# EventController class

The EventController class is a RESTful web service controller class in a Spring Boot application. It is responsible for handling HTTP requests related to the events and calling the appropriate methods in the EventService class.

The @RestController annotation indicates that this class is a RESTful web service controller. The @RequestMapping("/events") annotation sets the base URL for all the endpoints in this class to "/events".

The EventService class is autowired in the constructor and stored in a private final instance variable.

The getAllEvents() method handles the GET request to the "/events" endpoint and returns a list of all the events.

The createEvent() method handles the POST request to the "/events" endpoint and creates a new event by calling the createEvent() method in the EventService class and returning the created event.

The deleteEvent() method handles the DELETE request to the "/events/{id}" endpoint and deletes an event with the specified id by calling the deleteEvent() method in the EventService class.

The findByLocation() method handles the GET request to the "/location/{location}" endpoint and returns a list of events with the specified location by calling the findByLocation() method in the EventService class.

The findByCategory() method handles the GET request to the "/category/{category}" endpoint and returns a list of events with the specified category by calling the findByCategory() method in the EventService class.

The findByDateRange() method handles the GET request to the "/daterange" endpoint and returns a list of events within the specified date range by calling the findByDateRange() method in the EventService class.

# UserController class

This is the code for the **UserController** class in a Spring Boot REST API. The **UserController** class maps HTTP requests to specific methods that perform operations on the User entity. The class is annotated with **@RestController** and **@RequestMapping("/users")** which specify that it is a REST controller and all the mappings in this class will have the base URL of **/users**.

The class takes an instance of the **UserService** class as a constructor argument and uses this instance to perform operations on the User entity.

The following methods are defined in this class:

1. **getAllUsers** - This method returns a list of all users in the system. It maps to the GET request to **/users**.
2. **createUser** - This method creates a new user with the information provided in the request body. It maps to the POST request to **/users**.
3. **updateUser** - This method updates an existing user with the id provided in the path variable and the updated information in the request body. It maps to the PUT request to **/users/{id}**.
4. **deleteUser** - This method deletes a user with the id provided in the path variable. It maps to the DELETE request to **/users/{id}**.
5. **findByUsername** - This method returns a user with the username provided in the path variable. It maps to the GET request to **/users/{username}**.
6. **findEventsByUser** - This method returns all events created by the user with the username provided in the path variable. It maps to the GET request to **/users/{username}/events**.
7. **addEvent** - This method adds an event to the user with the username provided in the path variable. It maps to the POST request to **/users/{username}/events**.
8. **findEventsByCategory** - This method returns all events with the category provided in the path variable. It maps to the GET request to **/users/events/category/{category}**.
9. **findEventsByLocation** - This method returns all events at the location provided in the path variable. It maps to the GET request to **/users/events/location/{location}**.
10. **findEventsNearLocation** - This method returns all events near the user's location within the radius provided as request parameters. It maps to the GET request to **/users/events/near**.

# Event class (POJO)

This is a Java class that represents an **Event** object in a database. It is annotated with JPA (Java Persistence API) annotations to allow the class to be used for database mapping and persistence.

The class has fields for **id**, **name**, **category**, **location**, **dateTime**, **latitude**, **longitude**, and **price**, and each field has a corresponding column in the database table **event**. The class also has a field for a related **User** object and a relationship between **Event** and **User** is established using the **ManyToOne** and **JoinColumn** annotations.

The class has a constructor and getters and setters for each field, as well as an **toString** method for printing the class' fields.

# User class (POJO)

This is a Java class that defines the structure of a User entity in a Java Persistence API (JPA) application using Hibernate as the ORM (Object-Relational Mapping) framework.

The class is annotated with @Entity, which tells Hibernate that this class should be mapped to a database table with the name "user". The class has several fields, each annotated with @Column to indicate that it should be mapped to a column in the "user" table. The fields include "userid", "username", "email", and "password".

There is a one-to-many relationship between the User and Event entities, where a User can have many events. This is represented by the @OneToMany annotation on the "events" field and the "mappedBy" attribute that specifies the name of the field in the Event class that maps to the User.

The class also has a default constructor and another constructor that takes all fields as arguments. Additionally, it has getters and setters for all fields, as well as an overridden toString() method for debugging purposes.

# EventRepository

This is a JPA repository class for the **Event** entity. It extends the **JpaRepository** class and defines methods to query the database to retrieve **Event** objects.

It uses the **@Query** annotation to define a custom query to find events near a location based on the user's latitude and longitude and a given radius. The **findEventsNearLocation** method accepts the user's latitude and longitude as parameters and returns a list of **Event** objects that are within the specified radius.

It also defines several other methods to find **Event** objects based on location, category, date and time, and user. These methods use the methods provided by the **JpaRepository** class to perform the queries.

# UserRepository

This code defines a repository interface, **UserRepository**, that extends **JpaRepository** and is annotated with **@Repository**. The repository is responsible for handling the database interactions related to **User** entities.

It has two methods defined:

1. **findByUsername(String name)**: Finds a **User** entity with the given username.
2. **findById(Long id)**: Finds a **User** entity with the given id and returns the result as an **Optional** object.

This interface will provide basic CRUD (Create, Read, Update, Delete) operations as well as custom methods to interact with the database and retrieve the **User** entities.

# EventServiceImp class

This is the implementation of the EventService interface that provides the logic to manage events. It uses the EventRepository to perform CRUD operations on events and also provides methods to search events by location, category and date range.

It uses the **Autowired** annotation to automatically wire the EventRepository instance in the constructor and then uses this instance to perform all the necessary database operations. The methods in this class are straightforward and simply delegate the calls to the corresponding methods in the EventRepository.

# EventService interface

Great, this is the interface for the EventService which defines the methods that the service should provide. It defines the contract for the methods to get all events, create an event, delete an event, find events by location, find events by category, and find events by date range. This interface can be implemented by the concrete class EventServiceImp to provide the implementation for these methods.

# UserServiceImp class

This is the implementation of the UserService interface, which is used to perform CRUD (Create, Read, Update, Delete) operations on User objects. The implementation uses UserRepository and EventRepository to interact with the database to store and retrieve User and Event objects.

It defines the following methods:

* getAllUsers(): returns a list of all users in the database.
* createUser(User user): creates a new user in the database.
* getUserById(Long id): returns a User object for the given id, if it exists.
* updateUser(User updatedUser): updates an existing user in the database.
* deleteUser(Long id): deletes a user from the database for the given id.
* findByUsername(String username): returns a user for the given username, if it exists.
* addEvent(Event event, User user): adds an event to the database for a given user.
* findEventsByCategory(String category): returns a list of events for a given category.
* findEventsByLocation(String location): returns a list of events for a given location.
* findEventsByUser(User user): returns a list of events for a given user.
* findEventsNearLocation(double userLat, double userLng, double radius): returns a list of events near a given location.

# UserService interface

The UserService interface defines the behavior of the user service. The user service provides methods for performing CRUD operations on user objects and for finding events by various criteria such as location, category, and proximity to a given location. The methods in this interface are as follows:

1. getAllUsers(): Returns a list of all users
2. createUser(User user): Creates a new user
3. updateUser(User updatedUser): Updates an existing user
4. deleteUser(Long id): Deletes a user with a given id
5. findByUsername(String username): Finds a user by their username
6. findEventsByUser(User user): Finds events created by a given user
7. addEvent(Event event, User user): Adds an event to a user
8. findEventsByCategory(String category): Finds events by category
9. findEventsByLocation(String location): Finds events by location
10. findEventsNearLocation(double userLat, double userLng, double radius): Finds events near a given location.

# EventTest

This is a JUnit test for the **Event** class in your application. The test creates an instance of **User** and an instance of **Event**. It then uses the **assertEquals** method to check if the values of the properties of the **Event** instance are equal to the expected values. If all the assertions pass, then the test will pass. If any of the assertions fail, then the test will fail and the failing message will show what the expected value and the actual value are.

# UserTest

This is a JUnit test case for the User class in the com.example.events package. The test case tests the functionality of the User class by creating an instance of the class and asserting its attributes against expected values.

The test case defines a single test method named "testUser" which performs the following actions:

1. Creates an instance of the User class with the following parameters:
   * userid: 1L
   * username: "user"
   * email: "[user@example.com](mailto:user@example.com)"
   * password: "password"
   * events: null
2. Asserts the values of the User object's attributes against the expected values using the assertEquals method from the org.junit.Assert class:
   * The value of userid is asserted against 1L.
   * The value of username is asserted against "user".
   * The value of email is asserted against "[user@example.com](mailto:user@example.com)".
   * The value of password is asserted against "password".
   * The value of events is asserted against null using assertNull.

If all the assertions pass, the test case is considered to be successful. If any of the assertions fail, the test case is considered to be failed.