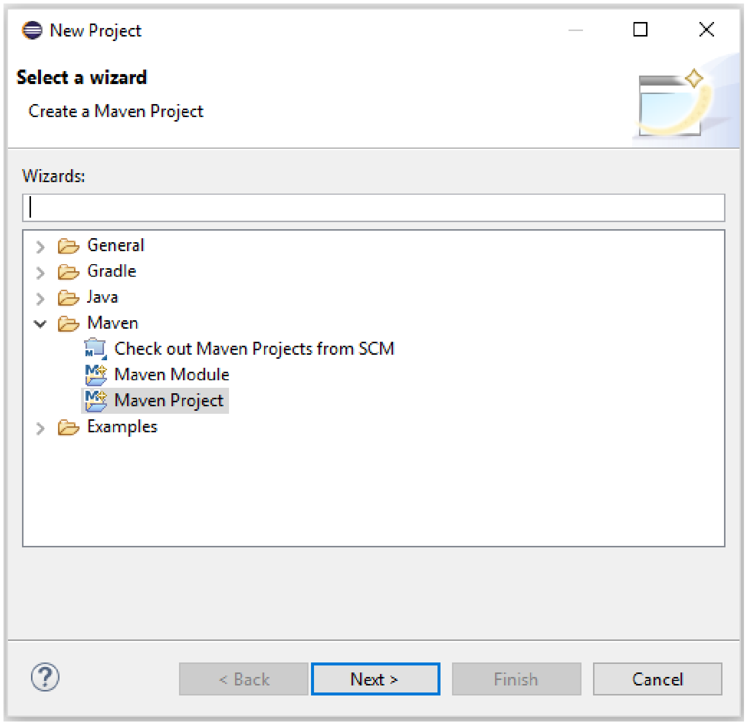
Social Media API Tutorial

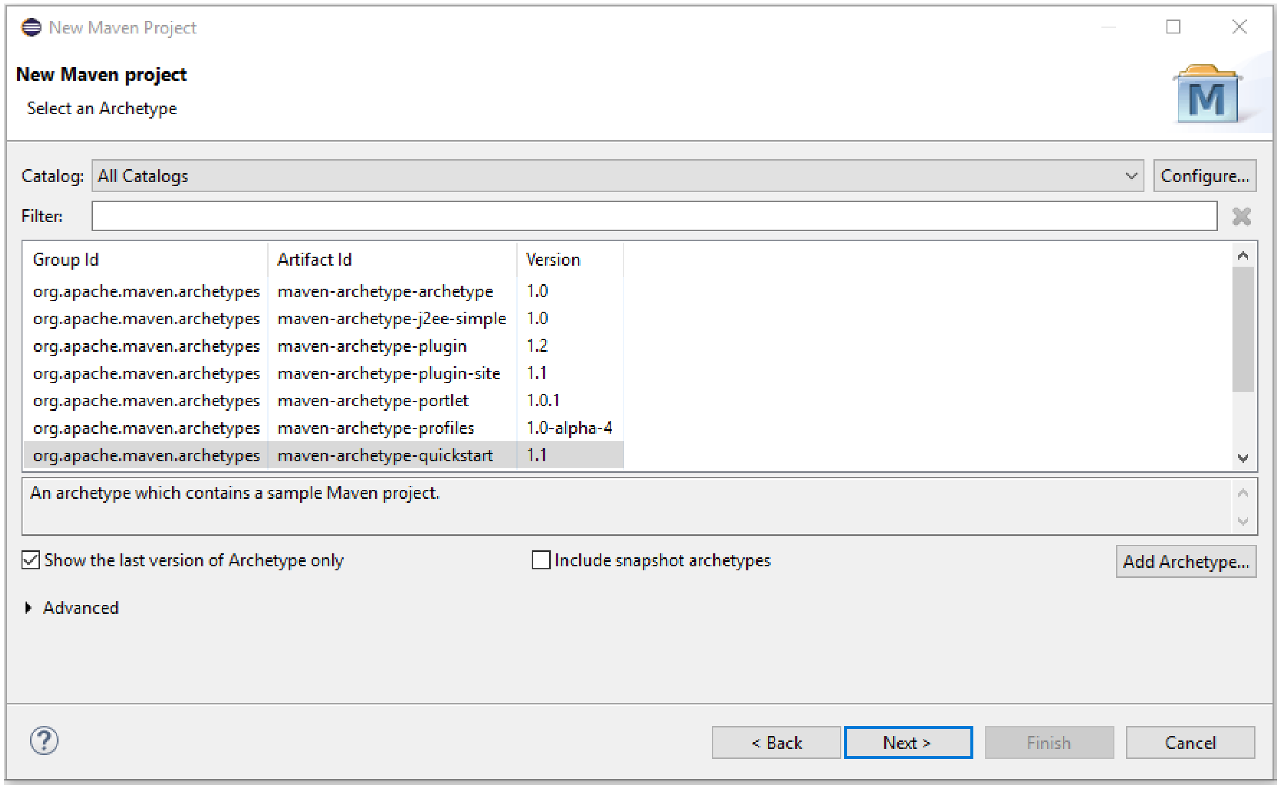
Select **File -> New -> Project...**

Select **Maven Project** under the Maven folder and click Next.



Leave the selected default of "Use default Workspace location" and click Next.

Leave the selected default Archetype and click Next.



New Maven project

Select an Archetype

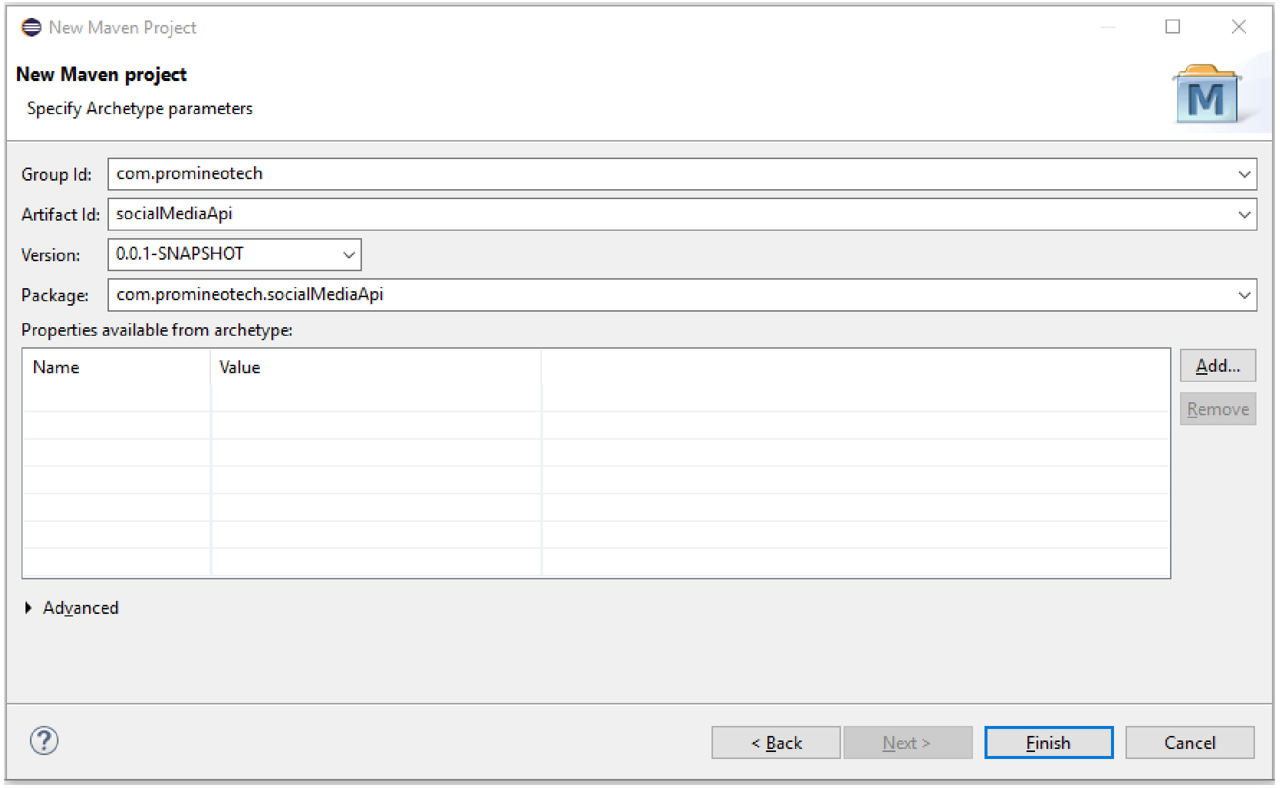
Catalog: All Catalogs

Group Id	Artifact Id	Version
org.apache.maven.archetypes	maven-archetype-archetype	1.0
org.apache.maven.archetypes	maven-archetype-j2ee-simple	1.0
org.apache.maven.archetypes	maven-archetype-plugin	1.2
org.apache.maven.archetypes	maven-archetype-plugin-site	1.1
org.apache.maven.archetypes	maven-archetype-portlet	1.0.1
org.apache.maven.archetypes	maven-archetype-profiles	1.0-alpha-4
org.apache.maven.archetypes	maven-archetype-quickstart	1.1

An archetype which contains a sample Maven project.

☑ Show the last version of Archetype only
☐ Include snapshot archetypes

Give your project a Group Id. The Group Id should follow conventions found here (essentially a domain name in reverse order) and will represent the base of all your projects. For example, our Group Id for this tutorial will be com.promineotech. Next, give your project an Artifact Id. The Artifact Id is the Id of the specific code base you are about to create, think of it like the project name. Our Artifact Id will be socialMediaApi.



New Maven project

Specify Archetype parameters

Group Id: com.promineotech

Artifact Id: socialMediaApi

Version: 0.0.1-SNAPSHOT

Package: com.promineotech.socialMediaApi

Properties available from archetype:

Name	Value

Notice that the Package name is derived from the Artifact Id concatenated to the end of the Group Id.

Click finish and your project will be created.

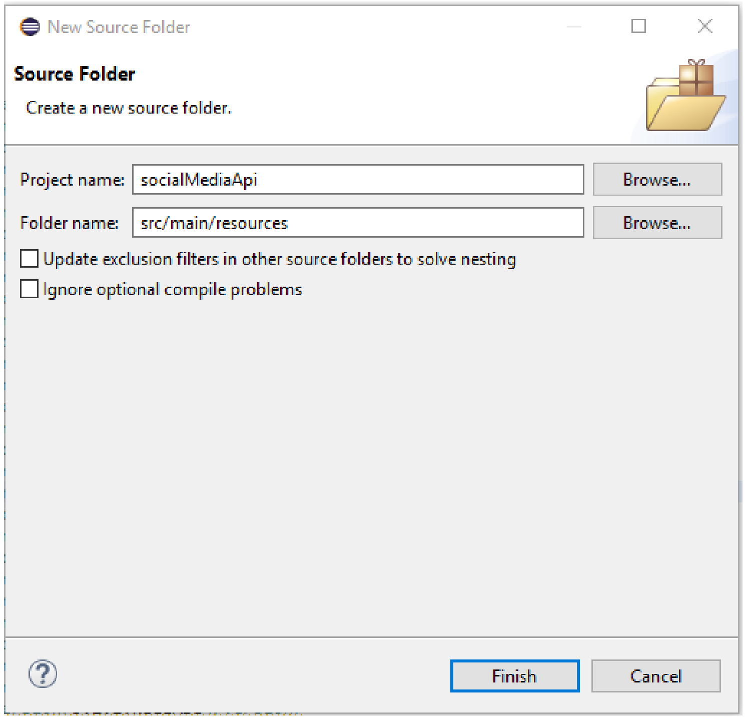
Open the pom.xml file and add the following parent element and dependencies. These dependencies will tell maven to pull in the external Java libraries we will use to create our API.

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.2.6.RELEASE</version>
</parent>


 <dependencies>
     <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-web</artifactId>
         <exclusions>
             <exclusion>
                 <groupId>org.springframework.boot</groupId>
                 <artifactId>spring-boot-starter-logging</artifactId>
             </exclusion>
         </exclusions>
     </dependency>
     <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-log4j2</artifactId>
     </dependency>
     <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-test</artifactId>
         <scope>test</scope>
     </dependency>
     <dependency>
         <groupId>mysql</groupId>
         <artifactId>mysql-connector-java</artifactId>
         <scope>runtime</scope>
     </dependency>
     <dependency>
         <groupId>com.jayway.jsonpath</groupId>
         <artifactId>json-path</artifactId>
         <scope>test</scope>
     </dependency>
     <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-data-jpa</artifactId>
     </dependency>
     <dependency>
         <groupId>org.springframework.security</groupId>
         <artifactId>spring-security-crypto</artifactId>
         <version>5.1.2.RELEASE</version>
     </dependency>
     <dependency>
         <groupId>junit</groupId>
         <artifactId>junit</artifactId>
         <version>3.8.1</version>
         <scope>test</scope>
     </dependency>
 </dependencies>
```

Note: When we run spring boot, if we see an error that says **Could not create connection to database server.** We may have a newer version of MySQL installed. To make our application work, we need to add a version element to the mysql dependency. For example:

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.13</version>
    <scope>runtime</scope>
</dependency>
```

Now that we have our dependencies set up, let's create a properties file that tells our API how to connect to the database it will be using. Right click on your project in the Package Explorer and select **New -> Source Folder**. Call the folder **src/main/resources** and click finish. You will see the newly created source folder appear under your project.


New Source Folder

Source Folder

Create a new source folder.

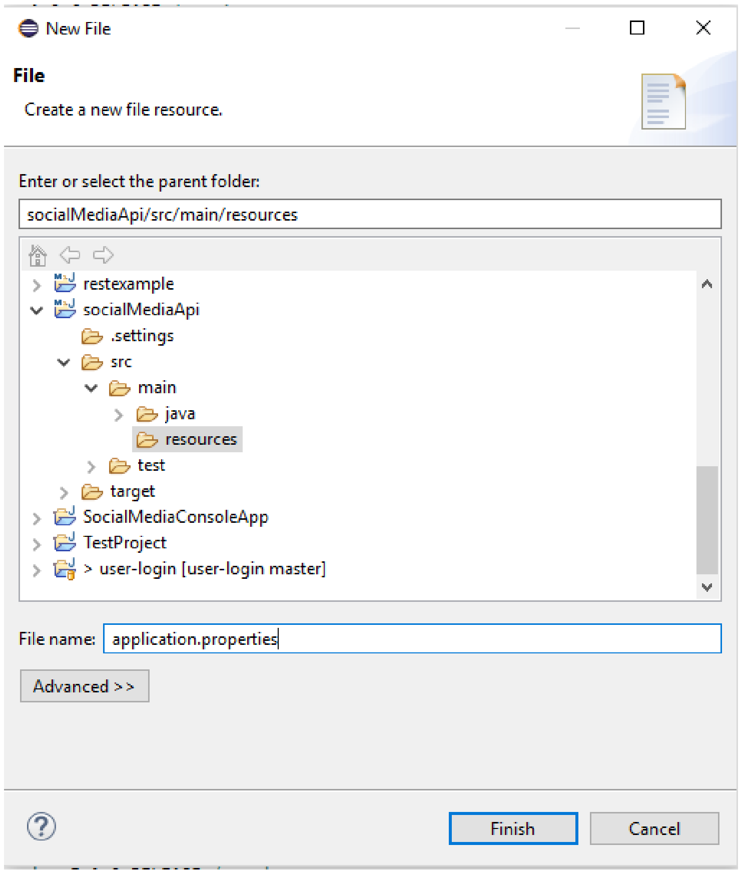
Project name: socialMediaApi

Folder name: src/main/resources

☐ Update exclusion filters in other source folders to solve nesting

☐ Ignore optional compile problems

Right click on the newly created resources folder and select **New -> File** and call the file **application.properties**.


New File

File

Create a new file resource.

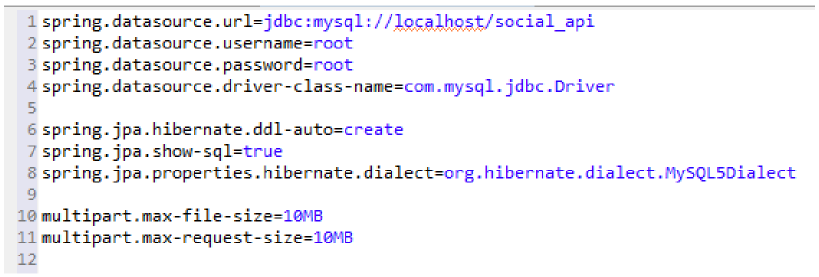
Enter or select the parent folder:

socialMediaApi/src/main/resources

```
> restexample
v socialMediaApi
    .settings
  v src
    v main
      > java
        resources
    > test
  > target
> SocialMediaConsoleApp
> TestProject
> > user-login [user-login master]
```

File name: application.properties

Inside application.properties, add the following lines.

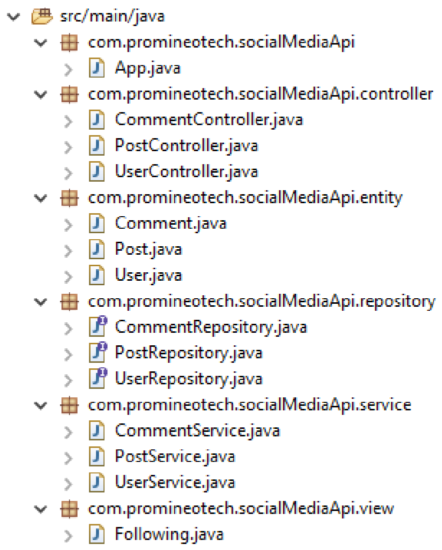


```
spring.datasource.url=jdbc:mysql://localhost/social_api
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.jdbc.Driver

spring.jpa.hibernate.ddl-auto=create
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect

multipart.max-file-size=10MB
multipart.max-request-size=10MB

```

The first four lines set up our connection to the database this API will be using. Note that you need to create the database in MySQL before this connection will work. To do that, log into your MySQL command line and run **create database social_api**. The next 3 lines tell our project how to interact with MySQL and what schema to use. We have spring.jpa.hibernate.ddl-auto set to create, this means that every time we start our application it will recreate the database schema, overwriting any data that is currently there. Once you finish writing all the code for your API you will want to change that value to update instead of create so that your data doesn't get blown away with each server bounce (restarting the server). The last two lines set size limits for file uploads; we include these lines in this project because uploading a profile picture will be part of the API.

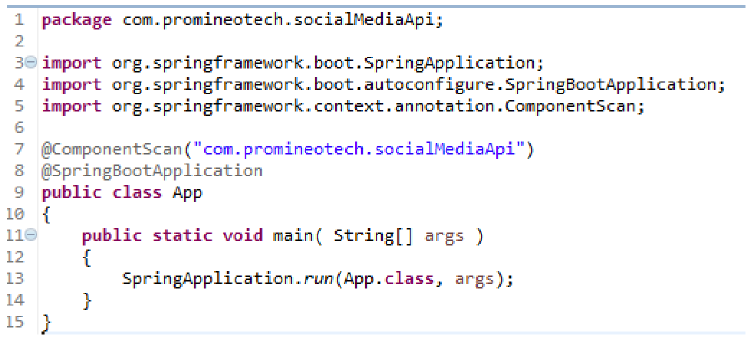
Right click on your project and select **New -> Folder** and call the folder **pictures**. This is where we will end up storing the profile pictures for our users.

Next, create the following packages, classes, and interfaces (the files in the com.promineotech.socialMediaApi.repository package are interfaces).



```
src/main/java
  com.promineotech.socialMediaApi
    App.java
  com.promineotech.socialMediaApi.controller
    CommentController.java
    PostController.java
    UserController.java
  com.promineotech.socialMediaApi.entity
    Comment.java
    Post.java
    User.java
  com.promineotech.socialMediaApi.repository
    CommentRepository.java
    PostRepository.java
    UserRepository.java
  com.promineotech.socialMediaApi.service
    CommentService.java
    PostService.java
    UserService.java
  com.promineotech.socialMediaApi.view
    Following.java
```

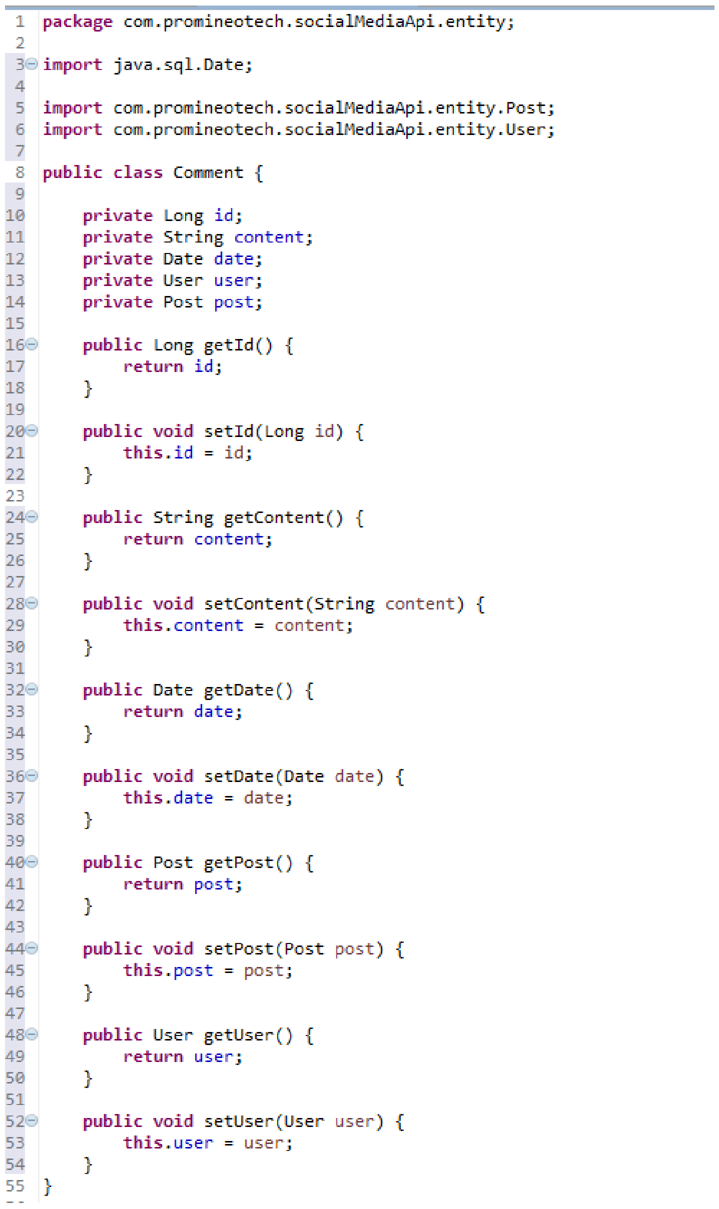
Once those are created, let's set our App class up to launch using Spring Boot. To do this, replace the automatically generated content of the App.java file with the following:



```java
package com.promineotech.socialMediaApi;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.ComponentScan;

@ComponentScan("com.promineotech.socialMediaApi")
@SpringBootApplication
public class App
{
    public static void main( String[] args )
    {
        SpringApplication.run(App.class, args);
    }
}
```

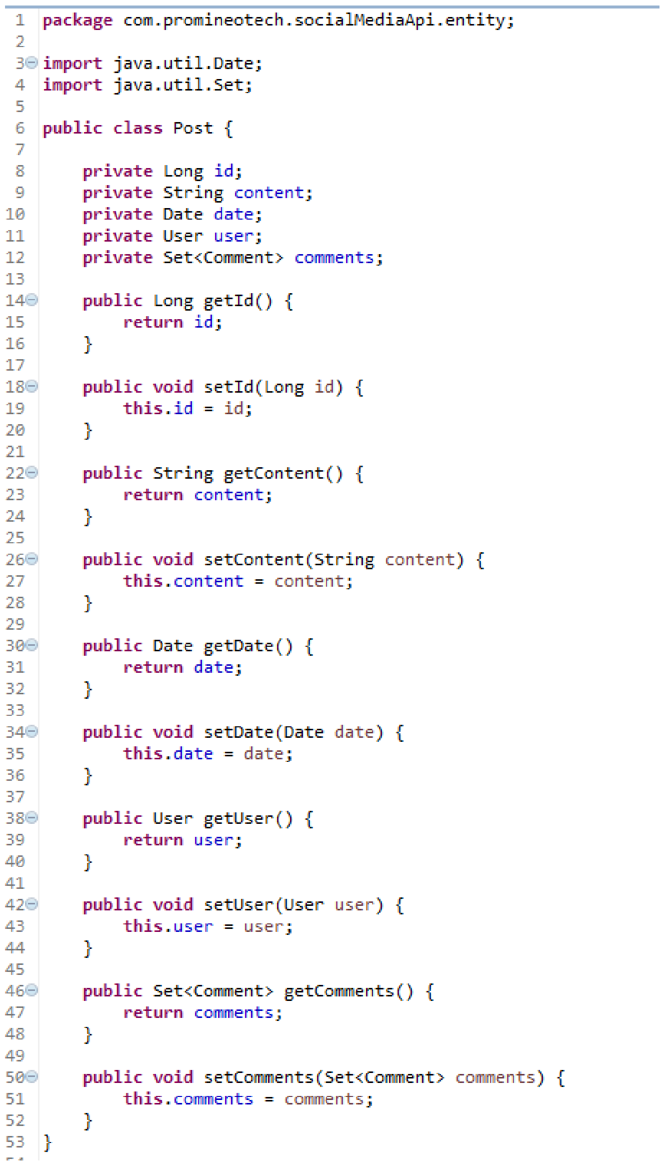
The **@ComponentScan** annotation tells Spring Boot what the base package is to scan for all the components we will be creating. These components are the classes we create that will be injected into other classes using **@Autowired**.

Next, let's go ahead and create the structure of our entities. Make your Comment, Post, and User classes look like the following:

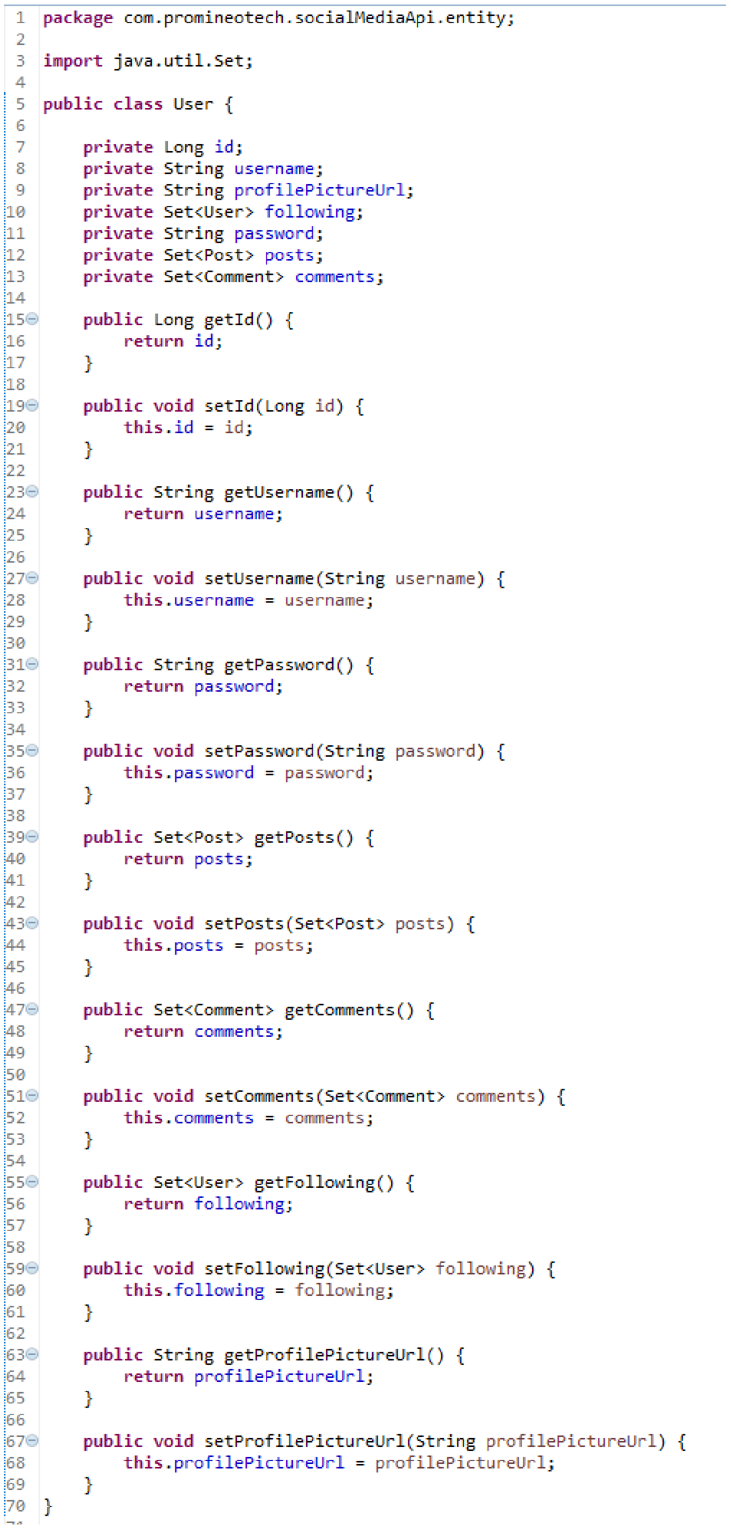


```java
package com.promineotech.socialMediaApi.entity;

import java.sql.Date;

import com.promineotech.socialMediaApi.entity.Post;
import com.promineotech.socialMediaApi.entity.User;

public class Comment {

    private Long id;
    private String content;
    private Date date;
    private User user;
    private Post post;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getContent() {
        return content;
    }

    public void setContent(String content) {
        this.content = content;
    }

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }

    public Post getPost() {
        return post;
    }

    public void setPost(Post post) {
        this.post = post;
    }

    public User getUser() {
        return user;
    }

    public void setUser(User user) {
        this.user = user;
    }
}
```

NOTE: please use java.util.Date instead of java.sql.Date shown in the above image.

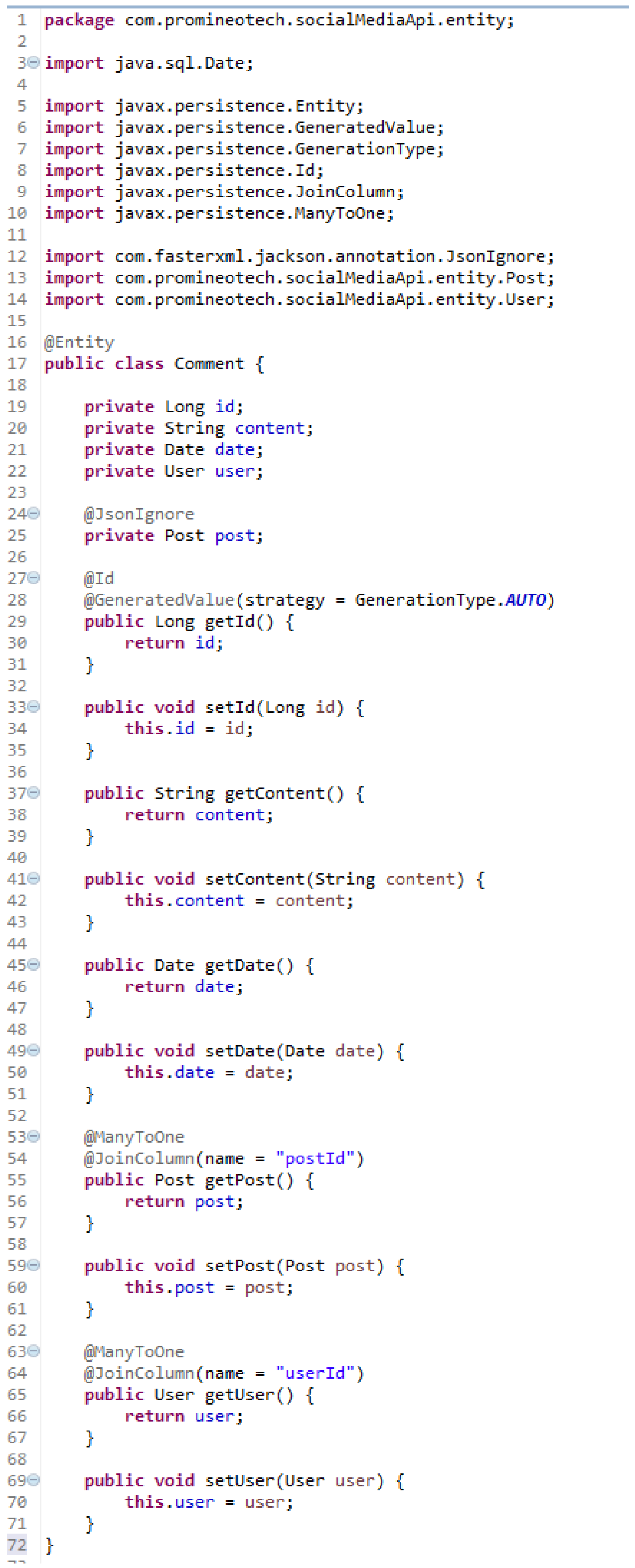


```java
package com.promineotech.socialMediaApi.entity;

import java.util.Date;
import java.util.Set;

public class Post {

    private Long id;
    private String content;
    private Date date;
    private User user;
    private Set<Comment> comments;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getContent() {
        return content;
    }

    public void setContent(String content) {
        this.content = content;
    }

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }

    public User getUser() {
        return user;
    }

    public void setUser(User user) {
        this.user = user;
    }

    public Set<Comment> getComments() {
        return comments;
    }

    public void setComments(Set<Comment> comments) {
        this.comments = comments;
    }
}
```



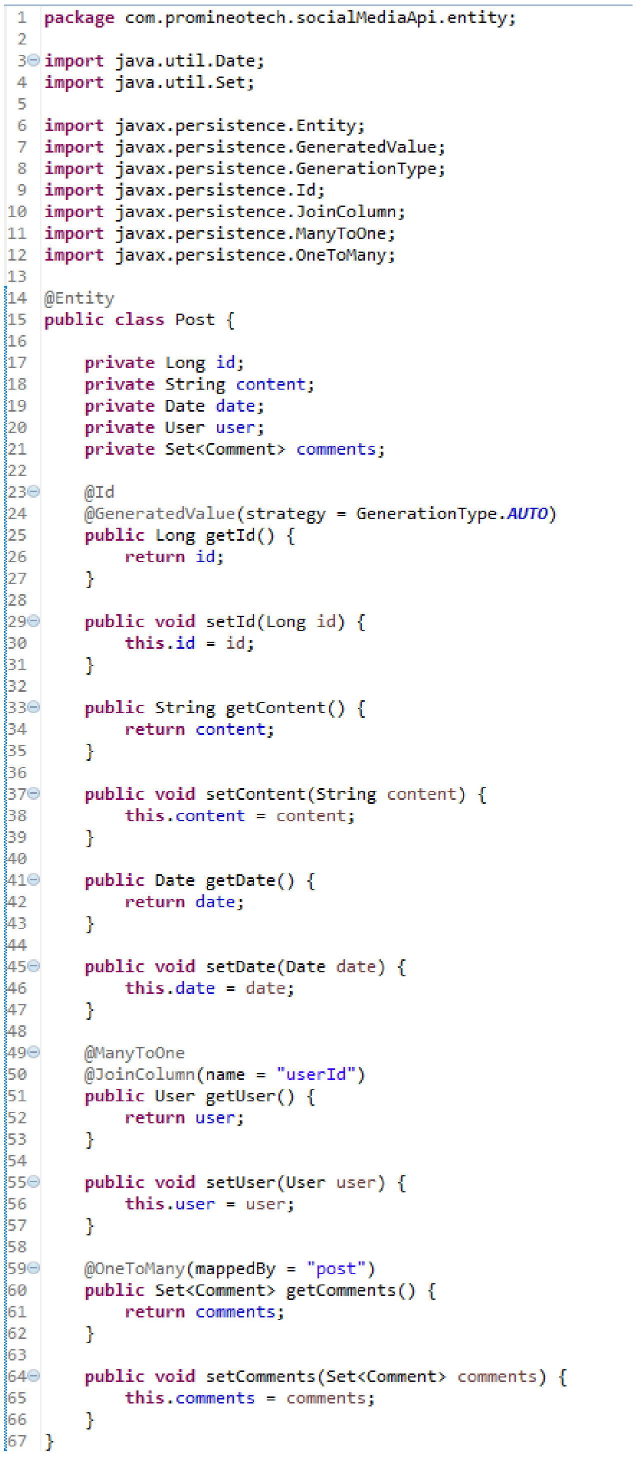
```java
package com.promineotech.socialMediaApi.entity;

import java.util.Set;

public class User {

    private Long id;
    private String username;
    private String profilePictureUrl;
    private Set<User> following;
    private String password;
    private Set<Post> posts;
    private Set<Comment> comments;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public Set<Post> getPosts() {
        return posts;
    }

    public void setPosts(Set<Post> posts) {
        this.posts = posts;
    }

    public Set<Comment> getComments() {
        return comments;
    }

    public void setComments(Set<Comment> comments) {
        this.comments = comments;
    }

    public Set<User> getFollowing() {
        return following;
    }

    public void setFollowing(Set<User> following) {
        this.following = following;
    }

    public String getProfilePictureUrl() {
        return profilePictureUrl;
    }

    public void setProfilePictureUrl(String profilePictureUrl) {
        this.profilePictureUrl = profilePictureUrl;
    }
}
```

The above classes show the fields that will be needed to represent our users, posts, and comments. Now that we have the basic structure in place, we need to add a few more things to set up the relationship mapping in the database, tell Spring Boot that these classes are entities mapped to database tables, and determine which fields will be sent back to the user when they request data and which will not. To do this, modify your entity classes to look like the following.



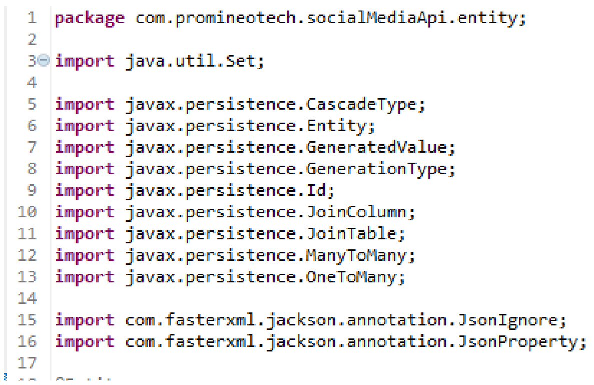
```java
package com.promineotech.socialMediaApi.entity;

import java.sql.Date;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;

import com.fasterxml.jackson.annotation.JsonIgnore;
import com.promineotech.socialMediaApi.entity.Post;
import com.promineotech.socialMediaApi.entity.User;

@Entity
public class Comment {

    private Long id;
    private String content;
    private Date date;
    private User user;

    @JsonIgnore
    private Post post;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getContent() {
        return content;
    }

    public void setContent(String content) {
        this.content = content;
    }

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }

    @ManyToOne
    @JoinColumn(name = "postId")
    public Post getPost() {
        return post;
    }

    public void setPost(Post post) {
        this.post = post;
    }

    @ManyToOne
    @JoinColumn(name = "userId")
    public User getUser() {
        return user;
    }

    public void setUser(User user) {
        this.user = user;
    }
}
```

The **@Entity** annotation tells JPA (which you will learn more about later) that this class maps to a table in the database. The **@JsonIgnore** annotation makes it so the annotated field is not part of the serialized output when sending an instance of this class to the user (it will be hidden from the JSON representation of this class). The **@Id** and **@GeneratedValue** annotations tell the program which field is the primary key in the database and how it should be created. Notice that we used GenerationType.AUTO, which means the primary key will auto increment in the database. The **@ManyToOne** and **@JoinColumn** annotations set up join relationships between this entity table in the database and other entity tables.

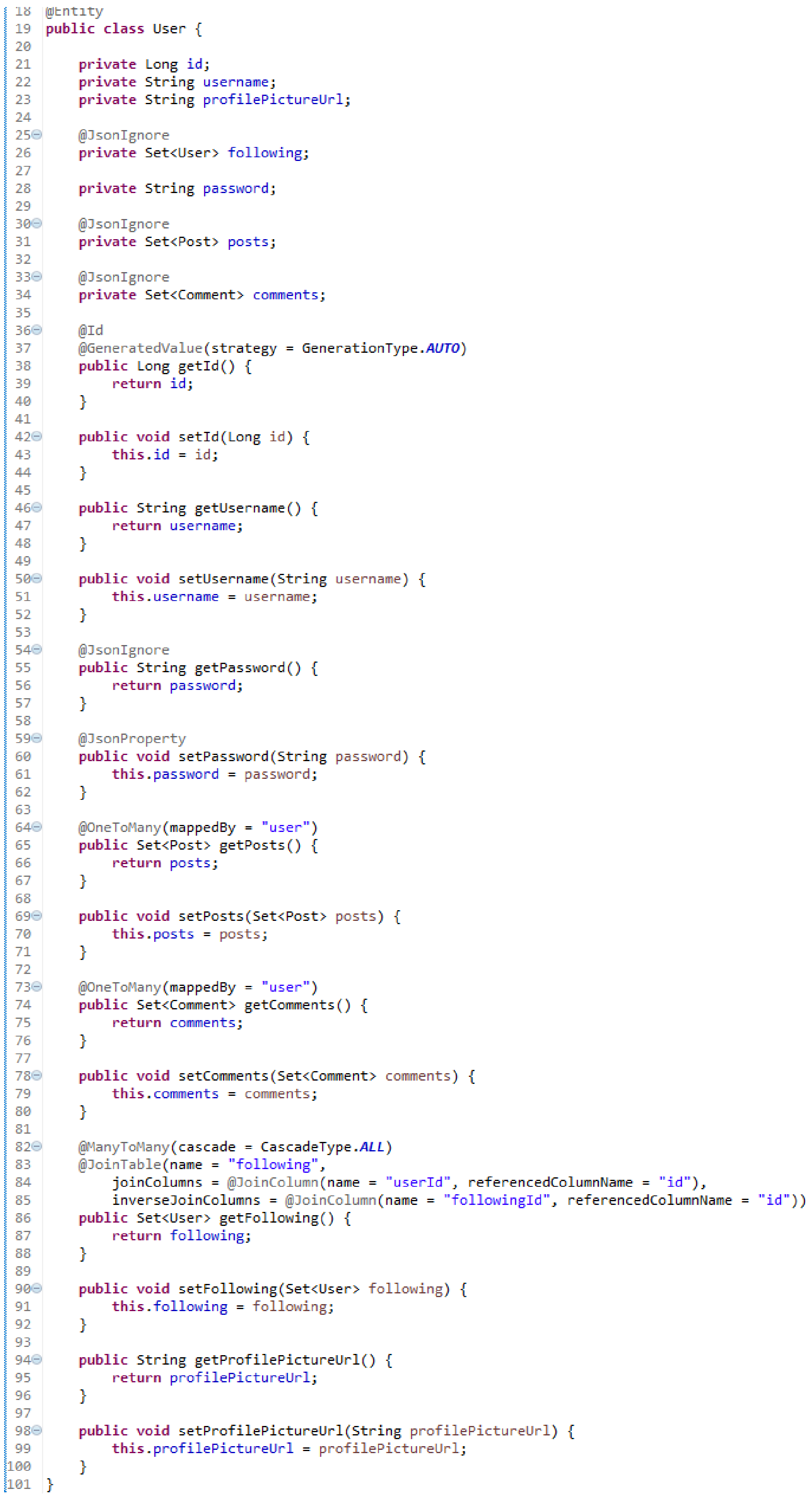


```java
package com.promineotech.socialMediaApi.entity;

import java.util.Date;
import java.util.Set;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.OneToMany;

@Entity
public class Post {

    private Long id;
    private String content;
    private Date date;
    private User user;
    private Set<Comment> comments;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getContent() {
        return content;
    }

    public void setContent(String content) {
        this.content = content;
    }

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }

    @ManyToOne
    @JoinColumn(name = "userId")
    public User getUser() {
        return user;
    }

    public void setUser(User user) {
        this.user = user;
    }

    @OneToMany(mappedBy = "post")
    public Set<Comment> getComments() {
        return comments;
    }

    public void setComments(Set<Comment> comments) {
        this.comments = comments;
    }
}
```

The only new annotation introduced in this class is the **@OneToMany** which is the counterpart to **@ManyToOne**. This tells JPA that all the instances of the Comment class that have this specific Post in their post field, are part of this set. In other words, a Post can have many comments associated with it, while each Comment can only belong to one Post.



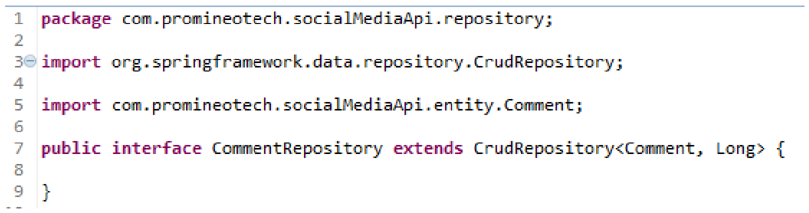
```java
package com.promineotech.socialMediaApi.entity;

import java.util.Set;

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.OneToMany;

import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonProperty;

```



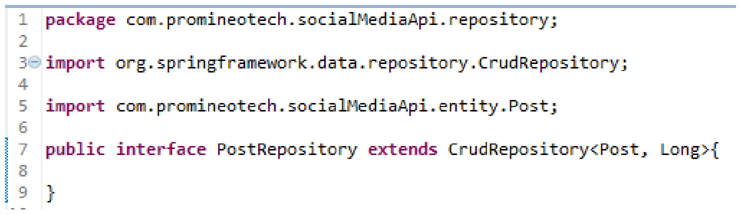
```java
@Entity
public class User {

    private Long id;
    private String username;
    private String profilePictureUrl;

    @JsonIgnore
    private Set<User> following;

    private String password;

    @JsonIgnore
    private Set<Post> posts;

    @JsonIgnore
    private Set<Comment> comments;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    @JsonIgnore
    public String getPassword() {
        return password;
    }

    @JsonProperty
    public void setPassword(String password) {
        this.password = password;
    }

    @OneToMany(mappedBy = "user")
    public Set<Post> getPosts() {
        return posts;
    }

    public void setPosts(Set<Post> posts) {
        this.posts = posts;
    }

    @OneToMany(mappedBy = "user")
    public Set<Comment> getComments() {
        return comments;
    }

    public void setComments(Set<Comment> comments) {
        this.comments = comments;
    }

    @ManyToMany(cascade = CascadeType.ALL)
    @JoinTable(name = "following",
        joinColumns = @JoinColumn(name = "userId", referencedColumnName = "id"),
        inverseJoinColumns = @JoinColumn(name = "followingId", referencedColumnName = "id"))
    public Set<User> getFollowing() {
        return following;
    }

    public void setFollowing(Set<User> following) {
        this.following = following;
    }

    public String getProfilePictureUrl() {
        return profilePictureUrl;
    }

    public void setProfilePictureUrl(String profilePictureUrl) {
        this.profilePictureUrl = profilePictureUrl;
    }
}
```

Here, we are introduced to the **@JsonProperty** annotation which is placed over the setter for the password field. We have **@JsonIgnore** over the getter so that a user's password is never serialized and returned as the result of an API call, but we want the password to be able to come in from a user when the are registering or logging in. The @JsonProperty annotation allows this to happen. We also notice the **@ManyToMany** and **@JoinTable** annotations. These have a little more going on, but they essentially set up a many-to-many relationship between entities in the database, declare what the join table will be named, and identify the primary keys that each entity will contribute to the join table. If you need a refresher on how many-to-many relationships work, consider looking back at the relational database curriculum or researching many-to-many relationships in SQL.

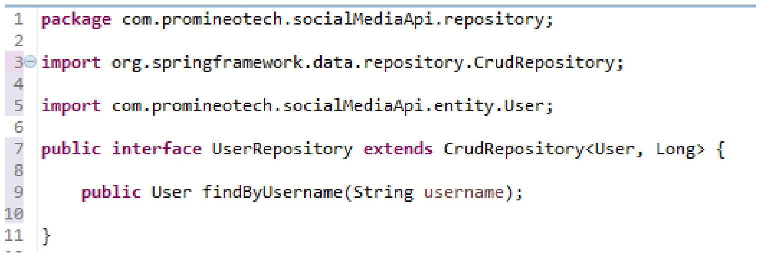
Now that we've completed the entities, let's create the repository interfaces. These interfaces will extend CrudRepository, which gives each interface we create a lot of already implemented functionality such as the ability to read and write to the database tables that correspond with the entities without us having to write any additional code! Go ahead and add the following code to your repository classes.



```java
package com.promineotech.socialMediaApi.repository;

import org.springframework.data.repository.CrudRepository;

import com.promineotech.socialMediaApi.entity.Comment;

public interface CommentRepository extends CrudRepository<Comment, Long> {

}
```



```java
package com.promineotech.socialMediaApi.repository;

import org.springframework.data.repository.CrudRepository;

import com.promineotech.socialMediaApi.entity.Post;

public interface PostRepository extends CrudRepository<Post, Long>{

}
```



```java
package com.promineotech.socialMediaApi.repository;

import org.springframework.data.repository.CrudRepository;

import com.promineotech.socialMediaApi.entity.User;

public interface UserRepository extends CrudRepository<User, Long> {

    public User findByUsername(String username);

}
```

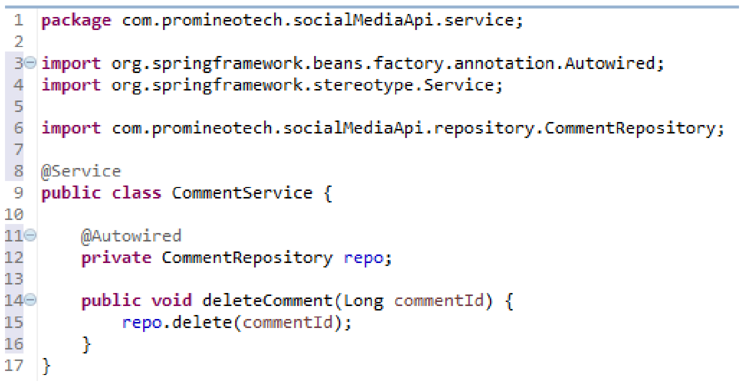
Notice that all three of these specify two classes in the CrudRepository's generic diamond operators - the entity class that this repository will manage (User, Post, or Comment) and the type that the primary key will be (Long). In addition to extending the CrudRepository, the UserRepository class also adds an abstract method, findByUsername. This method has no implementation, but it doesn't need one, because Spring Data does some magic under the hood and implements the functionality based on the name and method signature. So, just by **public User findByUsername(String username);** Spring Data knows to take a String, find the row in the database where the username matches that String, create an instance of the User class with the data from the retrieved row, and return that User.

Now that we have our repositories created, let's work on the service layer and our business logic. For Users, we know that we will need the following functionality.

The ability to:

- create a comment
- delete a comment

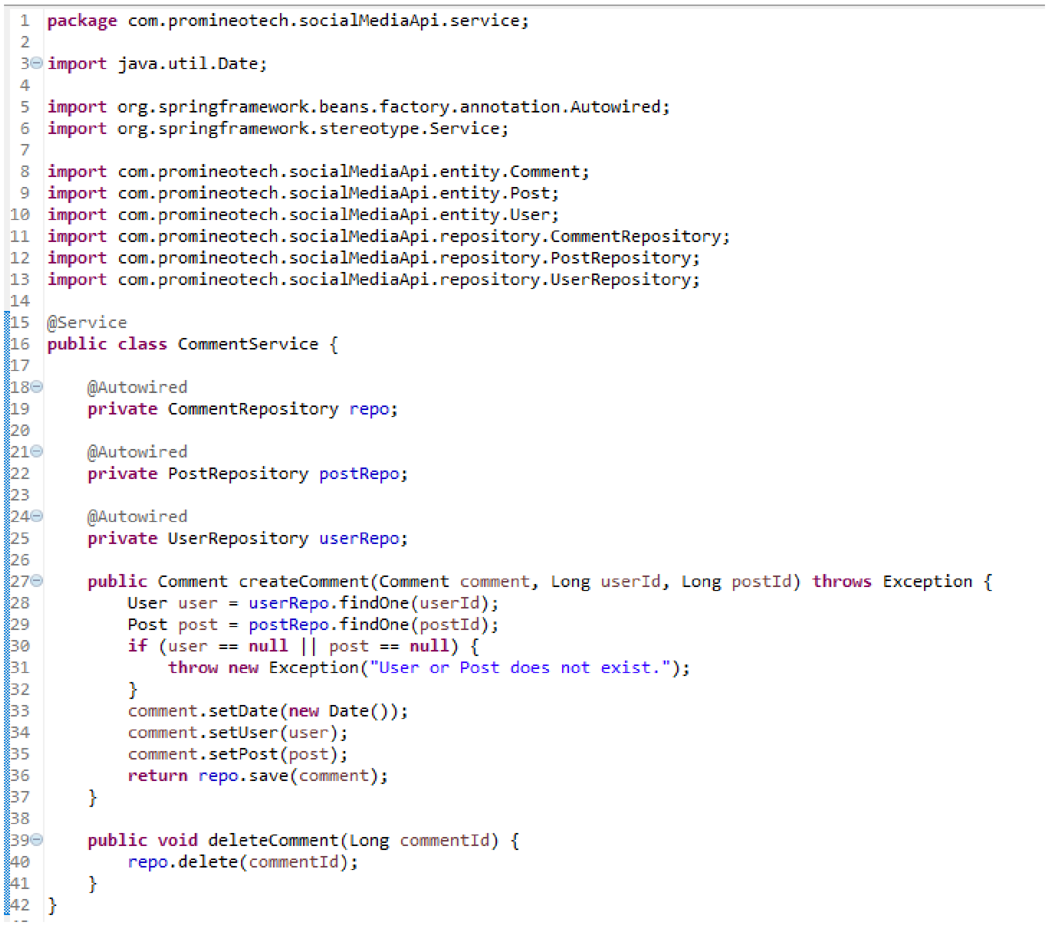
To delete a comment is a fairly simple process. Because of the functionality that Spring Data provides, we can just call the delete method on the repository. Add the following lines to your CommentService class.



```java
package com.promineotech.socialMediaApi.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.promineotech.socialMediaApi.repository.CommentRepository;

@Service
public class CommentService {

    @Autowired
    private CommentRepository repo;

    public void deleteComment(Long commentId) {
        repo.delete(commentId);
    }
}
```

The **@Service** annotation marks this class as a component to be picked up by the scanner and allows us to automatically inject an instance into any class that autowires it. You can see an example of the CommentRepository being injected using **@Autowired** as well. Notice how, with the injection of the CommentRepository, we don't have to instantiate an instance of the class anywhere, that is taken care of for us at runtime.

Next, let's add the ability to create a new comment. This one takes a bit more work as we need to be able to find the user that made the comment and the post the comment was made on. To do that, we will need to add the repositories for user and post, query the databases for the related user and post, and add those to the comment's fields before saving it. Just like this:

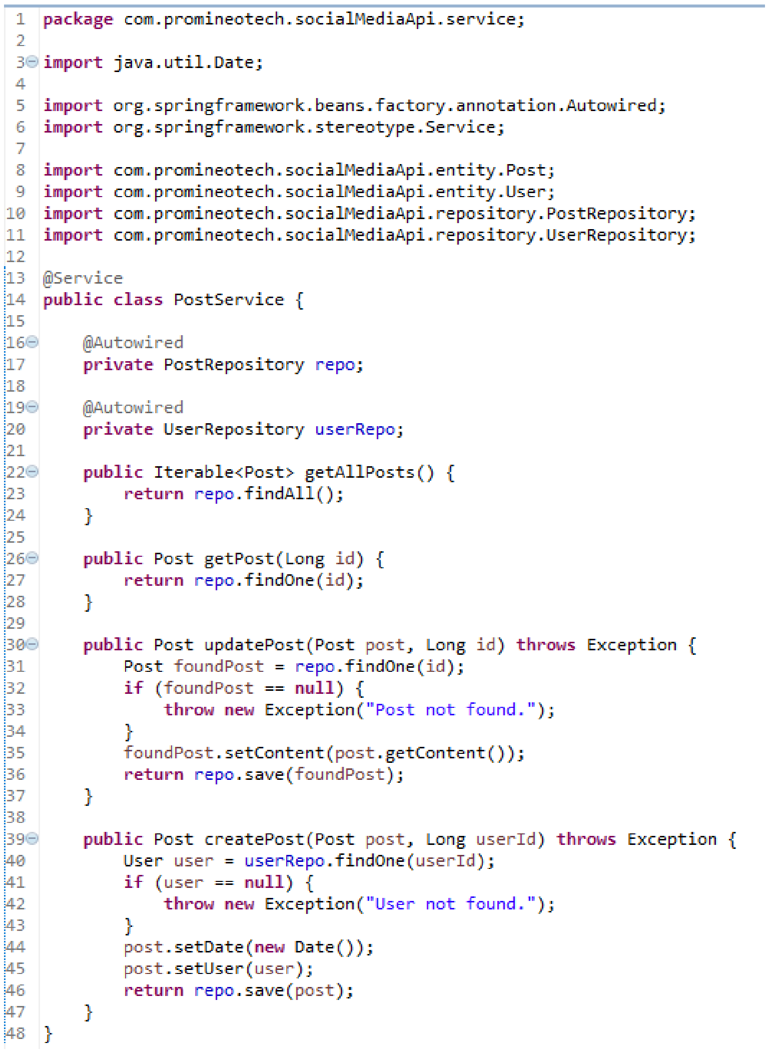


```java
package com.promineotech.socialMediaApi.service;

import java.util.Date;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.promineotech.socialMediaApi.entity.Comment;
import com.promineotech.socialMediaApi.entity.Post;
import com.promineotech.socialMediaApi.entity.User;
import com.promineotech.socialMediaApi.repository.CommentRepository;
import com.promineotech.socialMediaApi.repository.PostRepository;
import com.promineotech.socialMediaApi.repository.UserRepository;

@Service
public class CommentService {

    @Autowired
    private CommentRepository repo;

    @Autowired
    private PostRepository postRepo;

    @Autowired
    private UserRepository userRepo;

    public Comment createComment(Comment comment, Long userId, Long postId) throws Exception {
        User user = userRepo.findOne(userId);
        Post post = postRepo.findOne(postId);
        if (user == null || post == null) {
            throw new Exception("User or Post does not exist.");
        }
        comment.setDate(new Date());
        comment.setUser(user);
        comment.setPost(post);
        return repo.save(comment);
    }

    public void deleteComment(Long commentId) {
        repo.delete(commentId);
    }
}
```

Next, the PostService follows a similar pattern. We want to be able to:

- find all posts to display in a news feed of sorts
- find an individual post
- update a post
- create a post

We could add the ability to delete a post as well, if we wanted to.



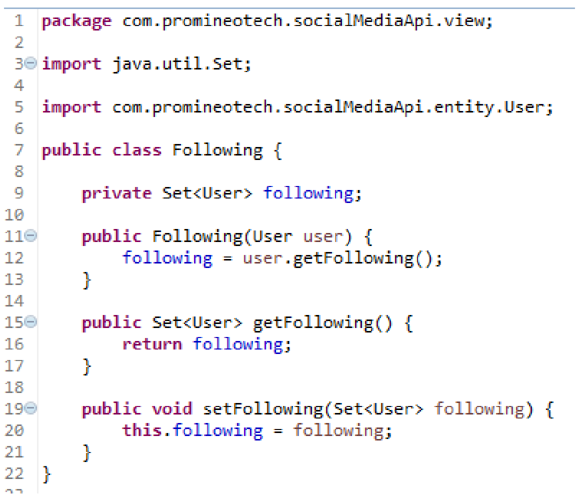
```java
package com.promineotech.socialMediaApi.service;

import java.util.Date;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.promineotech.socialMediaApi.entity.Post;
import com.promineotech.socialMediaApi.entity.User;
import com.promineotech.socialMediaApi.repository.PostRepository;
import com.promineotech.socialMediaApi.repository.UserRepository;

@Service
public class PostService {

    @Autowired
    private PostRepository repo;

    @Autowired
    private UserRepository userRepo;

    public Iterable<Post> getAllPosts() {
        return repo.findAll();
    }

    public Post getPost(Long id) {
        return repo.findOne(id);
    }

    public Post updatePost(Post post, Long id) throws Exception {
        Post foundPost = repo.findOne(id);
        if (foundPost == null) {
            throw new Exception("Post not found.");
        }
        foundPost.setContent(post.getContent());
        return repo.save(foundPost);
    }

    public Post createPost(Post post, Long userId) throws Exception {
        User user = userRepo.findOne(userId);
        if (user == null) {
            throw new Exception("User not found.");
        }
        post.setDate(new Date());
        post.setUser(user);
        return repo.save(post);
    }
}
```

The UserService is a little more involved than the PostService and CommentService. In the UserService we want the ability to:

- register or create users
- allow users to log in
- allow users to follow other users
- get a list of followed users for a given user
- update a user's profile picture

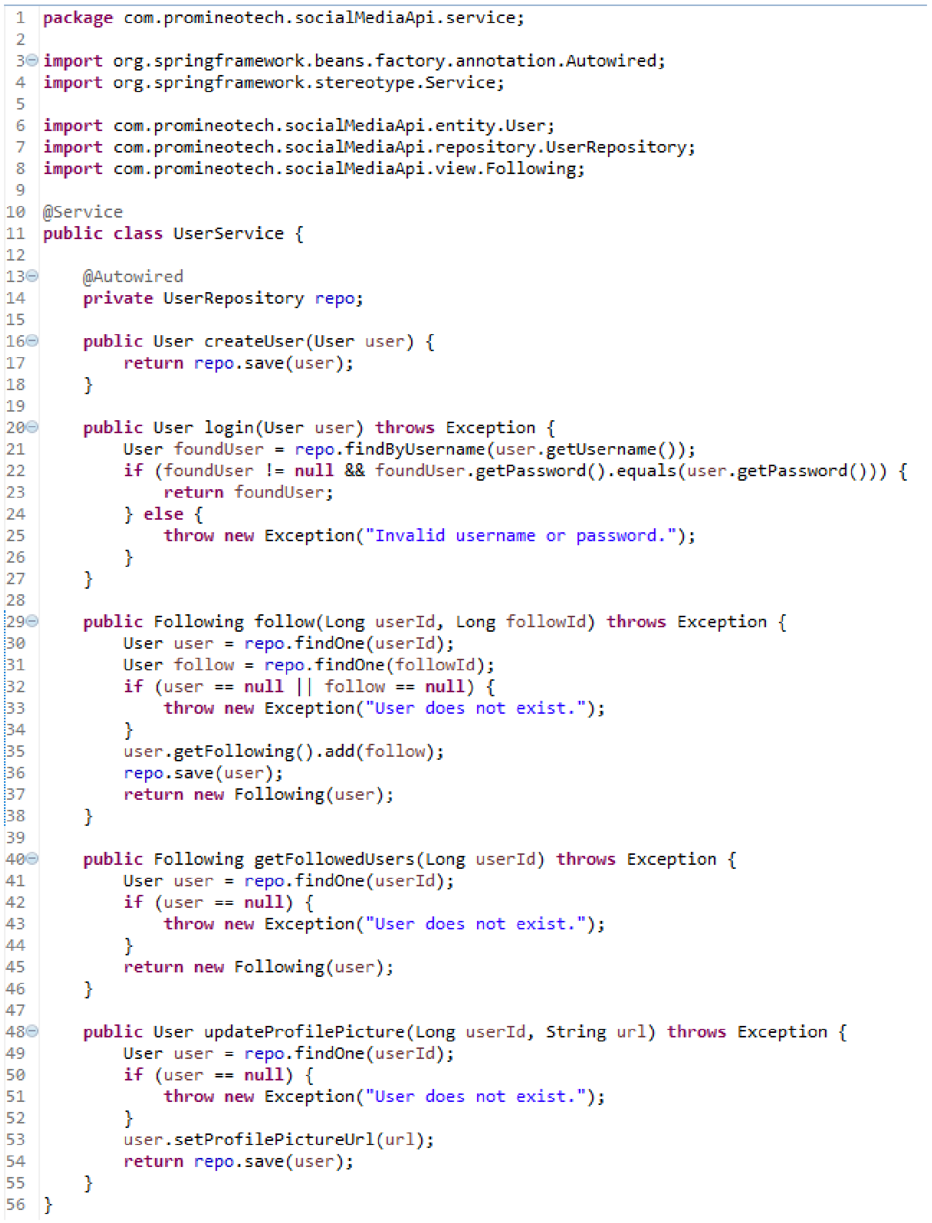
Note: this tutorial does not implement or cover security, we will learn more about that in a future lesson.

Before we can do this, however, we need to build out our Following class.



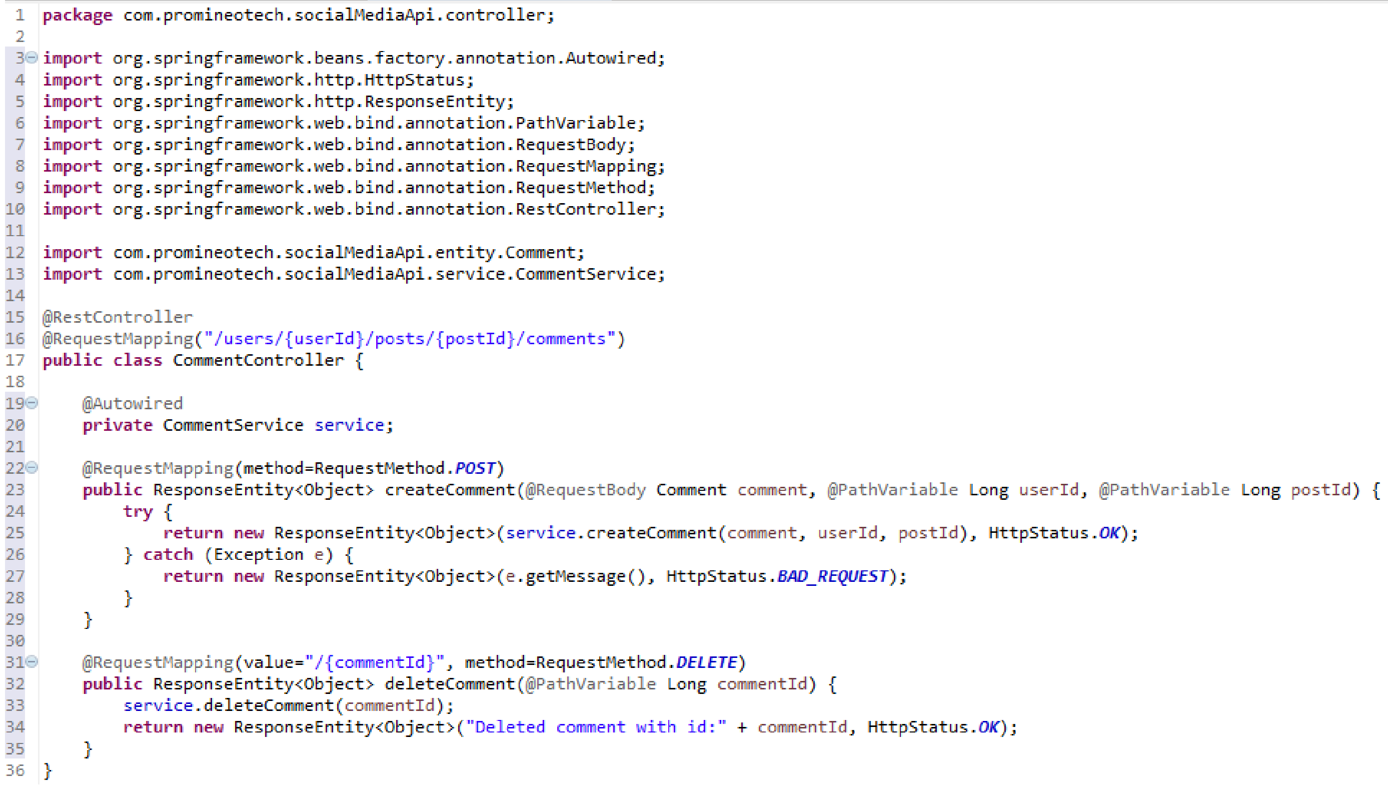
```java
package com.promineotech.socialMediaApi.view;

import java.util.Set;

import com.promineotech.socialMediaApi.entity.User;

public class Following {

    private Set<User> following;

    public Following(User user) {
        following = user.getFollowing();
    }

    public Set<User> getFollowing() {
        return following;
    }

    public void setFollowing(Set<User> following) {
        this.following = following;
    }
}
```

This is a simple class that contains a Set of Users that a given user follows. Now that we have this implemented, we can add the code for our UserService.



```java
package com.promineotech.socialMediaApi.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.promineotech.socialMediaApi.entity.User;
import com.promineotech.socialMediaApi.repository.UserRepository;
import com.promineotech.socialMediaApi.view.Following;

@Service
public class UserService {

    @Autowired
    private UserRepository repo;

    public User createUser(User user) {
        return repo.save(user);
    }

    public User login(User user) throws Exception {
        User foundUser = repo.findByUsername(user.getUsername());
        if (foundUser != null && foundUser.getPassword().equals(user.getPassword())) {
            return foundUser;
        } else {
            throw new Exception("Invalid username or password.");
        }
    }

    public Following follow(Long userId, Long followId) throws Exception {
        User user = repo.findOne(userId);
        User follow = repo.findOne(followId);
        if (user == null || follow == null) {
            throw new Exception("User does not exist.");
        }
        user.getFollowing().add(follow);
        repo.save(user);
        return new Following(user);
    }

    public Following getFollowedUsers(Long userId) throws Exception {
        User user = repo.findOne(userId);
        if (user == null) {
            throw new Exception("User does not exist.");
        }
        return new Following(user);
    }

    public User updateProfilePicture(Long userId, String url) throws Exception {
        User user = repo.findOne(userId);
        if (user == null) {
            throw new Exception("User does not exist.");
        }
        user.setProfilePictureUrl(url);
        return repo.save(user);
    }
}
```

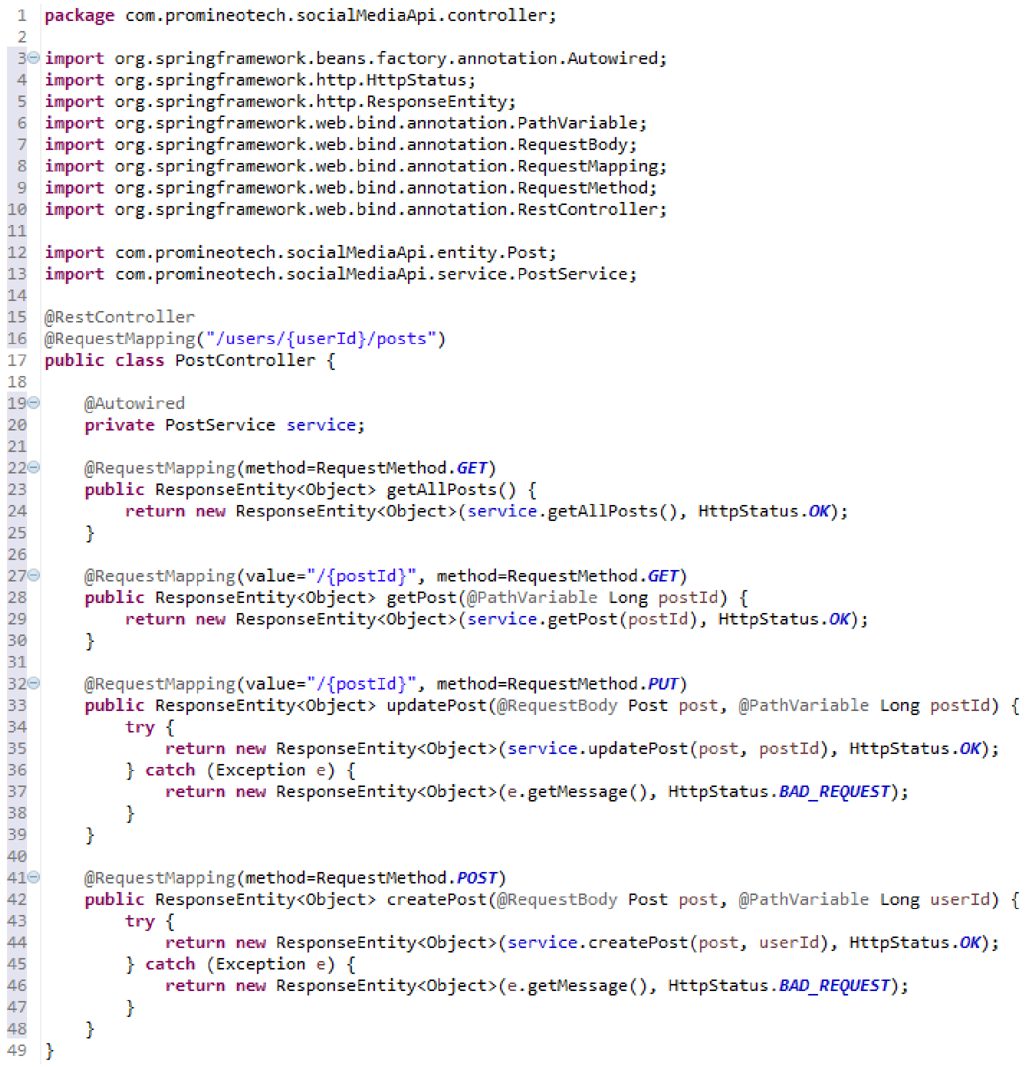
Now that we have all the service layer classes complete, let's finalize the project by creating the controller classes. The controllers are where we define the endpoints that this API will have. Below is the CommentController class that will have endpoints that allow a user to create or delete a comment.



```java
package com.promineotech.socialMediaApi.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import com.promineotech.socialMediaApi.entity.Comment;
import com.promineotech.socialMediaApi.service.CommentService;

@RestController
@RequestMapping("/users/{userId}/posts/{postId}/comments")
public class CommentController {

    @Autowired
    private CommentService service;

    @RequestMapping(method=RequestMethod.POST)
    public ResponseEntity<Object> createComment(@RequestBody Comment comment, @PathVariable Long userId, @PathVariable Long postId) {
        try {
            return new ResponseEntity<Object>(service.createComment(comment, userId, postId), HttpStatus.OK);
        } catch (Exception e) {
            return new ResponseEntity<Object>(e.getMessage(), HttpStatus.BAD_REQUEST);
        }
    }

    @RequestMapping(value="/{commentId}", method=RequestMethod.DELETE)
    public ResponseEntity<Object> deleteComment(@PathVariable Long commentId) {
        service.deleteComment(commentId);
        return new ResponseEntity<Object>("Deleted comment with id:" + commentId, HttpStatus.OK);
    }
}
```

The **@RestController** annotation tells Spring Boot that this class will be used to expose endpoints. The **@RequestMapping** annotation is used to set up which URL and which HTTP verb methods trigger a specific method. We have an @RequestMapping at the top of the class, before the class name, which makes all the endpoints in this class have a base URL of ***/users/{userId}/posts/{postsId}/comments***. The values userId and postId in curly braces are placeholders for path variables that will be replaced with values. For example, when an endpoint on this class is called the URL may look like ***/users/45/posts/3/comments***. The other two annotations we see here are the **@RequestBody** and **@PathVariable**. The request body is the payload that comes from the HTTP request and is deserialized into an instance of a class. The JSON fields that are passed in must match the same fields/schema of the class specified. Path variables are the values in place of the placeholders in the URL between curly braces.

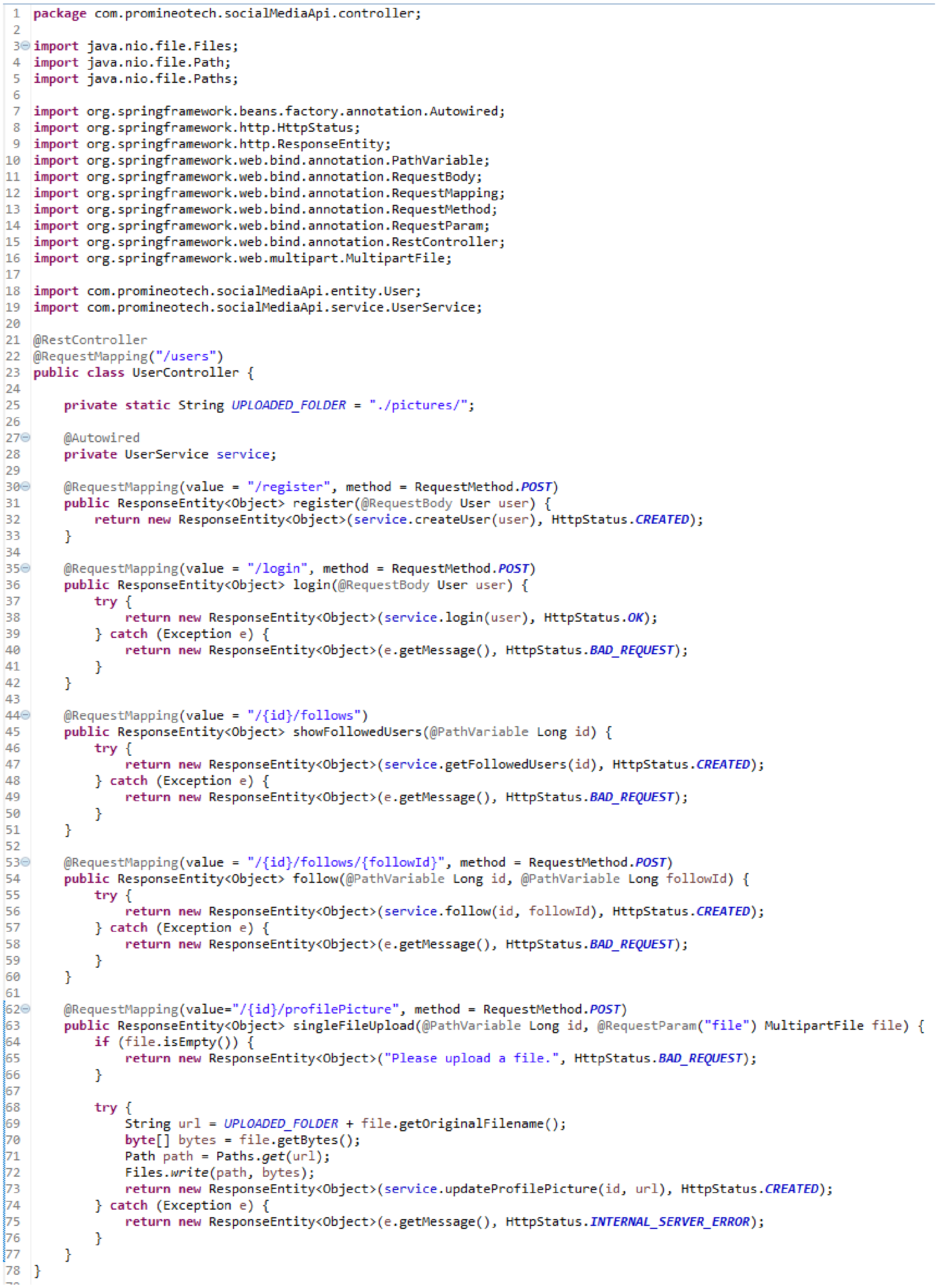
Next, let's write the necessary code to make our PostController endpoints work.



```java
package com.promineotech.socialMediaApi.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import com.promineotech.socialMediaApi.entity.Post;
import com.promineotech.socialMediaApi.service.PostService;

@RestController
@RequestMapping("/users/{userId}/posts")
public class PostController {

    @Autowired
    private PostService service;

    @RequestMapping(method=RequestMethod.GET)
    public ResponseEntity<Object> getAllPosts() {
        return new ResponseEntity<Object>(service.getAllPosts(), HttpStatus.OK);
    }

    @RequestMapping(value="/{postId}", method=RequestMethod.GET)
    public ResponseEntity<Object> getPost(@PathVariable Long postId) {
        return new ResponseEntity<Object>(service.getPost(postId), HttpStatus.OK);
    }

    @RequestMapping(value="/{postId}", method=RequestMethod.PUT)
    public ResponseEntity<Object> updatePost(@RequestBody Post post, @PathVariable Long postId) {
        try {
            return new ResponseEntity<Object>(service.updatePost(post, postId), HttpStatus.OK);
        } catch (Exception e) {
            return new ResponseEntity<Object>(e.getMessage(), HttpStatus.BAD_REQUEST);
        }
    }

    @RequestMapping(method=RequestMethod.POST)
    public ResponseEntity<Object> createPost(@RequestBody Post post, @PathVariable Long userId) {
        try {
            return new ResponseEntity<Object>(service.createPost(post, userId), HttpStatus.OK);
        } catch (Exception e) {
            return new ResponseEntity<Object>(e.getMessage(), HttpStatus.BAD_REQUEST);
        }
    }
}
```

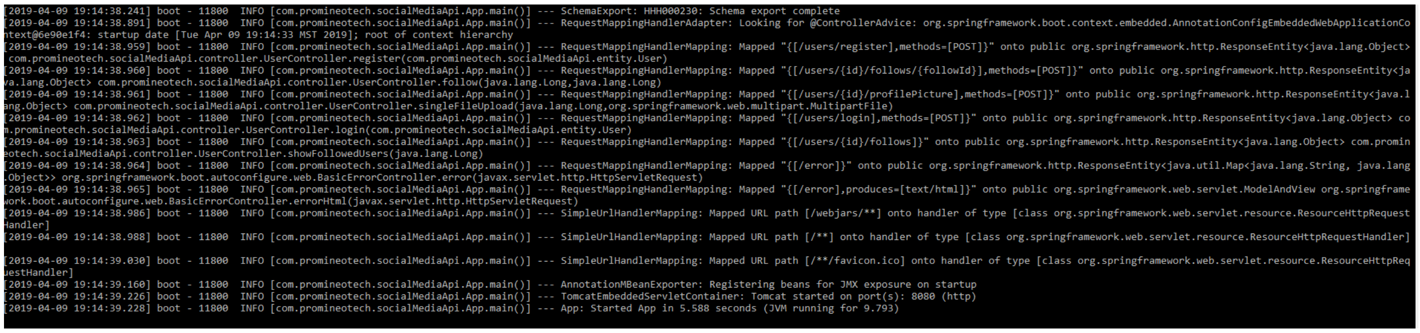
Notice how, in the controller layer, we don't do a whole bunch - we mostly just call the service layer and then return some result. This is because we want all the "heavy lifting" to be done in the service layer so that it is reusable from different controllers and other classes in the future. We don't want to have to rewrite the same functionality over and over, so putting it in a service layer where we can call those methods from other classes later on helps improve the maintainability or our code.

Finally, we have our UserController. This one is a bit more involved and contains endpoints mapped to methods that allow users to register, login, follow other users, and the special one - update a profile picture.



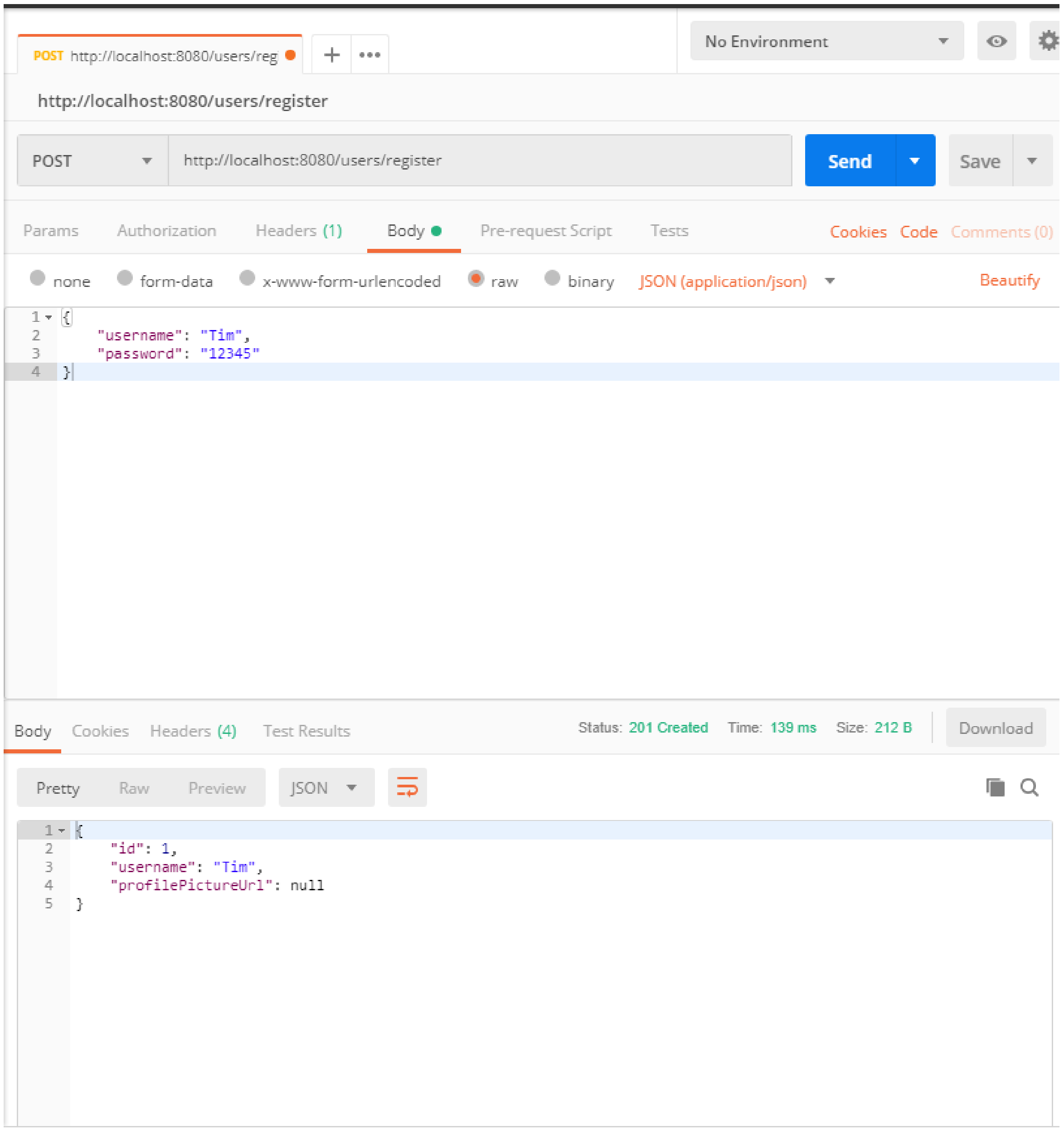
```java
package com.promineotech.socialMediaApi.controller;

import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.multipart.MultipartFile;

import com.promineotech.socialMediaApi.entity.User;
import com.promineotech.socialMediaApi.service.UserService;

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

    private static String UPLOADED_FOLDER = "./pictures/";

    @Autowired
    private UserService service;

    @RequestMapping(value = "/register", method = RequestMethod.POST)
    public ResponseEntity<Object> register(@RequestBody User user) {
        return new ResponseEntity<Object>(service.createUser(user), HttpStatus.CREATED);
    }

    @RequestMapping(value = "/login", method = RequestMethod.POST)
    public ResponseEntity<Object> login(@RequestBody User user) {
        try {
            return new ResponseEntity<Object>(service.login(user), HttpStatus.OK);
        } catch (Exception e) {
            return new ResponseEntity<Object>(e.getMessage(), HttpStatus.BAD_REQUEST);
        }
    }

    @RequestMapping(value = "/{id}/follows")
    public ResponseEntity<Object> showFollowedUsers(@PathVariable Long id) {
        try {
            return new ResponseEntity<Object>(service.getFollowedUsers(id), HttpStatus.CREATED);
        } catch (Exception e) {
            return new ResponseEntity<Object>(e.getMessage(), HttpStatus.BAD_REQUEST);
        }
    }

    @RequestMapping(value = "/{id}/follows/{followId}", method = RequestMethod.POST)
    public ResponseEntity<Object> follow(@PathVariable Long id, @PathVariable Long followId) {
        try {
            return new ResponseEntity<Object>(service.follow(id, followId), HttpStatus.CREATED);
        } catch (Exception e) {
            return new ResponseEntity<Object>(e.getMessage(), HttpStatus.BAD_REQUEST);
        }
    }

    @RequestMapping(value="/{id}/profilePicture", method = RequestMethod.POST)
    public ResponseEntity<Object> singleFileUpload(@PathVariable Long id, @RequestParam("file") MultipartFile file) {
        if (file.isEmpty()) {
            return new ResponseEntity<Object>("Please upload a file.", HttpStatus.BAD_REQUEST);
        }

        try {
            String url = UPLOADED_FOLDER + file.getOriginalFilename();
            byte[] bytes = file.getBytes();
            Path path = Paths.get(url);
            Files.write(path, bytes);
            return new ResponseEntity<Object>(service.updateProfilePicture(id, url), HttpStatus.CREATED);
        } catch (Exception e) {
            return new ResponseEntity<Object>(e.getMessage(), HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }
}
```

We can now test out our API. Open command prompt or terminal and navigate to your projects root directory. Run **mvn spring-boot:run** to start your server. You'll know the server has successfully started when you see **App: Started App in x seconds (JVM running for x)** as seen in the below image. Also, note that all the endpoints we defined are shown as Mapped. In addition to the endpoints we defined in our API, you can also see an [[/error]] and other miscellaneous endpoints. This does not mean that there is an error, rather it means that if you hit an endpoint that is not defined it will reroute you to the default error endpoint and send you an error message back.



```
[2019-04-09 19:14:38.241] boot - 11800  INFO [com.promineotech.socialMediaApi.App.main()] --- SchemaExport: HHH000230: Schema export complete
[2019-04-09 19:14:38.891] boot - 11800  INFO [com.promineotech.socialMediaApi.App.main()] --- RequestMappingHandlerAdapter: Looking for @ControllerAdvice: org.springframework.boot.context.embedded.AnnotationConfigEmbeddedWebApplicationCo
ntext@6e90e1f4: startup date [Tue Apr 09 19:14:33 MST 2019]; root of context hierarchy
[2019-04-09 19:14:38.959] boot - 11800  INFO [com.promineotech.socialMediaApi.App.main()] --- RequestMappingHandlerMapping: Mapped "{[/users/register],methods=[POST]}" onto public org.springframework.http.ResponseEntity<java.lang.Object>
 com.promineotech.socialMediaApi.controller.UserController.register(com.promineotech.socialMediaApi.entity.User)
[2019-04-09 19:14:38.960] boot - 11800  INFO [com.promineotech.socialMediaApi.App.main()] --- RequestMappingHandlerMapping: Mapped "{[/users/{id}/follows/{followId}],methods=[POST]}" onto public org.springframework.http.ResponseEntity<ja
va.lang.Object> com.promineotech.socialMediaApi.controller.UserController.follow(java.lang.Long,java.lang.Long)
[2019-04-09 19:14:38.961] boot - 11800  INFO [com.promineotech.socialMediaApi.App.main()] --- RequestMappingHandlerMapping: Mapped "{[/users/{id}/profilePicture],methods=[POST]}" onto public org.springframework.http.ResponseEntity<java.l
ang.Object> com.promineotech.socialMediaApi.controller.UserController.singleFileUpload(java.lang.Long,org.springframework.web.multipart.MultipartFile)
[2019-04-09 19:14:38.962] boot - 11800  INFO [com.promineotech.socialMediaApi.App.main()] --- RequestMappingHandlerMapping: Mapped "{[/users/login],methods=[POST]}" onto public org.springframework.http.ResponseEntity<java.lang.Object> co
m.promineotech.socialMediaApi.controller.UserController.login(com.promineotech.socialMediaApi.entity.User)
[2019-04-09 19:14:38.963] boot - 11800  INFO [com.promineotech.socialMediaApi.App.main()] --- RequestMappingHandlerMapping: Mapped "{[/users/{id}/follows]}" onto public org.springframework.http.ResponseEntity<java.lang.Object> com.promin
eotech.socialMediaApi.controller.UserController.showFollowedUsers(java.lang.Long)
[2019-04-09 19:14:38.964] boot - 11800  INFO [com.promineotech.socialMediaApi.App.main()] --- RequestMappingHandlerMapping: Mapped "{[/error]}" onto public org.springframework.http.ResponseEntity<java.util.Map<java.lang.String, java.lang
.Object>> org.springframework.boot.autoconfigure.web.BasicErrorController.error(javax.servlet.http.HttpServletRequest)
[2019-04-09 19:14:38.965] boot - 11800  INFO [com.promineotech.socialMediaApi.App.main()] --- RequestMappingHandlerMapping: Mapped "{[/error],produces=[text/html]}" onto public org.springframework.web.servlet.ModelAndView org.springframe
work.boot.autoconfigure.web.BasicErrorController.errorHtml(javax.servlet.http.HttpServletRequest)
[2019-04-09 19:14:38.986] boot - 11800  INFO [com.promineotech.socialMediaApi.App.main()] --- SimpleUrlHandlerMapping: Mapped URL path [/webjars/**] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequest
Handler]
[2019-04-09 19:14:38.988] boot - 11800  INFO [com.promineotech.socialMediaApi.App.main()] --- SimpleUrlHandlerMapping: Mapped URL path [/**] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]

[2019-04-09 19:14:39.030] boot - 11800  INFO [com.promineotech.socialMediaApi.App.main()] --- SimpleUrlHandlerMapping: Mapped URL path [/**/favicon.ico] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpReq
uestHandler]
[2019-04-09 19:14:39.160] boot - 11800  INFO [com.promineotech.socialMediaApi.App.main()] --- AnnotationMBeanExporter: Registering beans for JMX exposure on startup
[2019-04-09 19:14:39.226] boot - 11800  INFO [com.promineotech.socialMediaApi.App.main()] --- TomcatEmbeddedServletContainer: Tomcat started on port(s): 8080 (http)
[2019-04-09 19:14:39.228] boot - 11800  INFO [com.promineotech.socialMediaApi.App.main()] --- App: Started App in 5.588 seconds (JVM running for 9.793)
```

Now that the server is started up, we can send some test requests using Postman. Let's start by testing the endpoints defined in the UserController class. Send a POST request to **http://localhost:8080/users/register** to register a new user. We will include username and password in our JSON. Note that the 8080 in the URL is the default port that Spring Boot runs it's server on. You can stop your server by pressing ctrl + c or cmd + c depending on your operating system.



POST http://localhost:8080/users/register

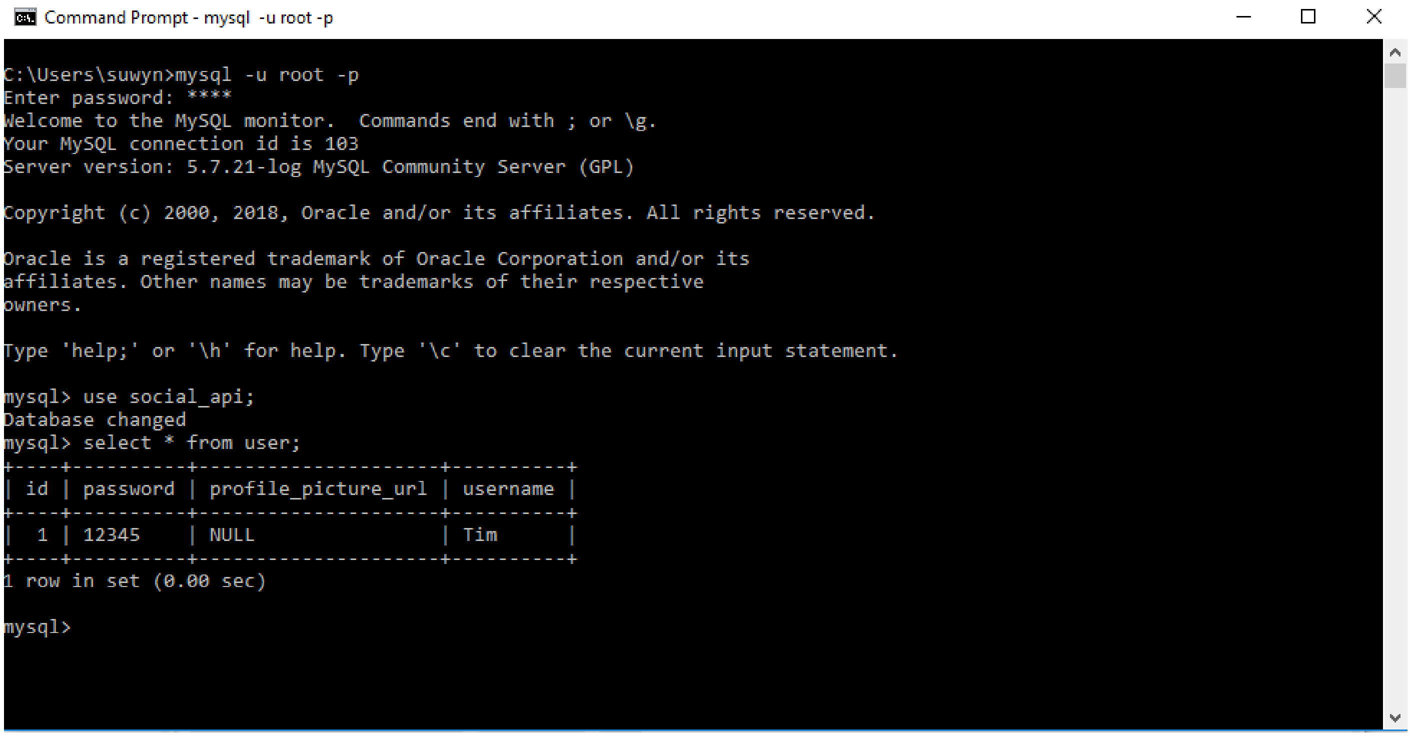
Body: raw, JSON (application/json)

```
{
    "username": "Tim",
    "password": "12345"
}
```

Status: 201 Created Time: 139 ms Size: 212 B

```
{
    "id": 1,
    "username": "Tim",
    "profilePictureUrl": null
}
```

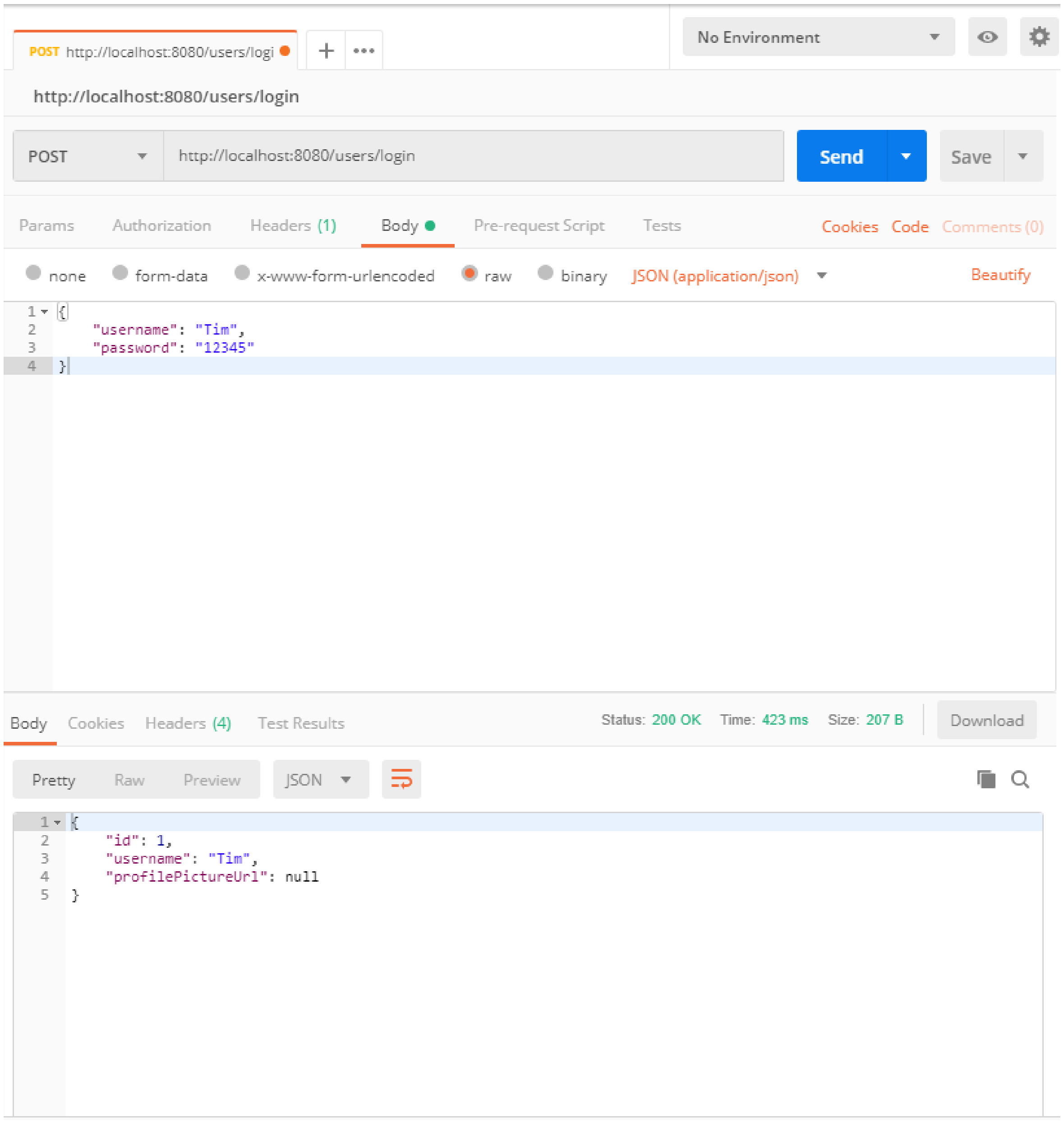
We can see that a new user was successfully register based on the response. We can also log into MySQL and run a query to validate the new data actually exists in the database.



Command Prompt - mysql -u root -p

```
C:\Users\suwyn>mysql -u root -p
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 103
Server version: 5.7.21-log MySQL Community Server (GPL)

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use social_api;
Database changed
mysql> select * from user;
+----+----------+---------------------+----------+
| id | password | profile_picture_url | username |
+----+----------+---------------------+----------+
|  1 | 12345    | NULL                | Tim      |
+----+----------+---------------------+----------+
1 row in set (0.00 sec)

mysql>
```

We can now test the rest of our endpoints as shown in the below images. Each image shows the HTTP verb, the URL, the request body (when needed), and the response.



http://localhost:8080/users/login

POST http://localhost:8080/users/login

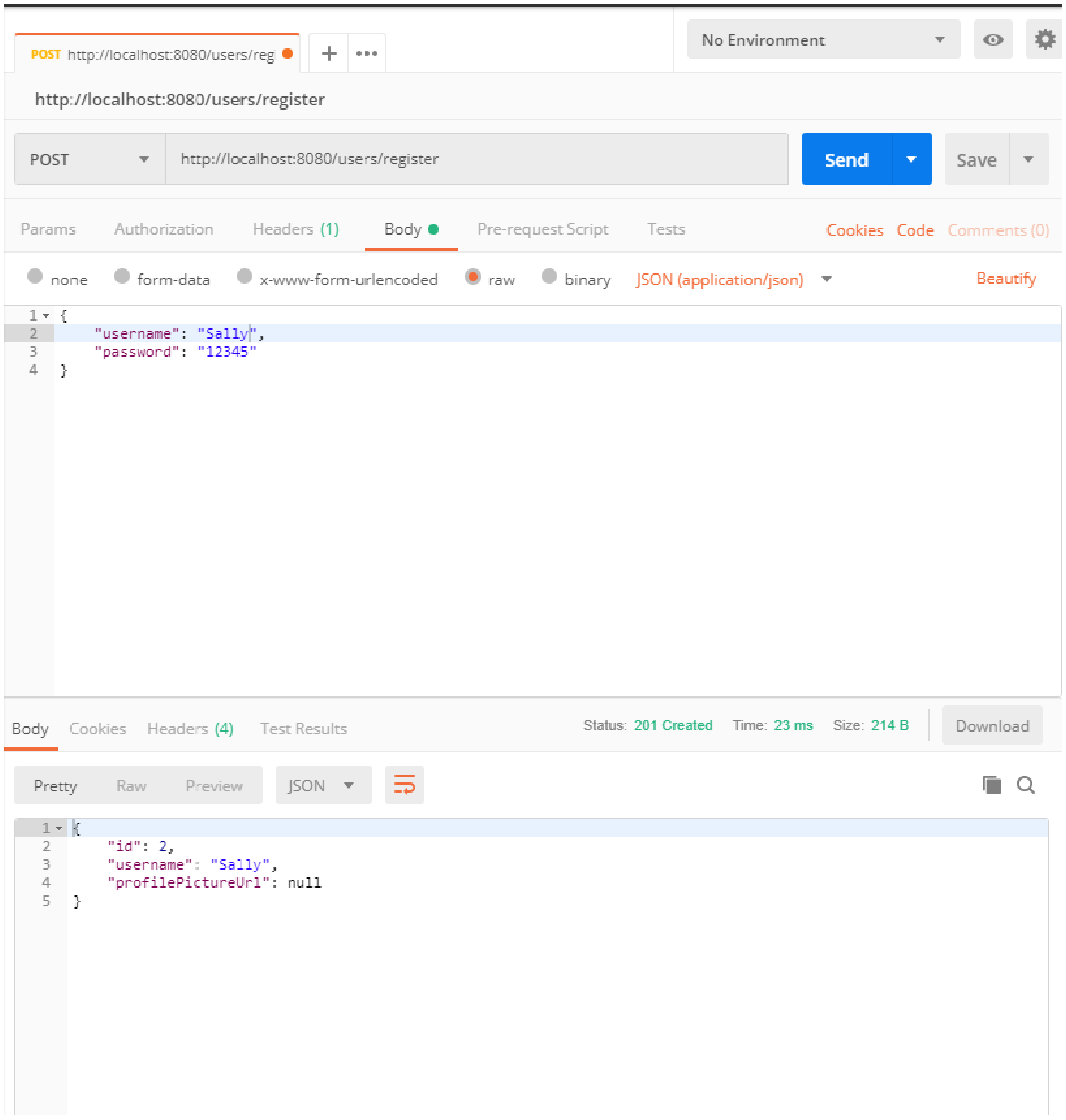
raw JSON (application/json)

```
{
    "username": "Tim",
    "password": "12345"
}
```

Status: 200 OK Time: 423 ms Size: 207 B

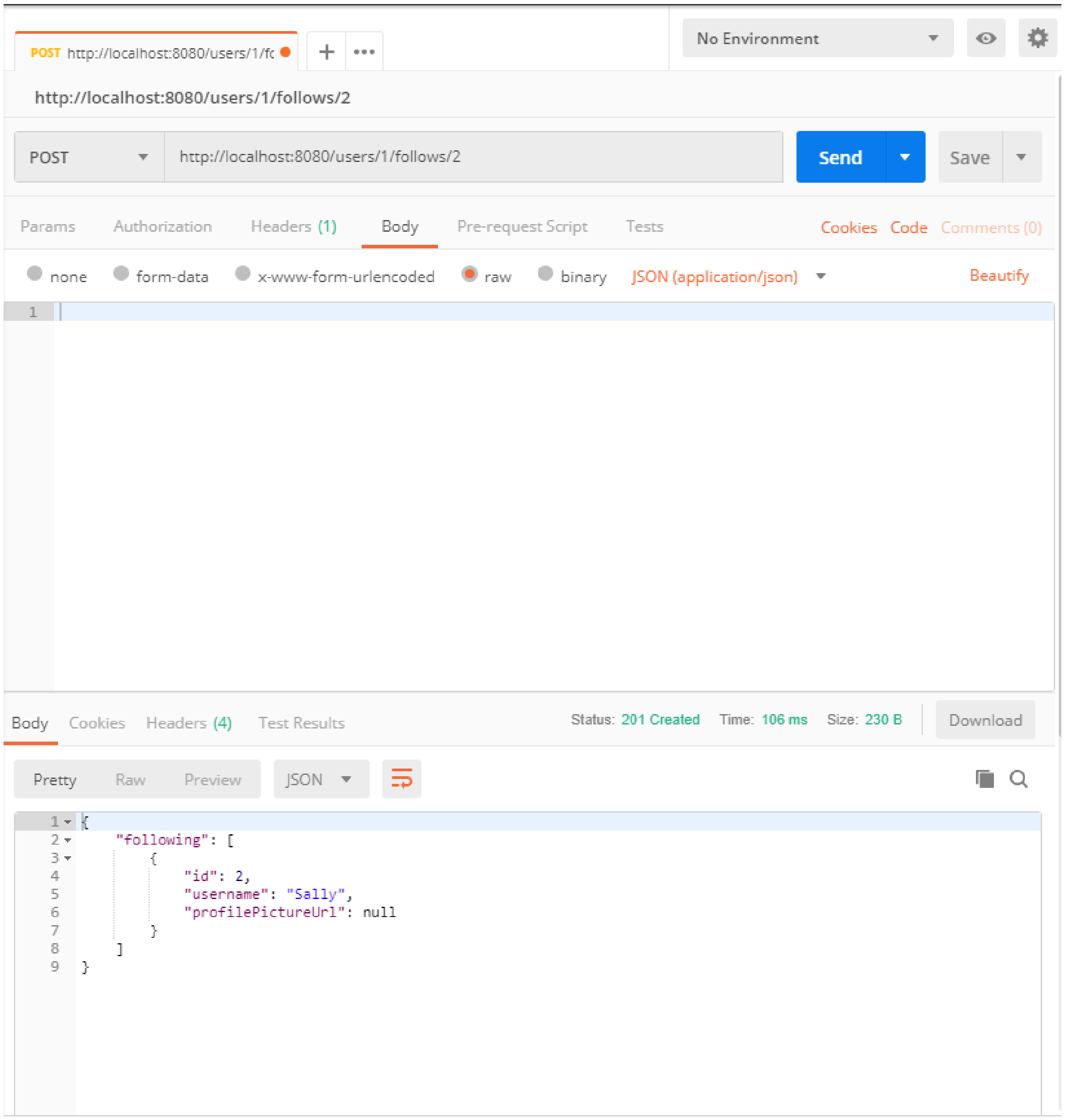
JSON

```
{
    "id": 1,
    "username": "Tim",
    "profilePictureUrl": null
}
```



```
{
    "username": "Sally",
    "password": "12345"
}
```

```
{
    "id": 2,
    "username": "Sally",
    "profilePictureUrl": null
}
```

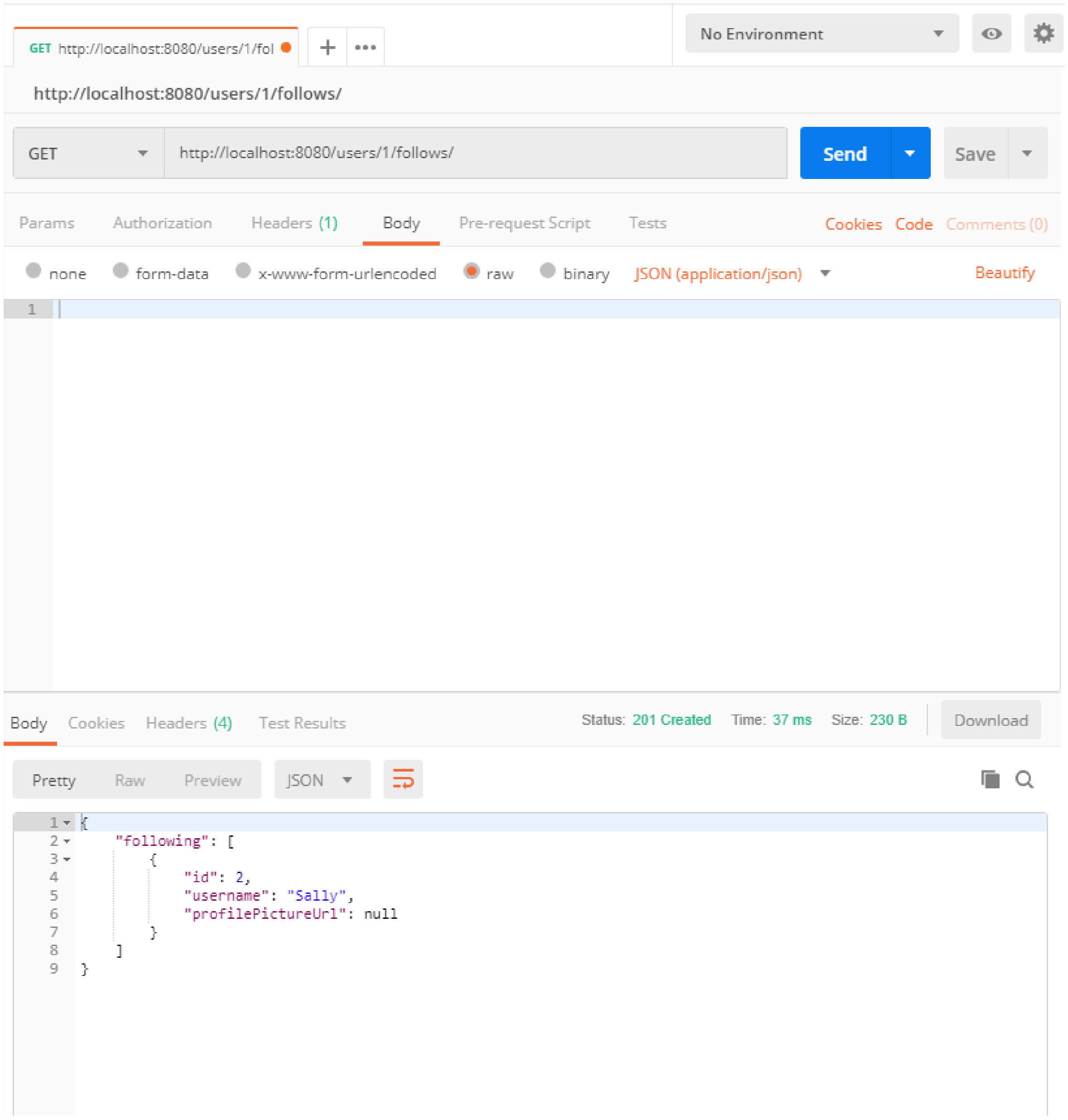


POST http://localhost:8080/users/1/follows/2

Body: raw, JSON (application/json)

Status: 201 Created Time: 106 ms Size: 230 B

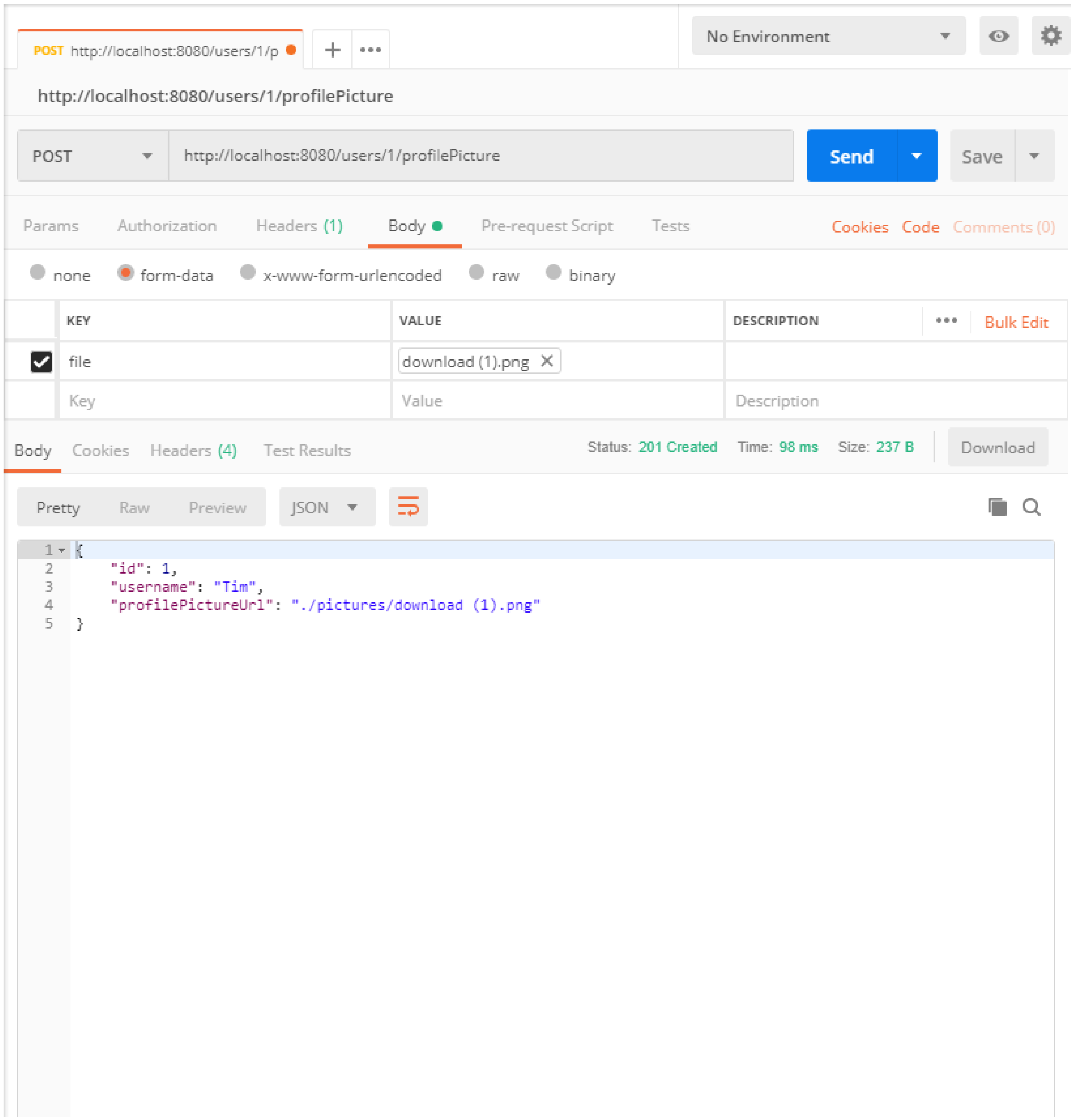
```
{
    "following": [
        {
            "id": 2,
            "username": "Sally",
            "profilePictureUrl": null
        }
    ]
}
```


GET http://localhost:8080/users/1/follows/

Body: raw, JSON (application/json)

Status: 201 Created Time: 37 ms Size: 230 B

```
{
    "following": [
        {
            "id": 2,
            "username": "Sally",
            "profilePictureUrl": null
        }
    ]
}
```

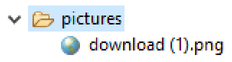

POST http://localhost:8080/users/1/profilePicture

Body: form-data

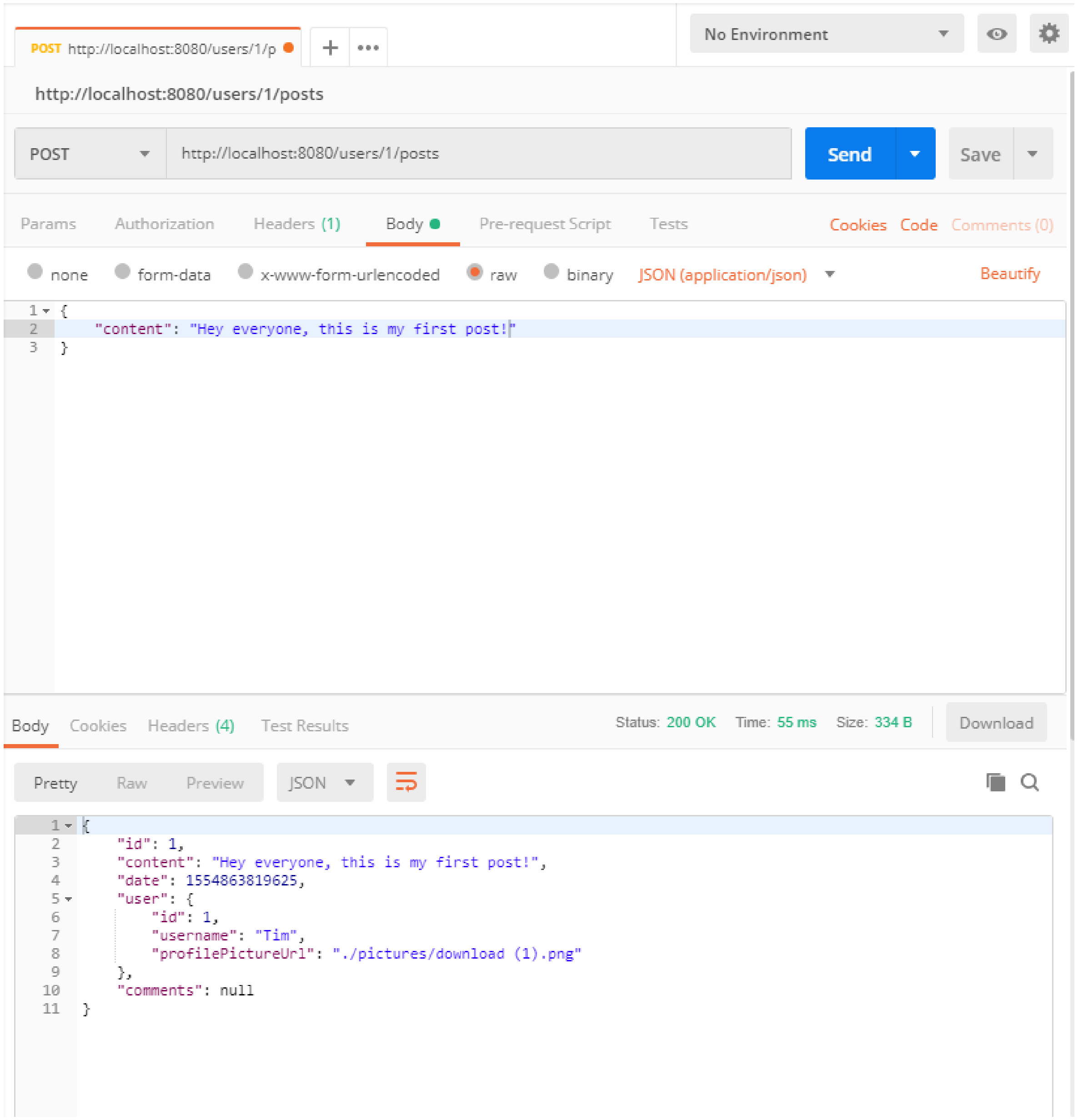
	KEY	VALUE	DESCRIPTION
✓	file	download (1).png	

Status: 201 Created Time: 98 ms Size: 237 B

```
{
    "id": 1,
    "username": "Tim",
    "profilePictureUrl": "./pictures/download (1).png"
}
```



NOTE: The image you uploaded shows up in your pictures folder.



http://localhost:8080/users/1/posts

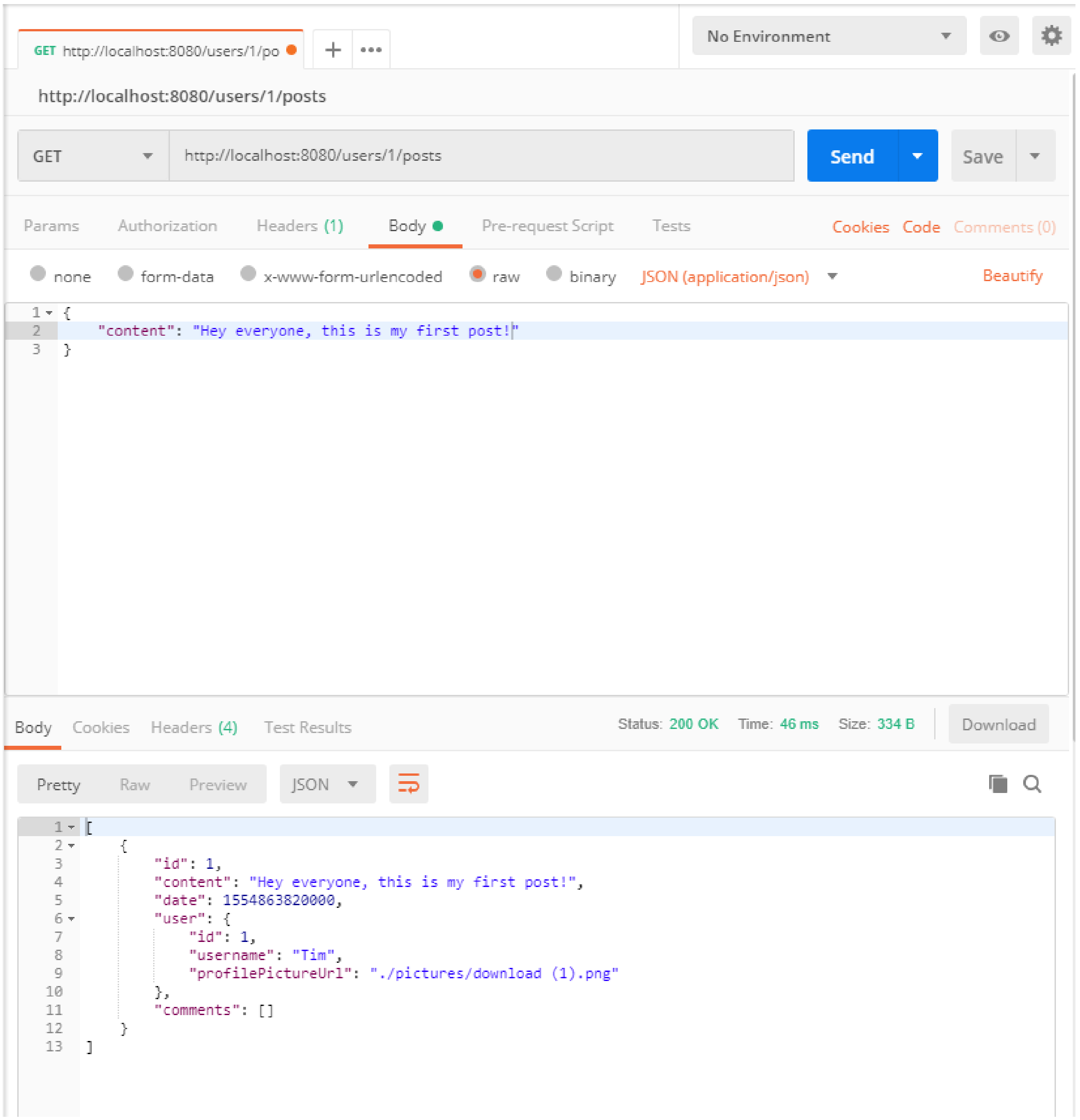
POST http://localhost:8080/users/1/posts

[✓ raw] JSON (application/json)

```
{
    "content": "Hey everyone, this is my first post!"
}
```

Status: 200 OK Time: 55 ms Size: 334 B

```
{
    "id": 1,
    "content": "Hey everyone, this is my first post!",
    "date": 1554863819625,
    "user": {
        "id": 1,
        "username": "Tim",
        "profilePictureUrl": "./pictures/download (1).png"
    },
    "comments": null
}
```



http://localhost:8080/users/1/posts

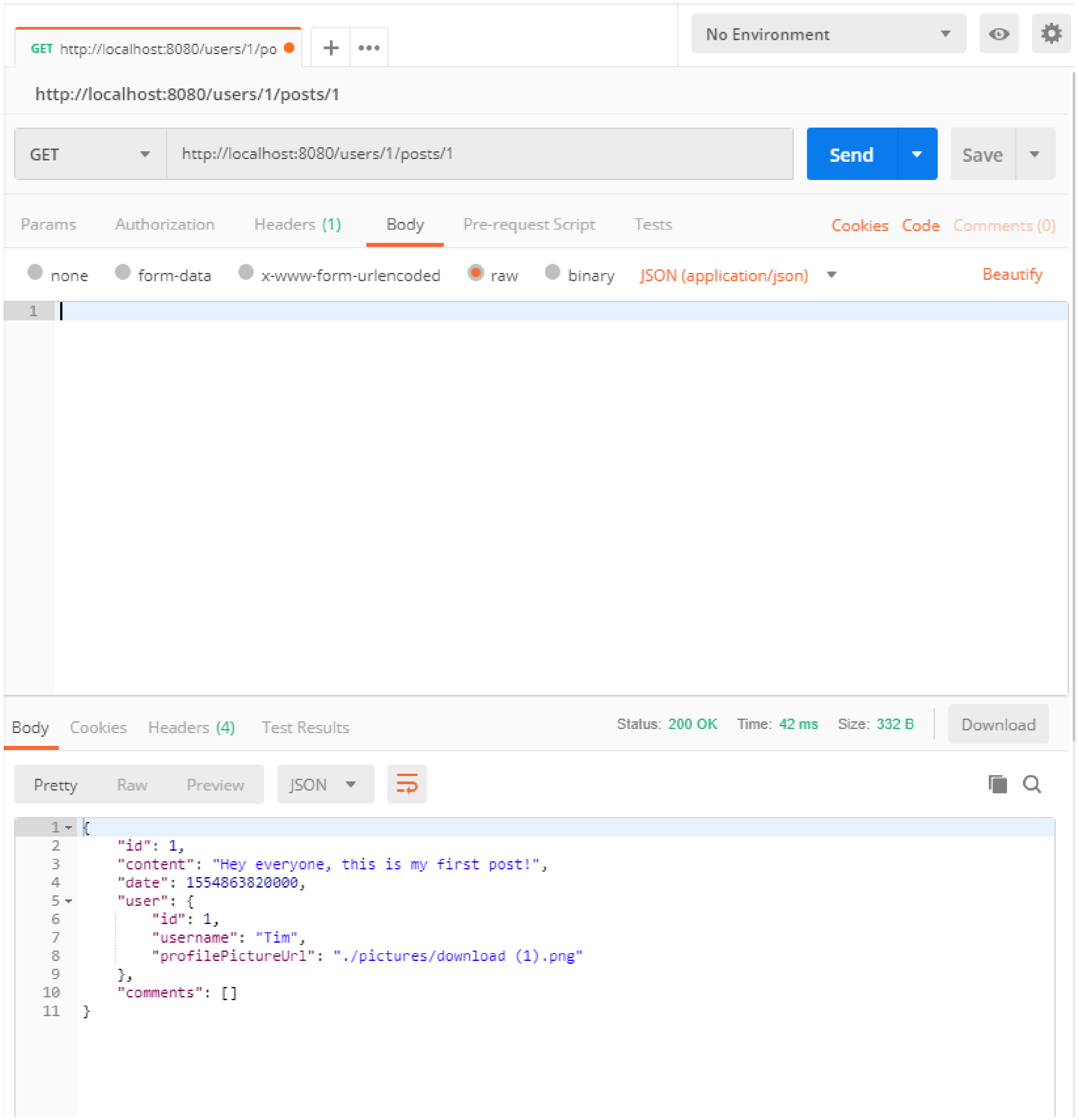
GET http://localhost:8080/users/1/posts

Body: [✓ raw] JSON (application/json)

```
{
    "content": "Hey everyone, this is my first post!"
}
```

Status: 200 OK Time: 46 ms Size: 334 B

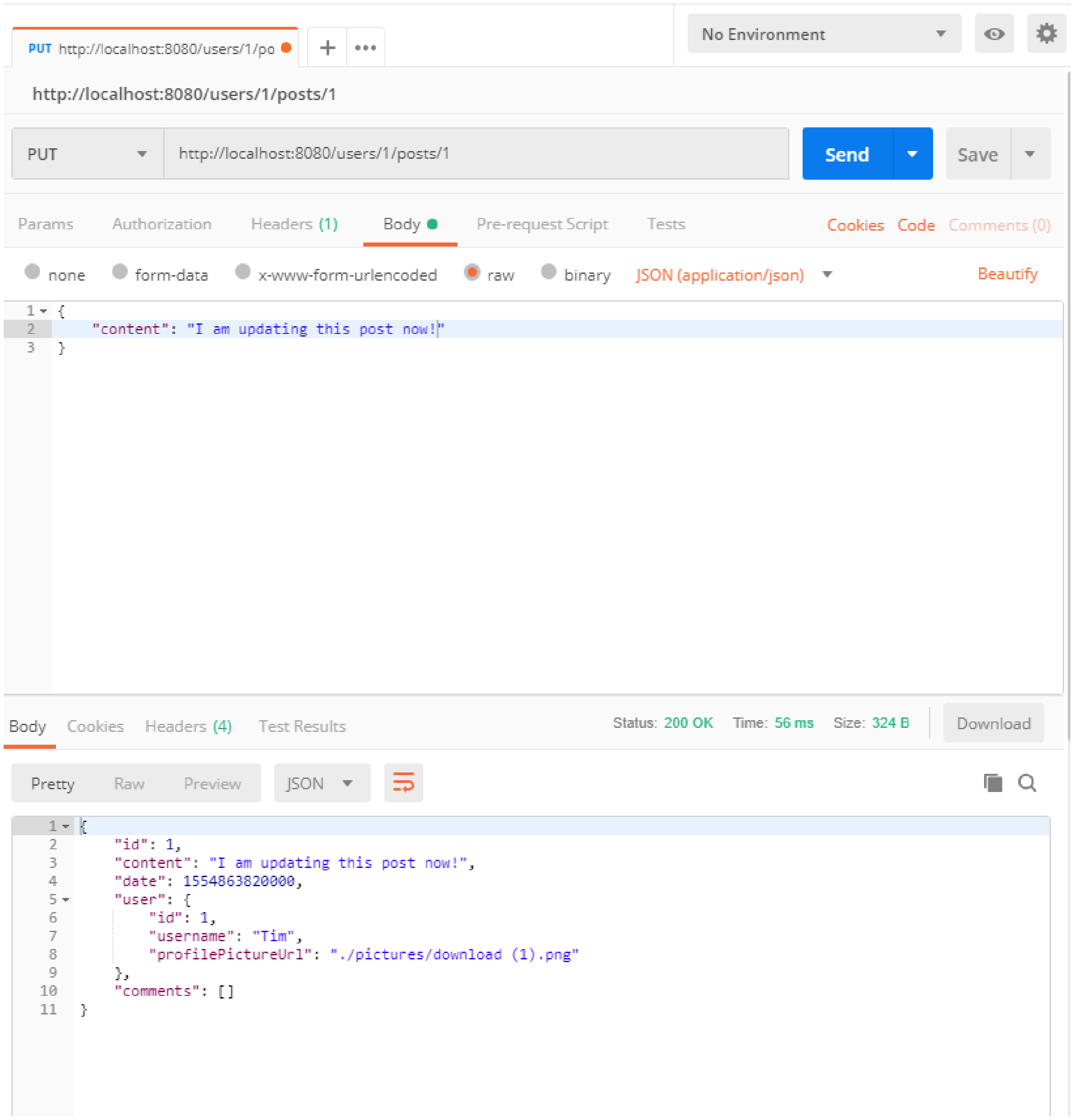
```
[
    {
        "id": 1,
        "content": "Hey everyone, this is my first post!",
        "date": 1554863820000,
        "user": {
            "id": 1,
            "username": "Tim",
            "profilePictureUrl": "./pictures/download (1).png"
        },
        "comments": []
    }
]
```



GET http://localhost:8080/users/1/posts/1

Status: 200 OK Time: 42 ms Size: 332 B

```
{
    "id": 1,
    "content": "Hey everyone, this is my first post!",
    "date": 1554863820000,
    "user": {
        "id": 1,
        "username": "Tim",
        "profilePictureUrl": "./pictures/download (1).png"
    },
    "comments": []
}
```



http://localhost:8080/users/1/posts/1

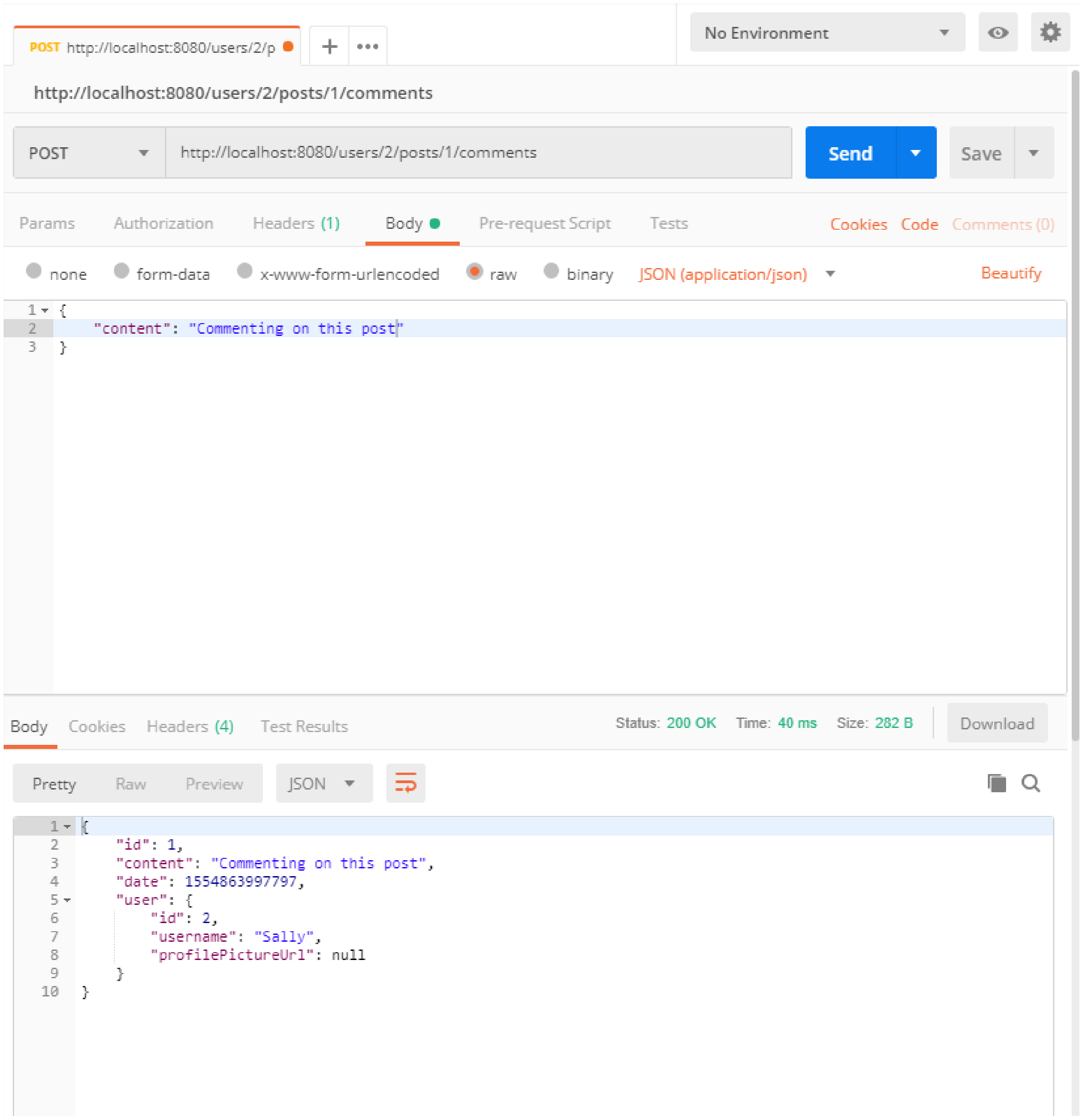
PUT http://localhost:8080/users/1/posts/1

Body: raw, JSON (application/json)

```
{
    "content": "I am updating this post now!"
}
```

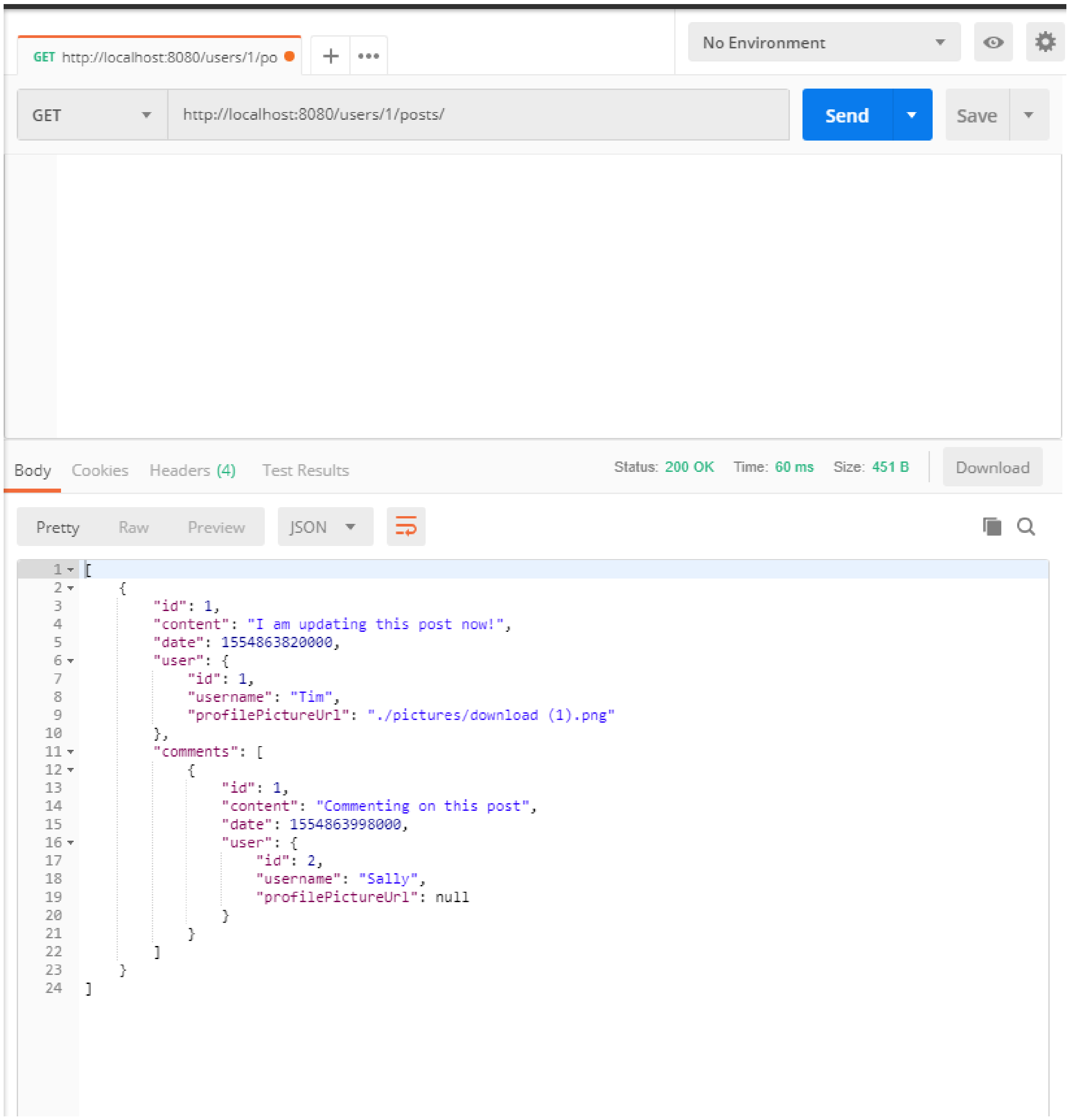
Status: 200 OK Time: 56 ms Size: 324 B

```
{
    "id": 1,
    "content": "I am updating this post now!",
    "date": 1554863820000,
    "user": {
        "id": 1,
        "username": "Tim",
        "profilePictureUrl": "./pictures/download (1).png"
    },
    "comments": []
}
```



```
{
    "content": "Commenting on this post"
}
```

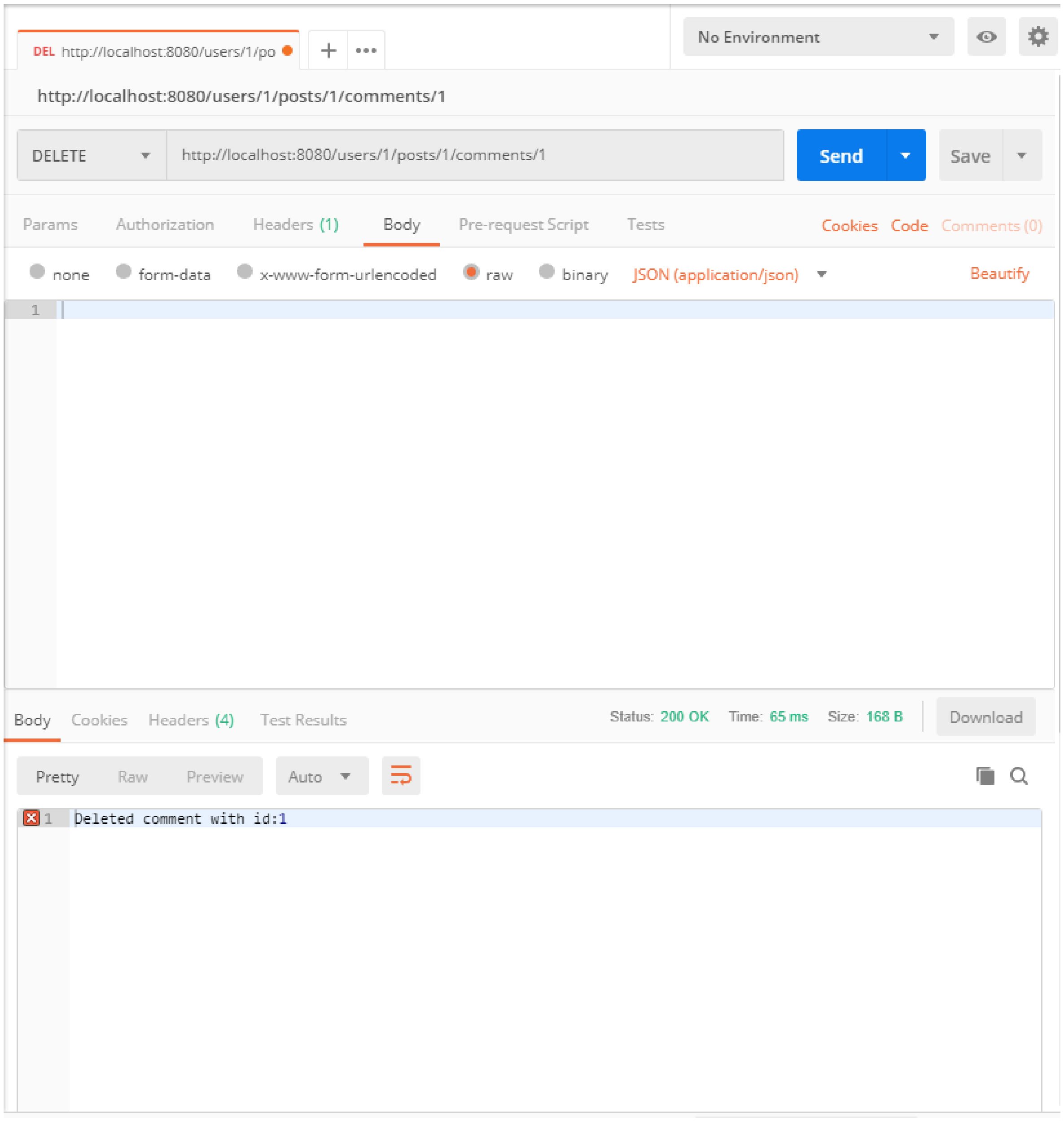
```
{
    "id": 1,
    "content": "Commenting on this post",
    "date": 1554863997797,
    "user": {
        "id": 2,
        "username": "Sally",
        "profilePictureUrl": null
    }
}
```



GET http://localhost:8080/users/1/posts/

Status: 200 OK Time: 60 ms Size: 451 B

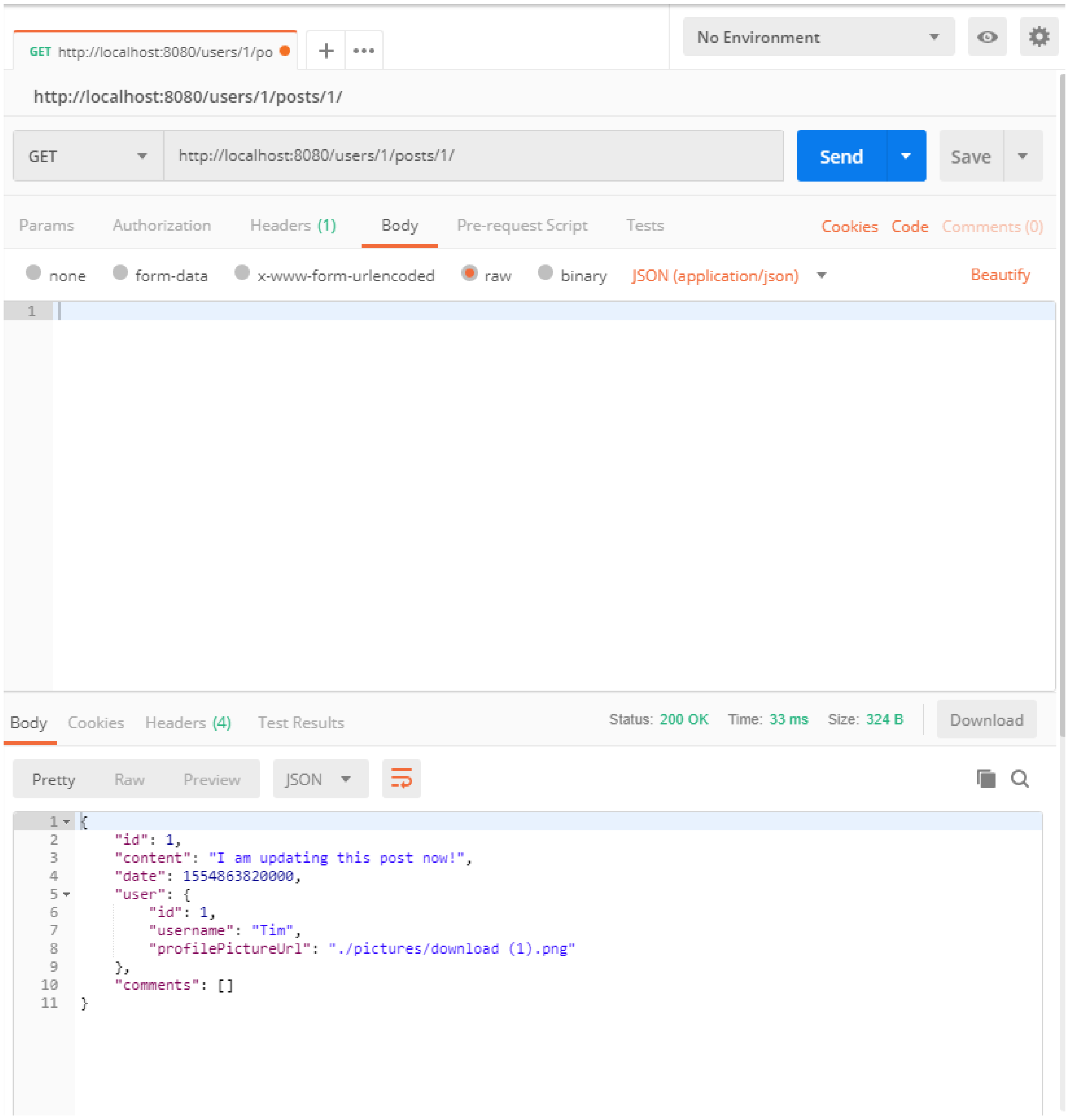
```
[
    {
        "id": 1,
        "content": "I am updating this post now!",
        "date": 1554863820000,
        "user": {
            "id": 1,
            "username": "Tim",
            "profilePictureUrl": "./pictures/download (1).png"
        },
        "comments": [
            {
                "id": 1,
                "content": "Commenting on this post",
                "date": 1554863998000,
                "user": {
                    "id": 2,
                    "username": "Sally",
                    "profilePictureUrl": null
                }
            }
        ]
    }
]
```



DELETE http://localhost:8080/users/1/posts/1/comments/1

Status: 200 OK Time: 65 ms Size: 168 B

```
Deleted comment with id:1
```



GET http://localhost:8080/users/1/posts/1/

Body: raw, JSON (application/json)

Status: 200 OK Time: 33 ms Size: 324 B

```
{
    "id": 1,
    "content": "I am updating this post now!",
    "date": 1554863820000,
    "user": {
        "id": 1,
        "username": "Tim",
        "profilePictureUrl": "./pictures/download (1).png"
    },
    "comments": []
}
```

And that's it! We've just created our Social Media API!

Last modified: Tuesday, 16 April 2019, 11:15 AM

◄ Week 1 Coding Assignment

Jump to...

LinkedIn Posts Week 1 ►

NAVIGATION

- Dashboard
  - Site home
  - Site pages
  - My courses
    - Java
    - MySQL

- spring boot
  - Participants
  - Badges
  - Competencies
  - Grades
  - General
  - Topic 1
    - Videos
    - Previous Recording 2021-01-26
    - Week 1 Research Discussion
    - Week 1 Coding Assignment
    - **Social Media API Tutorial**
    - LinkedIn Posts Week 1
    - Career Services Assignment 7 - Resume
  - Topic 2
  - Topic 3
  - Topic 4
  - Topic 5
  - Topic 6
- career services

You are logged in as Renee Dubuc (Log out)

English (en)
አማርኛ (am)
Afaan Oromoo (om)
Afrikaans (af)
Aragonés (an)
Aranés (oc_es)
Asturianu (ast)
Azərbaycanca (az)
Bahasa Melayu (ms)
Bairisch (bar)
Bamanankan (bm)
Bislama (bi)
Bosanski (bs)
Breizh (br)
Català (ca)
Català (Valencià) (ca_valencia)
Català per a Workplace (ca_wp)
Čeština (cs)
Crnogorski (mis)
Cymraeg (cy)
Dansk (da)
Dansk (kursus) (da_kursus)
Dansk Rum (da_rum)
Davvisámegiella (se)
Deutsch - Du (de_du)
Deutsch - Kids (de_kids)
Deutsch - Schweiz (de_ch)
Deutsch (de_wp)
Deutsch (de)
Deutsch community (de_comm)
Dolnoserbski (dsb)
Ebon (mh)
eesti (et)
English - Pirate (en_ar)
English - United States (en_us)
English (en)
English for kids (en_kids)

English for Workplace - United States (en_us_wp)
English for Workplace (en_wp)
English US - K12 (en_us_k12)
Español - Colombia (es_co)
Español - Internacional (es)
Español - México (es_mx)
Español - México para niños (es_mx_kids)
Español - Venezuela (es_ve)
Español para la Empresa (es_wp)
Esperanto (eo)
Euskara (eu)
Èʋegbe (ee)
Filipino (fil)
Finlandssvenska (sv_fi)
Føroyskt (fo)
Français - Canada (fr_ca)
Français (écriture inclusive) (fr_incl)
Français (fr)
Français pour Workplace (fr_wp)
Gaeilge (ga)
Gàidhlig (gd)
Galego (gl)
Gascon (oc_gsc)
Hausa (ha)
Hrvatski (hr_schools)
Hrvatski (hr)
ʻŌlelo Hawaiʻi (haw)
Igbo (ig)
Indonesian (id)
isiZulu (zu)
Íslenska (is)
Italiano (it)
Italiano per Workplace (it_wp)
Japanese - kids (ja_kids)
Kalaallisut (kl)
Kinyarwanda (rw)
Kiswahili (sw)
Kreyòl Ayisyen (hat)
Kurmanji (kmr)
Kvääni (fkv)
Laotian (lo)
Latin (la)
Latviešu (lv)
Lengadocian (oc_lnc)
Lëtzebuergesch (lb)
Lietuvių (lt)
Lithuanian (university) (lt_uni)
Lulesamisk (smj)
magyar (hu)
Malagasy (mg)
Māori - Tainui (mi_tn)
Māori - Waikato (mi_wwow)
Māori (Te Reo) (mi)
Mapudungún (arn)
Mirandés (mwl)
Mongolian (mn_mong)
Nederlands (nl)
Nederlands Workplace (nl_wp)
Norsk - bokmål (no)
Norsk - nynorsk (nn)
Norsk (no_gr)
Norsk bokmål (nb)
Norsk Workplace (no_wp)
O'zbekcha (uz)
Pidgin (pcm)
Polski (pl)
Português - Brasil (pt_br)
Português - Portugal (pt)
Português para Workplace (pt_br_wp)

Português para Workplace (pt_br_wp)
Qaraqalpaq tili (kaa)
Română (ro)
Romansh Sursilvan (rm_surs)
Samoan (sm)
Setswana (tn)
Shqip (sq)
Sicilianu (scn)
Slovenčina (sk)
Slovenščina (sl)
Soomaali (so)
Sørsamisk (sma)
Srpski (sr_lt)
Suomi (fi)
Suomi+ (fi_co)
Svenska (sv)
Tagalog (tl)
Tamil (ta)
Taqbaylit (kab)
Thai (th)
Tok Pisin (tpi)
Tongan (to)
Türkçe (tr)
Turkmen (tk)
Uyghur - latin (ug_lt)
VakaViti (fj)
Valencià_RACV (ca_valencia_racv)
Vietnamese (vi)
Wolof (wo)
Workplace în limba română (ro_wp)
Workplace 简体中文版 (zh_cn_wp)
Workplace 繁體中文版 (zh_tw_wp)
Workplace準拠の日本語 (ja_wp)
Ελληνικά (el)
Ελληνικά για παιδιά (el_kids)
Ελληνικά για σχολές (el_uni)
Ελληνικά για Χώρους Εργασίας (el_wp)
Башкорт теле (ba)
Беларуская (be)
Български (bg)
Кыргызча (ky)
Қазақша (kk)
Македонски (mk)
Монгол (mn)
Русский (ru)
Српски (sr_cr)
татар теле (tt)
Тоҷикӣ (tg)
Українська (uk)
ქართული (ka)
Հայերեն (hy)
עברית (he)
עברית בתי־ספר (he_kids)
עברית עבור Workplace (he_wp)
ئۇيغۇرچە (ug_ug)
اردو (ur)
بالعربي workplace (ar_wp)
پښتو (ps)
دری (prs)
سنڌي (sd_ap)
سۆرانی (ckb)
عربي (ar)
فارسی (fa)
ދިވެހި (dv)
ⵜⴰⵎⴰⵣⵉⵖⵜ (zgh)
ትግርኛ (ti)
नेपाली (ne)
बच्चों के लिए हिंदी (hi_kids)
मराठी (mr)