Stackoverflow

Sandu Bogdan Emilian

Group 30433/2

1. Introduction

Stack Overflow is a popular online platform where developers and programmers can post their coding related queries, and other members of the community can help provide answers and solutions. The platform has revolutionized the way developers collaborate and find solutions to coding problems.

This is a copy of Stack Overflow. It contains basic functionality. You can ask questions, add tags to questions, sort questions by tags, answer questions and possibly to upvote or downvote answers and questions.

1. Technology

* Spring boot: Java framework used for developing the backend of the application
* React JS: JavaScript framework used for developing the frontend of the application
* MySql: sql database used in the application
* IDE’s: IntelliJ Ultimate (BE) and WebStorm (FE)

Spring boot:

Java Spring Framework (Spring Framework) is a popular, open source, enterprise-level framework for creating standalone, production-grade applications that run on the Java Virtual Machine (JVM).

Java Spring Boot (Spring Boot) is a tool that makes developing web application and microservices with Spring Framework faster and easier through three core capabilities:

1. Autoconfiguration
2. An opinionated approach to configuration
3. The ability to create standalone applications

These features work together to provide you with a tool that allows you to set up a Spring-based application with minimal configuration and setup.

ReactJS:

The React.js framework is an open-source JavaScript framework and library developed by Facebook. It’s used for building interactive user interfaces and web applications quickly and efficiently with significantly less code than you would with vanilla JavaScript.

In React, you develop your applications by creating reusable components that you can think of as independent Lego blocks. These components are individual pieces of a final interface, which, when assembled, form the application’s entire user interface.

1. Use Case Diagrams:

In the Unified Modeling Language (UML), a use case diagram can summarize the details of your system's users (also known as actors) and their interactions with the system. To build one, you'll use a set of specialized symbols and connectors. An effective use case diagram can help your team discuss and represent:

* Scenarios in which your system or application interacts with people, organizations, or external systems
* Goals that your system or application helps those entities (known as actors) achieve
* The scope of your system

1. Diagram

   Description automatically generated­Architecture

The layered architecture pattern organizes components into horizontal layers, each with a specific role in the application, such as presentation or business logic. While there is no set number of layers, most architectures have four standard layers: presentation, business, persistence, and database. Sometimes, the business and persistence layers are combined, resulting in a three-layer architecture for smaller applications or a five or more layer architecture for more complex business applications.

Each layer in the architecture pattern has a specific responsibility, forming an abstraction around the work needed to fulfill a particular business request. For instance, the presentation layer is responsible for handling user interface and browser communication, while the business layer executes specific business rules associated with the request. Separation of concerns is a key feature of this pattern. Components within a layer handle only logic specific to that layer, making it easier to build effective roles and responsibilities models, test, govern, and maintain applications.

1. Package Diagrams

Diagram

Description automatically generatedPackage diagram, a kind of structural diagram, shows the arrangement and organization of model elements in middle to large scale project. Package diagram can show both structure and dependencies between sub-systems or modules, showing different views of a system, for example, as multi-layered (aka multi-tiered) application - multi-layered application model.

* ­­repository: contains the repositories, database operations
* dto: Data Transfer Object, contains the DTO’s of the app
* controller: contains the controllers for each route, handles the requests and mapps them accordingly
* service: contains the logic, each controller uses a service which handles the logic of the request
* entity: represents the tables in our application, their fields and relations
* security: contains the security part of our application (still needs some implementation)

1. Class Diagram

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

Graphical user interface

Description automatically generated

Entities:

* User: fields
  + private Long userId (id of user)
  + private String lastName
  + private String firstName
  + private String email
  + private String password
  + private String role
  + +getters, setters, constructors

Describes the user entity, its representation in the DB (subject to change).

* UserService: fields
  + UserRepository userRepository
  + Public List<UserDTO> retrieveUsers(): gets all users and maps them to the DTO
  + Public UserDTO getUserByID(Long cnp): gets a specific user by its id, after it maps it
  + Public USerDTO findUserByEmail(String email): gets a specific user by its email, after it maps it
  + Public long deleteUserByID(Long id): delete user with that id
  + Public User saveUser(User user): saves/updates an user
* UserControler: fields
  + UserService userService
  + Public List<UserDTO> retrieveUsers(): handles mapping and calls the corresponding method from the service
  + Public UserDTO getUserById(@PathVariable Long id): handles mapping and calls the corresponding method from the service
  + Public long delteUserById(@PathVariable Long id): handles mapping and calls the corresponding method from the service
  + Public User addUser(@RequestBody User user): handles mapping and calls the corresponding method from the service
  + Public User updateUser(@RequestBody User user): handles mapping and calls the corresponding method from the service

1. Diagram

   Description automatically generatedDatabase Diagram

Database currently contains 6 tables: users, content, questions, answers, tags, questions\_tags (SUBJECT TO CHANGE).

SQL SCRIPT:

drop schema if exists sd\_stack\_overflow ;

create schema sd\_stack\_overflow;

use sd\_stack\_overflow;

create table answers (answer\_id bigint not null auto\_increment, question\_id bigint, content\_id bigint, primary key (answer\_id)) engine=InnoDB;

create table content (content\_id bigint not null auto\_increment, created\_at datetime(6), description varchar(255), updated\_at datetime(6), user\_id bigint, primary key (content\_id)) engine=InnoDB;

create table questions (question\_id bigint not null auto\_increment, title varchar(255), content\_id bigint, primary key (question\_id)) engine=InnoDB;

create table questions\_tags (question\_id bigint not null, tag\_id bigint not null, primary key (question\_id, tag\_id)) engine=InnoDB;

create table tags (tag\_id bigint not null auto\_increment, name varchar(255), primary key (tag\_id)) engine=InnoDB;

create table users (user\_id bigint not null auto\_increment, e\_mail varchar(255), f\_name varchar(255), l\_name varchar(255), password varchar(255), role varchar(255), primary key (user\_id)) engine=InnoDB;

alter table answers add constraint FKqolg9rmukcj8w7j2j6f0os2wr foreign key (content\_id) references content (content\_id);

alter table questions add constraint FK8aq2mp5u9col4b9wm5m9kc5un foreign key (content\_id) references content (content\_id);

alter table questions\_tags add constraint FK7yq8xf1pqv8katljm8v8j8w3c foreign key (tag\_id) references tags (tag\_id);

alter table questions\_tags add constraint FK4u5xv906wfevngoe973bec6u0 foreign key (question\_id) references questions (question\_id);

1. Endpoints requests

* <http://localhost:8080/users/getAll> : gets all users

Output: [

    {

        "firstName": "Bogdan",

        "lastName": "Bogdan"

    }

]

* <http://localhost:8080/users/getById/1> : gets user with specific id

Output: {

    "firstName": "Bogdan",

    "lastName": "Bogdan"

}

* <http://localhost:8080/users/addUser> : adds a new User

Body: {

    "lastName": "Bogdan",

    "firstName": "Bogdan",

    "email":"email@email.com",

    "password": "$2a$10$ZvWza6v3MeixgZnpmaVDPOxfiEv0DzOWuOMUuNO.UQGnAJ9q94YAm",

    "role": "user"

}

Output: {

    "userId": 1,

    "lastName": "Bogdan",

    "firstName": "Bogdan",

    "email": "email@email.com",

    "password": "$2a$10$ZvWza6v3MeixgZnpmaVDPOxfiEv0DzOWuOMUuNO.UQGnAJ9q94YAm",

    "role": "user"

}

* <http://localhost:8080/questions/addQuestion> : adds a new Question

Body: {

    "title": "Question2",

    "content":

    {

        "userId":1,

        "description": "sampletext2"

    }

}

Output: {

    "questionId": 1,

    "title": "Question2",

    "content": {

        "contentId": 1,

        "userId": 1,

        "description": "sampletext2",

        "createdAt": "2023-03-19T20:47:11.6113446",

        "updatedAt": "2023-03-19T20:47:11.6113446"

    },

    "tags": **null**

}

* <http://localhost:8080/answers/addAnswer> : adds a new Answer

Body: {

    "questionId":1,

    "content":

    {

        "userId":1,

        "description": "hellow :p"

    }

}

Output: {

    "answerId": 1,

    "questionId": 1,

    "content": {

        "contentId": 2,

        "userId": 1,

        "description": "hellow :p",

        "createdAt": "2023-03-19T20:47:34.9654111",

        "updatedAt": "2023-03-19T20:47:34.9654111"

    }

}

* <http://localhost:8080/answers/getAll> : gets all answers

Output: [

    {

        "answerId": 1,

        "questionId": 1,

        "content": {

            "contentId": 2,

            "userId": 1,

            "description": "hellow :p",

            "createdAt": "2023-03-19T20:47:34.965411",

            "updatedAt": "2023-03-19T20:47:34.965411"

        }

    }

]

* <http://localhost:8080/questions/getAll> : gets all questions

Output: [

    {

        "questionId": 1,

        "title": "Question2",

        "content": {

            "contentId": 1,

            "userId": 1,

            "description": "sampletext2",

            "createdAt": "2023-03-19T20:47:11.611345",

            "updatedAt": "2023-03-19T20:47:11.611345"

        },

        "tags": [

            {

                "tagId": 3,

                "name": "haskell"

            },

            {

                "tagId": 1,

                "name": "c++"

            }

        ]

    }

]

1. Front End Architecture

React is a popular front-end library for building user interfaces. When building a React project, developers typically follow a specific architectural pattern to organize their code and ensure that it is easy to maintain, scale, and test. Here are some key elements of the architecture of a React project:

Component-Based Architecture: React projects are organized around components, which are reusable, independent pieces of UI that can be composed together to create complex user interfaces. Components in React are typically written as classes or functions, and they encapsulate both the visual representation and the behavior of a specific part of the UI.

Virtual DOM: React uses a virtual DOM (Document Object Model) to optimize performance and minimize the number of updates needed to the real DOM. The virtual DOM is a lightweight representation of the actual DOM, and it allows React to efficiently update the UI by comparing the differences between the virtual and real DOMs and only making the necessary changes.

Unidirectional Data Flow: In a React project, data flows in one direction from the parent components down to the child components. This ensures that the data remains consistent and predictable, and it makes it easier to debug and test the application.

Redux: Redux is a state management library that is often used in React projects to manage the application state. Redux provides a centralized store that holds the state of the entire application, and it allows components to access and update the state in a predictable and consistent way.

JSX: JSX is a syntax extension for JavaScript that allows developers to write HTML-like code in their JavaScript files. JSX makes it easier to write and maintain complex UIs, and it provides a more declarative and intuitive way of working with React components.

Webpack: Webpack is a module bundler that is often used in React projects to bundle and optimize the application code. Webpack allows developers to split their code into smaller, more manageable chunks, and it provides features such as code splitting, lazy loading, and tree shaking to optimize the performance of the application.

Overall, the architecture of a React project is designed to be modular, scalable, and easy to maintain. By following the component-based architecture, using the virtual DOM, managing state with Redux, and using tools like Webpack, developers can build complex and performant user interfaces that are easy to work with and maintain over time.

SRC:

* Api folder: calls to the backend
* Components: components used
* Images: images used
* Pages: Overall pages, contain components
* Store: Redux global state of the app

1. Routing

* “/”: Auth Page: implemented but not used yet
* “/users”: List of Users
* “/questions”: List of questions
* “/home”: Landing Page/ Presentation: not implemented yet
* “/questions/:questionId”: Detailed page of question, includes answers