Spring Boot + Spring Security + JWT + MySQL + React Full Stack Polling app - Part 4

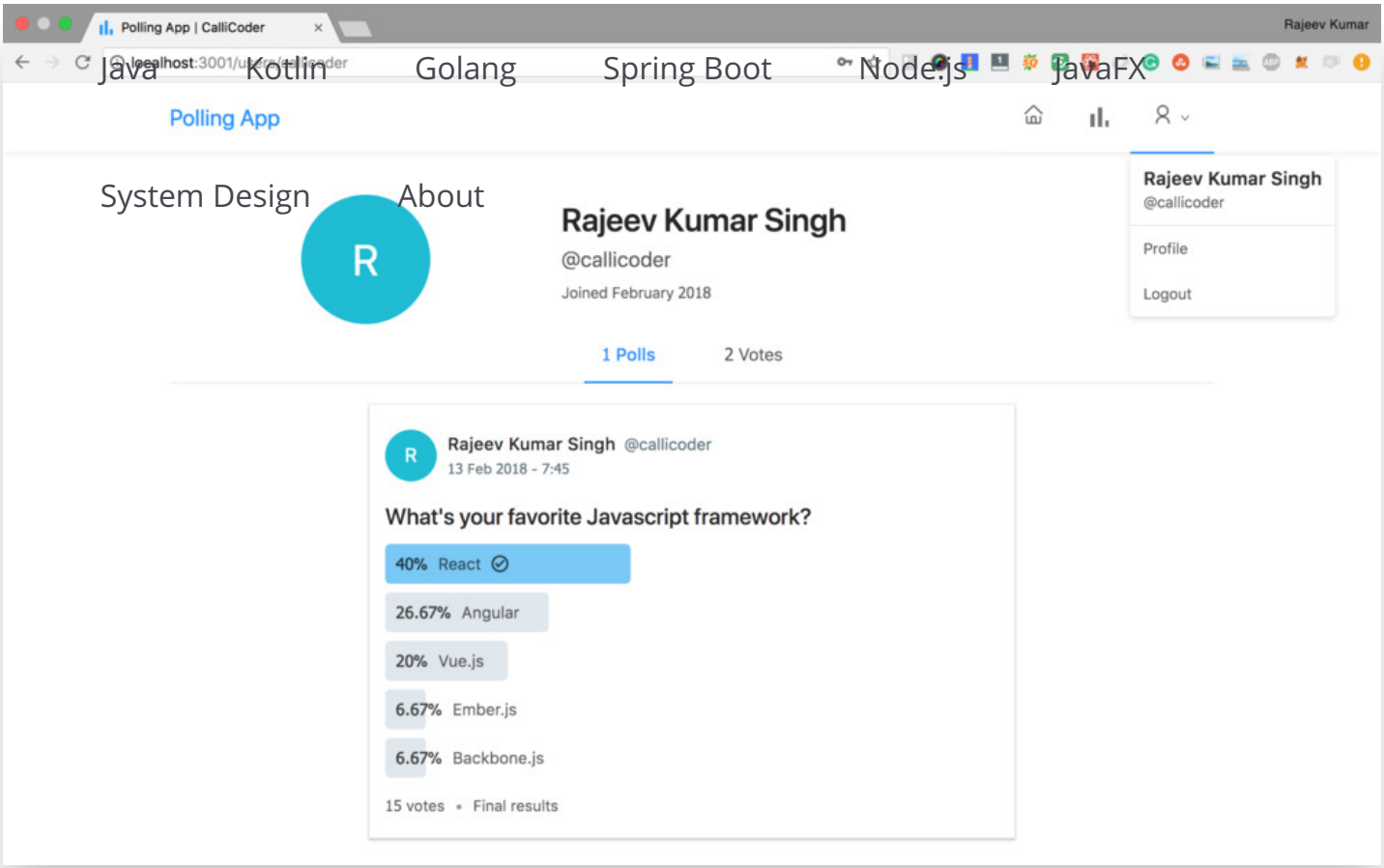
Rajeev Kumar Singh • Spring Boot • Feb 10, 2018 • 9 mins read

Welcome to the 4th and final part of my full stack app development series with Spring Boot, Spring Security, JWT, MySQL and React.

In the last 3 articles, we created the backend project, setup spring security with JWT authentication, and written the rest APIs for Login, Signup, Polls, and Users.

In this article, We'll build the client side of the application using React and Ant Design.

> You can find the complete source code for the project on Github. The project is also hosted on AWS free tier for you to explore. Check out the live demo of at https://polls.callicoder.com.



Bootstrapping the Front End Project

Follow the steps below to setup the front-end project.

1. Installing create-react-app

First of all, we'll install `create-react-app`, a command line tool for creating react apps

```
npm install -g create-react-app
```

2. Creating the app

Now let's create the app using `create-react-app` tool by typing the following command -


```
create-react-app polling-app-client
```

3. Installing Additional Dependencies

We'll be using the following additional dependencies in our project -

1. And Design: An excellent react based user interface library for designing the user interface.
2. React Router: Client side routing solution for react apps.

Let's install these dependencies by typing the following command

```
cd polling-app-client
npm install antd react-router-dom --save
```

We'll also need some dev dependencies to customize Ant Design's theme and enable on-demand component import. Type the following command to install these dev dependencies -

```
npm install react-app-rewired babel-plugin-import react-app-rewire-less -
```

4. Configuring Ant Design

Now let's configure ant design and customize its theme by overriding `less` variables.

We'll use react-app-rewired to enable customization. Open `package.json` file and replace the following scripts


```
"scripts": {
  "start": "react-scripts start",
  "build": "react-scripts build",
  "test": "react-scripts test --env=jsdom",
  "eject": "react-scripts eject"
}
```

with these scripts -

```
"scripts": {
  "start": "react-app-rewired start",
  "build": "react-app-rewired build",
  "test": "react-app-rewired test --env=jsdom",
  "eject": "react-scripts eject"
}
```

- **Overriding configurations with config-overrides.js**

  Now create a file named `config-overrides.js` in the root directory of the project and add the following code to it -

```
const { injectBabelPlugin } = require('react-app-rewired');
const rewireLess = require('react-app-rewire-less');

module.exports = function override(config, env) {
    config = injectBabelPlugin(['import', { libraryName: 'antd', styl
```

```
            "@layout-body-background": "#FFFFFF",
            "@layout-header-background": "#FFFFFF",
            "@layout-footer-background": "#FFFFFF"
        },
        javascriptEnabled: true
    })(config, env);
    return config;
};
```

Notice how we're overriding Ant Design's default less variables to customize the theme as per our needs.

5. Running the App

We're done with all the configurations. Let's run the app by typing the following command -

```
npm start
```

Exploring the directory structure of the Project

Following is the directory structure of the complete front-end project. You can check out the entire source code on Github.

```
polling-app-client
  ↳ public
        ↳ favicon.png
        ↳ index.html
        ↳ manifest.json
```

```
↳ app
    ↳ App.css
    ↳ App.js
    ↳ common
        ↳ AppHeader.css
        ↳ AppHeader.js
        ↳ LoadingIndicator.js
        ↳ NotFound.css
        ↳ NotFound.js
        ↳ PrivateRoute.js
        ↳ ServerError.css
        ↳ ServerError.js
    ↳ constants
        ↳ index.js
    ↳ poll
        ↳ NewPoll.css
        ↳ NewPoll.js
        ↳ Poll.css
        ↳ Poll.js
        ↳ PollList.css
        ↳ PollList.js
    ↳ user
        ↳ login
            ↳ Login.css
            ↳ Login.js
        ↳ profile
            ↳ Profile.css
            ↳ Profile.js
        ↳ signup
            ↳ Signup.css
            ↳ Signup.js
```

```
        ↳ APIUtils.js
        ↳ Colors.js
        ↳ Helpers.js
    ↳ index.css
    ↳ index.js
    ↳ logo.svg
    ↳ poll.svg
    ↳ registerServiceWorker.js
  ↳ config-overrides.js
  ↳ package.json
```

In this article, I'll give you a brief idea of how the project is structured and what each directory and files do.

Going through every piece of code in this blog will be very time consuming, and is absolutely unnecessary. The code is simple and self-explanatory. A basic knowledge of React is needed to understand the code.

You can download the code from Github and go through it. I'll always be here to help if you get stuck at something.

All right! Let's now understand some of the important pieces of code in our front-end project.

Understanding the front-end code

index.js

This file is the main entry point of our react application -

```
import React from 'react';
```

```
import App from './app/App';
import registerServiceWorker from './registerServiceWorker';

import { BrowserRouter as Router } from 'react-router-dom';

ReactDOM.render(
    <Router>
        <App />
    </Router>,
    document.getElementById('root')
);


registerServiceWorker();
```

In the above script, we simply render the `App` component in an html element with id `root` (This html element is available in `public/index.html` file).

src/app/App.js

It defines the `App` component. The `App` component is the main top-level component of our application. It defines the primary layout and routing, loads the currently logged in user, and passes the `currentUser` and `isAuthenticated` property to other components.

```
import React, { Component } from 'react';
import './App.css';
import {
  Route,
  withRouter,
  Switch
} from 'react-router-dom';
```

```
import { ACCESS_TOKEN } from '../constants';

import PollList from '../poll/PollList';
import NewPoll from '../poll/NewPoll';
import Login from '../user/login/Login';
import Signup from '../user/signup/Signup';
import Profile from '../user/profile/Profile';
import AppHeader from '../common/AppHeader';
import NotFound from '../common/NotFound';
import LoadingIndicator from '../common/LoadingIndicator';
import PrivateRoute from '../common/PrivateRoute';

import { Layout, notification } from 'antd';
const { Content } = Layout;

class App extends Component {
  constructor(props) {
    super(props);
    this.state = {
      currentUser: null,
      isAuthenticated: false,
      isLoading: false
    }
    this.handleLogout = this.handleLogout.bind(this);
    this.loadCurrentUser = this.loadCurrentUser.bind(this);
    this.handleLogin = this.handleLogin.bind(this);

    notification.config({
      placement: 'topRight',
      top: 70,
      duration: 3
```

```
}

loadCurrentUser() {
  this.setState({
    isLoading: true
  });
  getCurrentUser()
  .then(response => {
    this.setState({
      currentUser: response,
      isAuthenticated: true,
      isLoading: false
    });
  }).catch(error => {
    this.setState({
      isLoading: false
    });
  });
}

componentWillMount() {
  this.loadCurrentUser();
}

// Handle Logout, Set currentUser and isAuthenticated state which will
handleLogout(redirectTo="/", notificationType="success", description="
  localStorage.removeItem(ACCESS_TOKEN);

  this.setState({
    currentUser: null,
    isAuthenticated: false
```

```
  this.props.history.push(redirectTo);

  notification[notificationType]({
    message: 'Polling App',
    description: description,
  });
}

/*
 This method is called by the Login component after successful login
 so that we can load the logged-in user details and set the currentUse
 isAuthenticated state, which other components will use to render thei
*/
handleLogin() {
  notification.success({
    message: 'Polling App',
    description: "You're successfully logged in.",
  });
  this.loadCurrentUser();
  this.props.history.push("/");
}

render() {
  if(this.state.isLoading) {
    return <LoadingIndicator />
  }
  return (
      <Layout className="app-container">
        <AppHeader isAuthenticated={this.state.isAuthenticated}
          currentUser={this.state.currentUser}
```

```
        <Content className="app-content">
          <div className="container">
            <Switch>
              <Route exact path="/"
                render={(props) => <PollList isAuthenticated={this.stat
                    currentUser={this.state.currentUser} handleLogout=
              </Route>
              <Route path="/login"
                render={(props) => <Login onLogin={this.handleLogin} {
              <Route path="/signup" component={Signup}></Route>
              <Route path="/users/:username"
                render={(props) => <Profile isAuthenticated={this.state
              </Route>
              <PrivateRoute authenticated={this.state.isAuthenticated}
              <Route component={NotFound}></Route>
            </Switch>
          </div>
        </Content>
      </Layout>
    );
  }
}

export default withRouter(App);
```

src/common - Common Components

- **AppHeader.js**: Header component which renders Login & SignUp buttons for unauthenticated users, and Home, Profile & Create Poll buttons for

- **LoadingIndicator.js**: It is used by other components to render a loading indicator while an API call is in progress.
- **NotFound.js**: We use this in `App` component to render a 404 Not Found page if none of the routes match the current url.
- **PrivateRoute.js**: A meta component that redirects to `/login` if the user is trying to access a protected page without authentication.
- **ServerError.js**: Other components use this to render a 500 Server Error page if any API responds with a 500 error which the component can't handle.

src/constants

I've defined all the global constants in `src/constants/index.js` file for other components use -

```
export const API_BASE_URL = 'http://localhost:5000';
export const ACCESS_TOKEN = 'accessToken';

export const POLL_LIST_SIZE = 30;
export const MAX_CHOICES = 6;
export const POLL_QUESTION_MAX_LENGTH = 140;
export const POLL_CHOICE_MAX_LENGTH = 40;

export const NAME_MIN_LENGTH = 4;
export const NAME_MAX_LENGTH = 40;

export const USERNAME_MIN_LENGTH = 3;
export const USERNAME_MAX_LENGTH = 15;

export const EMAIL_MAX_LENGTH = 40;
```

```
export const PASSWORD_MIN_LENGTH = 6;
export const PASSWORD_MAX_LENGTH = 20;
```


src/poll

- **NewPoll.js**: Renders the Poll creation form.
- **PollList.js**: This component is used to render a list of polls. It is used to render all polls on the home page. We also use this in user's profile page to render the list of polls created by that user, and the list of polls in which that user has voted.
- **Poll.js**: It is used by the `PollList` component to render a single Poll.

src/user

- **login/Login.js**: The Login component renders the Login form calls the login API to authenticate a user.
- **signup/Signup.js**: It renders the registration form and contains a bunch of client-side validations. It's an interesting component to check out if you want to learn how to do form validations in React.
- **profile/Profile.js**: The profile page renders a user's public profile. It displays user's basic information, the list of polls that the user has created, and the list of polls in which the user has voted.

src/util

- **APIUtils.js**: All the Rest API calls are written in this script. It uses the `fetch` API to make requests to the backend server.
- **Colors.js**: This util is used to get a random color to use in user's avatar.

Over to you

I always prefer writing practical end-to-end tutorials on this blog. Full stack end-to-end tutorials like this one give you a broader picture of the application and lets you think through the challenges that are faced in all the stacks.

I hope that I didn't overwhelm you with lots of code, and you were able to follow along and learn the concepts.

If something is unclear and you want me to explain it in detail, then let me know in the comment section below. I'll do my best to explain it to you.

At last, I have some homework for you :) There are still a lot of stuff that we can improve in our polls app. I want you to work on them and submit a pull request on Github.

So here are some of the improvements that will make the polls app more awesome -

- **Email verification**: There is no email verification right now in our application. Anyone can register with a random email. Write the logic to send an email verification mail to newly registered users with a verification link and verify user's email once he clicks on the link.
- **Edit Profile**: Add an edit profile page where a user can change his name, username, email, and password.
- **Forgot Password**: Add forgot password functionality in the app.

I encourage you to implement the above functionalities. You will find many articles on the internet explaining how to implement things like email verification, forgot password etc. If you don't find a proper answer on the internet, then write to me in the comment section below or send me an email. I'll help you out.

nhuthuynh • 8 days ago

how can you deploy backend to AWS Elastic Beantalk and front end to AWS S3 and link them together?

Reply • Share ›

Rajeev Kumar Singh Mod → nhuthuynh • 6 days ago

Hi,

I've already written a tutorial on how to deploy a spring boot app on AWS.

For deploying the front-end - first build the react app using `npm run-script build` command. After that, upload the entire `build` folder in AWS S3.

You just need to make one change in the front-end code to connect it with the spring boot backend deployed on AWS.

Open `src/constants/index.js` file and change the `API_BASE_URL` to the url of spring boot backend app.

Cheers,
Rajeev

Reply • Share ›

test user • 19 days ago

Hi,
When I run this project and try to register a new user to the App I got

resource"

• Reply • Share ›

DESH ➦ test user • 11 days ago

Hi test user,
Did you manage to fix "full authentication is required to access this resource" error, if you did please share with me... I am really struggling with this one!
Thanks.

• Reply • Share ›

Rajeev Kumar Singh Mod ➦ DESH • 6 days ago

Hey guys,

If you can share your code on Github, I'll check that out. It's difficult to just guess what might be missing without looking at the code.

Cheers,
Rajeev

• Reply • Share ›

test user ➦ DESH • 8 days ago

not yet...

• Reply • Share ›

sarathraj • 22 days ago

Hey Rajeev, You made my life so easy with all tutorials. I have only one question. i want to replace 'MySQL' with 'oracle'. Whats steps need to follow. Thanks.

• Reply • Share ›

Kishore Kumar Korada • 23 days ago

Hi Rajeev,

What about log out senario? Will the token be invalidated after logout? How it gets handled by login, accessing resources and logout?

• Reply • Share ›

Rajeev Kumar Singh Mod ➦ Kishore Kumar Korada • 23 days ago

Hi Krishna,

Please check this comment. I've answered your question there.

anywhere. Ideally a token should be invalidated when the user logs out. But since JWT is not stored, it can't be invalidated unless it expires.

To Logout, you simply remove the token from the client. Check the part 4 of this tutorial where logout is implemented like this.

However, there are some other considerations while using JSON Web Tokens that you should be aware of. Check this StackOverflow answer for details.

Cheers,
Rajeev

Reply • Share ›

furotyst furotyst • 23 days ago

Hi Rajeev,

After adding react front end, I constantly receive "User not found with id : 2" exception in backend during new user creation (/signup).

The issue is placed in JwtAuthenticationFilter I do believe. "getJwtFromRequest(request)" method returns non-null value and then user with id=2 is retrieved from jwt. Then, the user (id=2) couldn't be found by repository (which is fine as it does not exist) and exception is thrown.

Please, notice this is not the case for the Postman. Using postman, I can singup the user and getJwtFromRequest(request) method returns null and the problematic part of the code is not run.

Please, could you clarify what I've done wrong

Reply • Share ›

Rajeev Kumar Singh Mod → furotyst furotyst • 23 days ago

Hi,

Can you please clear the accessToken that is set in your browser's localStorage? Try signing up after clearing that. It should work.

Cheers,
Rajeev

Reply • Share ›

MaxSinclair • a month ago

Rajeev Kumar Singh Mod → MaxSinclair • a month ago

Hi,
Have you enabled CORS as described in `Part2`?

1 • Reply • Share ›

MaxSinclair → Rajeev Kumar Singh • a month ago

That fixed it. Thank you so much! And sorry for being sloppy :p

• Reply • Share ›

Blitz NLify • a month ago

Hello Rajeev,
Would it be possible to make this application real-time using websockets?

• Reply • Share ›

Venkatesh • a month ago

Hello Rajeev, Thanks for the tutorial.

I am getting "Encoded password does not look like BCrypt" error while checking the "/signin" api in postman.
My password column in database is varchar with 100 as limit. And password is stored in an encrypted format.
Could you please help?

• Reply • Share ›

test user • a month ago

Hi,
Thanks for this tutorial!
One question, could you please let me know how can I fix the following issue?

• Reply • Share ›

Hi,

There is a bug in the new version of less. Check this out.

To fix the above error, Open `package.json` file and fix the `react-app-rewire-less` version to "2.1.0". (Remove ^).

Reply • Share ›

test user → Rajeev Kumar Singh • a month ago

Hi, thanks for your effort. I saw this article, but unfortunatelly it doesn't resolve the bug.

Reply • Share ›

Rajeev Kumar Singh Mod → test user • a month ago

Check this issue. You can enable javascript in `config-overrides.js` like this -

```
module.exports = function override(config,
  config = injectBabelPlugin(
    ["import", { libraryName: "antd", style
    config
  );
  config = rewireLess.withLoaderOptions({
    modifyVars: {
      "@layout-body-background": "#FFFFFF",
      "@layout-header-background": "#FFFFFF
      "@layout-footer-background": "#FFFFFF
    },
    javascriptEnabled: true
  })(config, env);
  return config;
};
```

Reply • Share ›

test user → Rajeev Kumar Singh • 18 days ago

Thanks a lot. Finally this solution resolved this issue.

Reply • Share ›

Mateusz Piorowski • 2 months ago

Amazing, well done and clean tutorial. Everything working as expected :)

If You could try convert it into webflux....that would be even more amazing :)

Reply • Share ›

Thakib Kayodé Adéchinan Salami • 2 months ago

Thanks so much! It's was amazing and rewarding to follow the tutorial

Amir Amir • 2 months ago
what about giving an example with angular 6 ? :)
Reply • Share ›

Vitali Bassov • 3 months ago
Hello Rajeev,
I've got a problem trying to run this project:
The registration works perfectly, but when I'm trying to login, then I have the following exception:
java.lang.ClassNotFoundException: javax.xml.bind.DatatypeConverter
at
java.base/jdk.internal.loader.BuiltinClassLoader.loadClass(BuiltinClassL
~[na:na]
at
java.base/jdk.internal.loader.ClassLoaders$AppClassLoader.loadClass(
~[na:na]
at java.base/java.lang.ClassLoader.loadClass(ClassLoader.java:499) ~
[na:na]
at io.jsonwebtoken.impl.Base64Codec.decode(Base64Codec.java:26)
~[jjwt-0.9.0.jar:0.9.0]
at
io.jsonwebtoken.impl.DefaultJwtBuilder.signWith(DefaultJwtBuilder.java
~[jjwt-0.9.0.jar:0.9.0]
at
com.example.polls.security.JwtTokenProvider.generateToken(JwtToken
~[classes/:na]
at
com.example.polls.controller.AuthController.authenticateUser(AuthCont
~[classes/:na]

Do you know what is it?
Reply • Share ›

sampath karupakula ➦ Vitali Bassov • 2 months ago
<dependency>
<groupid>javax.xml.bind</groupid>
<artifactid>jaxb-api</artifactid>
<version>2.3.0</version>
</dependency>

add this, it will solve the problem, form java 9 onward, javax.xml.bind package has been removed.
1 Reply • Share ›

Can you clean try cleaning and re-building your project? Also, Please check that you have defined `app.jwtSecret` and `app.jwtExpirationInMs` properties in `application.properties` file.
Also, let me know which version of Java are you using?

Reply • Share ›

Mykola • 3 months ago

Hello Rajeev,
Thank you for this amazing tutorial! You described the whole information very good.
But I faced one issue with user authorization: I cannot register any user because on server side I am getting UserNotFoundException.
Problem is that in JwtAuthenticationFilter customUserDetailsService.loadUserById(userId) returns wrong ID (always userId==4 for unknown reasons). I suppose this is cached data problem, because in another browser this problem is absent.
And on client side I am getting TypeError: Cannot read property 'accessToken' of undefined.
Can you explain me more detail about the generation of JWT on client side?

Reply • Share ›

Rajeev Kumar Singh Mod → Mykola • 3 months ago

Hi,

For the first problem, you can just clear your localStorage and try again.

I'm not sure why the second issue is coming. I'll have more idea if you can give me the complete error including the method and the line number where this error is coming.

Also, JWT is not created on the client side. It is created on the server and sent to the client on login. The client stores this JWT in localStorage and includes it in the Authorization header of all the requests.

Reply • Share ›

Jack • 4 months ago

Hi! Your tutorials are awesome!
Can you please give a brief steps how did you manage to upload two different projects (client and server) on aws? Did you connect them for

Rajeev Kumar Singh Mod ➦ Jack • 4 months ago

I connected them into one project.

You can build the react client using `npm run-script build` command. It will produce a `build` directory. You can then copy everything from the `build` directory to the `src/main/resources/static` directory of the spring boot server and deploy the server to AWS.

I use elastic beanstalk to deploy the spring boot app. Check out this article to learn how to deploy spring boot apps on AWS using Elastic Beanstalk.

1 • Reply • Share ›

Jack ➦ Rajeev Kumar Singh • 4 months ago

Works like a charm! Thank you very much!

• Reply • Share ›

Freddy Daniel Muro Giurfa • 4 months ago

Hi Rajeev,

Thanks for the awesome tutorial!.

I have one question... how could you manage to do the navigation with a Sidebar?

I created a component for the Sidebar and placed after the first <layout> (at App.js) just before the Header.

```
import React, { Component } from 'react';
import { Menu, Icon } from 'antd';
import {
Link
} from 'react-router-dom';

import PollList from '../poll/PollList';

const SubMenu = Menu.SubMenu;
const MenuItemGroup = Menu.ItemGroup;
```

see more

• Reply • Share ›

Rajeev Kumar Singh Mod ➦ Freddy Daniel Muro Giurfa • 4 months ago

should work with that. Are you getting any error?

• Reply • Share ›

Sushil GC • 4 months ago

Hi Rajeev.
awesome tutorial
Could you help/share the code how to add signin using spring social(facebook etc).
I could not find a single tutorial over a internet with jwt and spring social.

• Reply • Share ›

Blitz NLify • 4 months ago

Hello sir, just a question, does plain text password is intentional? just refer to the image I've attached.

• Reply • Share ›

Rajeev Kumar Singh Mod → Blitz NLify • 4 months ago

Passwords are stored as encrypted in the database.

But it's not encrypted on the client side before sending to the server.

That is fine if you're using SSL.

• Reply • Share ›

junior • 4 months ago