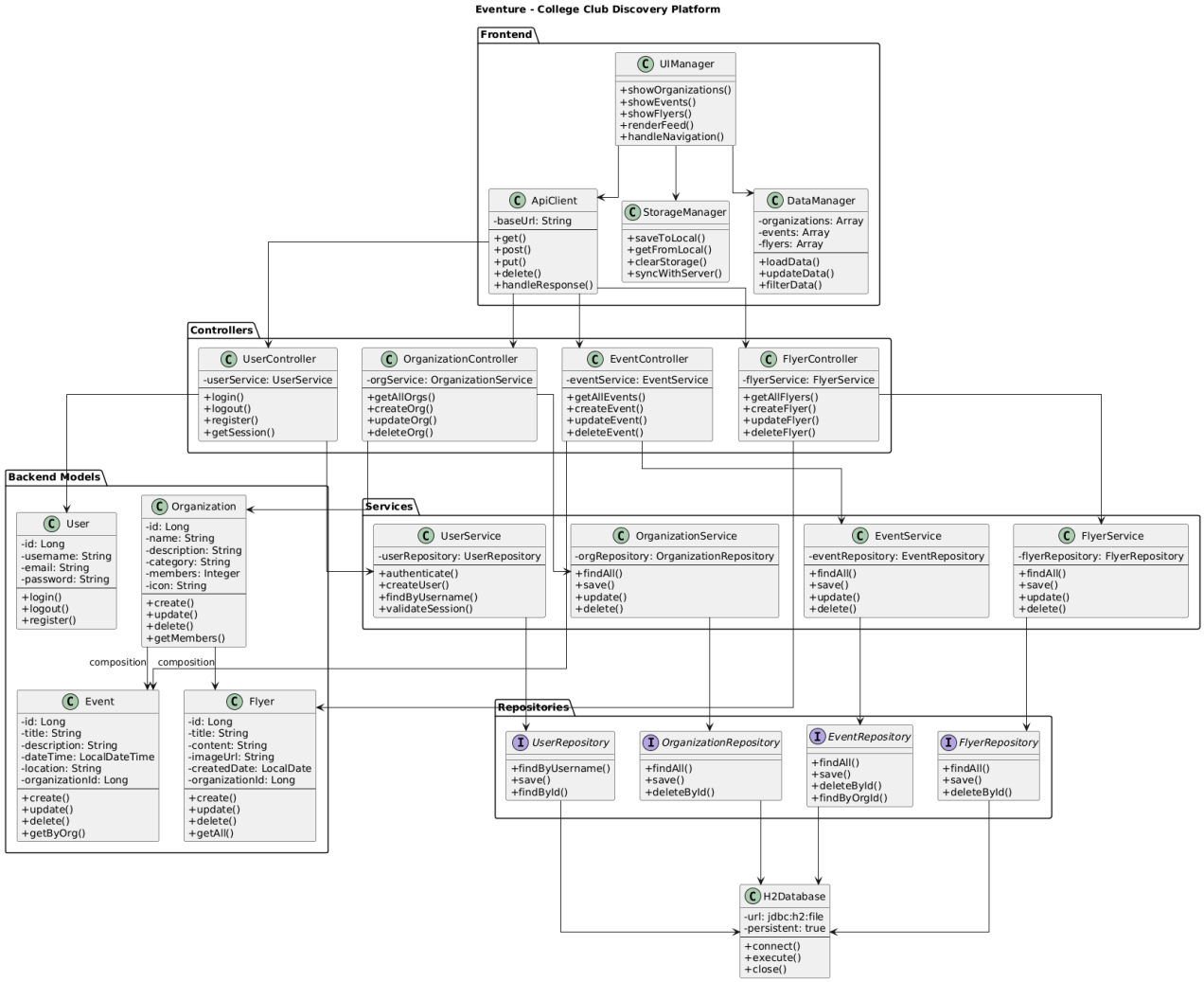
```
    PRIMARY KEY (id),
    UNIQUE KEY uq_users_username (username),
    UNIQUE KEY uq_users_email (email)
);
CREATE TABLE org (
    id BIGINT NOT NULL AUTO_INCREMENT,
    org_name      VARCHAR(255) NOT NULL,
    org_owner     VARCHAR(255) NOT NULL,
    org_followers BIGINT NOT NULL,
    PRIMARY KEY (id)
);
CREATE TABLE flyers (
    id BIGINT NOT NULL AUTO_INCREMENT,
    org_id          BIGINT      NOT NULL,
    flyer_advert    VARCHAR(255) NOT NULL,
    file_path       VARCHAR(255) NOT NULL,
    popularity_score INT        NOT NULL,
    PRIMARY KEY (id),
    KEY fk_flyers_org (org_id),
    CONSTRAINT fk_flyers_org
      FOREIGN KEY (org_id)
      REFERENCES org (id)
      ON DELETE CASCADE
);


CREATE TABLE comments (
    id      BIGINT   NOT NULL AUTO_INCREMENT,
    user_id  BIGINT   NOT NULL,
    flyer_id BIGINT   NOT NULL,
    content  VARCHAR(500) NOT NULL,
    timestamp TIMESTAMP   NOT NULL,
```

```
    PRIMARY KEY (id),
    KEY fk_comments_user (user_id),
    KEY fk_comments_flyer (flyer_id),
    CONSTRAINT fk_comments_user
      FOREIGN KEY (user_id)
      REFERENCES users (id)
      ON DELETE CASCADE,
    CONSTRAINT fk_comments_flyer
      FOREIGN KEY (flyer_id)
      REFERENCES flyers (id)
      ON DELETE CASCADE
);

CREATE TABLE saved_flyers (
    id       BIGINT NOT NULL AUTO_INCREMENT,
    user_id  BIGINT,
    flyer_id BIGINT,
    PRIMARY KEY (id),
    KEY fk_saved_user (user_id),
    KEY fk_saved_flyer (flyer_id),
    CONSTRAINT fk_saved_user
      FOREIGN KEY (user_id)
      REFERENCES users (id)
      ON DELETE CASCADE,
    CONSTRAINT fk_saved_flyer
      FOREIGN KEY (flyer_id)
      REFERENCES flyers (id)
      ON DELETE CASCADE
);

CREATE TABLE user_feed (
```

```
  id      BIGINT NOT NULL AUTO_INCREMENT,
  user_id BIGINT NOT NULL,
  org_id  BIGINT NOT NULL,
  PRIMARY KEY (id),
  KEY fk_feed_user (user_id),
  KEY fk_feed_org (org_id),
  CONSTRAINT fk_feed_user
    FOREIGN KEY (user_id)
    REFERENCES users (id)
    ON DELETE CASCADE,
  CONSTRAINT fk_feed_org
    FOREIGN KEY (org_id)
    REFERENCES org (id)
    ON DELETE CASCADE
);
```

2.4 Global Control Flow

The frontend is event driven, nothing happens without the user's input, it then asks the backend's endpoints to perform a task and then waits for a response that it's completed. The backend can complete multiple tasks concurrently using springboot's embedded web server tomcat that has a thread pool waiting for http requests.


Eventure - College Club Discovery Platform
Frontend
UIManager
+showOrganizations()
+showEvents()
+showFlyers()
+renderFeed()
+handleNavigation()
ApiClient
-baseUrl: String
+get()
+post()
+put()
+delete()
+handleResponse()
StorageManager
+saveToLocal()
+getFromLocal()
+clearStorage()
+syncWithServer()
DataManager
-organizations: Array
-events: Array
-flyers: Array
+loadData()
+updateData()
+filterData()
Controllers
UserController
-userService: UserService
+login()
+logout()
+register()
+getSession()
OrganizationController
-orgService: OrganizationService
+getAllOrgs()
+createOrg()
+updateOrg()
+deleteOrg()
EventController
-eventService: EventService
+getAllEvents()
+createEvent()
+updateEvent()
+deleteEvent()
FlyerController
-flyerService: FlyerService
+getAllFlyers()
+createFlyer()
+updateFlyer()
+deleteFlyer()
Backend Models
User
-id: Long
-username: String
-email: String
-password: String
+login()
+logout()
+register()
Organization
-id: Long
-name: String
-description: String
-category: String
-members: Integer
-icon: String
+create()
+update()
+delete()
+getMembers()
composition
composition
Event
-id: Long
-title: String
-description: String
-dateTime: LocalDateTime
-location: String
-organizationId: Long
+create()
+update()
+delete()
+getByOrg()
Flyer
-id: Long
-title: String
-content: String
-imageUrl: String
-createdDate: LocalDate
-organizationId: Long
+create()
+update()
+delete()
+getAll()
Services
UserService
-userRepository: UserRepository
+authenticate()
+createUser()
+findByUsername()
+validateSession()
OrganizationService
-orgRepository: OrganizationRepository
+findAll()
+save()
+update()
+delete()
EventService
-eventRepository: EventRepository
+findAll()
+save()
+update()
+delete()
FlyerService
-flyerRepository: FlyerRepository
+findAll()
+save()
+update()
+delete()
Repositories
UserRepository
+findByUsername()
+save()
+findById()
OrganizationRepository
+findAll()
+save()
+deleteById()
EventRepository
+findAll()
+save()
+deleteById()
+findByOrgId()
FlyerRepository
+findAll()
+save()
+deleteById()
H2Database
-url: jdbc:h2:file
-persistent: true
+connect()
+execute()
+close()