CALLICODER

Spring Boot + Spring Security + JWT + MySQL + React Full Stack Polling App - Part 1

Rajeev Singh • Spring Boot • Feb 5, 2018 • 9 mins read

Hello and Welcome to the first part of an exciting series of blog posts where you will learn how to build an end-to-end full stack polling app similar to twitter polls.

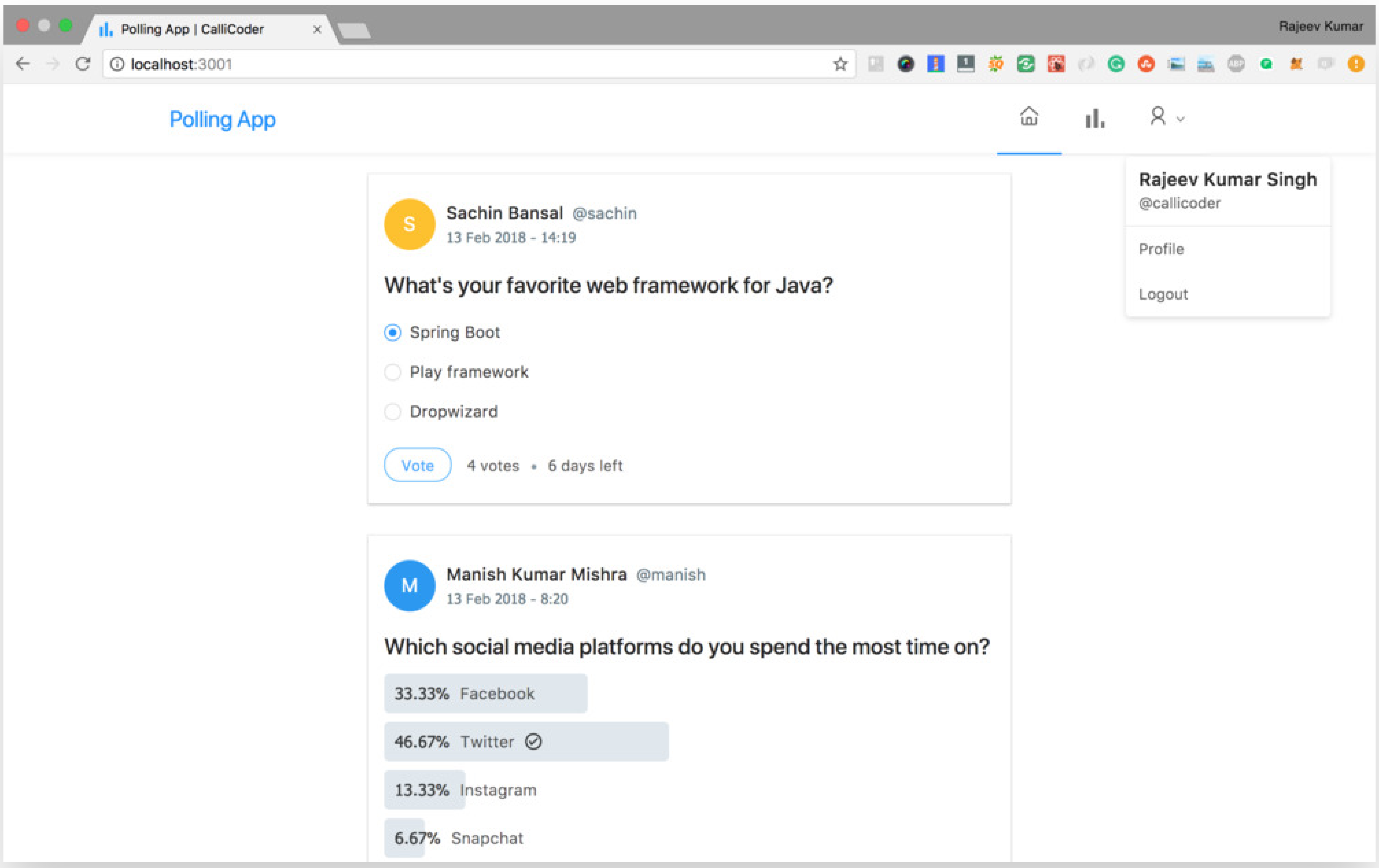
We'll build the backend server using Spring Boot where we'll use Spring Security along with JWT authentication. We'll use MySQL database for storage.

The front-end application will be built using React. We'll also use Ant Design for designing our user interface.

In the end of this tutorial series, you'll have built a fully-fledged polling application from scratch *like a boss*.

> The complete source code of the project is hosted on Github. You can refer that anytime if you get stuck at something.

Following is the screenshot of the final version of our application -



Sachin Bansal @sachin
13 Feb 2018 - 14:19

What's your favorite web framework for Java?

- (●) Spring Boot
- ( ) Play framework
- ( ) Dropwizard

Vote 4 votes • 6 days left

Manish Kumar Mishra @manish
13 Feb 2018 - 8:20

Which social media platforms do you spend the most time on?

33.33% Facebook
46.67% Twitter ✓
13.33% Instagram
6.67% Snapchat

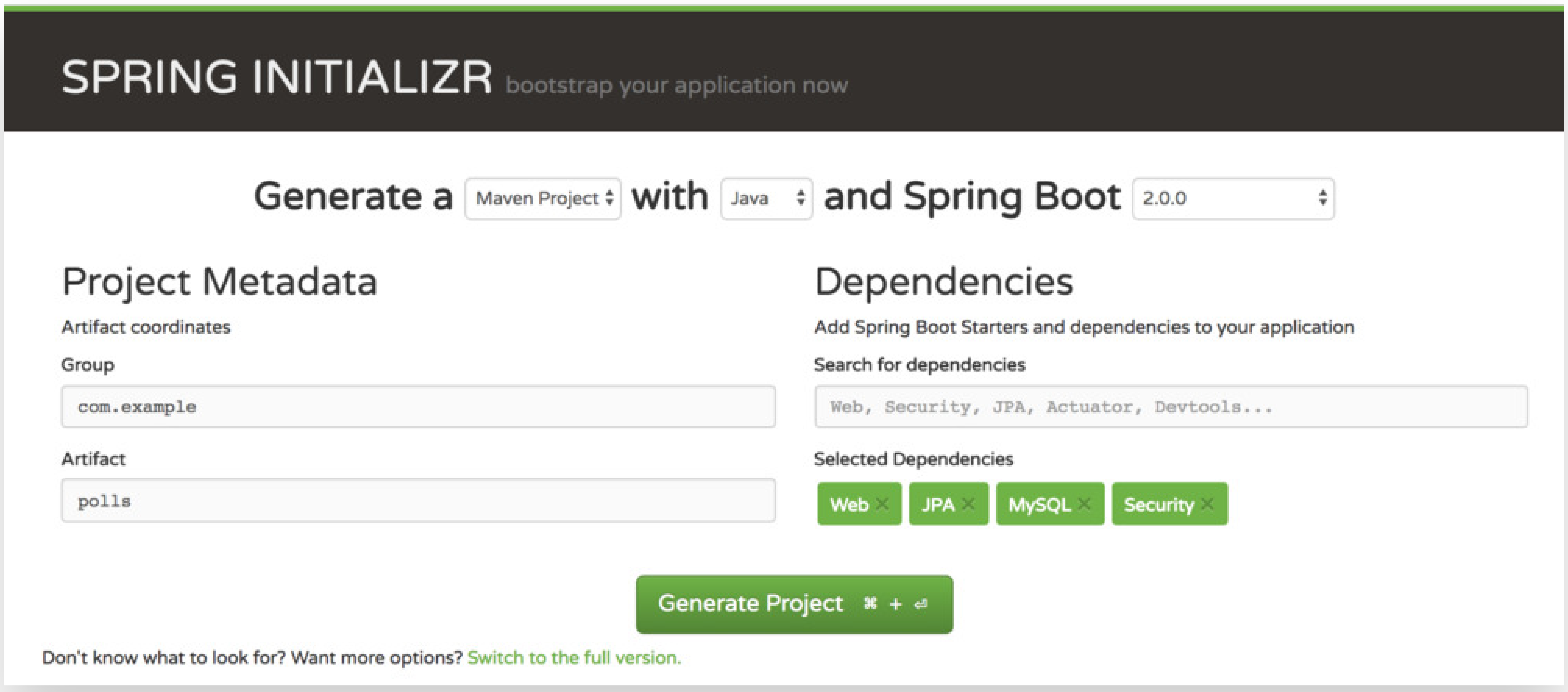
Looks great, isn't it? Well, then let's start building it from scratch...

In this article, We'll set up the backend project using Spring Boot and define the basic domain models and repositories.

Creating the Backend Application using Spring Boot

Let's bootstrap the project using Spring Initialzr web tool -

1. Open http://start.spring.io
2. Enter **polls** in Artifact field.



SPRING INITIALIZR bootstrap your application now

Generate a Maven Project with Java and Spring Boot 2.0.0

Project Metadata

Artifact coordinates

Group

com.example

Artifact

polls

Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies

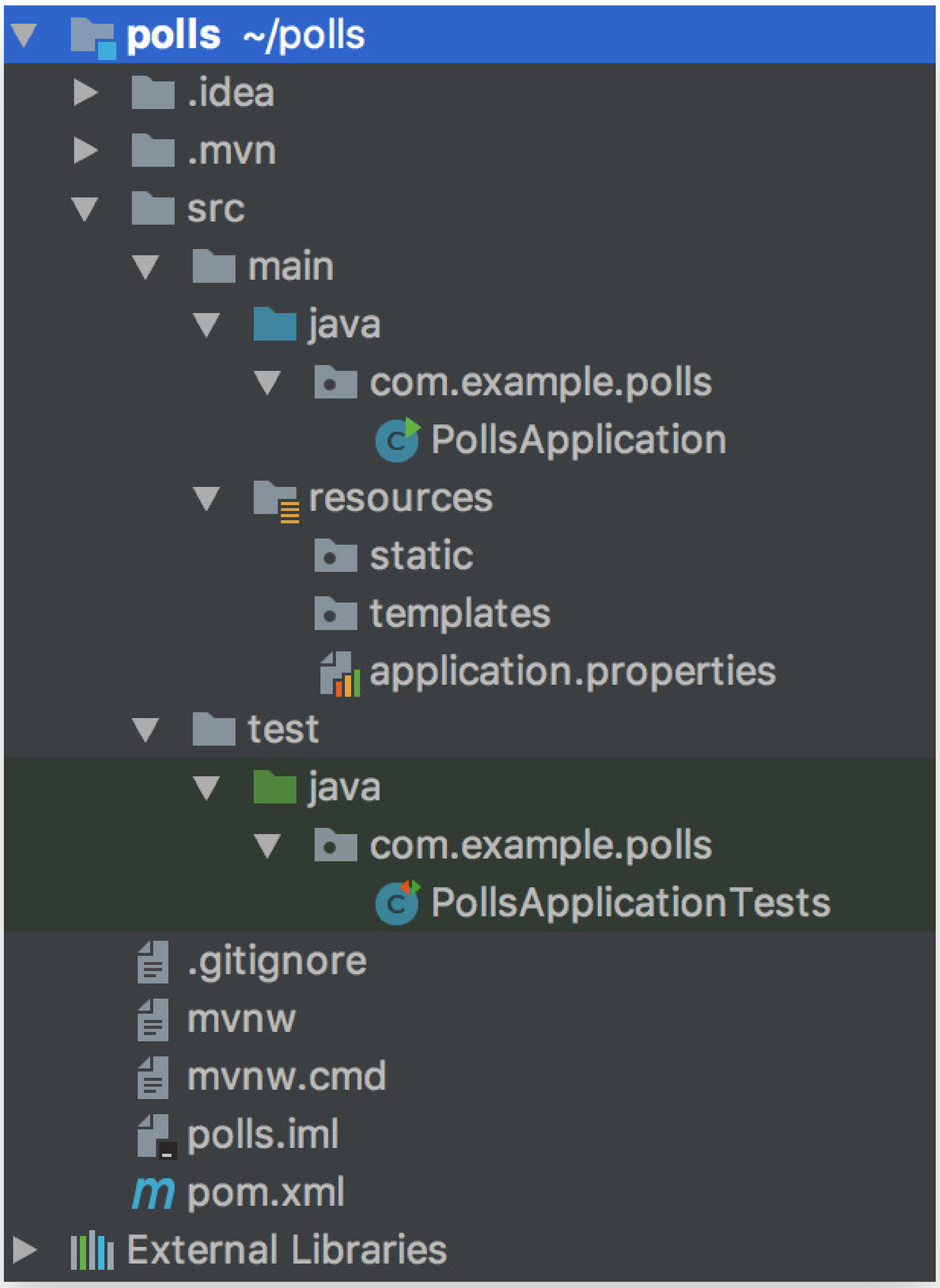
Web, Security, JPA, Actuator, Devtools...

Selected Dependencies

Web JPA MySQL Security

Don't know what to look for? Want more options? Switch to the full version.

Once the project is downloaded, unzip it and import it into your favorite IDE. The directory structure of the project will look like this-



```
polls ~/polls
  .idea
  .mvn
  src
    main
      java
        com.example.polls
          PollsApplication
      resources
        static
        templates
        application.properties
    test
      java
        com.example.polls
          PollsApplicationTests
  .gitignore
  mvnw
  mvnw.cmd
  polls.iml
  pom.xml
External Libraries
```

We'll need to add few additional dependencies to our project. Open `pom.xml` file from the root directory of your generated project and add the following to the `<dependencies>` section -

```xml
<!-- For Working with Json Web Tokens (JWT) -->
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.0</version>
</dependency>

<!-- For Java 8 Date/Time Support -->
<dependency>
    <groupId>com.fasterxml.jackson.datatype</groupId>
    <artifactId>jackson-datatype-jsr310</artifactId>
</dependency>
```

Configuring the Server, Database, Hibernate and Jackson

Let's now configure the server, database, hibernate, and jackson by adding the following properties to the `src/main/resources/application.properties` file -

```
## Server Properties
server.port= 5000

## Spring DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)
spring.datasource.url= jdbc:mysql://localhost:3306/polling_app?useSSL=false&serverTimezone=UTC&useLegacyDatetimeCode=false
spring.datasource.username= root
spring.datasource.password= callicoder

## Hibernate Properties

# The SQL dialect makes Hibernate generate better SQL for the chosen database
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect
spring.jpa.hibernate.ddl-auto = update

## Hibernate Logging
logging.level.org.hibernate.SQL= DEBUG

# Initialize the datasource with available DDL and DML scripts
spring.datasource.initialization-mode=always

## Jackson Properties
spring.jackson.serialization.WRITE_DATES_AS_TIMESTAMPS= false
spring.jackson.time-zone= UTC
```

All the above properties are self-explanatory. I've set hibernate's `ddl-auto` property to `update`. This will automatically create/update the tables in the database according to the entities in our application.

The Jackson's `WRITE_DATES_AS_TIMESTAMPS` property is used to disable serializing Java 8 Data/Time values as timestamps. All the Date/Time values will be serialized to ISO date/time string.

Before proceeding further, please create a database named `polling_app` in MySQL and change the `spring.datasource.username` and `spring.datasource.password` properties as per your MySQL installation.

We'll be using Java 8 Data/Time classes in our domain models. We'll need to register JPA 2.1 converters so that all the Java 8 Date/Time fields in the domain models automatically get converted to SQL types when we persist them in the database.

Moreover, We'll set the default timezone for our application to UTC.

Open the main class `PollsApplication.java` and make the following modifications to it-

```java
package com.example.polls;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.autoconfigure.domain.EntityScan;
import org.springframework.data.jpa.convert.threeten.Jsr310JpaConverters;

import javax.annotation.PostConstruct;
import java.util.TimeZone;

@SpringBootApplication
@EntityScan(basePackageClasses = {
        PollsApplication.class,
        Jsr310JpaConverters.class
})
public class PollsApplication {

    @PostConstruct
    void init() {
        TimeZone.setDefault(TimeZone.getTimeZone("UTC"));
    }

    public static void main(String[] args) {
        SpringApplication.run(PollsApplication.class, args);
    }
}
```

Creating the domain models

Our application will allow new users to register and login to our application. Every User will have one or more roles. The roles associated with a user will be used in future to decide whether the user is authorized to access a particular resource on our server or not.

In this section, We'll create the `User` and `Role` domain models. All the domain models will be stored in a package named `model` inside `com.example.polls`.

1. `User` model

The `User` model contains the following fields -

1. `id` : Primary Key
2. `username` : A unique username
3. `email` : A unique email
4. `password` : A password which will be stored in encrypted format.
5. `roles` : A set of roles. (Many-To-Many relationship with `Role` entity)

Here is the complete `User` class -

```
import com.example.polls.model.audit.DateAudit;
import org.hibernate.annotations.NaturalId;
import javax.persistence.*;
import javax.validation.constraints.Email;
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.Size;
import java.util.HashSet;
import java.util.Set;

@Entity
@Table(name = "users", uniqueConstraints = {
        @UniqueConstraint(columnNames = {
            "username"
        }),
        @UniqueConstraint(columnNames = {
            "email"
        })
})
public class User extends DateAudit {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotBlank
    @Size(max = 40)
    private String name;

    @NotBlank
    @Size(max = 15)
    private String username;

    @NaturalId
    @NotBlank
    @Size(max = 40)
    @Email
    private String email;

    @NotBlank
    @Size(max = 100)
    private String password;

    @ManyToMany(fetch = FetchType.LAZY)
    @JoinTable(name = "user_roles",
            joinColumns = @JoinColumn(name = "user_id"),
            inverseJoinColumns = @JoinColumn(name = "role_id"))
    private Set<Role> roles = new HashSet<>();

    public User() {

    }

    public User(String name, String username, String email, String password) {
        this.name = name;
        this.username = username;
        this.email = email;
        this.password = password;
    }
    }

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

    public String getUsername() {
        return username;
    }

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

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPassword() {
        return password;
    }

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

    public Set<Role> getRoles() {
        return roles;
    }

    public void setRoles(Set<Role> roles) {
        this.roles = roles;
    }
}
```

The `User` class extends the `DateAudit` class that we'll define shortly. The `DateAudit` class will have `createdAt` and `updatedAt` fields that will be used for auditing purposes.

2. `Role` model

The `Role` class contains an `id` and a `name` field. The `name` field is an `enum`. We'll have a fixed set of pre-defined roles. So it makes sense to make the role name as `enum`.

Here is the complete code for `Role` class -

```
import org.hibernate.annotations.NaturalId;
import javax.persistence.*;

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

    @Enumerated(EnumType.STRING)
    @NaturalId
    @Column(length = 60)
    private RoleName name;

    public Role() {

    }

    public Role(RoleName name) {
        this.name = name;
    }

    public Long getId() {
        return id;
    }

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

    public RoleName getName() {
        return name;
    }

    public void setName(RoleName name) {
        this.name = name;
    }
}
```

RoleName enum

Following is the `RoleName` enum -

```
package com.example.polls.model;

public enum  RoleName {
    ROLE_USER,
    ROLE_ADMIN
}
```

I have defined two roles namely `ROLE_USER` and `ROLE_ADMIN`. You're free to add more roles as per your project requirements.

3. `DateAudit` model

We'll use JPA's `AuditingEntityListener` to automatically populate `createdAt` and `updatedAt` values when we persist an entity.

Here is the Complete `DateAudit` class (I've created a package named `audit` inside `com.example.polls.model` package to store all the auditing related models) -

```
package com.example.polls.model.audit;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import org.springframework.data.annotation.CreatedDate;
import org.springframework.data.annotation.LastModifiedDate;
import org.springframework.data.jpa.domain.support.AuditingEntityListener;
import javax.persistence.Column;
import javax.persistence.EntityListeners;
import javax.persistence.MappedSuperclass;
import java.io.Serializable;
import java.time.Instant;

@MappedSuperclass
@EntityListeners(AuditingEntityListener.class)
@JsonIgnoreProperties(
        value = {"createdAt", "updatedAt"},
        allowGetters = true
)
public abstract class DateAudit implements Serializable {

    @CreatedDate
    @Column(nullable = false, updatable = false)
    private Instant createdAt;

    @LastModifiedDate
    @Column(nullable = false)
    private Instant updatedAt;

    public Instant getCreatedAt() {
        return createdAt;
    }

    public void setCreatedAt(Instant createdAt) {
        this.createdAt = createdAt;
    }

    public Instant getUpdatedAt() {
        return updatedAt;
    }

    public void setUpdatedAt(Instant updatedAt) {
        this.updatedAt = updatedAt;
    }
}
```

To enable JPA Auditing, we'll need to add `@EnableJpaAuditing` annotation to our main class or any other configuration classes.

Let's create an `AuditingConfig` configuration class and add the `@EnableJpaAuditing` annotation to it.

We're creating a separate class because we'll be adding more auditing related configurations later. So it's better to have a separate class.

```java
package com.example.polls.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.data.jpa.repository.config.EnableJpaAuditing;

@Configuration
@EnableJpaAuditing
public class AuditingConfig {
    // That's all here for now. We'll add more auditing configurations later.
}
```

Creating the Repositories for accessing `User` and `Role` data

Now that we have defined the domain models, Let's create the repositories for persisting these domain models to the database and retrieving them.

All the repositories will go inside a package named `repository`. So let's first create the `repository` package inside `com.example.polls`.

1. `UserRepository`

Following is the complete code for `UserRepository` interface. It extends Spring Data JPA's `JpaRepository` interface.

```java
package com.example.polls.repository;

import com.example.polls.model.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import java.util.List;
import java.util.Optional;

@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByEmail(String email);

    Optional<User> findByUsernameOrEmail(String username, String email);

    List<User> findByIdIn(List<Long> userIds);

    Optional<User> findByUsername(String username);

    Boolean existsByUsername(String username);

    Boolean existsByEmail(String email);
}
```

2. `RoleRepository`

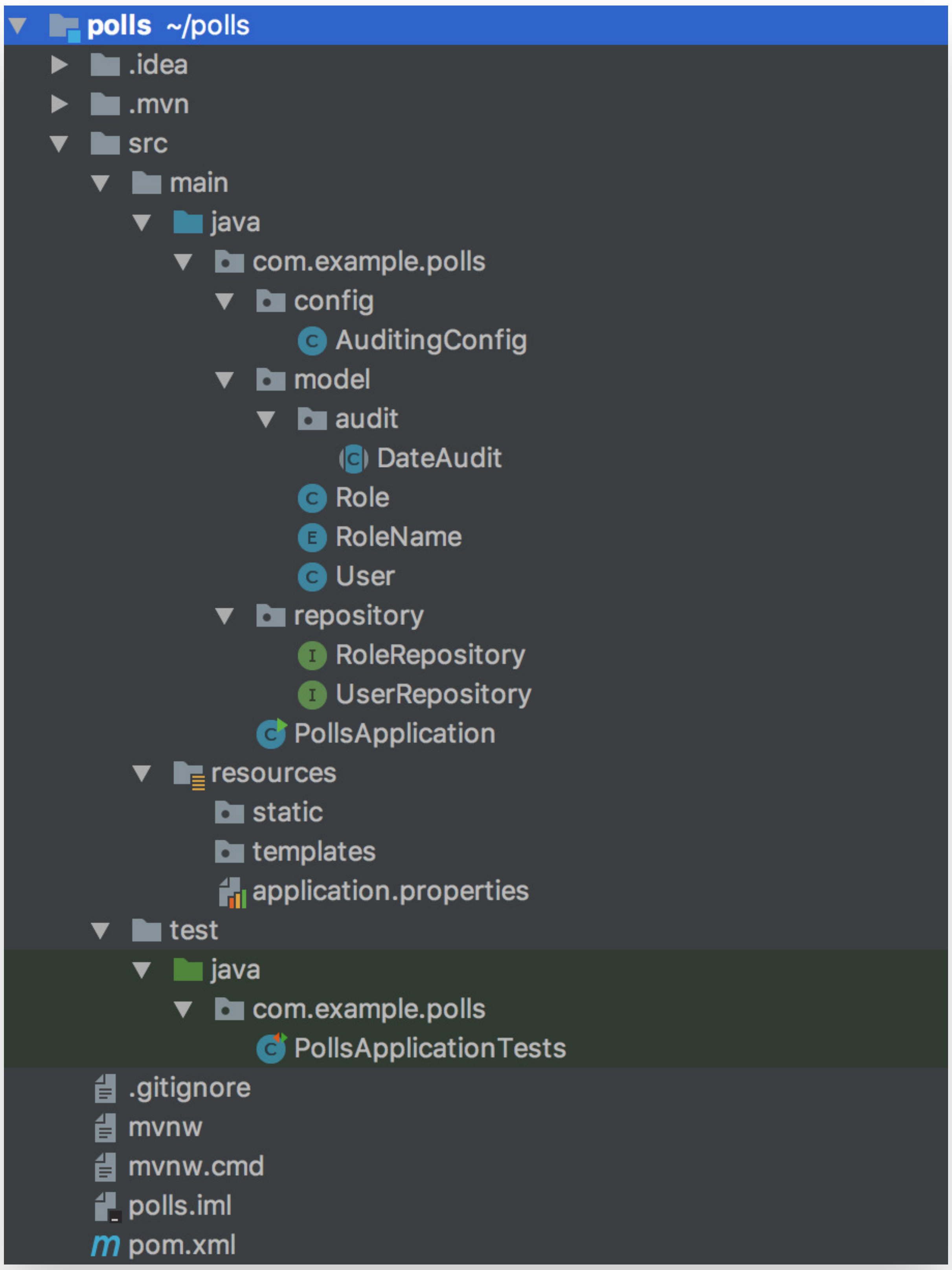
Following is the `RoleRepository` interface. It contains a single method to retrieve a `Role` from the `RoleName` -

```
import com.example.polls.model.Role;
import com.example.polls.model.RoleName;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import java.util.Optional;

@Repository
public interface RoleRepository extends JpaRepository<Role, Long> {
    Optional<Role> findByName(RoleName roleName);
}
```

Exploring the current setup and Running the Application

After creating all the above models, repositories and configurations, our current project should look like this -



```
polls ~/polls
  .idea
  .mvn
  src
    main
      java
        com.example.polls
          config
            AuditingConfig
          model
            audit
              DateAudit
            Role
            RoleName
            User
          repository
            RoleRepository
            UserRepository
          PollsApplication
      resources
        static
        templates
        application.properties
    test
      java
        com.example.polls
          PollsApplicationTests
  .gitignore
  mvnw
  mvnw.cmd
  polls.iml
  pom.xml
```

You can run the application by typing the following command from the root directory of your project -

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

Check out the logs and make sure that the server starts successfully.

Write to me in the comment section, if the server doesn't start successfully for you. I'll help you out.

Creating Default Roles

We'll have a fixed set of predefined roles in our application. Whenever a user logs in, we'll assign `ROLE_USER` to it by default.

For assigning the roles, they have to be present in the database. So let's create the two default roles in the database by executing the following insert statements -

```
INSERT INTO roles(name) VALUES('ROLE_USER');
INSERT INTO roles(name) VALUES('ROLE_ADMIN');
```

What's Next?

In the next chapter of this series, we'll learn how to configure Spring Security in our project and add functionalities to register a new user and log them in.

Read Next: Full Stack Polling App with Spring Boot, Spring Security, JWT, MySQL and React - Part 2

Liked the Article? Share it on Social media!

Twitter Facebook Linkedin Reddit

56 Comments **CalliCoder** **Fredi Tansari**

Recommend 9 Tweet Share **Sort by Newest**

Join the discussion…

Dawid Orzycki • 18 days ago
Hello,

I have issue with project build.
Two errors whas caused by 2 methods in User repository : existByUsername and existByEmail
Once I deleted them project was build.
I use Java 11.
Reply • Share ›

Shay091 ➦ Dawid Orzycki • 7 days ago
I used Java 11 I didnt have that issue
Reply • Share ›

Patrick Shi • 2 months ago
Hi, Rajeev, thanks for this beautiful project. Can I ask how long did it take to create it? front-end and backend respectively
Reply • Share ›

kevin vinograd • 3 months ago
The generated access token tells me that it is invalid when I encode in jwt.io.
can be?
Reply • Share ›

4 ^ | v • Reply • Share ›

Shay091 ➦ Your Mom • 7 days ago

Yeah same, it think its optional

^ | v • Reply • Share ›

Andres Rodriguez • 4 months ago

Amazing tutorial !! I have learned many things !!!! You're the best! I don't have enough words to thank you for your effort !!!! many thanks

1 ^ | v • Reply • Share ›

Anh Tuan Le • 5 months ago

Hello! I have an issue while run application. There it is:
Pls help me

e!

2 ^ | v 1 • Reply • Share ›

Varun Upadhyay • 5 months ago • edited

Facing
`java.sql.SQLException: Access denied for user 'root'@'localhost'`

Using root and password in the properties file. Any help will be appreciated.

^ | v • Reply • Share ›

Hamza Ouni ➦ Varun Upadhyay • 4 months ago

you should verify that username and password in the properties files are compatible with your RDMS(mysql ...) username and password

^ | v • Reply • Share ›

Jamal Fazlur Rahman • 5 months ago

i want to ask something,
when first time i run the application, it can start normally. but at the second time, i got this message:

```
Failed to execute SQL script statement #1 of URL [file:
.../target/classes/data.sql]: INSERT INTO roles(name)
VALUES('ROLE_USER'); nested exception is
java.sql.SQLIntegrityConstraintViolationException: Duplicate entry
'ROLE_USER' for key 'UK_nb4h0p6txrmfc0xbrd1kglp9t' -> [Help 1]
```

If i delete the data.sql, the apps can be run normally..

i have set the application.properties like this:
`spring.jpa.hibernate.ddl-auto = update`

^ | v • Reply • Share ›

Vipul Sharma • 6 months ago

Hey, great tutorial.Just wanted to confirm that we need to create the tables in our database on our own, right? I did not find any mention of it in the tutorial and my application won't start.

^ | v • Reply • Share ›

Rajeev Singh Mod ➦ Vipul Sharma • 6 months ago • edited

You don't need to create tables manually. You just need to create the database and update the username/password in application.properties file.

The following property does the magic of creating the tables when you start the application -

```
spring.jpa.hibernate.ddl-auto = update
```

It finds all the classes annotated with `@Entity` and creates the corresponding tables.

Hey thanks for the reply, it did not create the other tables until i had created the roles table on my own using query first.

Reply • Share ›

Udaya Sooriyan • 6 months ago

when i import this project into eclipse only polling-app-Server is showing... why polling-app-client showing inside eclipse???
which file is calling each other on runtime?

Reply • Share ›

Rajeev Singh Mod → Udaya Sooriyan • 6 months ago

Hi, `polling-app-client` is a react project. Eclipse may not recognize that. I recommend using a code editor like VSCode or Atom for React projects.

Reply • Share ›

Vimal • 8 months ago

Question: Why use AuditingEntityListener to automatically populate createdAt and updatedAt values when we persist an entity, when you can do this in the database itself? Is there a reason for this.

2 • Reply • Share ›

Devon • 9 months ago • edited

Hello!
What is reason of using Role as separate Entity and in this class use RoleName enum, why you didn't put RoleName enum in User entity at once?

1 | 1 • Reply • Share ›

Rajeev Singh Mod → Devon • 6 months ago

Hi,

A given application can have a lot of roles. And every role can have a bunch of permissions associated with them. Although, we don't have permissions in this project, But the project is extendable to use permissions.

Moreover, Roles can be separately added, modified, and managed. You may want to add a new role, remove an existing role, or rename a role. If we put the Role name inside User entity, then if you want to modify a role, you'll have to change all User rows who have that role.

Therefore, It's better to have it as a separate entity. User and Role has a many to many relationship between them.

Thanks,
Rajeev

Reply • Share ›

Marwane Erradja → Devon • 6 months ago

did you have an answer for this ?

Reply • Share ›

Harry Kurniansyah • 9 months ago

Hello Guys,
I have an issue, there it is.
Can you all help me with this issue? I've been tried all of the solutions in stackoveflow, but can't help it.

| 1 • Reply • Share ›

Joe Barne • 9 months ago

Hello all...
This is a beauty of a project! very useful and super for experience.
I was able to build my own secured angular based application and it worked like a charm.

My question is, what modification would I have to make to get working (Beyond the getters and setters)??

If anyone could help I would really appreciate, I've been on to this for quiet sometime now

^ | v • Reply • Share ›

Vipul Sharma → Joe Barne • 6 months ago

how did you create the user table using query?

^ | v • Reply • Share ›

Chris Moore • 10 months ago

This whole series is awesome! I'm about to build my first Java Spring project and I intend to use this tutorial as a guide. Thank you so much for taking the time and effort to do this great piece of work.

One point - '9 min read' - really! :')

^ | v • Reply • Share ›

Vipul Sharma → Chris Moore • 6 months ago

hey, did you create the user table on your own? as in using a query?

^ | v • Reply • Share ›

Eduardo Greco • a year ago

I have this problem.

^ | v • Reply • Share ›

Soumya Ranjan Jena • a year ago

the hierarchy of the type user repository is inconsistent getting error please help me out

^ | v • Reply • Share ›

Valmar Júnior • a year ago

Rajeev, hello again!

One question, how can i achieve the result of "findByUsernameOrEmail" using QueryDSL?

18 ^ | v • Reply • Share ›

Vipul Sharma → Valmar Júnior • 6 months ago

how did you create the user table?

^ | v • Reply • Share ›

Ugogbuzue Okwubanego • a year ago

Many thanks for the insight you provided in this tutorial. I have learnt a whole lot especially JWT authentication using spring boot. Thanks.
Regards,
Jaytee

^ | v • Reply • Share ›

Vipul Sharma → Ugogbuzue Okwubanego • 6 months ago

how did you create the user table using query?

^ | v • Reply • Share ›

Fahad • a year ago

Hello there,

Thank you soo much for this tutorial, it was very amusing and beneficial. I have a question regarding storing the passwords in the database, is that a valid way to do it or should we use something with salt and SH1 encryption? for a production application.

Thanks,
Fahad

Bcrypt is perfectly fine for securing passwords in production applications. Following are some online resources that you can check for more details -

Do any security experts recommend bcrypt for password storage?

Why is BCrypt more secure than just storing a salt and an encrypted password in the database?

Why you should use BCrypt to hash passwords

Cheers,
Rajeev
^ | v 1 • Reply • Share ›

Tuğberk Göç • a year ago
When I run mvn spring-boot:run command I got that problem. Is there anyone knows that?

^ | v • Reply • Share ›

Rajeev Singh Mod → Tuğberk Göç • a year ago
Hi,

Looks like you're using Java 9. The JAXB APIs have been removed from Java 9. Please add them explicitly with the following dependency -

```
<dependency>
    <groupid>javax.xml.bind</groupid>
    <artifactid>jaxb-api</artifactid>
    <version>2.3.0</version>
</dependency>
```

Check this stackoverflow answer for more details.
^ | v • Reply • Share ›

Langton Favor Rebel Mudyiwa • a year ago
Hie, thank you for the article , i have been following along but when i run the programme its giving me the below error:

"Failed to create query for method public abstract java.util.Optional com.springReact.SpringReactdemo.Repository.UserRepository.findByUserNan
Unable to locate Attribute with the the given name [userName] on this ManagedType [com.springReact.SpringReactdemo.Entity.Audit.DateAudit]"

Any idea what causes this?
21 ^ | v • Reply • Share ›

Rajeev Singh Mod → Langton Favor Rebel Mudyiwa • a year ago
Hi,

The name of the method should be `findByUsername`. It is case-sensitive. It should exactly match the corresponding field name in the `User` class.

Regards,
Rajeev
^ | v • Reply • Share ›

Marcus Aurelius • a year ago
Man, the server worked perfectly. I'm going to the part 2 now. See you there.
^ | v • Reply • Share ›

Vipul Sharma → Marcus Aurelius • 6 months ago
how did you create the user table?
^ | v • Reply • Share ›

gotqks2 • a year ago

In this example, the server put a token in the body when responding. The rfc document recommends headers, but it is not required. If both are possible, what are the advantages of each?

^ | ⌄ • Reply • Share ›

seanbruceful • a year ago

Hi thank you for such good tutorial. I am tring use what I learned to build my project, Following part1 and par2 of the tutorial, the app runs perfectlly, signing, logining, authenticating no problem. but I encountered an error when I following part 3. the error is "java.sql.SQLException: Value '0000-00-00 00:00:00' can not be represented as java.sql.Timestamp" it occurs when I try to use api for fetching data. I'm trying so hard to resolve the error by myself but failed. what could be the prosible reason for such error.

^ | ⌄ • Reply • Share ›

Saravanan Subramani • a year ago

Excellent! Very helpful

Can you suggest how to call these services via POSTMAN?
createPoll
getPollById
getPollById

THanks,

^ | ⌄ 1 • Reply • Share ›

Saravanan Subramani ➜ Saravanan Subramani • a year ago

Please ignore my query. i just gone through the part-2 and understood how to test this application using POSTMAN.

^ | ⌄ • Reply • Share ›

Koko Handokoko • 2 years ago • edited

Thx before for great tutorial, i have issue when i run mvn got error, it says

Unable to close ApplicationContext

here a screenshot

for details
https://gist.githubusercont...

very appreciate it for any response

^ | ⌄ • Reply • Share ›

Jasenko • 2 years ago

Any suggestions how to add swagger doc to this sample? Configuring class that extends WebMvcConfigurationSupport breaks Security context holder, current user principal values are null

^ | ⌄ 1 • Reply • Share ›

Sumaya Manzoor • 2 years ago

Generally, I don't make comments on sites, however, I need to say that this post really pushed me to do as such thing.

1 ^ | ⌄ • Reply • Share ›

Blitz NLify • 2 years ago

Hello I just ask if you have spring boot security mysql with bcrypt authentication not the jwt method.

^ | ⌄ • Reply • Share ›

Rajeev Singh Mod ➜ Blitz NLify • 2 years ago encrypted using a cryptographic algorithm like SHA-256, and store this token in the database against the given user.

When the user logs in, create this token and send it to the client. After that, on every request, extract this token from the `Authorization` header and verify it against the database.

You'll need to change only two classes. You'll need to replace `JwtTokenProvider` with your own `TokenProvider` implementation, and modify `JWTAuthenticationFilter` as per your `TokenProvider`