Tasks with low level complexity

Task 3

Build a RESTful API for a simple social media application using Spring Boot, Hibernate, and PostgreSQL. The application should allow users to create and view posts, follow other users, and like posts. Each post should have a title, body, and author. Use Hibernate to persist the post and user data in the database.

Steps taken:

1. ***Asked ChatGPT to build the REST API for me***:

You

Build a RESTful API for a simple social media application using Spring Boot, Hibernate, and PostgreSQL. The application should allow users to create and view posts, follow other users, and like posts. Each post should have a title, body, and author. Use Hibernate to persist the post and user data in the database.

ChatGPT

Sure, let's build a simple RESTful API for a social media application using Spring Boot, Hibernate, and PostgreSQL. Before you start, make sure you have Java, Maven, and PostgreSQL installed on your system.

Step 1: Set Up the Project

1. Create a new Spring Boot project using Spring Initializer: https://start.spring.io/
   - Project: Maven Project
   - Language: Java
   - Spring Boot: Latest stable version
   - Packaging: JAR
   - Dependencies: Spring Web, Spring Data JPA, PostgreSQL Driver
2. Click "Generate" and extract the downloaded ZIP file.

Step 2: Configure Database

In `src/main/resources/application.properties`, configure the PostgreSQL database connection:

properties Copy code

```
spring.datasource.url=jdbc:postgresql://localhost:5432/your-database
spring.datasource.username=your-username
spring.datasource.password=your-password
spring.jpa.hibernate.ddl-auto=update
```

Step 3: Create Entity Classes

Create the following entity classes for User and Post in the `src/main/java/com/example/socialmedia/model` package:

Java Copy code

```java
// User.java
@Entity
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String username;

    // Getter and Setter methods
}

// Post.java
@Entity
@Table(name = "posts")
public class Post {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;
    private String body;

    @ManyToOne
    @JoinColumn(name = "author_id")
    private User author;

    // Getter and Setter methods
}
```

Step 4: Create Repositories

Create repositories for User and Post:

java Copy code

```java
// UserRepository.java
public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByUsername(String username);
}

// PostRepository.java
public interface PostRepository extends JpaRepository<Post, Long> {
    List<Post> findByAuthorIdOrderByCreatedAtDesc(Long authorId);
}
```

Step 5: Create Services

Create services for User and Post:

java Copy code

```java
// UserService.java
public interface UserService {
    User createUser(User user);
    Optional<User> getUserById(Long id);
    Optional<User> getUserByUsername(String username);
}

// PostService.java
public interface PostService {
    List<Post> getPostsByAuthorId(Long authorId);
    Post createPost(Post post);
    Optional<Post> getPostById(Long id);
    List<Post> getAllPosts();
    void deletePost(Long postId);
}
```

Step 6: Implement Service and Controller Classes

Implement the services and controllers for User and Post. Create the necessary methods to handle user registration, post creation, and other functionalities.

Step 7: Test the API

Write tests to ensure that your API works as expected.

Step 8: Run the Application

Run your Spring Boot application, and your API should be accessible at `` `http://localhost:8080` ``.

This is a basic outline to get you started. Depending on your requirements, you may need to add more features, error handling, security, and other enhancements to make your social media application robust and secure.

2. As I was not fully satisfied with the answer began to ask more granular questions:

ChatGPT

Absolutely, let's modify the project setup to include Lombok as a dependency when generating the project on Spring Initializer.

1. Visit https://start.spring.io/.
2. Fill in the project details:
   - Project: Maven Project
   - Language: Java
   - Spring Boot: Latest stable version
   - Packaging: JAR
   - Dependencies: Spring Web, Spring Data JPA, PostgreSQL Driver, Lombok
3. Click "Generate" and download the ZIP file.
4. Extract the downloaded ZIP file to get your project.

Now, your project setup includes Lombok as a dependency, and you can use Lombok annotations in your code to reduce boilerplate code. For example, you can use `` `@Data` `` on your entity classes to automatically generate getters, setters, `` `equals()` ``, `` `hashCode()` ``, and `` `toString()` `` methods.

Here's an example of how you can modify the `User` and `Post` entities with Lombok:

java Copy code

```java
// User.java
@Entity
@Table(name = "users")
@Data
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String username;
}

// Post.java
@Entity
@Table(name = "posts")
@Data
public class Post {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;
    private String body;

    @ManyToOne
    @JoinColumn(name = "author_id")
    private User author;
}
```

With Lombok, you don't need to write explicit getters, setters, or `toString()` methods; Lombok takes care of generating them during compilation.

Remember to install the Lombok plugin for your IDE (e.g., IntelliJ or Eclipse) to enable better support for Lombok annotations.

M **You**

Is the @Data supposed to be used on Entity classes? I believe it will be conflictual with JPA specifications.

‹ 2/2 ›

ChatGPT

You're correct, and I appreciate your clarification. While using Lombok's `@Data` on entities is a common practice, it can lead to some issues in certain scenarios, particularly with JPA. The `@Data` annotation generates a `toString()` method that includes all fields, which may lead to infinite recursion when entities have bidirectional relationships.

For JPA entities, it's generally recommended to use more specific Lombok annotations to avoid potential issues. Here's a safer approach:

java Copy code

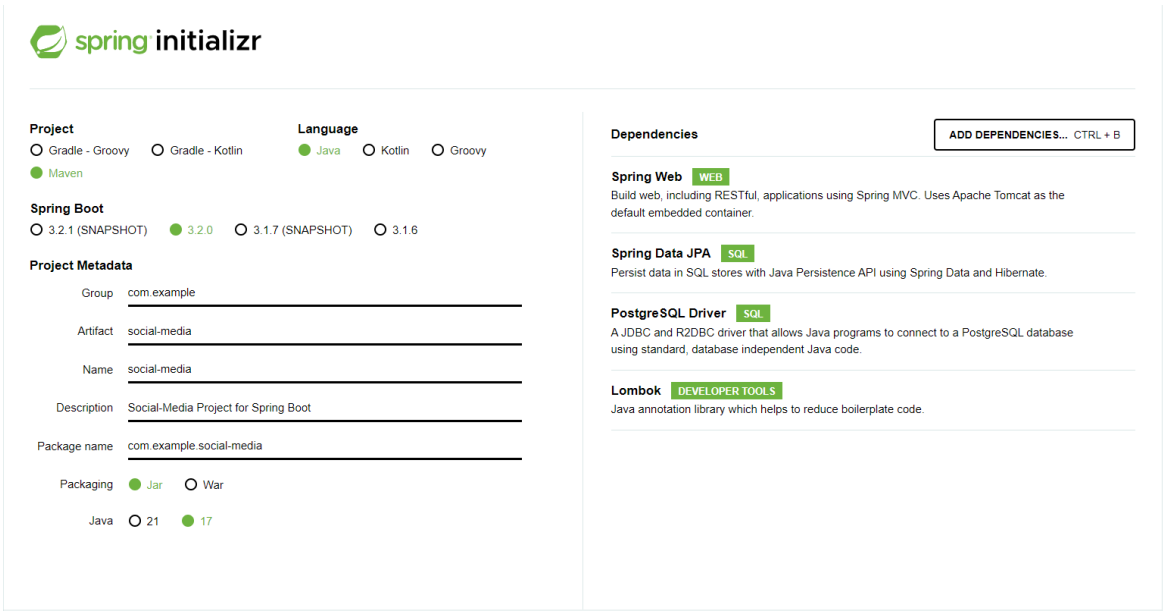
```java
// User.java
@Entity
@Table(name = "users")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String username;
}

// Post.java
@Entity
@Table(name = "posts")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class Post {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;
    private String body;

    @ManyToOne
    @JoinColumn(name = "author_id")
    private User author;
}
```

In this example:

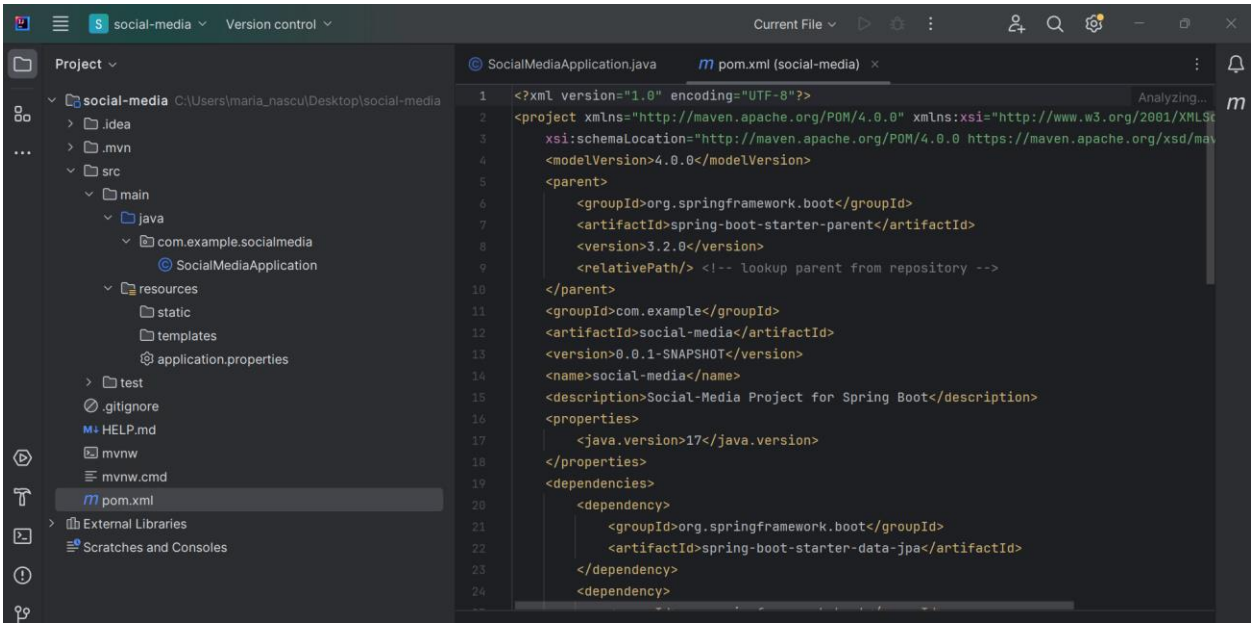
- `@Getter` and `@Setter` are used to generate getters and setters for the fields.
- `@NoArgsConstructor` generates a no-args constructor, which is required by JPA.
- `@AllArgsConstructor` generates an all-args constructor, which can be useful in certain scenarios.

Feel free to adjust the annotations based on your specific needs, and always test thoroughly to ensure that your entities work well with your chosen JPA provider (Hibernate, in this case).

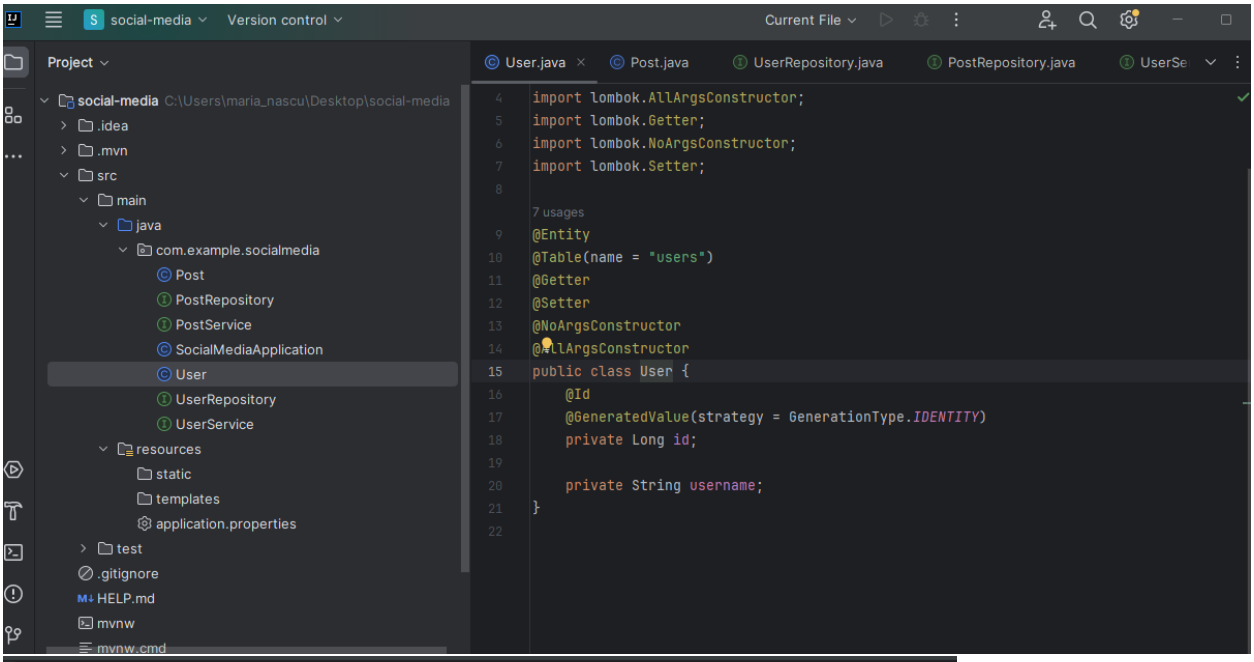
3. ***Being more satisfied with the answer started generating the structure of the project (with its dependencies) using Spring Initializer:***

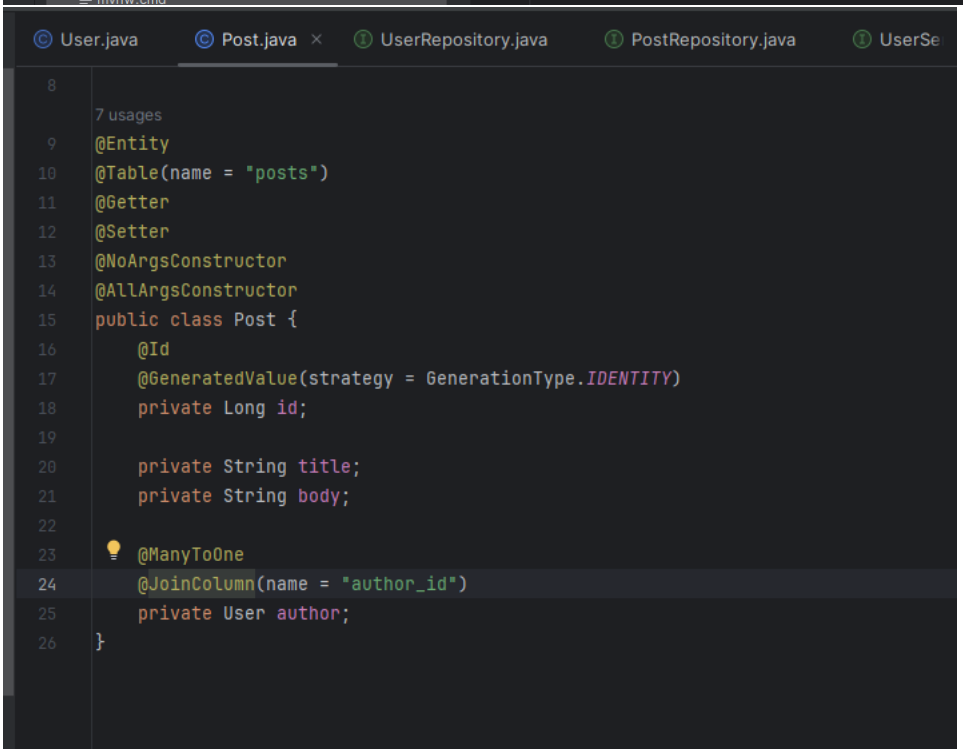


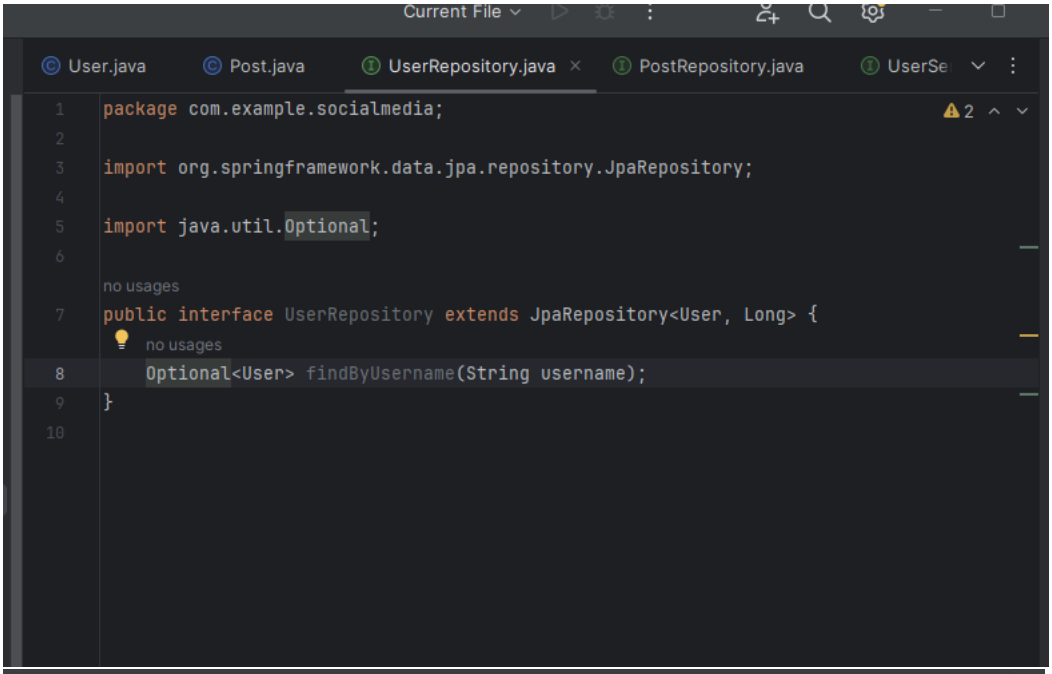
The application was generated, and I opened it up using my IDE:

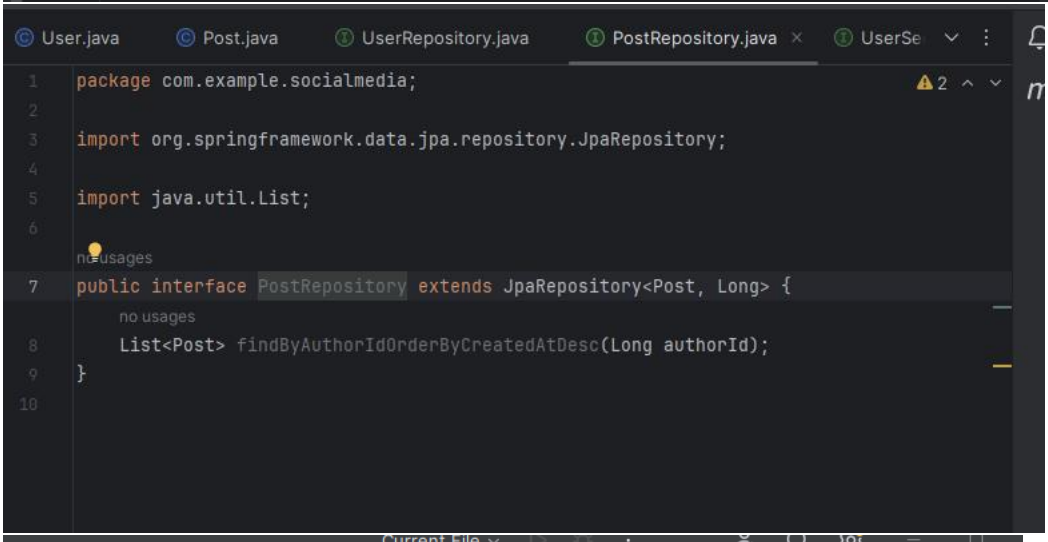


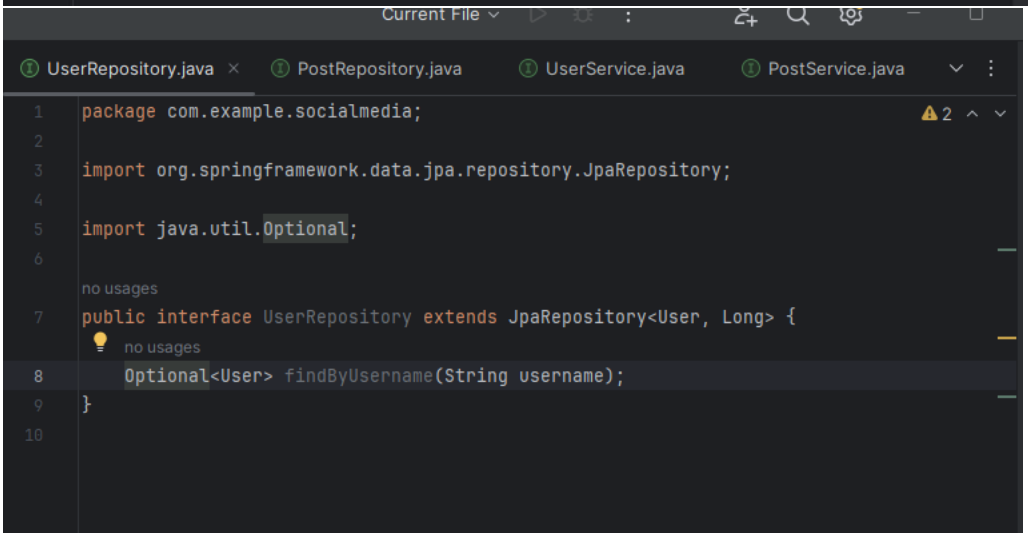
4. Started adding the ChatGPT generated code (solved out the imports too):

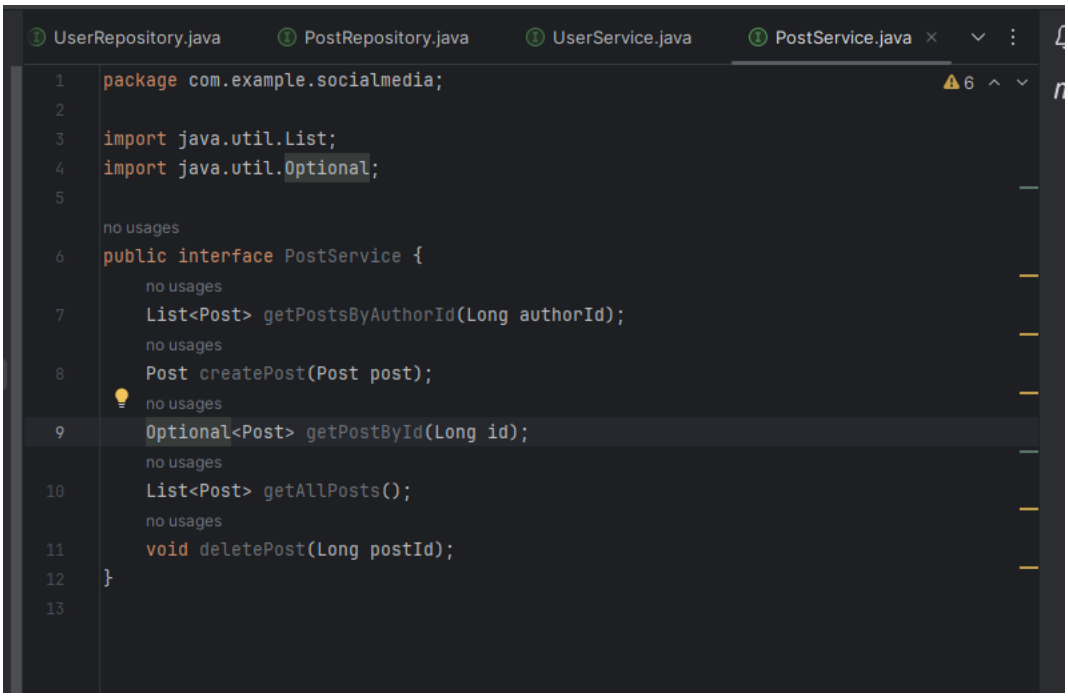

social-media
Version control
Current File
Project
social-media C:\Users\maria_nascu\Desktop\social-media
.idea
.mvn
src
main
java
com.example.socialmedia
Post
PostRepository
PostService
SocialMediaApplication
User
UserRepository
UserService
resources
static
templates
application.properties
test
.gitignore
HELP.md
mvnw
mvnw.cmd
User.java
Post.java
UserRepository.java
PostRepository.java
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
7 usages
@Entity
@Table(name = "users")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class User {
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;
private String username;
}


User.java
Post.java
UserRepository.java
PostRepository.java
7 usages
@Entity
@Table(name = "posts")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class Post {
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;
private String title;
private String body;
@ManyToOne
@JoinColumn(name = "author_id")
private User author;
}


Current File
User.java
Post.java
UserRepository.java
PostRepository.java
UserSe
package com.example.socialmedia;
2
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.Optional;
no usages
public interface UserRepository extends JpaRepository<User, Long> {
no usages
Optional<User> findByUsername(String username);
}


User.java
Post.java
UserRepository.java
PostRepository.java
UserSe
package com.example.socialmedia;
2
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.List;
no usages
public interface PostRepository extends JpaRepository<Post, Long> {
no usages
List<Post> findByAuthorIdOrderByCreatedAtDesc(Long authorId);
}


Current File
UserRepository.java
PostRepository.java
UserService.java
PostService.java
package com.example.socialmedia;
2
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.Optional;
no usages
public interface UserRepository extends JpaRepository<User, Long> {
no usages
Optional<User> findByUsername(String username);
}



5. Started giving granular instructions and implemented the code:

You

Please generate the code for the service implementation classes and for the controllers too.

ChatGPT

Certainly! Below are sample implementations for the service and controller classes. Note that these are simplified examples, and in a real-world application, you would need to add more features, error handling, and security measures.

Service Implementations:

`UserServiceImpl.java`

java — Copy code

```java
@Service
public class UserServiceImpl implements UserService {

    @Autowired
    private UserRepository userRepository;

    @Override
    public User createUser(User user) {
        return userRepository.save(user);
    }

    @Override
    public Optional<User> getUserById(Long id) {
        return userRepository.findById(id);
    }

    @Override
    public Optional<User> getUserByUsername(String username) {
        return userRepository.findByUsername(username);
    }
}
```

`PostServiceImpl.java`

```java
@Service
public class PostServiceImpl implements PostService {

    @Autowired
    private PostRepository postRepository;

    @Override
    public List<Post> getPostsByAuthorId(Long authorId) {
        return postRepository.findByAuthorIdOrderByCreatedAtDesc(author
    }

    @Override
    public Post createPost(Post post) {
        return postRepository.save(post);
    }

    @Override
    public Optional<Post> getPostById(Long id) {
        return postRepository.findById(id);
    }

    @Override
    public List<Post> getAllPosts() {
        return postRepository.findAll();
    }

    @Override
    public void deletePost(Long postId) {
        postRepository.deleteById(postId);
    }
}
```

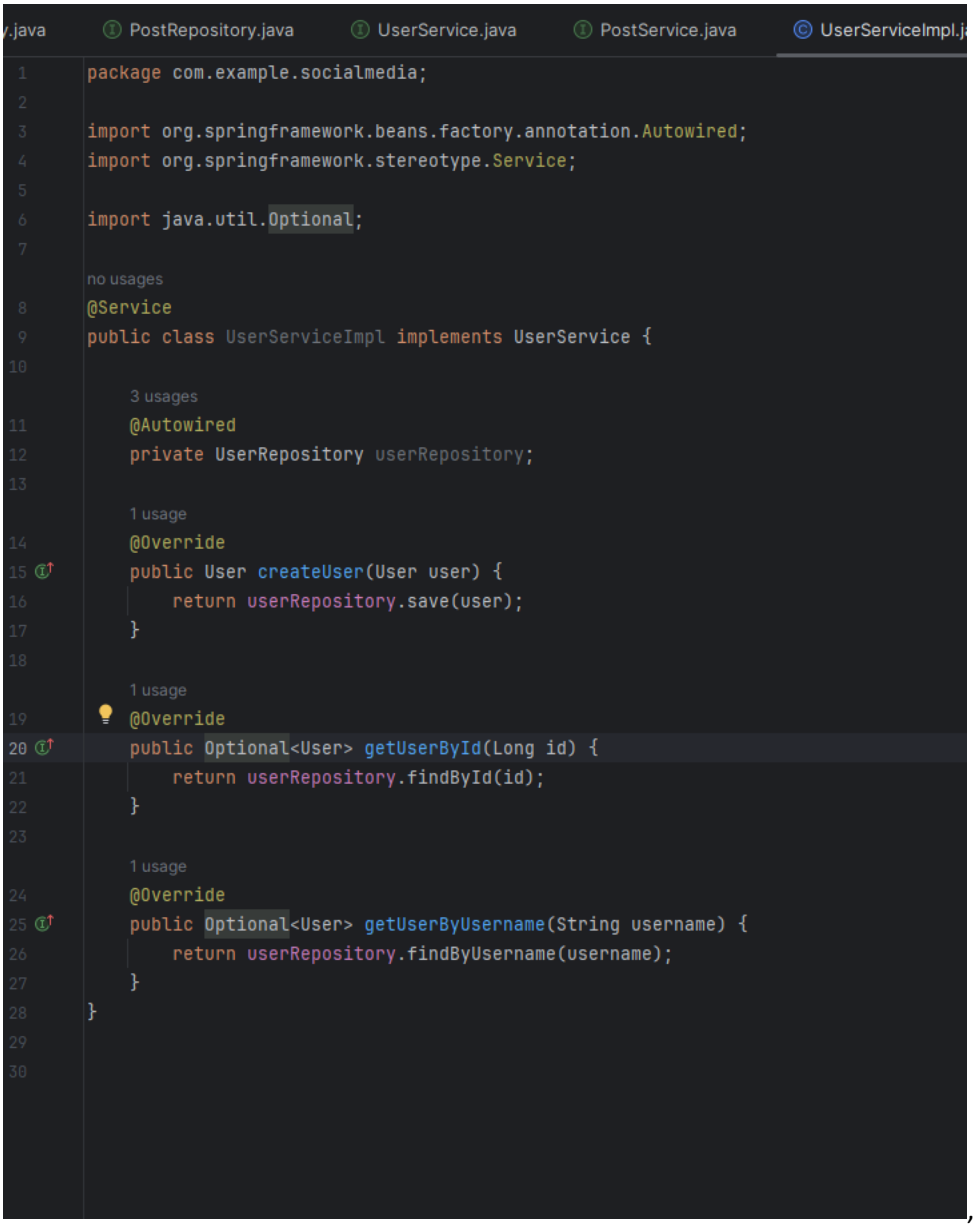
Controllers:

`UserController.java`

```java
@RestController
@RequestMapping("/api/users")
public class UserController {

    @Autowired
    private UserService userService;

    @PostMapping
    public ResponseEntity<User> createUser(@RequestBody User user) {
        User createdUser = userService.createUser(user);
        return new ResponseEntity<>(createdUser, HttpStatus.CREATED);
    }

    @GetMapping("/{userId}")
    public ResponseEntity<User> getUserById(@PathVariable Long userId)
        Optional<User> user = userService.getUserById(userId);
        return user.map(value -> new ResponseEntity<>(value, HttpStatus
                .orElseGet(() -> new ResponseEntity<>(HttpStatus.NOT
    }

    @GetMapping("/by-username/{username}")
    public ResponseEntity<User> getUserByUsername(@PathVariable String
        Optional<User> user = userService.getUserByUsername(username);
        return user.map(value -> new ResponseEntity<>(value, HttpStatus
                .orElseGet(() -> new ResponseEntity<>(HttpStatus.NOT
    }
}
```

`PostController.java`

```java
@RestController
@RequestMapping("/api/posts")
public class PostController {

    @Autowired
    private PostService postService;

    @PostMapping
    public ResponseEntity<Post> createPost(@RequestBody Post post) {
        Post createdPost = postService.createPost(post);
        return new ResponseEntity<>(createdPost, HttpStatus.CREATED);
    }

    @GetMapping("/{postId}")
    public ResponseEntity<Post> getPostById(@PathVariable Long postId)
        Optional<Post> post = postService.getPostById(postId);
        return post.map(value -> new ResponseEntity<>(value, HttpStatus
                .orElseGet(() -> new ResponseEntity<>(HttpStatus.NOT
    }

    @GetMapping("/all")
    public ResponseEntity<List<Post>> getAllPosts() {
        List<Post> posts = postService.getAllPosts();
        return new ResponseEntity<>(posts, HttpStatus.OK);
    }

    @GetMapping("/by-author/{authorId}")
    public ResponseEntity<List<Post>> getPostsByAuthorId(@PathVariable
        List<Post> posts = postService.getPostsByAuthorId(authorId);
        return new ResponseEntity<>(posts, HttpStatus.OK);
    }

    @DeleteMapping("/{postId}")
    public ResponseEntity<Void> deletePost(@PathVariable Long postId) {
        postService.deletePost(postId);
        return new ResponseEntity<>(HttpStatus.NO_CONTENT);
    }
}
```


.java
PostRepository.java
UserService.java
PostService.java
UserServiceImpl.j
package com.example.socialmedia;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.Optional;
no usages
@Service
public class UserServiceImpl implements UserService {
3 usages
@Autowired
private UserRepository userRepository;
1 usage
@Override
public User createUser(User user) {
return userRepository.save(user);
}
1 usage
@Override
public Optional<User> getUserById(Long id) {
return userRepository.findById(id);
}
1 usage
@Override
public Optional<User> getUserByUsername(String username) {
return userRepository.findByUsername(username);
}
}


ry.java
PostRepository.java
UserService.java
PostService.java
package com.example.socialmedia;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;
import java.util.Optional;
no usages
@Service
public class PostServiceImpl implements PostService {
5 usages
@Autowired
private PostRepository postRepository;
1 usage
@Override
public List<Post> getPostsByAuthorId(Long authorId) {
return postRepository.findByAuthorIdOrderByCreatedAtDesc(auth
}
1 usage
@Override
public Post createPost(Post post) {
return postRepository.save(post);
}
1 usage
@Override
public Optional<Post> getPostById(Long id) {
return postRepository.findById(id);
}
1 usage
@Override
public List<Post> getAllPosts() {
return postRepository.findAll();
}

```java
import java.util.Optional;

no usages
@RestController
@RequestMapping("/api/users")
public class UserController {

    3 usages
    @Autowired
    private UserService userService;

    no usages
    @PostMapping
    public ResponseEntity<User> createUser(@RequestBody User user) {
        User createdUser = userService.createUser(user);
        return new ResponseEntity<>(createdUser, HttpStatus.CREATED);
    }

    no usages
    @GetMapping("/{userId}")
    public ResponseEntity<User> getUserById(@PathVariable Long userId) {
        Optional<User> user = userService.getUserById(userId);
        return user.map(value -> new ResponseEntity<>(value, HttpStatus.OK))
                .orElseGet(() -> new ResponseEntity<>(HttpStatus.NOT_FOUND));
    }

    no usages
    @GetMapping("/by-username/{username}")
    public ResponseEntity<User> getUserByUsername(@PathVariable String username) {
        Optional<User> user = userService.getUserByUsername(username);
        return user.map(value -> new ResponseEntity<>(value, HttpStatus.OK))
                .orElseGet(() -> new ResponseEntity<>(HttpStatus.NOT_FOUND));
    }
}
```

6. Connected to the database and tried running the application, but the application failed to run because there was no property ‘createdAt’ in the Post entity, so I got the following error:
Caused by: java.lang.IllegalArgumentException: Failed to create query for method public abstract java.util.List com.example.socialmedia.PostRepository.findByAuthorIdOrderByCreatedAtDesc(java.lang.Long); No property 'createdAt' found for type 'Post'

   Asked ChatGPT to correct this error and then ran the code again and it was successful:

You

The code provided for the PostRepository seems to be incorrect as I get this error: "Cau
java.lang.IllegalArgumentException: Failed to create query for method public abstract j
com.example.socialmedia.PostRepository.findByAuthorIdOrderByCreatedAtDesc(java.
No property 'createdAt' found for type 'Post' ". Can uou update the code?

ChatGPT

Certainly, I apologize for the oversight. It seems there was a misunderstanding in the property name used in the query method. Let's update the `PostRepository` and the `Post` entity to use a proper property name. If your `Post` entity has a `createdAt` field, we can modify the repository accordingly.

Assuming your `Post` entity has a `createdAt` field, you can update the code as follows:

Update `Post` Entity:

java — Copy code

```java
@Entity
@Table(name = "posts")
@Data
public class Post {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;
    private String body;

    @ManyToOne
    @JoinColumn(name = "author_id")
    private User author;

    @Column(name = "created_at")  // Add this annotation to map the cre
    private LocalDateTime createdAt;  // Assuming createdAt is of type

    // Getter and Setter methods
}
```

Update `PostRepository`:

java — Copy code

```java
// PostRepository.java
public interface PostRepository extends JpaRepository<Post, Long> {
    List<Post> findByAuthorIdOrderByCreatedAtDesc(Long authorId);
}
```

By adding `@Column(name = "created_at")` to the `createdAt` field in the `Post` entity, you specify the column name in the database. The `findByAuthorId0rderByCreatedAtDesc` method in the repository should now work correctly.

Make sure that the field name in your `Post` entity matches the one used in the `@Column` annotation. Adjust the field type and name accordingly based on your actual entity structure.

You

Where is the LocalDateTime from? I need to add the import inside the Post class.

ChatGPT

You're correct; I apologize for the oversight. `LocalDateTime` is part of the `java.time` package, and you need to import it into your `Post` class. Here's the corrected `Post` class with the import statement:

ChatGPT
You're correct; I apologize for the oversight. `LocalDateTime` is part of the `java.time` package, and you need to import it into your `Post` class. Here's the corrected `Post` class with the import statement:

java Copy code

```java
import java.time.LocalDateTime;

@Entity
@Table(name = "posts")
@Data
public class Post {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;
    private String body;

    @ManyToOne
    @JoinColumn(name = "author_id")
    private User author;

    @Column(name = "created_at")  // Add this annotation to map the cre
    private LocalDateTime createdAt;  // Import this from java.time

    // Getter and Setter methods
}
```

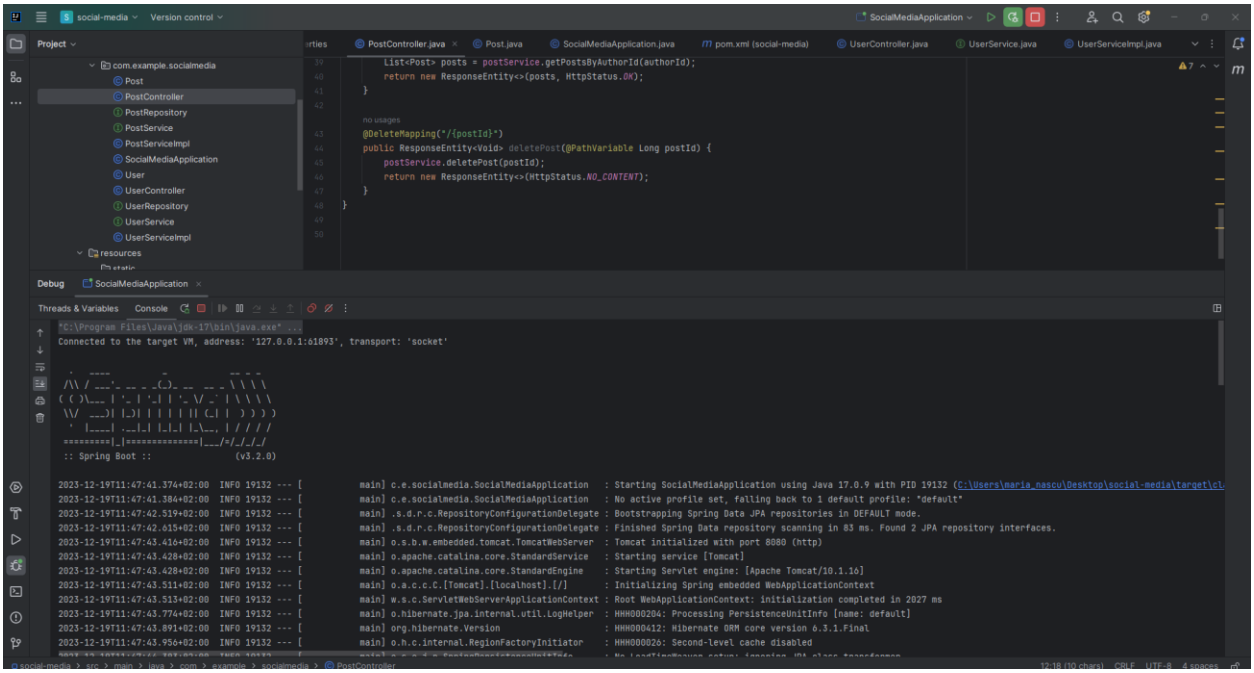
Don't forget to add the following import statement at the top of your `Post` class:

java Copy code

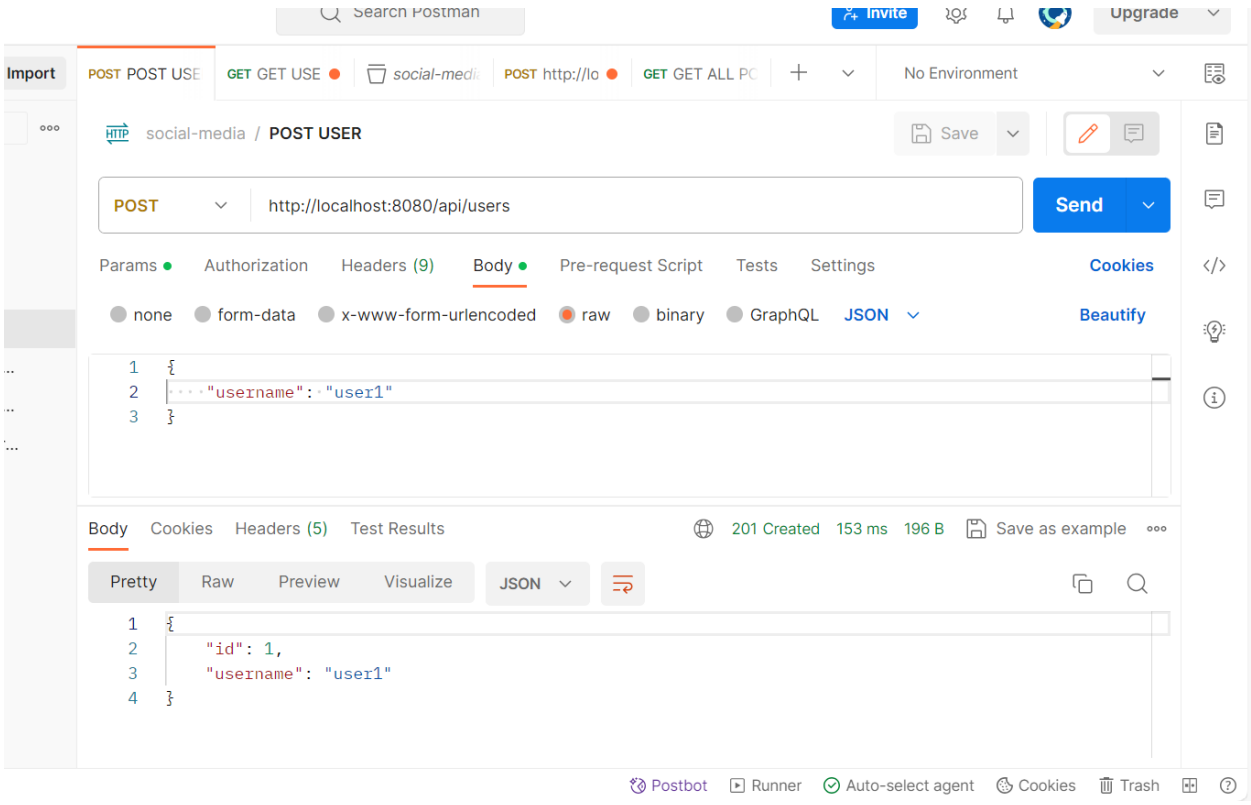
```java
import java.time.LocalDateTime;
```

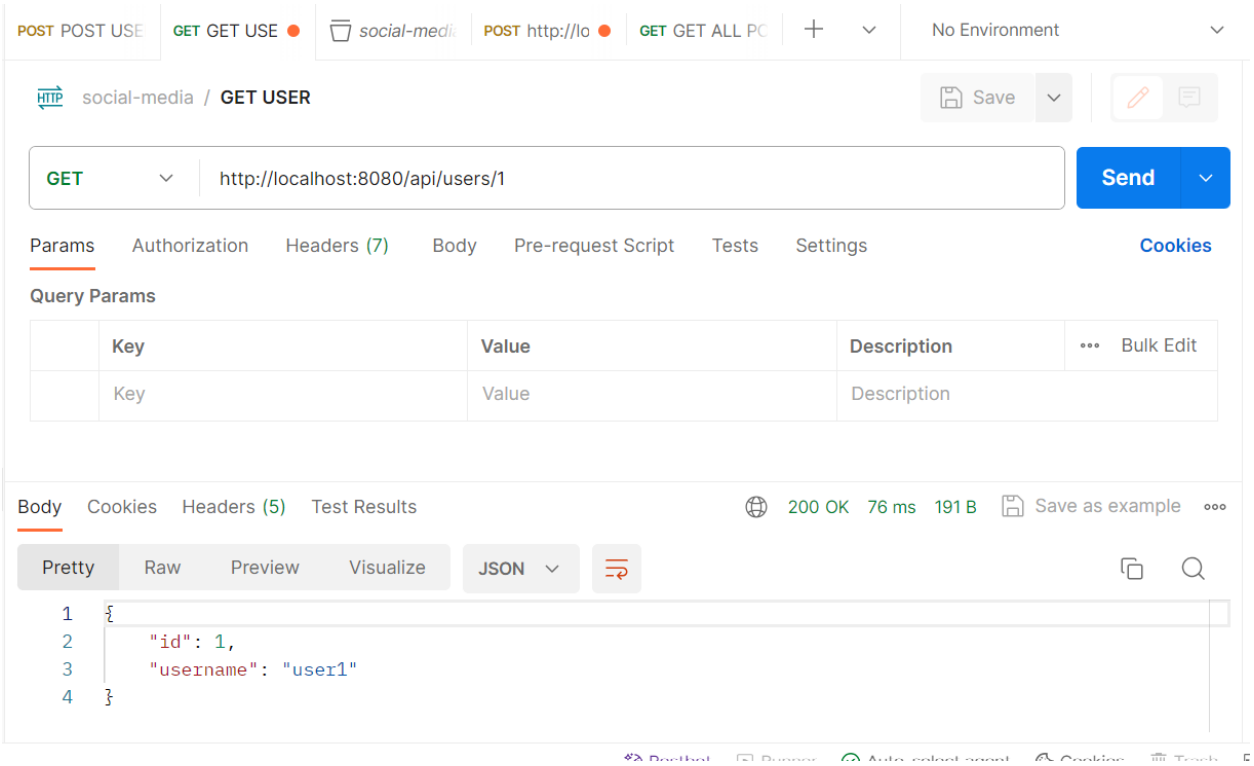
Now, your `Post` entity should work correctly with the `LocalDateTime` type for the `createdAt` field. Adjust the import statement based on your actual `java.time` package structure if necessary.

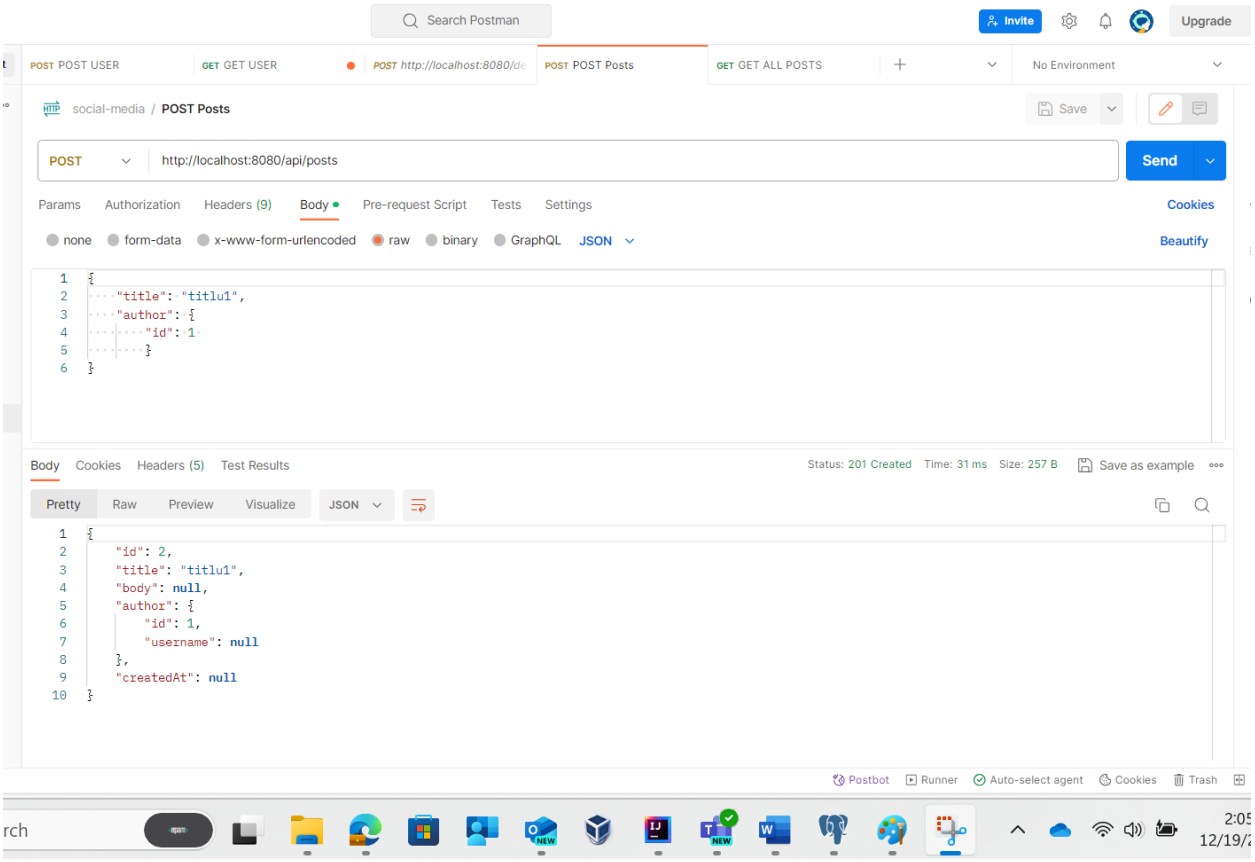
7. Ran the application and it started:

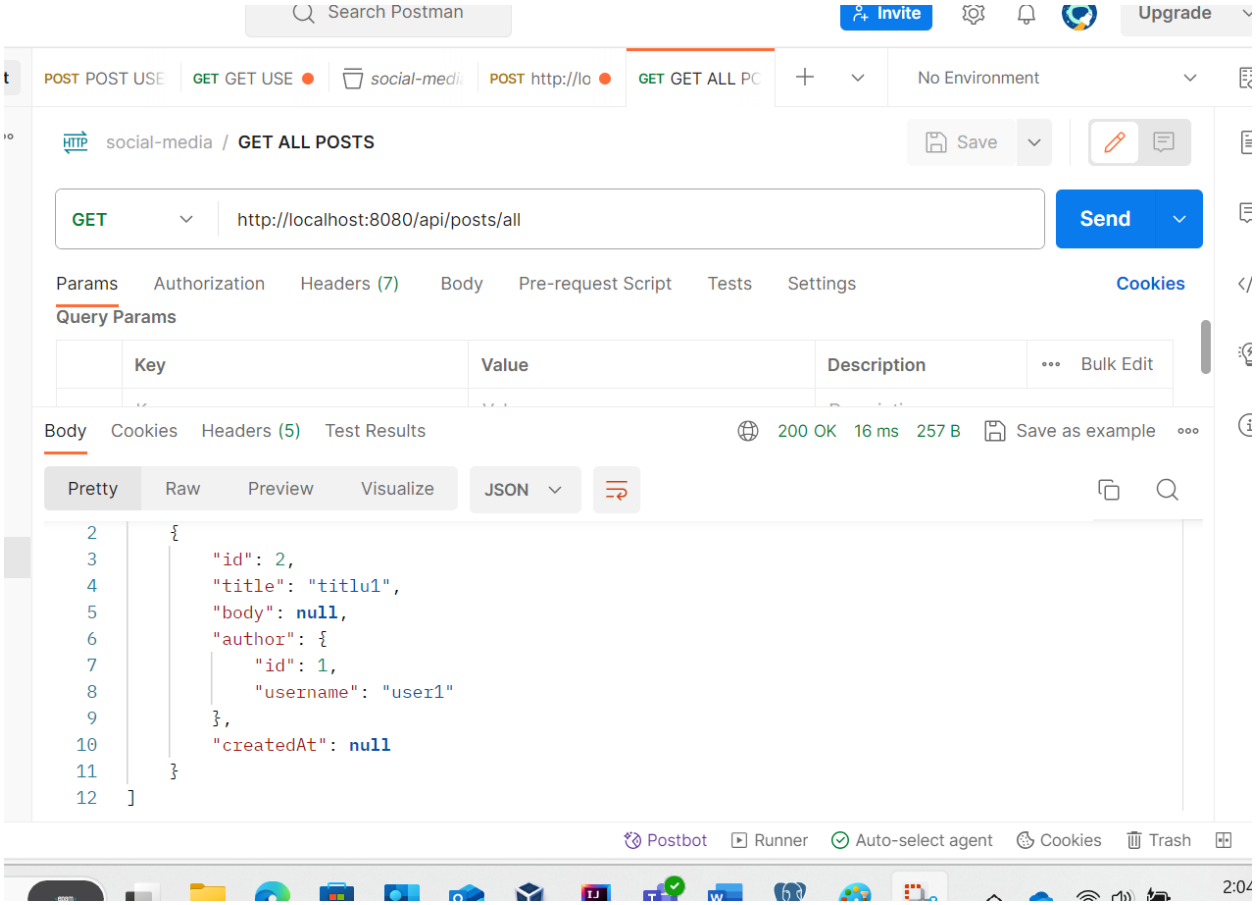


8. Tested some of the endpoints and they seem to work:

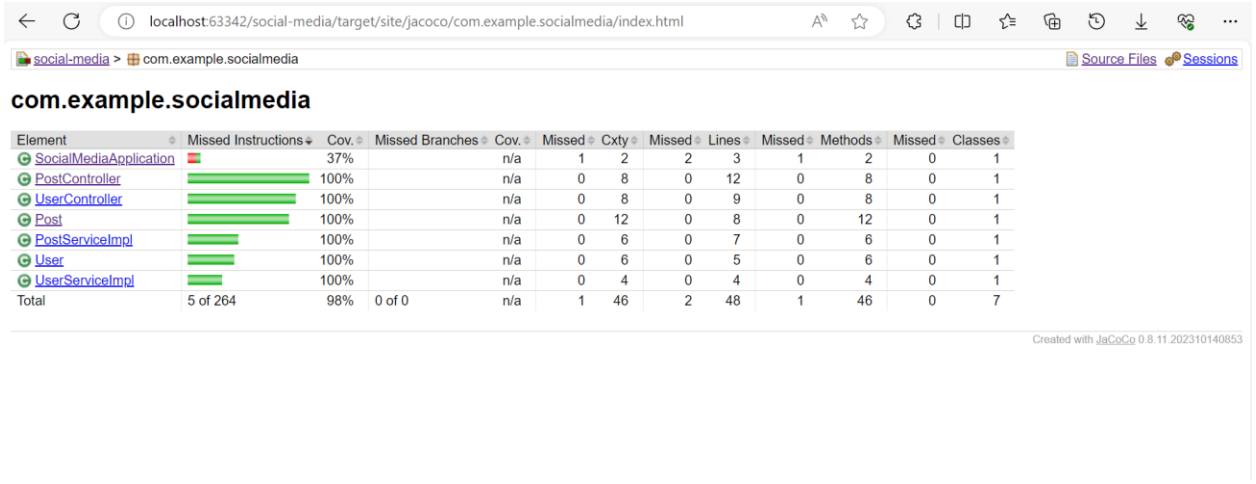



POST POST USE
GET GET USE
social-media
POST http://lo
GET GET ALL PC
No Environment
social-media / GET USER
Save
GET
http://localhost:8080/api/users/1
Send
Params
Authorization
Headers (7)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Key
Value
Description
Body
Cookies
Headers (5)
Test Results
200 OK
76 ms
191 B
Save as example
Pretty
Raw
Preview
Visualize
JSON
{
"id": 1,
"username": "user1"
}


Search Postman
Invite
Upgrade
POST POST USER
GET GET USER
POST http://localhost:8080/de
POST POST Posts
GET GET ALL POSTS
No Environment
social-media / POST Posts
Save
POST
http://localhost:8080/api/posts
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"title": "titlu1",
"author": {
"id": 1
}
}
Body
Cookies
Headers (5)
Test Results
Status: 201 Created
Time: 31 ms
Size: 257 B
Save as example
Pretty
Raw
Preview
Visualize
JSON
{
"id": 2,
"title": "titlu1",
"body": null,
"author": {
"id": 1,
"username": null
},
"createdAt": null
}
Postbot
Runner
Auto-select agent
Cookies
Trash



9. Asked ChatGPT to add tests to the project (unit, integration and end-to-end) – also added JaCoCo's dependency in the pom.xml. The code coverage shown by JaCoCo is 98%.



10. Generated the README.md using ChatGPT as well.

ChatGPT

Certainly! Below is a basic `README.md` file for your Spring Boot application. Feel free to customize it based on the specific details of your project.

markdown Copy code

```
# Social Media Application

## Description
This is a simple social media application built with Spring Boot, Hiber

## Requirements
- Java 8 or later
- Maven
- PostgreSQL database

## Setup

### Database Configuration
1. Create a PostgreSQL database for the application.
2. Update the database configuration in `src/main/resources/application

   ```properties
   spring.datasource.url=jdbc:postgresql://your-database-host:your-data
   spring.datasource.username=your-database-username
   spring.datasource.password=your-database-password
```

Running the Application

Using Maven

1. Open a terminal and navigate to the project root directory.
2. Run the following command to build the project:

bash Copy code

```
mvn clean install
```

3. Run the application:

bash Copy code

```
mvn spring-boot:run
```

4. The application will be accessible at `http://localhost:8080`.

Using IDE

1. Import the project into your favorite IDE (Eclipse, IntelliJ, etc.).
2. Run the `SocialMediaApplication` class as a Java application.

API Endpoints

- **Users:**
  - Create User: `POST /api/users`
  - Get User by ID: `GET /api/users/{userId}`
  - Get User by Username: `GET /api/users/by-username/{username}`
- **Posts:**
  - Create Post: `POST /api/posts`
  - Get Post by ID: `GET /api/posts/{postId}`
  - Get Posts by Author ID: `GET /api/posts/by-author/{authorId}`
  - Get All Posts: `GET /api/posts/all`
  - Delete Post: `DELETE /api/posts/{postId}`
```

Example Requests

- Create User:

```bash
curl -X POST -H "Content-Type: application/json" -d '{"username":"john
```

- Create Post:

```bash
curl -X POST -H "Content-Type: application/json" -d '{"title":"New Pos
```

- Get User by ID:

```bash
curl http://localhost:8080/api/users/1
```

- Get Post by ID:

```bash
curl http://localhost:8080/api/posts/1
```

Contributing

Feel free to contribute to the development of this application by opening issues and pull requests.

License

This project is licensed under the MIT License.

```sql
Replace the placeholder values in the database configuration section an
```

You

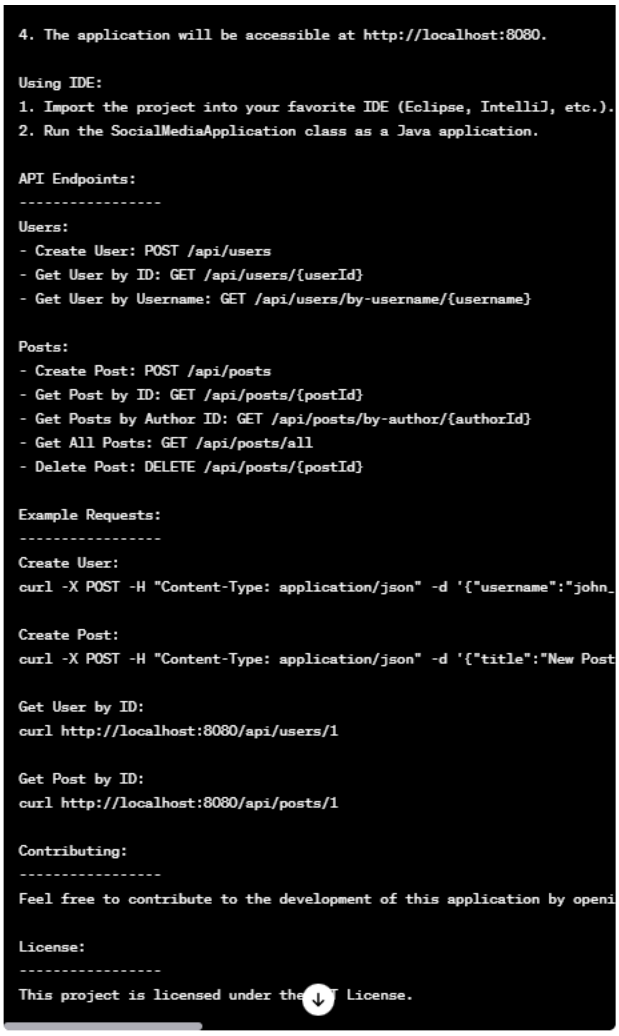
Can you make it in a text format will all the necessary data?

ChatGPT

Certainly! Here's the `README.txt` file with the necessary information:

plaintext — Copy code

```
Social Media Application

Description:
-----------------
This is a simple social media application built with Spring Boot, Hiber

Requirements:
-----------------
- Java 8 or later
- Maven
- PostgreSQL database

Setup:
-----------------
Database Configuration:
1. Create a PostgreSQL database for the application.
2. Update the database configuration in src/main/resources/application.

   spring.datasource.url=jdbc:postgresql://your-database-host:your-data
   spring.datasource.username=your-database-username
   spring.datasource.password=your-database-password

Running the Application:

Using Maven:
1. Open a terminal and navigate to the project root directory.
2. Run the following command to build the project:

   mvn clean install

3. Run the application:

   mvn spring-boot:run

4. The application will be accessible at http://localhost:8080
```



```
4. The application will be accessible at http://localhost:8080.

Using IDE:
1. Import the project into your favorite IDE (Eclipse, IntelliJ, etc.).
2. Run the SocialMediaApplication class as a Java application.

API Endpoints:
----------------
Users:
- Create User: POST /api/users
- Get User by ID: GET /api/users/{userId}
- Get User by Username: GET /api/users/by-username/{username}

Posts:
- Create Post: POST /api/posts
- Get Post by ID: GET /api/posts/{postId}
- Get Posts by Author ID: GET /api/posts/by-author/{authorId}
- Get All Posts: GET /api/posts/all
- Delete Post: DELETE /api/posts/{postId}

Example Requests:
----------------
Create User:
curl -X POST -H "Content-Type: application/json" -d '{"username":"john_

Create Post:
curl -X POST -H "Content-Type: application/json" -d '{"title":"New Post

Get User by ID:
curl http://localhost:8080/api/users/1

Get Post by ID:
curl http://localhost:8080/api/posts/1

Contributing:
----------------
Feel free to contribute to the development of this application by openi

License:
----------------
This project is licensed under the T License.
```

11. Here's the full conversation log with ChatGPT: https://chat.openai.com/share/9c6d2f51-b501-4f69-93ff-eb74f67762eb