StackOverflow Application

Motogna Alexandru

**BACKEND**

Architecture Diagram:

A picture containing text, electronics

Description automatically generated

* The client requests from the server and receives responses.
* The controller receives the requests and calls the service layer to execute them.
* The repository layer/database accesses the data using dependency injection, which is modelled by the model layer.

Class Diagram:

Diagram

Description automatically generated

(the dependencies from the Model layer have been omitted to keep the diagram more clean).

Database Diagram:

Diagram

Description automatically generated

End points(localhost:8081):

1. User Controller(/users):

* Get all users (/getAll, GET)
* Get user by id (/get, GET, Request parameter id – the user’s id)
* Delete user (/delete, DELETE, Request parameter id – the user’s id)
* Login user – get user from login credentials (/login, POST, Request Body – the credentials)
* Create user (/create, POST, Request body – the user’s object as JSON)
* Update user (/update, PUT, Request body – the user’s object as JSON)
* Ban user (/ban, PUT, Request param id – the user’s id)

1. Question Controller(/questions):

* Get all questions (/getAll, GET)
* Get question by id (/get, GET, Request parameter id – the question’s id)
* Get all from author by id (/getAllFromUser, GET, Request parameter authorid – the author’s id)
* Delete question (/delete, DELETE, Request parameter id – the question’s id)
* Create question (/create, POST, Request body – the question’s object as JSON, Request Param authorid - the author’s id)
* Update question (/update, PUT, Request body – the question’s object as JSON)
* Add tag to question (/addTag, POST, Request Param qid - the question’s id, Request Param tagid - the tag’s id)
* Remove tag from question (/removeTag, DELETE, Request Param qid - the question’s id, Request Param tagid - the tag’s id)
* Get all votes for a question (/votes, GET, Request parameter id – the question’s id)
* Vote question (upvote or downvote) (/upvote, POST, Request Param qid - the question’s id, Request Param userid - the user’s id who is voting, Request Param upvote – true if upvote or false if downvote)

1. Answer Controller (/answers):

* Get all answers (/getAll, GET)
* Get answer by id (/get, GET, Request parameter id – the answers id)
* Get all from author by id (/getAllFromQuestion, GET, Request parameter qid – the question’s id)
* Delete answer (/delete, DELETE, Request parameter id – the answer’s id)
* Create answer (/create, POST, Request body – the answer’s object as JSON, Request Param authorid - the author’s id)
* Update answer (/update, PUT, Request body – the answer’s object as JSON)
* Get all votes for an answer (/votes, GET, Request parameter id – the answer’s id)
* Vote answer (upvote or downvote) (/upvote, POST, Request Param aid - the answer’s id, Request Param userid - the user’s id who is voting, Request Param upvote – true if upvote or false if downvote)

1. Tag Controller (/tags):

* Get all tags (/getAll, GET)
* Get tag by id (/get, GET, Request parameter id – the tag’s id)
* Create tag (/create, POST, Request body – the tag’s object as JSON)
* Delete tag (/delete, DELETE, Request parameter id – the tag’s id)

**Frontend**

Implemented using Vue.js using Nuxt. The implementation is done using pages and components. Both pages and components are composed of 3 things:

* Template: the html body of the page
* Scripts: the javascript part of the page
* Style: the css part

Pages are the content that is being displayed once the webpage is being accessed. Components are part of the pages and used to prevent duplicate code and to generate a component for each element of a list in the JSON object that is being displayed.

Pages:

* Login page: Page for logging in. Can navigate to create account page. Once logged in, redirected to main page. (/login)
* Create account page: Page for creating account. Can navigate to login page. Once created an account, redirected to main page. (/register)
* Main page: Can redirect to profile page, questions page or create question page. (/mainpage)
* Profile page: Can view user data and all questions submitted given by user. (/profile)
* Questions page: Can view all submitted questions, clicking on one, redirects to its question view page. (/questions)
* Questions/id page: View the question and all its answers give. Can redirect to create answer page and upvote/downvote the question and the answers. (/questions/id)
* Create question page: Text input fields to input a question. (/createquestion)
* Edit question page: Text input fields to edit a question. (/questions/edit)
* Create Answer page: Text input fields to input an answer. (/createanswer)
* Edit Answer page: Text input fields to edit an answer. (/editanswer)
* Banned page: A simple page to show the user that was banned (/forbidden)

Only the login and create account pages can be accessed without being logged-in.

Components:

* Logo: Component containing a photo of the logo.
* TagItem: Is a box containing the tag name
* QuestionItem: Is a display of the question data. Contains a TagItem component for each tag.
* AnswerItem: Is a display of the answer data

Relevant screenshots:

* Login page

Graphical user interface, text, application

Description automatically generated

* Questions page

Graphical user interface

Description automatically generated with medium confidence

* Question View page
* Graphical user interface, application

  Description automatically generated

Connection between Frontend and Backend

The communication between the frontend and backend is done with the axios JavaScript library. When a certain data or action to the database is needed the respective endpoint from the backend is called.

**Bibliography:**

<https://vuejs.org/>

<https://nuxtjs.org/>

<https://vuetifyjs.com/en/>

<https://www.baeldung.com/spring-boot-vue-js>

<https://www.javatpoint.com/spring-boot-architecture>

<https://app.diagrams.net/>

<https://v2.vuejs.org/v2/cookbook/using-axios-to-consume-apis.html?redirect=true>