**API REPORT**

**Introduction**

APIs are a widely used tool for providing clients with secure access to data and services. In the context of healthcare, APIs play a vital role in facilitating seamless interactions between various stakeholders, including doctors, patients, and administrators, in a hospital management system. Our hospital API aims to streamline healthcare operations and enhance patient care by enabling authorized users to perform a wide range of operations efficiently. The API is designed with robust security measures, ensuring that only authorized users can access sensitive data and perform permitted operations. The app allows simple operations like the creation of appointments and visits.

**Technology:**

Node.js

Express.js

Postgresql

Elasticsearch

**Express API**

The Express Application is the main API that handles all the requests. The app has multiple endpoints. There are three types of users, admin, patient and doctor. For each of the users there are separate routes through which they can perform all their operations. The API provides facilities for authorization so that only authorized users can perform operations, blocking all the unauthorized requests.

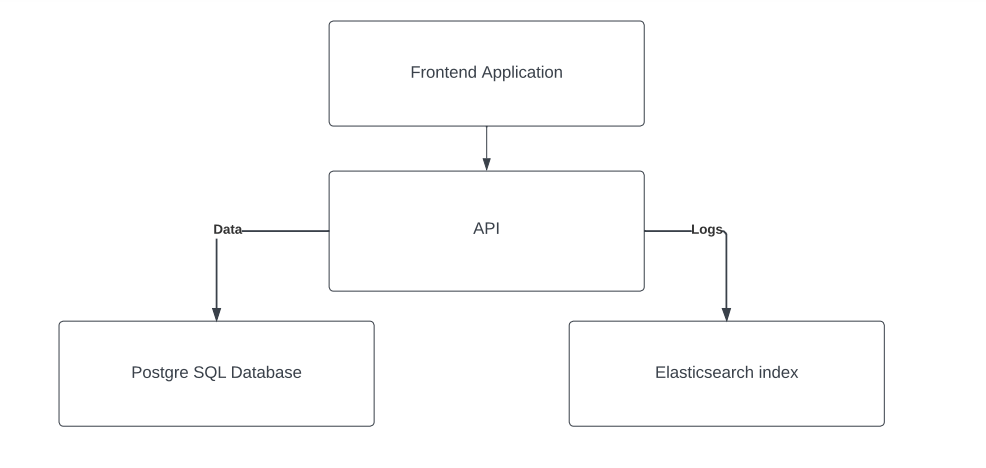
**PostgreSQl**

Postgresql is our database to which our API connects with. All the data is fetched through from the database and all the insertions are made. Apart from checks, authorization and authentication in the API Layer, there are also constraints added on the database layer to ensure data validation. Node.js provides with pg-sql client that helps us connect to our database from our application.

**Elasticsearch**

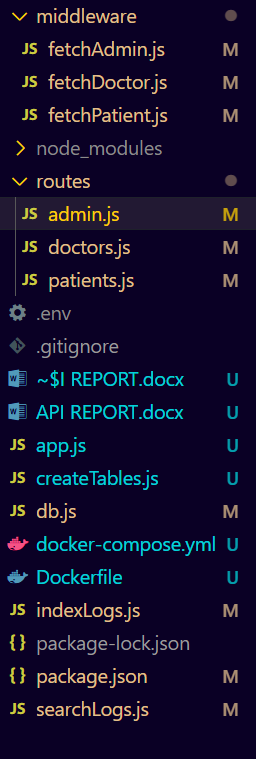
Elasticsearch is a datastore used for enhancing search queries and develop efficient search engines. As our data is stored in postgresql database, the logs of all operations are stored in an Elasticsearch index. These logs are used for keeping security checks and help detect in a rise in malicious activity. The logs can be helpful in the further enhancement and development of the application. The elasticsearch client @elasticsearch/elasticsearch aids in connecting with elasticsearch.

**API Architecture**

****

The API has a basic architecture. The Frontend sends some request to the database (GET, POST, PUT, DELETE). The API handles the requests as specified in the endpoints. The APi first checks if the user making the request is authorized, after the user is authorized, it checks for data validation and upon passing all the checks, the API inserts/fetches data to/from the database. Along that, the API also generates a log and indexes the log in elasticsearch.

**Folder Structure**

****

In the root folder, we have our main app.js file which is initially run when the server is started. In the root folder we have other files that contain different functions.

**Db.js**

This file has the function for connecting with the postgresql database.

**createTables.js**

When the docker container for the app is initialized for the first time on a machine, the file creates all the tables and schema.

**indexLogs.js**

This file has the function for indexing the logs in elasticsearch index.

**searchLogs.js**

This file fetches all the logs from the elasticsearch database. This function is used by the admin to check the logs.

Apart from the route folder we have a node\_modules folder that contains all the dependancies, a routes folder and a middlewasre folder

**Routes Folder**

The routes folder contains all the routes that are used by different users for making requests to the API. As mentioned before, there are three levels of user, doctor, patient and Admin. For each of these users there is a separate file that contains their request endpoints.

**Middleware Folder**

The middleware folder contains the functions that are passed as a middleware to the API requests to check if the user is authorized to make the request.

**Authorization**

Authorization is done in this application through a module in Node.js called jsonwebtoken. This module is used to create a token when the user logs in. This token is then stored in the browser cookies which are passed in the request header whenever the user makes any request. If the required token is present in the request header, then the user is authorized and allowed to proceed with the operation otherwise the API does not proceed with the request.

**Password Hashing**

Like any other application, the API does not store the passwords as a plain text but first hashes them and then stores that hashed password in the database. Bcrypt module in Node.js helps us for password hashing and comparing any password entered for logging in with the hashed password present in the database.

**API Endpoints**

The API has a number of endpoints. There are three users and different endpoints for each user, the details of these end points is as following:

**Admin Endpoints**

**admin/login** is the endpoint for the admin to login.

**admin/create** is the endpoint through which an admin can create another admin.

**admin/createpatient** is the endpoint through which admin can create new patients.

**admin/createdoctor** is the endpoint for creating new doctors.

**admin/getadmins** for fetching all the admins information from the database.

**admin/getdoctors** is the endpoint for fetching the information regarding all the doctors

**admin/get patients** is the request for fetching patients information.

**admin/patient** is a request of method delete through which the admin can mark the patient as deleted in the database.

**admin/doctor** is a delete request for deleting a doctor account.

**admin/getpatientbyid** fetches any patient from the database from the patient id

**admin/getpatientbyemail** fetches a patient from database through email.

**admin/getdoctorbyid** fetches a doctor from the database using the id key in the database.

**admin/getdoctorbyemail** fetches doctor from the database using the email of the doctor.

**admin/getadminbyid** helps the admin to fetch any admin (including himself) from the database using the admin id.

**admin/getadminbyemail** fetches admin information with the desired email.

**admin/logs** allows admin to view all logs from elasticsearch.

**Patient Endpoints**

**patient/login** allows the patient to login and creates a patient token for authorization.

**patient/createappointment** allows patient to create an appointment with any doctor.

**patient/createvisit** is used to create a new visit.

**patient/updatestatus** is helpful for updating the status of an appointment to booked, cancelled or completed.

**patient/allappointments** helps the patient to fetch all the appointments the patient has made.

**patient/allvisits** helps the patient to fetch all visits that belong to the patient.

**patient/allexams** allows the patient to fetch all the exams of the patient.

**patient/delete** allows the patient to delete account.

**Doctor Endpoints**

**doctor/login** allows the doctor to login and creates a doctor token for authorization.

**doctor/createappointment** allows doctor to create an appointment with any patient.

**doctor/createvisit** is used to create a new visit.

**doctor/createexam** allows doctor to make new exams.

**doctor/updatestatus** is helpful for updating the status of an appointment to booked, cancelled or completed.

**doctor/allappointments** helps the doctor to fetch all the appointments the doctor has made.

**doctor/allvisits** helps the doctor to fetch all visits that belong to the doctor.

**doctor/allexams** allows the doctor to fetch all the exams of the doctor.

**doctor/delete** allows the doctor to delete account.

**Dockerization**

The application has been dockerized so that it can be easily hosted and run on other machines without the need of installing all the dependancies required by the application. The docker compose file creates three containers, one for the express api, one for the postgresql and one container for containing all the dependancies of elasticsearch.

**Docker-Compose**

Docker Compose is a tool for defining and running multi-container Docker applications. It uses a YAML file (usually named docker-compose.yml) to configure the services, networks, and volumes required for your application. This file allows to define multiple services, their configurations, and how they interact with each other.

**The Three Containers:**

**Express API Container:**

The Express API container holds the Node.js application that serves as the hospital management API. This container is based on a Docker image that includes Node.js and any required application code, configuration files, and dependencies. The Dockerfile used to build this image specifies the necessary steps to create the environment for the Node.js application.

**PostgreSQL Container:**

The PostgreSQL container contains the PostgreSQL database used to store and manage the hospital management data. This container is based on the official PostgreSQL Docker image and includes the database setup, user credentials, and any initial data required for the application to function.

**Elasticsearch Container:**

The Elasticsearch container holds the Elasticsearch service and its dependencies. This container uses the official Elasticsearch Docker image or a custom image that includes the necessary configuration for Elasticsearch.

**Database Schema**

The database schema comprises of six tables. An admin table, a patients table, a doctors table, an appointments table, a visits table and exams table

**Admin Table**

As the name suggests, the admin table contains the data regarding the admins. It contains the id, admin name, the admin email, the admin password and the created\_by i.e which other admin created this admin account. The created\_by is set to 0 for admin that is present by default. The id attribute plays the role of primary key.

**Patients Table**

The patients table contains all the information for a patient. The attributes(columns) of the table include patient id(primary key), email, name, password, deleted and created\_by. Three deleted attribute shows that if the account is marked deleted(deactivated) while the created\_by attribute is a foreign key that refers to the id in admin table. It is the id of the admin who created the patient account.

**Doctors Table**

The doctors table has the same keys as the patients table id, email, name, password, deleted and created\_by where created\_by is the foreign key from the admin table.

**Appointments Table**

The appointments table has the keys id, patient\_id, doctor\_id, start\_time, end\_time, status and visit\_id. patient\_id, doctor\_id and visit\_id are the foreign keys. Patient\_id refers to the id in patients table. It is the id of the patient who has the appointment with the doctor. Doctor\_id refers to the doctor who has the appointment with the patient. Visit\_id is the visit id from the visits table that refers to the visit in which the appointment was scheduled.

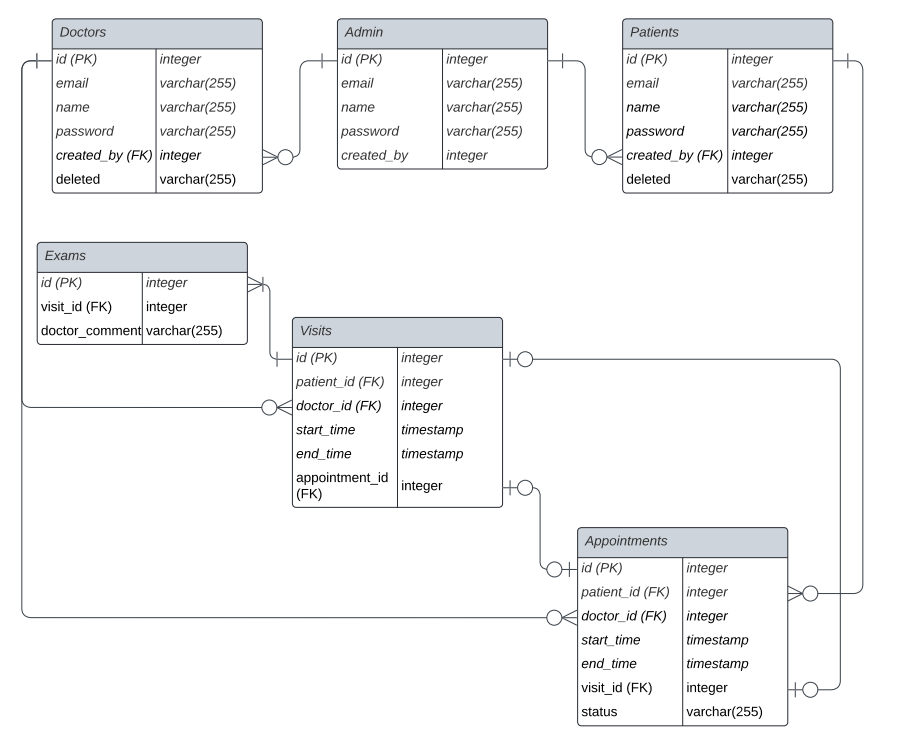
**Visits Table**

The visits table contains the keys id, patient\_id, doctor\_id, start\_time, end\_time, appointment\_id. Patient\_id, doctor\_id and appointment\_id are the foreign keys that refer to the id attribute in patients, doctors and appointments table respectively.

**Exams Table**

The exams table contains the exams that are taken during any visit. Its attributes are id, visit\_id and doctor\_comment where visit\_id is the foreign key referring to the id key in visits table.

**ER Diagram**

****