SD Laboratory Assignment –

Stack Overflow

Laboratory teacher: Eng. Bindea Bogdan Nicusor

Student: Petruta George-Calin

Group: 30434

Contents

[1. Introduction 3](#_Toc102475264)

[2. Technical Stack 4](#_Toc102475265)

[2.1 Programming language 4](#_Toc102475266)

[2.2 Frontend framework 4](#_Toc102475267)

[2.3 Data storage and querying 5](#_Toc102475268)

[2.4 API testing and services 5](#_Toc102475269)

[3. Implementation 5](#_Toc102475270)

[4. Use cases 6](#_Toc102475271)

[4.1 User use case 6](#_Toc102475272)

[4.2 Admin use case 6](#_Toc102475273)

[4.3 Detailed use case: 6](#_Toc102475274)

[4.4 Log in: 7](#_Toc102475275)

[4.5 See question: 9](#_Toc102475276)

[4.6 Add question: 10](#_Toc102475277)

[4.7 Add answer: 10](#_Toc102475278)

[4.8 Like/dislike answer/question: 11](#_Toc102475279)

[5. Architecture 11](#_Toc102475280)

[5.1 Backend architecture 11](#_Toc102475281)

[5.2 Database diagram: 13](#_Toc102475282)

[5.3 Class diagram 13](#_Toc102475283)

[5.4 Frontend architecture 14](#_Toc102475284)

[5.5 Frontend – backend connection 14](#_Toc102475285)

[6. Testing 16](#_Toc102475286)

[6.1 Create new user: 16](#_Toc102475287)

[6.2 Add question: 17](#_Toc102475288)

[6.3 Add answer: 17](#_Toc102475289)

[6.4 Get user: 18](#_Toc102475290)

[6.5 Get question: 18](#_Toc102475291)

[6.6 Get answer: 18](#_Toc102475292)

[6.7 Get all users: 18](#_Toc102475293)

[6.8 Get all questions: 19](#_Toc102475294)

[6.9 Get all answers: 19](#_Toc102475295)

[6.10 Delete user: 19](#_Toc102475296)

[6.11 Delete question: 19](#_Toc102475297)

[6.12 Delete answer: 19](#_Toc102475298)

[7. Bibliography 20](#_Toc102475299)

# Introduction

The application to be designed is a very simple copy of Stack Overflow. It will permit the user to enter questions with tags, like comment and question, see and search questions based on their title and/or tags and comment on questions.

The user is not able to do anything before logging in besides, of course, sign-up.

The users can like each other’s contributions, but not their own. They can also edit or delete their own material on the application when they want.

Beside the feature of adding new content on the web application, there will be a system which will rank the users depending on their contributions to the site.

Their score will be showed on every comment, and it will be updated when they receive reactions from others and depending on that, the score will increase or decrease.

Also, beside the normal user, there will be an/some admins which can ban/unban users if there is a need for that and delete/edit materials that are found to be against the rules of the application.

On short, the application will have 2 types of users: Admin and User and each of them will have the following available features:

* Admin:
  + Add users
  + Delete/Update comments/questions
  + Ban/Unban users
* User:
  + Add/edit/delete question
  + Add/edit/delete answer
  + Like/dislike content
  + Create account
  + Search questions

# Technical Stack

## Programming language

The project will be done using as a programming language **Javascript (React library).**

**Java version used: 17**

## Frontend framework

The UI will be done using the **React** library.

**Visual studio code** is used as the text editor for the development of the UI.

## Data storage and querying

The application used for storing the database and executing queries is the Mysql Workbench.

## API testing and services

In order to test the correct functionality of the different endpoints of the website application, **Postman** will be used.

# Implementation

The application has the component grouped together depending on what they do or what they represent.

The main groups are:

* Components
  + Answer
    - Contains a class for the page of answer of edit
  + Client
    - Contains 2 classes, one for editing a new client (user) and one for listing all the users
  + errorPages
    - Contains error pages that will be displayed by the application when certain events and actions are performed
  + Login
    - Contains 2 classes: one that handles the log in and one that handle the sign up
  + Question
    - Contains 3 classes: one for listing the questions, one for editing a question and one for showing/displaying a question page
  + Search
    - Contains a class, where the search page in rendered and the search operation is done
  + Service
    - Contains 4 classes which handle some operations such as adding a like and creation of empty objects
  + Types
    - Contains classes used for entity mapping
* Main:
  + Contains classes for the main page rendering, the dashboard and routing of the application
* Testing:
  + Contains classes for testing (not really used).

# Use cases

## Diagram Description automatically generatedUser use case

## Admin use case

Diagram

Description automatically generated

## Detailed use case:

**User logs in, comments on a question and like another answer:**

The user accesses the application and logs in.

After he successfully logs in, he can then browse through the questions and select one of them by clicking on the button to the right.

A new page will be displayed, showing the information about the question: title, author, vote count, tags and content.

Below this information added by the author, the user can find a comment section.

There, the user can like other’s user answer. Below the comment section he will find a button to add a new comment. After he enters the comment and press the button, the answer will be shown on the question page.

## Log in:

When starting the application, the user will see a login button which when pressed will redirect the user towards a page where he can log in or create a new account.

Graphical user interface, application

Description automatically generated

If the user logs in and the account if banned, he will receive a page saying that his account was banned.

Text

Description automatically generated

If the password is wrong or the account does not exist, another error page will be displayed.

Graphical user interface, text, application

Description automatically generated

Table

Description automatically generated with low confidenceIf the password and the account is correct, then the user is redirected to the main page where he will see a list of questions.

## See question:

Graphical user interface, text, application, email

Description automatically generatedWhen the user’s presses the button “See question” he will be redirected towards a new page containing information about that question such as: answer, likes, author.

## Add question:

Graphical user interface, text, application, email

Description automatically generatedWhen the user’s presses the button “Add question” he will be redirected towards a new page containing information for a new question: title, text and tags.

After adding the information for the question and pressing the “Save” button, a new question will be added and the user will be redirected towards the question list page.

## Add answer:

Graphical user interface, text, application

Description automatically generatedWhen the user’s is in a question page, he has the possibility of adding a new answer by pressing the button “Add answer”. This will redirect him towards a new page where he can add the answer and then he will be redirected back to the question page, where he will then see his newly added comment.

## Like/dislike answer/question:

When the user’s is in a question page, he has the possibility of liking or disliking every comment/question that is not his. After he likes/dislikes that answer/question the button will become unavailable. Also, the score for the users will be modified depending on the type of the content and the type of the like.

Graphical user interface, text, application

Description automatically generated

Fig 3.6.1 Image before adding a like to an already disliked question

Graphical user interface, text, application, chat or text message

Description automatically generated

Fig 3.6.2 Image after adding a like to an already disliked question

# Architecture

## 5.1 Backend architecture

The architecture used for the application is called Layered Architecture. It consists in 4 layers, each of them calling through request the layers (most of the time only the next layer is called. In case of a closed layered architecture only the one below is called) and then obtaining the information from bottom to up as shown in the picture below:

The 4 layers are called:

* + Presentation
  + Business
  + Persistence
  + Database

The presentation layer represent the view and what the user can see. Usually here we have the classes that are used in UI or in frontend.

The business layer consists of classes that do all the logic for obtaining the information for the presentation layer. Here you can find classes such as: mappers, services and/or configuration.

In the persistence layer most of the classes are used for obtaining and storing data from the database. In the application, the main classes that exists in this layer are marked as “DTO” (Data Transfer Object) and is used to obtain information from executing queries. Using the mappers described in the business layer, we obtain the model classes used by the application (which are not visible directly to the user yet).

The database layer contains the databases which is accessed through different properties files containing the necessary information for connecting to the database and retrieving the data.Diagram

Description automatically generated

Figure 4.1: An example of a closed layered architecture

## Database diagram:

Diagram

Description automatically generated

## Class diagram

Diagram

Description automatically generated with medium confidence

## 5.4 Frontend architecture

The architecture used for the front-end is called Component Architecture and Unidirectional Architecture. This architecture depends on components when rendering pages, breaking up the rendered page in more, smaller, specific components.

For example, when displaying a list of elements, first we create a component for that given element.

Diagram, timeline

Description automatically generatedThe Unidirectional Architecture is used for directing the data through the application using states.

Fig 4.1 Unidirectional Data Flow Architecture

In react, this architecture is implemented through the usage of components’ props and state. When an event takes place, an action is done and the state is modified, resulting in a new rendering of the page using the new state.

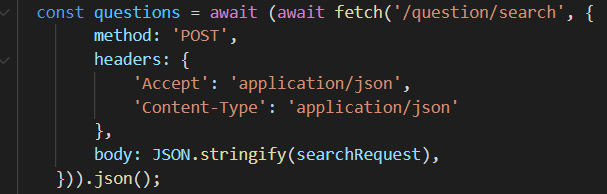
## Frontend – backend connection

This chapter will present the way the frontend connects to the backend and what is the flow of the information:

The basic flow of information at the highest level can be described as follows:

* The user presses a button that has an associated function / method
* The associated method is called
* The method contains a call to the backend through a fetch call on a URL
* The backend receives the URL
* It goes through the list of controllers and methods and find which is the corresponding method for the URL
* After finding the method, extracts the needed information: path variables, request body, etc. and then send the request towards the service
* In the service, the request is sent to the repository
* Repository will do a query on the database depending on the button pressed by the user and the action desired.
* The query will return an answer that will be mapped to a java object and then send back to the service
* The service sends it back to the controller
* The controller receives the information and then throw it back to the frontend
* The frontend will re-render with the new information
* The user observes the results of his actions

Examples of frontend calls to the backend:

* 
* Call used to send a search request for questions and then retrieve the information for the filtered questions
* URL: {baseUrl}/question/search
* Method: POST
* RequestBody: SearchRequest object
* 
* Call used to send a request for all question. It returns every entry from the database in question table
* URL: {baseUrl}/question/all
* Method: GET
* Text

  Description automatically generated
* Call used to send a request for deleting a question
* URL: {baseUrl}/question/delete/{questionId}
* Method: GET
* PathVariable: idQuestion – the id of the question to be deleted

# Testing

It was done mostly manually, going through each feature and see the results and then modifying the code if needed.

The manual testing was done by verifying the data displayed to the user and then verifying if the data was correctly updated in the database.

There are no real tests written, just some trials for the main component.

The main part of the application was tested manually through the UI, but some additional unit tests were added in the application.

Beside those, some tests were done using Postman. Below, some of the requests from Postman will be described.

**Disclaimer: the information from the body/url might need to be changed in order to work on your computer!**

## Create new user:

**Method: POST**

**URL:** <http://localhost:8080/users/create>

**BODY:**

{

    "idRole": 1,

    "username": "maria",

    "password": "pere",

    "email": "ana@gmail.com",

    "banned": **false**

}

## Add question:

**Method: POST**

**URL:** [http://localhost:8080 /question/create](http://localhost:8080/users/create)

**BODY:**

{

    "question":

    {

        "idUser": 3,

        "title": "Java",

        "text": "gere?",

        "creationDate": "2018-01-25T21:34:55"

    },

    "tags": [

        {

            "idTag" : "5",

            "name" : "java"

        }

    ]

}

## Add answer:

**Method: POST**

**URL:** [http://localhost:8080/answer/create](http://localhost:8080/answer/create%20)

**BODY:**

{

    "idUser": 2,

    "idQuestion": 4,

    "text": "perge",

    "creationDate": "2018-01-25T21:34:55"

}

## Get user:

**Method: GET**

**URL:** [http://localhost:8080 /users/2](http://localhost:8080/users/create)

**BODY: EMPTY**

## Get question:

**Method: GET**

**URL:** <http://localhost:8080/question/2>

**BODY: EMPTY**

## Get answer:

**Method: GET**

**URL:** <http://localhost:8080/answer/2>

**BODY: EMPTY**

## Get all users:

**Method: GET**

**URL:** [http://localhost:8080 /users/all](http://localhost:8080%20/users/all)

**BODY: EMPTY**

## Get all questions:

**Method: GET**

**URL:** <http://localhost:8080/question/all>

**BODY: EMPTY**

## Get all answers:

**Method: GET**

**URL:** <http://localhost:8080/answer/all>

**BODY: EMPTY**

## Delete user:

**Method: GET**

**URL:** [http://localhost:8080 /users/delete/2](http://localhost:8080%20/users/delete/2)

**BODY: EMPTY**

## Delete question:

**Method: GET**

**URL:** <http://localhost:8080/question/delete/2>

**BODY: EMPTY**

## Delete answer:

**Method: GET**

**URL:** <http://localhost:8080/answer/delete/2>

**BODY: EMPTY**

# Bibliography

1. <https://suas.readme.io/docs/why-unidirectional-architectures>
2. <https://www.postman.com/>
3. <https://azeynalli1990.medium.com/software-architecture-patterns-for-front-end-development-9e43e43cdfb3>
4. <https://testing-library.com/docs/react-testing-library/intro/>
5. <https://cs.uwaterloo.ca/~m2nagapp/courses/CS446/1195/Arch_Design_Activity/Layered.pdf>
6. <https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html>
7. <https://heap.io/topics/what-is-a-tech-stack>
8. <https://www.postman.com/>
9. <https://online.visual-paradigm.com/diagrams/templates/use-case-diagram/use-case-diagram-example-website-extend-and-include-use-case/>
10. <https://www.baeldung.com/java-spring-mockito-mock-mockbean>