Documentation of Full stack application with spring boot and React.

Date:20/01/2021

**VVIP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**GitHub Fundamentals:**

To connect your project to your GitHub portfolio, go to your project folder and run the command.

**git init (**this should automatically checkout the master branch or check the branch using git branch command**)**

Then run the command **git add .**

Then run the command **git commit -m “Message for the commit”**

Your current branch will be master. To view all the branches, use command git branch.

Push the code to the master branch using **git push** command.

Make the necessary changes, create a new branch branch0 using command git branch branch0.

Now checkout this branch branch0 using **git checkout branch0** command,

Make necessary changes, do the Git add . and Git commit, then do the git push to the branch0.

Now if all the test cases are passed, check out the master branch and merge master and branch0 using:

**git merge branch0 command** and then finally use the git push command.

**Technical Knowledge gained:**

You have created a new class “Project” with **@Entity** annotation.

**@Entity** annotation maps the class to the table in java.

To define the primary key for the table you have to specify **@Id** command for the variable you want to use as “primary key”.

Primary Key Generation Strategy: There are different types to generate primary keys for entities, one of way is:

**@GeneratedValue (strategy = GenerationType.IDENTITY)**

The above statement specifies the auto increment feature for the primary key. There are 3 other primary key generation strategies, make sure you learn them when you have time.

All the above annotations are part of the **javax. persistence** package. Know about this package.

**21/01/2021**

**Technical Knowledeg gained:**

ORM basics:

Reference: **https://www.infoworld.com/article/3379043/what-is-jpa-introduction-to-the-java-persistence-api.html**

As part of the application architecture, the ORM layer is responsible for managing the conversion of software objects to interact with the tables and columns in a relational database.

By default, the name of the object being persisted becomes the name of the table, and fields become columns. Once the table is set up, each table row corresponds to an object in the application. Object mapping is configurable, but defaults tend to work well.

JDBC:

JDBC is a layer of abstraction that lets an application issue SQL commands without thinking about the underlying database implementation.

CRUD Repository: For doing the crud operations on the persisted objects we have a interface called org.springframework.data.repository.CrudRepository . Any class which implements CrudRepository can use all the methods in the interface. save method is used to insert the persisted object in the table

Annotations:

@RestController: It is an annotations that binds java classes to web and so it exists in the project org.springframework.web.bind package.

@RequestMapping: Used for posting a json request through postman to the specified path.

@PostMapping: used to post the response body for the request with the http status.

@RequestBody: accepts the json object and maps it to the class specified in the parantheses

@ResponseEntity: For mapping the class as the response body with the HTTP status.

For now, the full stack web application has 5 packages namely:

Base Package: Com.fullstack.ppmtool and 4 subpackages:

com.fullstack.ppmtool.domain ( which has the project class)

com.fullstack.ppmtool.exceptions for handling exceptions

com.fullstack.ppmtool.repositories for projectRepository(it has project repository class)

com.fullstack.ppmtool.services for the persistent layer( It has the project service class which communicates with the repository class)

com.fullstack.ppmtool.web (it has controller class which basically communicates with the web)

Communication as of now:

**Controller class (also talks to exception class)🡪 service (also talks to exception class) 🡪 Repository --🡪 JPA (Project Record)**

**ProjectController Class:**

Why the class is used: It is a controller in MVC pattern, it is used to map the web requests to handler methods and serve the requests using POST, GET methods. We have @Controller and @RestController patterns to mention the class as the controller bean in the spring boot. Controller communicates with the service class to implement the required functionality. So we have to do the dependency injection of the service bean to this class using “@AutoWired” annotation.

**Keywords:**

**@RequestMapping (“path”):** This annotation maps HTTP requests to handler methods of MVC and REST controllers.

**@PostMapping (“path”):** This annotation triggers a handler method that can return the information that needs to be posted back.

**@GetMapping(“/{pathVariable}”):** fetches the data for the request using path variable. Here path variable is a request param and can be used in the handler methods.

**@DeleteMapping(“/{pathVariable}”):** This annotation is used for executing delete requests.

**Response Entity<>:** It the structure of the response body that is sent as a HTTP response to the request. We can send the objects of the class by returning Response Entity<Object> object.

**Some more keywords:**

**@Valid:**

**@ResponseBody:**

**@PathVariable:** to use the request parameters in the methods.

**ProjectService Class:**

This class is used to implement the functionality requested by the controller by communicating with the repository. @Service annotation is used to represent the class as a service bean. Since service data relies on repository implementations, DI of the project Repository class should be done on the service class using “@AutoWired” annotation.

**ProjectRepository Class:**

This class is used to represent the repository of all the projects and is specified with @Repository. It extends the CRUD repository interface from the data repository package in the spring framework.

The interface comes will all the CRUD methods that can applied on the repository instances.

CRUDRepository has 2 parameters : Entity class type and primary key type

Examples: **public** **interface** ProjectRepository **extends** CrudRepository<Project , Long>

**Introduction to react:**

**1.React is a javascript library , created by facebook that allows components based development and also the use of JSX.**

**2.JSX is a syntax like HTML**

**3.3 main codes of the react workspace are index.html, index.js and app.js.**

**4.index.html is the end point of the application.**

**5.index.js is the function which tells which rendered views go to which specific element.**

**6.app.js is a function which returns the rendered views.**

**7.The components created can be reused in any part of the code just by importing them with the path.**

**8.The return function accepts can return only one specific element.**

**Steps to create and start a react app:**

**1.Use the command** npm create-react-app <app name> to create the react application.

2.Upto now you need only 2 packages “ bootstrap” and “react-router-dom” . To install these packages , use the commands npm install react-router-dom and npm install bootstrap.

Workflow of the project:

1.First create the project using the command npm create-react-app <app name>

2.A react project will be created with all the necessary dependencies installed.

3. The main files in the project are index.html , index.js and app.js

4. In the index. Js , it will be already mentioned that rendered view for the index.html is app.js. So the work should be on app.js.

5.First create a new folder components, to list all the components. Then create class level components called Dashboard and Header.js.

Why these are class level components: The dashboard and navigation bar states and lifecycles are needed for the projects. If we don’t need them we can create functional level components.

6.In the Dashboard.js use the command ‘rcc’ to get the entire definition of the class level component and make sure you return the content needed for the dashboard.

7.In the Header.js return all the content for the navigation bar.

8. In the app.js return <Header /> and <Dashboard /> under a single <div> </div> element

Adding Routing part for the project:

1.To add the routing functionality to the application install the package npm-react-route dom.

2.Under the components folder , add a project folder and create a file create Project button. This button will be used to create the project. Import the module {Link} from the react-router-dom to use <Link></Link> attribute. This attribute can be thought of as like <a href=”#”></a>. The link will redirect to path “/addProject”.

3.In the app.js folder, then import {BrowserRouter and Route} packages a wrap the return element inside the <BrowserRouter><./BrowserRouter> element to use the routing functionalities.

4.To route the createprojectbutton to the addproject component, write the code using Route module as:

<Route path =”/addProject” component=”AddProject”/>.