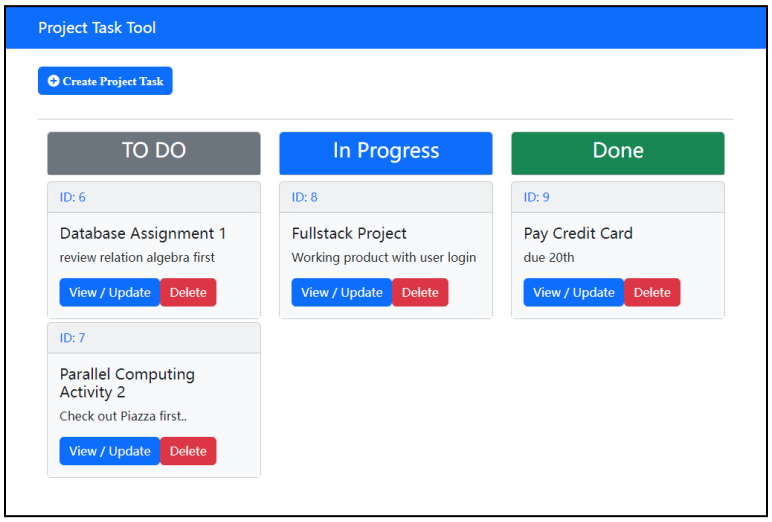
To-do List

Introduction:

This is a to-do list full-stack project with Spring and React allows users to add, update, and delete tasks. The backend with Spring performs the CRUD operations with instructions received through api request from the frontend with React, and the project utilizes Redux for state management to reduce the frequency of api calls.



Demo Figure

Credit to Agile Intelligence for the tutorial: https://www.youtube.com/watch?v=PkQivzgsLxM&list=PLhxN8qSgOT20PXVZo4eksXMfsmp1Ndnwb

Tech:

- Backend **Spring**:
  - Sprintboot
  - Maven Compiler
  - Spring web MVC
  - H2 Database
  - javax.validation

    -
- Frontend **React**:
    - React-router-dom v6
    - React-redux
    - Redux-thunk
    - Axios
    - Bootstrap

Helpful Guide:
React: https://developmentarc.gitbooks.io/react-indepth/content/

Application Structure

- **Spring Web App Layers:**
    - **Domain**
        - Defines Database Entity attributes
    - **Repository**
        - Interface that extends CrudRepository<db, id_type>
    - **Service**
        - Class annotated with @Service
    - **Web**
        - Class annotated with @RestController @RequestMapping and @CrossOrigin
        - Autowired Service object
    - ***pom.xml** to declare all the dependencies

- **React App Structure:**
    - **Index.html**
        - The HTML page that is going to get rendered
    - **Index.js**
        - Main render method that renders App into index.html in div “root”
    - **App.js**
        - Where all the components going to meet
        - Routings
    - ***package.json** to declare all the dependencies
    - **Redux**

        - State management library (keep state in the client side)
        - API is stateless
    - **Thunk**
        - Dispatch actions from Store to API

Backend with Spring

1. Spring Initializer
    a. start.spring.io (bootstrap the application)
    b. Configuration: Maven project, Java, Spring Boot 2.7.3, Jar packaging, and Java 17
    c. Dependencies: Spring Boot DevTools, Spring Web, H2 (clear DB everytime server restart), Spring Data JPA, H2 Database, MySQL Driver
2. AWS SDK for Java Maven
    a. https://mvnrepository.com/artifact/com.amazonaws/aws-java-sdk
    b. Paste the Maven dependency in pom.xml
3. H2-Database Test with DevTool
    a. http://localhost:8080/h2-console/
    b. Since 2.3.0, default value of generate-unique-name is true, we need to add a few setting in src/main/resources/application.properties
        i. spring.datasource.generate-unique-name=false
        ii. spring.h2.console.enabled=true
        iii. spring.datasource.url=jdbc:h2:mem:testdb
    c. Change the JDBC URL to jdbc:h2:mem:testdb
    d. Go to your user directory C:\users\your_name
        i. Pre-create the test database file “test.mv.db”
4. Add dependencies for API validation
    a.
```
        <dependency>
            <groupId>javax.validation</groupId>
            <artifactId>validation-api</artifactId>
            <version>2.0.0.Final</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-validation</artifactId>
        </dependency>
    </dependencies>
```
5. Enjoy! You can now start implementing backend app logic!

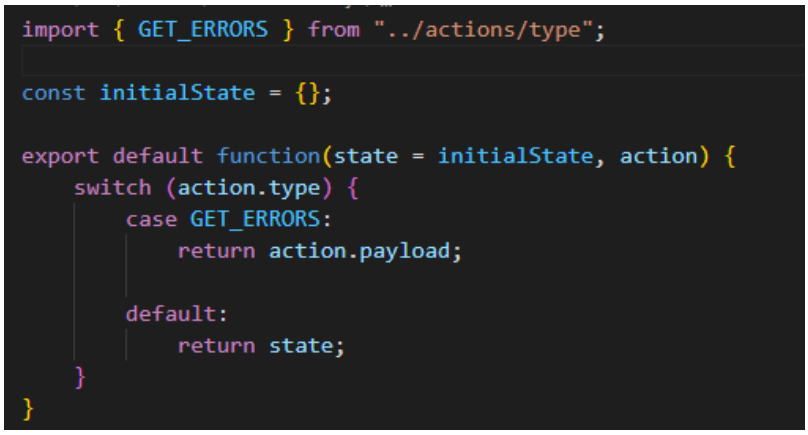
Frontend with React.js

1. Root folder, create project
    a. “npm init react-app name-of-your-app”
    b. or “npx create-react-app name-of-your-app”
2. Install dependencies
    a. redux react-redux redux-thunk
    b. react-router-dom
    c. axios
    d. bootstrap
    e. Classnames
3. In App.js
    a. Make App a class that extends Component
    b. import React, { Component } from 'react';
    c. import "bootstrap/dist/css/bootstrap.min.css";
    d. import {BrowserRouter, Routes, Route, withRouter} from "react-router-dom";
        i. react-router-dom version 6 it is a bit different
            1. https://stackoverflow.com/questions/69832748/error-error-a-route-is-only-ever-to-be-used-as-the-child-of-routes-element
        ii. Routing: (https://www.youtube.com/watch?v=UjHT_NKR_gU)
            1. Surround the App.js return with <BrowserRouter>< ../>
            2. Surround <Route /> with <Routes>< ../>
            3. <Route /> takes
                a. Path, for routing url
                b. Element, for component, in element = {<component />}
            4. <Link to=”path”> </Link>
        iii. History:
            1. To access props.history, we need withRouter
            2. export default withRouter(ComponentName)
        iv. Example:

1. 

```
class App extends Component {
  render() {
    return (
      <BrowserRouter>
        <div className="App">
        <Navbar />
        <Routes>
          <Route exact path="/" element={<ProjectBoard />} />
          <Route exact path="/addProjectTask" element={<AddProjectTask />} />
        </Routes>
        </div>
      </BrowserRouter>
    );
  }
}
```

4. Create folder “components” under “src” for components
    a. short-cut for creating component from extension
        i. “rcc” for react class component
        ii. “rfc” for react function component
5. Redux storage and Reducers
    a. https://github.com/reduxjs/redux-devtools/tree/main/extension#installation
    b. Create “store.js” under “src/”
        i. import {createStore, applyMiddleware, compose} from "redux";
        ii. import thunk from "thunk";
    c. Create a new folder “reducers” under “src/”, and create “index.js” in it
        i. import { combineReducers } from "redux";
    d. In App.js
        i. Import { Provider } from “react-redux”;
            1. Then wrap the entire App.js return in Provider tag pairs
            2. <Provider store = {store}> … </Provider>
        ii. import store from “./store”;
    e. Create reducer for action:
        i. Create new file “type.js” under “src/actions/”
            1. export const GET_ERRORS = "GET_ERRORS";
            2. actions/ is where the action logics are implemented
        ii. Create new file “errorsReducer.js” under “src/reducers/”

1. 

```
import { GET_ERRORS } from "../actions/type";

const initialState = {};

export default function(state = initialState, action) {
    switch (action.type) {
        case GET_ERRORS:
            return action.payload;

        default:
            return state;
    }
}
```

   iii. Add this reader in “src/reducers/index.js” in combineReducer export
      1. erros: errorReducer,

f. dispatch({})
   i. Async dispatch =>
g. .propTypes
   i. Import PropTypes from “prop-types”;
h. Connect
   i. Import { connect } from “react-redux”;

Progress Tracking:

- Project Creation
- Spring Web App Layers setup
- Define Entity(summary, acceptanceCriteria, status)
- Implement Post request for adding an entity
- Validation check on Post request

- Implement Get request for get all
- Implement Delete request for delete
- Create React app (dependencies..)
- Set up the main HTML page

- Set up HTML for add/update
- Routing and Link with react-router-dom

- Implement the addProjectTask component (states and events)
- Set up Redux and Reducers (still in progress)

- Implement Redux store with createStore (should be updated to configure store in the future)
- Set up actions addProjectTask (store to backend through api post request)
- Implemented errorsReducer in reducers
- Error handling when bad add request with no summary (no redirect + err msg)

- Add get all request in projectTaskAction and save them in redux
- Sort out the tasks in projectBoard for display

- Add delete request in projectTaskAction and auto re-render using redux
- Set up update page, load the updating values when visiting
- Add update request in projectTaskAction and auto re-render using redux

Working Product: