Swiggy i++ Stage 2 Final Project

By Sujata Dwivedi

Provides the functionality for routes handling and also makes the development faster and easier.

Most preferred way for authentication in APIs because it is stateless and easy to manage.

It helps in providing types and other functionality like classes and object which makes debugging of the code much easier compared to Javascript.

Used to containerize the project for easy setup.

Famous ORM used for security purposes and will help in future to migrate to another database with minimal development efforts.

Because there were clear relations between the different entities and the schema was not loose.

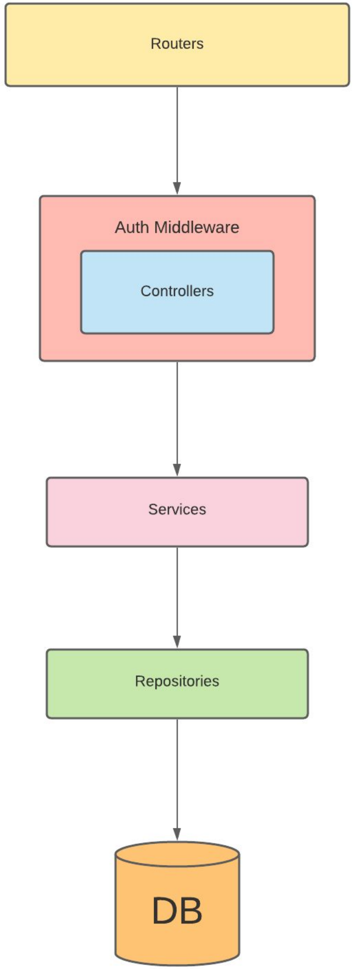
Handles all the routes

Auth middleware handles jwt authentication

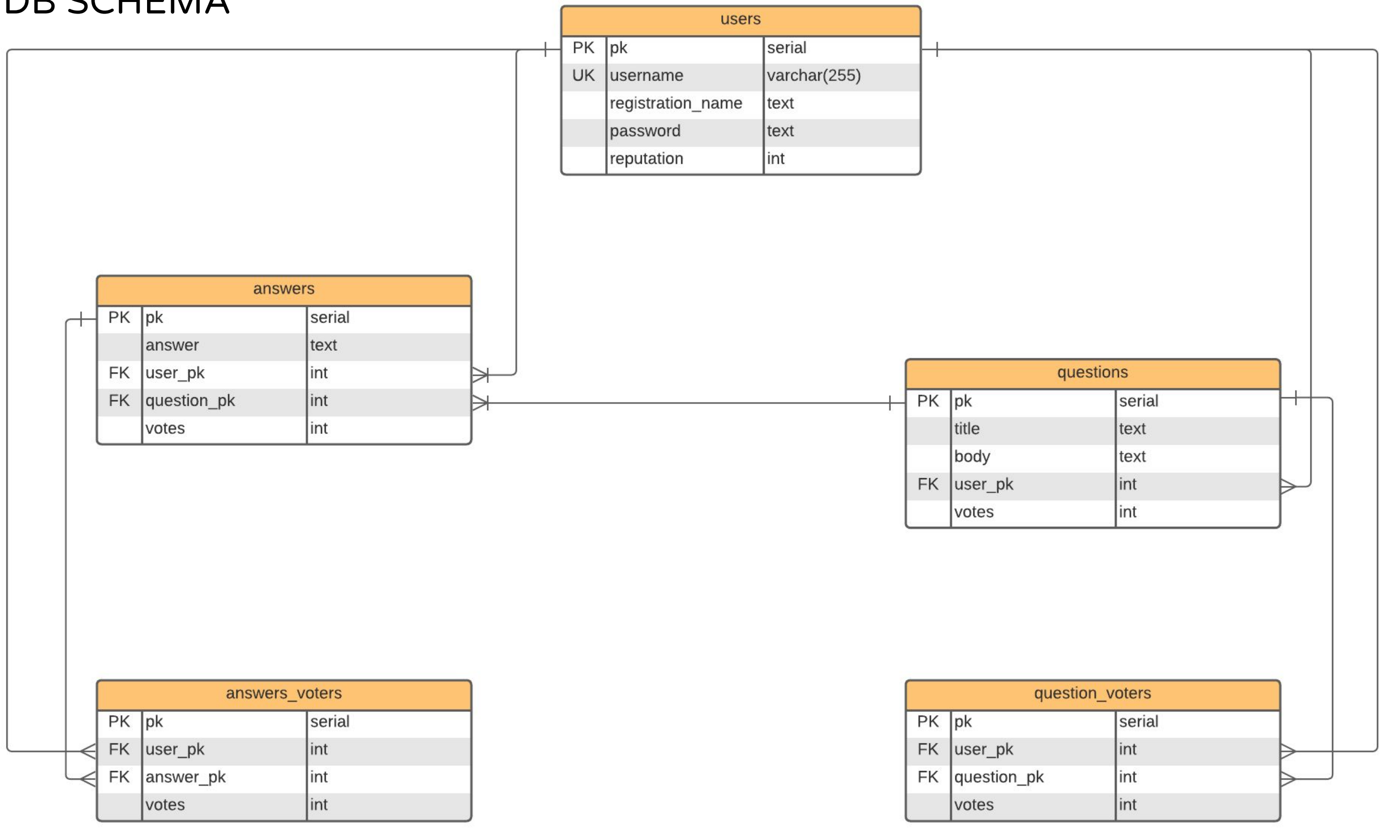
Controllers - All http request processing and response formation is done here.

Services handles core business logic

Repositories handles all the database queries with the help of entities.



DB SCHEMA



users

PK	pk	serial
UK	username	varchar(255)
	registration_name	text
	password	text
	reputation	int

answers

PK	pk	serial
	answer	text
FK	user_pk	int
FK	question_pk	int
	votes	int

questions

PK	pk	serial
	title	text
	body	text
FK	user_pk	int
	votes	int

answers_voters

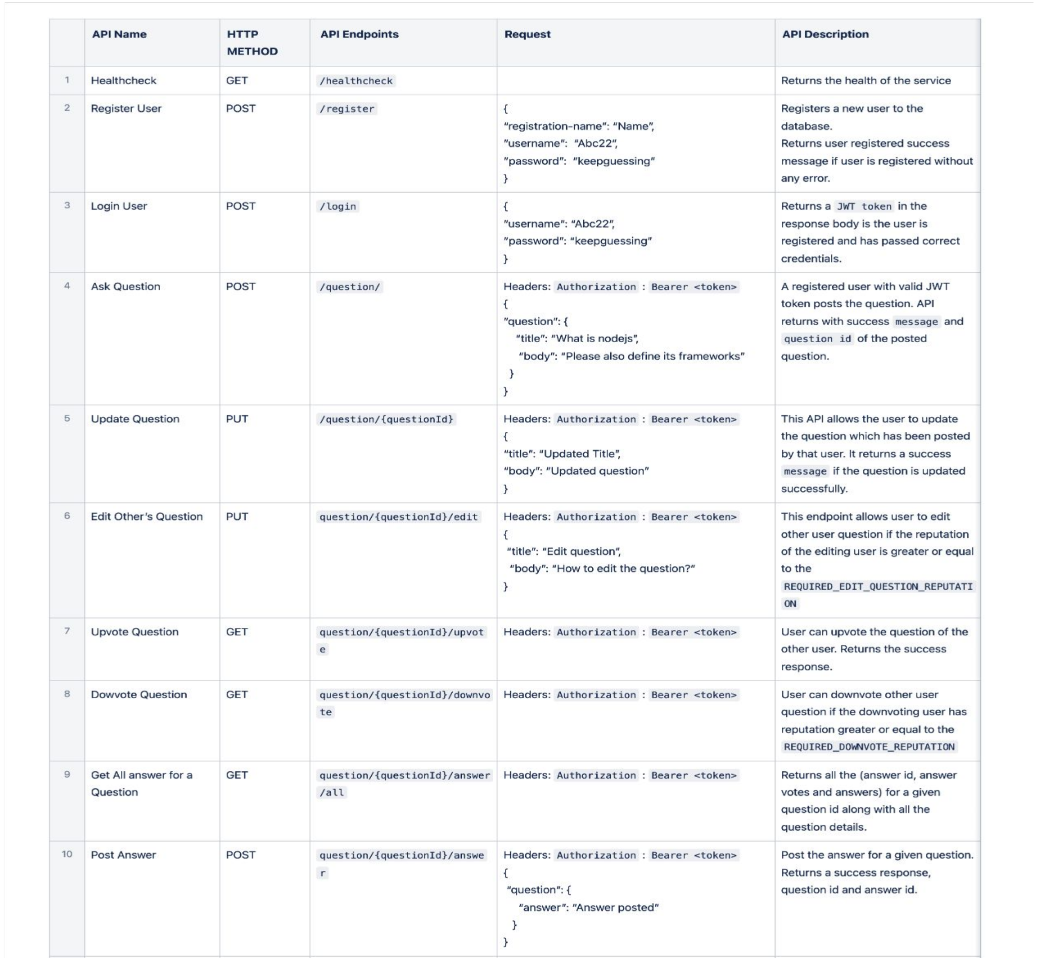
PK	pk	serial
FK	user_pk	int
FK	answer_pk	int
	votes	int

question_voters

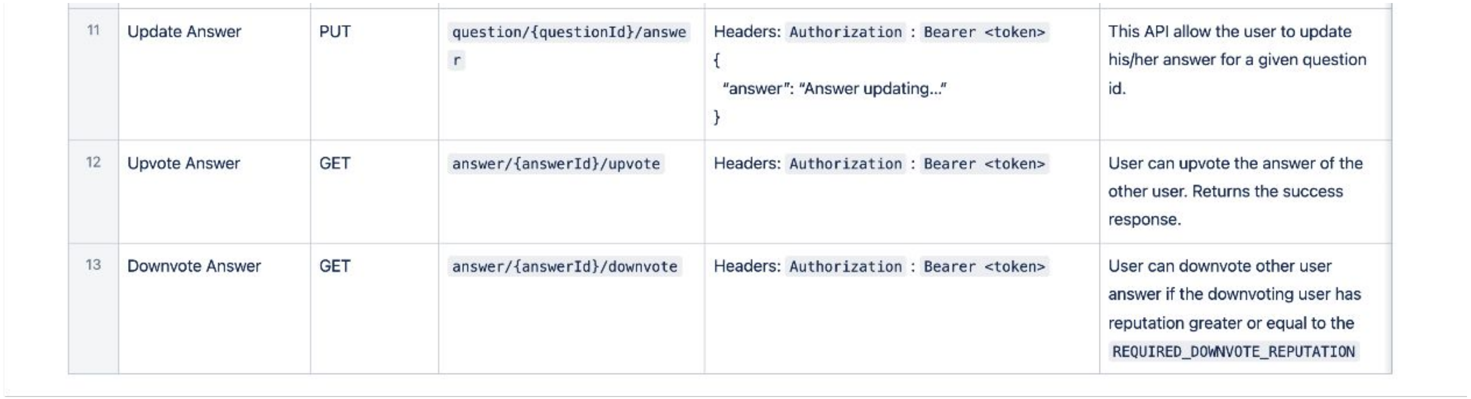
PK	pk	serial
FK	user_pk	int
FK	question_pk	int
	votes	int

Features of Stackunderflow

1. Register User
2. Login User
3. Post Question
4. Answer Question
5. Update Question
6. Edit other’s question - Reputation requirement
7. Upvote Question
8. Downvote Question - Reputation requirement
9. Update Answer
10. Upvote Answer
11. Downvote Answer - Reputation requirement
12. Get all answer for a question



	API Name	HTTP METHOD	API Endpoints	Request	API Description
1	Healthcheck	GET	`/healthcheck`		Returns the health of the service
2	Register User	POST	`/register`	{ "registration-name": "Name", "username": "Abc22", "password": "keepguessing" }	Registers a new user to the database. Returns user registered success message if user is registered without any error.
3	Login User	POST	`/login`	{ "username": "Abc22", "password": "keepguessing" }	Returns a `JWT token` in the response body is the user is registered and has passed correct credentials.
4	Ask Question	POST	`/question/`	Headers: `Authorization` : `Bearer <token>` { "question": { "title": "What is nodejs", "body": "Please also define its frameworks" } }	A registered user with valid JWT token posts the question. API returns with success `message` and `question id` of the posted question.
5	Update Question	PUT	`/question/{questionId}`	Headers: `Authorization` : `Bearer <token>` { "title": "Updated Title", "body": "Updated question" }	This API allows the user to update the question which has been posted by that user. It returns a success `message` if the question is updated successfully.
6	Edit Other's Question	PUT	`question/{questionId}/edit`	Headers: `Authorization` : `Bearer <token>` { "title": "Edit question", "body": "How to edit the question?" }	This endpoint allows user to edit other user question if the reputation of the editing user is greater or equal to the `REQUIRED_EDIT_QUESTION_REPUTATION`
7	Upvote Question	GET	`question/{questionId}/upvote`	Headers: `Authorization` : `Bearer <token>`	User can upvote the question of the other user. Returns the success response.
8	Dowvote Question	GET	`question/{questionId}/downvote`	Headers: `Authorization` : `Bearer <token>`	User can downvote other user question if the downvoting user has reputation greater or equal to the `REQUIRED_DOWNVOTE_REPUTATION`
9	Get All answer for a Question	GET	`question/{questionId}/answer/all`	Headers: `Authorization` : `Bearer <token>`	Returns all the (answer id, answer votes and answers) for a given question id along with all the question details.
10	Post Answer	POST	`question/{questionId}/answer`	Headers: `Authorization` : `Bearer <token>` { "question": { "answer": "Answer posted" } }	Post the answer for a given question. Returns a success response, question id and answer id.



11	Update Answer	PUT	`question/{questionId}/answer`	Headers: `Authorization` : `Bearer <token>`
				{
				"answer": "Answer updating..."
				}
					This API allow the user to update his/her answer for a given question id.

12	Upvote Answer	GET	`answer/{answerId}/upvote`	Headers: `Authorization` : `Bearer <token>`	User can upvote the answer of the other user. Returns the success response.
13	Downvote Answer	GET	`answer/{answerId}/downvote`	Headers: `Authorization` : `Bearer <token>`	User can downvote other user answer if the downvoting user has reputation greater or equal to the `REQUIRED_DOWNVOTE_REPUTATION`

Fig. 3

Thank You!