Expense Tracker Project - Full Stack Application Overview (Spring Boot + React.js)

Project Summary

This project is a full-stack **Expense Tracker** application built using **Spring Boot** for the backend and **React.js** for the frontend. It allows users to create, read, update, and delete (CRUD) their expense entries.

The app follows a standard client-server architecture, with the React frontend making REST API calls to the Spring Boot backend. The backend interacts with a relational database (likely H2/MySQL/PostgreSQL based on configuration) to persist data.

Key Functional Features

- List all expenses
- Add new expense
- Edit/update existing expense
- Delete an expense
- Data persists using backend and a database

Backend - Spring Boot

Technologies and Frameworks

- Spring Boot
- Spring Data JPA
- RESTful APIs
- Maven (project build tool)
- Java

Spring Boot Annotations Used

- `@RestController` - marks a class as a REST API controller.
- `@RequestMapping` - maps URL paths to controller methods.
- `@GetMapping`, `@PostMapping`, `@PutMapping`, `@DeleteMapping` - handle HTTP methods.
- `@Entity` - marks a class as a JPA entity (maps to DB table).
- `@Table`, `@Column` - for table and field configurations.
- `@Id`, `@GeneratedValue` - primary key generation.
- `@Autowired` - dependency injection.
- `@Repository` - to create DAO components for data operations.
- `@Service` - business logic layer.

REST API Endpoints

Method	Endpoint	Description
GET	`/api/expenses`	Get all expenses
GET	`/api/expenses/{id}`	Get a specific expense
POST	`/api/expenses`	Create a new expense
PUT	`/api/expenses/{id}`	Update an existing expense
DELETE	`/api/expenses/{id}`	Delete an expense

Data Flow in Backend

1. **API Call from Frontend** -> Hits a controller (`@RestController`).
2. **Controller** -> Forwards the request to a service class (`@Service`).
3. **Service** -> Performs logic and interacts with the database via a repository (`@Repository`).
4. **Repository** -> Extends `JpaRepository` to automatically provide CRUD methods.
5. **Entity** -> Represents expense record in the database.

Database Configuration

- Configured in `application.properties` or `application.yml`.
- Contains DB URL, username, password, driver class.
- Uses Spring Data JPA to map Java objects to relational tables.

Frontend - React.js

Technologies Used

- React.js (Functional Components)
- React Hooks (`useState`, `useEffect`)
- Axios or Fetch API for API requests
- JavaScript / JSX
- HTML/CSS for styling

Key Components

1. **App Component**: Root component; renders main layout.
2. **ExpenseList Component**: Fetches and displays all expenses.
3. **ExpenseForm Component**: Used for creating and updating expenses.
4. **Header/NavBar Component**: Contains branding/navigation (if present).

Event Handling & Data Flow in Frontend

1. **On Component Mount (e.g., `useEffect`)**:
   - Fetches expenses from backend using GET request.
   - Sets data to state using `useState`.

2. **On Form Submit**:
   - Gathers form input values.
   - Sends POST or PUT request to backend.
   - Updates UI based on response.

3. **On Delete Button Click**:
   - Sends DELETE request to backend.
   - Filters out deleted item from local state to update UI.

Axios Usage

- Axios is likely used for REST API calls (GET, POST, PUT, DELETE).
- Example:

axios.get("/api/expenses")

```
  .then(response => setExpenses(response.data))
  .catch(error => console.error(error));
```

Proxy Configuration (Frontend to Backend)

- In the React project, the `package.json` file contains a line like:

"proxy": "http://localhost:8080"

- This forwards API calls (like `/api/expenses`) from React's development server to the Spring Boot backend, avoiding CORS issues.

Installed Node Packages & Their Purpose

Package	Purpose
react	Core React library for building UI
react-dom	Enables DOM rendering for React components
react-scripts	Development build tools (scripts, bundlers, etc.)
axios	For making HTTP requests from frontend
@testing-library/react	For writing component tests (optional)
web-vitals	Reports app performance metrics (optional)

Additional packages might be included depending on styling or state management preferences.

Full Stack Integration

Communication Between Frontend & Backend

- Frontend makes RESTful API calls to backend endpoints (`/api/expenses`).
- Backend processes the request and communicates with the DB.
- Backend sends JSON responses to frontend.
- Frontend updates the UI dynamically based on API responses.

Deployment Checklist

Frontend

- [ ] Set proxy in `package.json` for local development.
- [ ] Use `.env` for storing API URLs.
- [ ] Run `npm run build` to generate optimized production build.
- [ ] Host on Netlify, Vercel, or any static host.
- [ ] Configure custom domain and HTTPS.

Backend

- [ ] Update `application.properties` for production (or use `application-prod.properties`).
- [ ] Enable CORS for production domains.
- [ ] Use environment variables for DB credentials.
- [ ] Deploy on Heroku, AWS, or other cloud service.
- [ ] Connect to managed SQL database (AWS RDS, Heroku Postgres, etc.).

Shared

- [ ] Set up CI/CD pipeline (e.g., GitHub Actions).
- [ ] Monitor logs and exceptions.
- [ ] Use version control and clear commit messages.

Final Notes for Maintenance

- Always validate data on both client and server sides.
- Handle API errors gracefully on the frontend.
- Use logging (`SLF4J`/`Logback`) in backend for monitoring.
- Keep component logic modular and reusable.
- Update dependencies regularly to avoid vulnerabilities.

This documentation provides a complete overview for any new developer to understand, maintain, and enhance the Expense Tracker project in a production environment.