1 – Introduction

Stack Overflow is a question-and-answer platform that has become a reference for developers around the world. The Stack Overflow clone project aims to offer an open-source alternative to this platform by allowing users to ask questions and receive answers from other members of the community.

The main features of this Stack Overflow clone project will include:

- The ability to ask questions and answer those of other users.
- A voting system to evaluate the quality of questions and answers.
- A tag system to classify questions by subject.
- The ability to search for questions based on their content or tag.
- User profiles to allow users to track their activity on the site and gain visibility.
- A notification system to inform users of any activity related to their questions and answers.

The Stack Overflow clone project will be developed using modern technologies such as Nodejs for the backend, AngularJS for the frontend, and MongoDB for the database. It will be designed to be easily extensible and adaptable to the needs of users.

2 – Technology

To develop my project, I have chosen to use several modern technologies that will allow me to create a robust and high-performance application. For the backend, I opted for Node.js, a JavaScript development platform that offers great flexibility and high performance. I used Visual Studio Code IDE to develop my code and Postman to test my endpoints. For the frontend, I chose AngularJS, a JavaScript framework that makes it easy to create interactive and dynamic user interfaces. Finally, for the database, I chose MongoDB, a NoSQL database that offers great scalability and flexibility in data management. I used Mongo Compass to manage and visualize my data.

I chose these technologies because of their specific advantages for my project. Node.js is a fast and flexible technology that will allow me to create a RESTful API that can easily integrate with my user interface. AngularJS is a well-established JavaScript framework that offers great ease of use for frontend developers, while also being highly performant. Finally, MongoDB is a very popular NoSQL database that will allow me to efficiently store my application data and easily query it through its advanced query system. I also used Swagger-Autogen to automatically generate clear and precise documentation for my API. With these technologies and tools, I am confident that I will be able to achieve my goals for this project and provide a high-quality application to my users.

3 – Use diagram cases

A use case diagram is a graphical representation of the interactions between the actors (users) and the system being developed. It shows the different ways in which a user can interact with the system to achieve a specific goal. The diagram consists of use cases, actors, and relationships between them.

For my application, which is a StackOverflow clone, the use case diagram would include the following:

- Actors: Regular users, moderators, and administrators
- Use cases:
    - Search for questions/answers
    - Ask a question
    - Answer a question
    - Comment on a question/answer
    - Vote on a question/answer
    - Flag a question/answer
    - Edit a question/answer
    - Delete a question/answer
    - Mark a question as duplicate
    - Accept an answer
    - Add tags to a question
    - Follow a question
    - Message a user
    - Access user profile
    - Access moderation tools (for moderators and administrators)

The relationships between the actors and the use cases would show which actors are involved in each use case, and how they interact with the system to achieve their goals. This use case diagram would help to ensure that all the necessary features are included in the system and that the interactions between the users and the system are clearly defined.

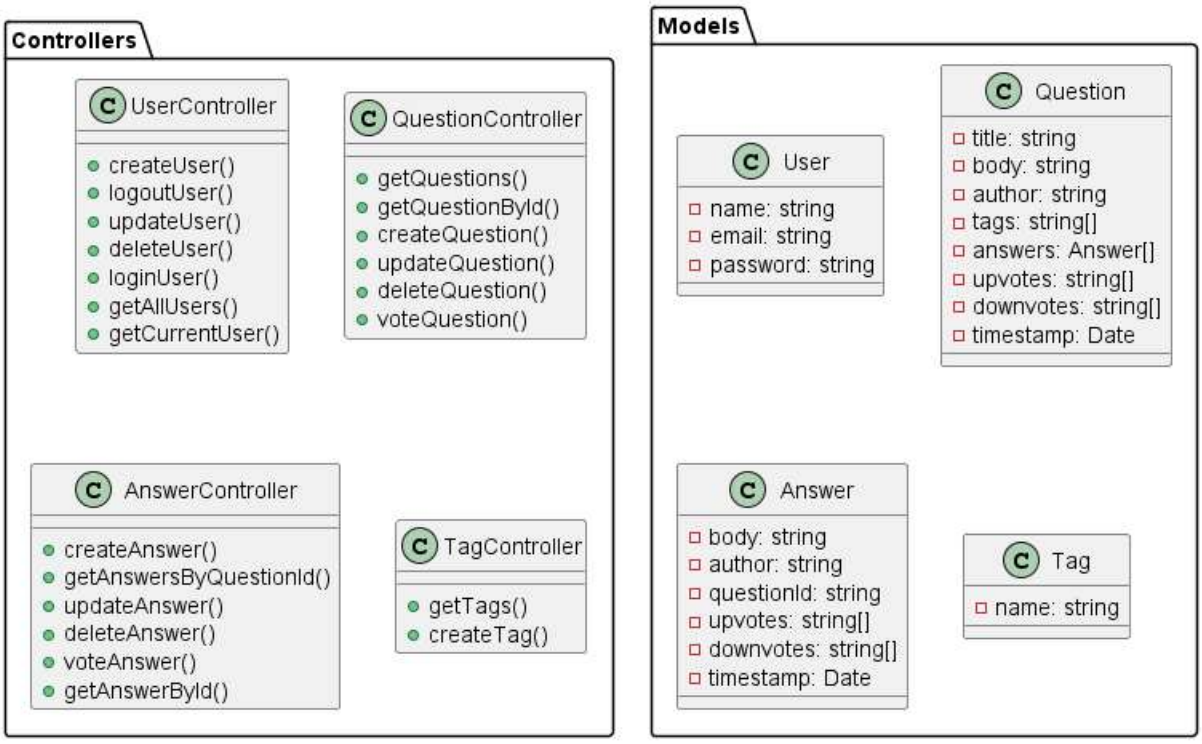
4 – Architecture

1. Config: The config folder typically contains configuration files for your application. This may include environment-specific settings, database configuration, logging configuration, and any other configuration files that your application requires.
2. Controller: The controller layer is responsible for handling incoming requests from the client and processing them. It is the intermediary between the user interface and the business logic layer. In this folder, you would typically have files that define the logic for each endpoint of your API or web application.

3. Middlewares: The middlewares layer sits between the incoming request and the controller layer, and it provides common functionalities for your application. These may include authentication, request validation, error handling, and other common tasks. In this folder, you would typically have files that define the logic for each middleware that your application requires.
4. Models: The models folder contains the data models that define the structure of your data and the relationships between them. This layer typically interacts with the database or other data storage systems and is responsible for retrieving and storing data.
5. Routes: The routes layer is responsible for defining the endpoints of your API or web application and mapping them to the corresponding controller methods. In this folder, you would typically have files that define the URL routes and the controller methods that handle each route.
6. Scripts: The scripts folder typically contains scripts that automate tasks related to your application, such as running tests, database migrations, or other maintenance tasks.
7. Services: The services layer contains the business logic of your application. This layer is responsible for implementing the actual functionality of your application, such as processing user input, performing calculations, and interacting with other systems.
8. Utils: The utils folder typically contains utility functions and helper modules that are used throughout your application. These may include commonly used functions or modules that provide additional functionality to your application.

In summary, the architecture of your application seems to follow a typical layered architecture, where each layer has a specific responsibility and interacts with the layers above and below it. The config, scripts, and utils folders provide supporting functionality for your application, while the controller, middlewares, models, routes, and services folders form the core of your application's functionality.

5 – Package Diagram



5 – Class Diagram

A class diagram is a type of diagram used in object-oriented programming (OOP) to visualize the structure of a system and the relationships and dependencies among its classes. It shows the classes, their attributes, and the methods that can be called on them, along with the relationships between the classes.

For the application you provided, the class diagram includes four main classes: User, Question, Answer, and Tag, along with several controllers that manipulate them.

The User class has three private attributes: name, email, and password. This class represents the users of the system, and their attributes are self-explanatory.

The Question class has several private attributes: title, body, author, tags, answers, upvotes, downvotes, and timestamp. The title and body attributes contain the text of the question, while author is the user who posted the question. The tags attribute is an array of strings that represent the tags associated with the question. The answers attribute is an array of Answer objects that represent the answers to the question. The upvotes and downvotes attributes are arrays of user IDs that have voted on the question, and the timestamp attribute is the date and time when the question was posted.

The Answer class has several private attributes: body, author, questionId, upvotes, downvotes, and timestamp. The body attribute contains the text of the answer, while author is the user who posted the answer. The questionId attribute is a string that

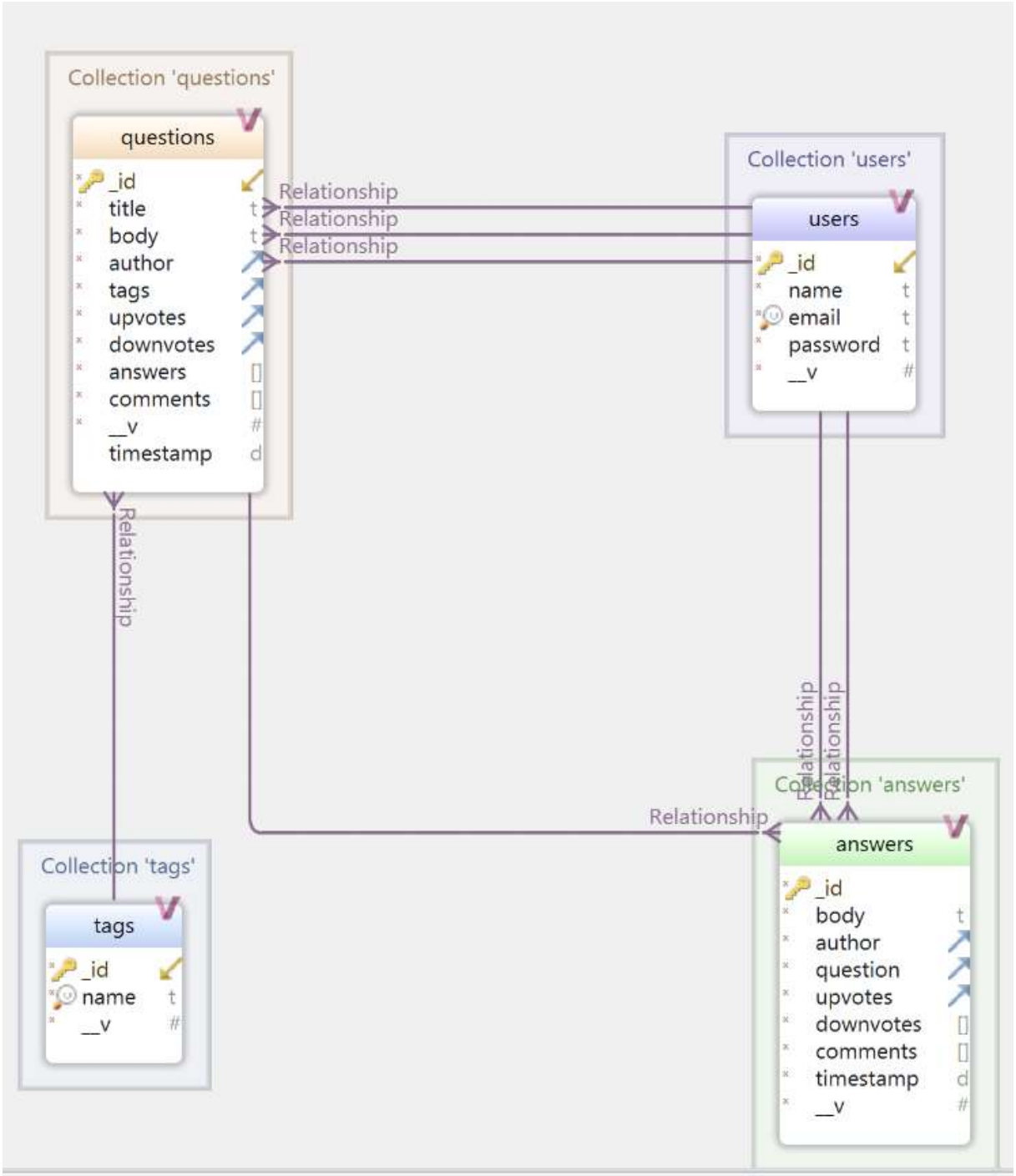
represents the ID of the question that the answer is associated with. The upvotes and downvotes attributes are arrays of user IDs that have voted on the answer, and the timestamp attribute is the date and time when the answer was posted.

The Tag class has a single private attribute: name. This class represents the tags that can be associated with questions, and the name attribute is a string that represents the tag name.

The controllers in the class diagram include UserController, QuestionController, AnswerController, and TagController. These controllers provide methods for creating, updating, deleting, and retrieving instances of the classes they manipulate. For example, the UserController has methods for creating, updating, deleting, and retrieving User objects, while the QuestionController has methods for creating, updating, deleting, and retrieving Question objects.

Overall, the class diagram provides a high-level overview of the entities and their relationships in the system, which can be useful for understanding the structure and behavior of the system.

7 – Database Diagram


Collection 'questions'
questions
_id
title
body
author
tags
upvotes
downvotes
answers
comments
__v
timestamp
Relationship
Relationship
Relationship
Collection 'users'
users
_id
name
email
password
__v
Relationship
Relationship
Relationship
Relationship
Collection 'answers'
answers
_id
body
author
question
upvotes
downvotes
comments
timestamp
__v
Collection 'tags'
tags
_id
name
__v

8 - Endpoints

REST API 1.0.0

[Base URL: localhost:3000/]

Schemes

HTTP

default

GET /users/current

Parameters Try it out

Name	Description
authorization string *(header)*	authorization

Responses Response content type application/json

Code	Description
200	OK
401	Unauthorized
500	Internal Server Error

GET /users/all

Parameters Try it out

Name	Description
authorization `string` *(header)*	authorization

Responses Response content type application/json

Code	Description
200	OK
401	Unauthorized
500	Internal Server Error

POST /users

Parameters Try it out

Name	Description

body
`object`
(body)
	Example Value Model
	
	{
	"name": "any",
	"email": "any",
	"password": "any"
	}
	
	Parameter content type
	application/json

Responses Response content type application/json

Code	Description
200	OK
400	Bad Request

Code	Description
422	Unprocessable Entity
500	Internal Server Error

POST /login

Parameters Try it out

Name	Description

body
object
(body)
	Example Value Model
	
	{
	"email": "any",
	"password": "any"
	}
	
	Parameter content type
	application/json

Responses Response content type application/json

Code	Description
200	OK
400	Bad Request
422	Unprocessable Entity
500	Internal Server Error

POST /logout

Parameters Try it out

Name	Description
authorization string	authorization

Name	Description
(header)	

Responses Response content type: application/json

Code	Description
200	OK
401	Unauthorized
500	Internal Server Error

PUT /users/{userId}

Parameters Try it out

Name	Description
userId * required string *(path)*	userId
authorization string *(header)*	authorization

body
object
(body)
	Example Value Model
	{
	"name": "any",
	"email": "any",
	"password": "any"
	}
	Parameter content type
	application/json

Responses Response content type: application/json

Code	Description
200	OK
400	Bad Request
401	Unauthorized
500	Internal Server Error

DELETE /users/{userId}

Parameters Try it out

Name	Description
userId * required string *(path)*	userId
authorization string *(header)*	authorization

Responses Response content type application/json

Code	Description
200	OK
401	Unauthorized
500	Internal Server Error

GET /questions

Parameters Try it out

No parameters

Responses Response content type: application/json

Code	Description
200	OK
500	Internal Server Error

POST /questions

Parameters Try it out

Name	Description
authorization string *(header)*	authorization

body
object
(body)
	Example Value Model
	{
	"title": "any",
	"tags": "any",
	"body": "any"
	}
	Parameter content type
	application/json

Responses Response content type: application/json

Code	Description
200	OK
401	Unauthorized
500	Internal Server Error

GET /questions/{questionId}

Parameters Try it out

Name	Description
questionId * required `string` *(path)*	questionId

Responses Response content type application/json

Code	Description
200	OK
404	Not Found
500	Internal Server Error

PUT /questions/{questionId}

Parameters Try it out

Name	Description
questionId * required `string` *(path)*	questionId
authorization `string` *(header)*	authorization

body
`object`
(body)
	Example Value Model
	
	{
	"title": "any",
	"tags": "any",
	"body": "any"
	}
	
	Parameter content type
	application/json

Responses Response content type application/json

Code	Description
200	OK
401	Unauthorized
404	Not Found
500	Internal Server Error

DELETE /questions/{questionId}

Parameters Try it out

Name	Description
questionId * required string *(path)*	questionId
authorization string *(header)*	authorization

Responses Response content type application/json

Code	Description
200	OK
401	Unauthorized
404	Not Found
500	Internal Server Error

POST /questions/{questionId}/vote

Parameters Try it out

Name	Description
questionId * required `string` *(path)*	questionId
authorization `string` *(header)*	authorization
body `object` *(body)*	**Example Value** Model `{ "vote": "any" }` **Parameter content type** application/json

Responses Response content type application/json

Code	Description
200	OK
400	Bad Request
401	Unauthorized
404	Not Found
500	Internal Server Error

GET /questions/{questionId}/answers

Parameters Try it out

Name	Description
questionId * required `string` *(path)*	questionId

Responses Response content type: application/json

Code	Description
200	OK
500	Internal Server Error

POST /questions/{questionId}/answers

Parameters Try it out

Name	Description
questionId * required string *(path)*	questionId
authorization string *(header)*	authorization

body
object
(body)
	Example Value Model
	{
	"body": "any"
	}
	Parameter content type
	application/json

Responses Response content type: application/json

Code	Description
201	Created
401	Unauthorized
500	Internal Server Error

GET /answers/{answerId}

Parameters Try it out

Name	Description
answerId * required string *(path)*	answerId

Responses Response content type application/json

Code	Description
200	OK
404	Not Found
500	Internal Server Error

PUT /answers/{answerId}

Parameters Try it out

Name	Description
answerId * required string *(path)*	answerId
authorization string *(header)*	authorization

Name	Description

body
object
(body)
	Example Value Model
	
	{
	"body": "any"
	}
	
	Parameter content type
	application/json

Responses Response content type application/json

Code	Description
200	OK
401	Unauthorized
404	Not Found
500	Internal Server Error

DELETE /answers/{answerId}

Parameters Try it out

Name	Description
answerId * required string *(path)*	answerId
authorization string *(header)*	authorization

Responses Response content type application/json

Code	Description
200	OK
401	Unauthorized
404	Not Found
500	Internal Server Error

POST /answers/{answerId}/vote

Parameters Try it out

Name	Description
answerId * required string *(path)*	answerId
authorization string *(header)*	authorization

body
object
(body)
	Example Value Model
	{
	"vote": "any"
	}
	Parameter content type
	application/json

Responses Response content type application/json

Code	Description
200	OK
400	Bad Request
401	Unauthorized
404	Not Found

Code	Description
500	Internal Server Error

GET /tags

Parameters Try it out

No parameters

Responses Response content type application/json

Code	Description
200	OK
500	Internal Server Error

POST /tags

Parameters Try it out

Name	Description
authorization `string` *(header)*	authorization
body `object` *(body)*	**Example Value** Model `{ "name": "any" }` **Parameter content type** application/json

Responses Response content type application/json

Code	Description
201	Created
401	Unauthorized
500	Internal Server Error