22/23

SOUTH EAST EUROPEAN UNIVERSITY



“EventFinder”

Students: Isra Zeqiri, Agnesa Kriezi, Zana Kriezi

* **Project overview**

EventFinder project, is a project where the users have the ability to both post events and discover events posted by other users.

The event posting feature allows users to create and publish events on the platform, providing details such as date, time, venue, description, and other related information. This feature provides a platform for event organizers and creators to promote their events and reach a wider audience.

On the other hand, the event discovery feature provides users with a user-friendly interface to search and discover events happening in a specific location or date range. They can filter the events based on their preferences and view the details of the events they are interested in attending.

This combination of features creates a comprehensive event discovery and management platform that benefits both event organizers and attendees.

* **System Architecture:**

The system consists two main classes: Event and Users. The classes **Event** and **User** represent entities in a database and are annotated with JPA annotations to define their relationship and mapping to the database tables. The **Event** class has a many-to-one relationship with the **User** class, which means that each **Event** belongs to a single **User**, but each **User** can have multiple **Events**.

The **Event** class contains information about an event, such as the name, category, location, date and time, latitude, longitude, and price. The **User** class contains information about a user, such as the username, email, and password.

The system architecture uses a layered architecture, where the classes represent the persistence layer and the database tables represent the data storage layer. The persistence layer communicates with the data storage layer through Hibernate, which handles all the database operations such as creating, reading, updating, and deleting records.

In summary, the system architecture is a Java-based JPA architecture with a relational database, using Hibernate as the ORM framework to map the Java objects to the database tables and handle database operations.

* **Dependencies**

1. Spring Boot Starter: This is a set of useful starter dependencies for building Spring Boot applications, such as spring-boot- starter-web for building web applications and spring-boot- starter-data-jpa for working with JPA repositories.
2. Spring Web: This provides support for building web applications, including RESTful web services.
3. Spring Test: This provides support for testing the application, including JUnit and Mockito.
4. The MySQL driver: This is used to connect to a MySQL database from a Spring application. The dependency includes the MySQL Connector/J library, which provides a Java driver for communicating with a MySQL server.

# *DDL script/s

DDL (Data Definition Language) scripts are used to define the structure of a database. They are used to create, alter, and drop database objects such as tables, indexes, and constraints. In this project there are present 2 DDL scripts: Event and Users.

**Event**

CREATE TABLE `event` (

**`id`** bigint(20) NOT NULLAUTO\_INCREMENT,

**`category**` varchar(255) DEFAULT NULL,

**`date\_time`** datetime DEFAULT NULL,

**`latitude**` double DEFAULT NULL,

 `**location`** varchar(255) DEFAULT NULL,

**`longitude`** double DEFAULT NULL,

**`name`** varchar(255) DEFAULT NULL,

**`price`** double DEFAULT NULL,

**`user\_id`** bigint(20) DEFAULT NULL,

**PRIMARY KEY** (`id`),  KEY `FKi8bsvlthqr8lngsyshiqsodak` (`user\_id`),

**CONSTRAINT** `FKi8bsvlthqr8lngsyshiqsodak` FOREIGN KEY (`user\_id`) REFERENCES `user` (`user\_id`) ) ENGINE=InnoDB AUTO\_INCREMENT=2 DEFAULT CHARSET=utf8mb4

**Users**

CREATE TABLE `user` (

**`user\_id`** bigint(20) NOT NULL AUTO\_INCREMENT,

**`email`** varchar(255) DEFAULT NULL,

**`password`** varchar(255) DEFAULT NULL,

**`username`** varchar(255) DEFAULT NULL,

**PRIMARY KEY** (`user\_id`) ) ENGINE=InnoDB DEFAULT CHARSET=utf8

# *Rest endpoints

The project includes endpoints for each class, including the Event and User

### The following endpoints are provided for the EVENT class in the code:

GET http://localhost:8080/events/getAllEvents

POST http://localhost:8080/events/createEvent

DELETE http://localhost:8080/events/deleteEvent/{{id}}

GET http://localhost:8080/events/findByLocation/{{location}}

GET http://localhost:8080/events/findEventsByDateRange/daterange

GET http://localhost:8080/events/findByCategory/{{category}}

### The following endpoints are provided for the USER class in the code:

GET http://localhost:8080/users/getAllUsers

POST http://localhost:8080/users/createUser

PUT http://localhost:8080/users/updateUser/{{id}}

DELETE http://localhost:8080/users/deleteUser/{{id}}

GET http://localhost:8080/users/findByUsername/{{username}}

GET http://localhost:8080/users/findEventsByUser/{{username}}/events

GET http://localhost:8080/users/findEventsNearLocation/events/near

# *Results and Evaluation

## Testing

## These are the test methods that we did

* void testEvent ()
* void testUser()
* Code snippets for testing

|  |  |
| --- | --- |
| @Test  public void testEvent() {  User user = new User(1L, "John Doe", "johndoe@example.com", "password", null);  Event event = new Event(1L, "Music Festival", "Music", "New York", LocalDateTime.now(), 40.71, 74.00, 100.00, user);  assertNotNull(event);  assertEquals(1L, event.getId().longValue());  assertEquals("Music Festival", event.getName());  assertEquals("Music", event.getCategory());  assertEquals("New York", event.getLocation());  assertEquals(100.00, event.getPrice(), 0);  assertEquals(user, event.getUser());  } | This test checks the functionality of the User class. The test creates a User object with specific values for its properties (userid, username, email, password, and events). Then, using JUnit's assertNotNull method, the test confirms that the object has been created successfully. The test also uses assertEquals methods to verify that the properties of the User object have the expected values. The test also confirms that the events property is null. If any of the assertions fail, the test will result in a failure. |

|  |  |
| --- | --- |
| @Test  public void testUser() {  User user = new User(1L, "John Doe", "johndoe@example.com", "password", null);  assertNotNull(user);  assertEquals(1L, user.getUserid().longValue());  assertEquals("John Doe", user.getUsername());  assertEquals("johndoe@example.com", user.getEmail());  assertEquals("password", user.getPassword());  assertNull(user.getEvents());  } | This test method is testing the creation and properties of a User object in a Java Spring Boot application. The code first creates a new User object with specified parameters. Then, it uses JUnit's assertNotNull method to ensure that the object is not null. The code then uses JUnit's assertEquals method to check if the object's properties (userid, username, email, password, and events) are equal to the expected values. The first assertEquals method checks if the userid property is equal to 1, the second checks if the username property is equal to "John Doe", the third checks if the email property is equal to "[johndoe@example.com](mailto:johndoe@example.com)", the fourth checks if the password property is equal to "password", and the fifth checks if the events property is equal to null.. |