मुंबई विद्यापीठ
University of Mumbai
Re-accredited with A++ Grade
(CGPA 3.65) by NAAC (3rd Cycle 2021)

Search to Arrival

Submitted in the partial fulfilment of the requirements of the degree of

Master of Science (Computer Science)

By

Tejas Thakare

Under the Guidance of
Dr. Jyotshna Dongardive

University Department of Computer Science, Kalina, Santacruz (E), Mumbai - 400098

Year: 2021-2023

मुंबई विद्यापीठ
University of Mumbai
Re-accredited with A++ Grade
(CGPA 3.65) by NAAC (3rd Cycle 2021)

Search to Arrival

Submitted in the partial fulfilment of the requirements of the degree of

Master of Science (Computer Science)

By

Thakare Tejas Vijay

Seat no ____________

Under the Guidance of
Dr. Jyotshna Dongardive

University Department of Computer Science, Kalina, Santacruz (E), Mumbai - 400098

Year: 2021-2023

CERTIFICATE

This is to certify that, **Mr. Thakare Tejas Vijay, Seat No. _______________** studying in MSc Computer Science, Semester – IV has satisfactorily completed the project on **" Search to Arrival** "as laid down by the **University of Mumbai** for the year 2021-2023.

Dr. Jyotshna Dongardive
Project Guide

Dr. Jyotshna Dongardive
Head
Dept. of Computer Science

Examiner

Date:

Place: Mumbai

ACKNOWLEDGEMENT

This project was undertaken and expedited under the expert guidance of Dr. Jyotshna Dongardive, for the course titled Project Implementation. The technologies and methodologies of the subjects that were taught, mentored, and supported have been incorporated in this project, without which this project would not have been conceptualized, taken shape and reached the present stage.

I take this opportunity to extend my gratitude for your exemplary guidance, mentorship, and encouragement throughout the course of this project.

Thanking You,

Yours Sincerely.

Thakare Tejas Vijay

TABLE OF CONTENT

Chapter 1: Introduction

In today's fast-paced world, students require easy access to information about the colleges they plan to attend. They need to be able to quickly find the location of the college, the time it takes to travel there, basic details about the college, and contact information. Additionally, students often need to navigate the complex public transportation systems in major cities, such as Mumbai.

To address this need, your project leverages the power of Google Maps to provide students with an easy-to-use interface that provides all the information they require in one place. By incorporating travel time data, the project helps students plan their commute and avoid delays. The project also includes information about each college, including its location, basic details about the college, and contact information.

Moreover, the project aims to provide students with a feature similar to the Chalo app for BEST buses in Mumbai. This feature allows students to view real-time bus locations, check bus schedules, and receive alerts about bus delays or changes in routes.

Overall, your project provides students with a comprehensive tool that simplifies the process of finding and accessing information about Mumbai University-affiliated colleges. It streamlines the process of planning and commuting to college, making it easier for students to focus on their studies and achieve their academic goals.

1.1 Tools and Techniques:

To create this web application, we have used various tools and techniques to ensure its efficiency and effectiveness. We have utilized the Mern stack, which includes MongoDB, Express, React, and Node.js, to develop our application's frontend and backend. We have also used Google map services like Google maps JavaScript API, Direction API services, Geocode API, and Place API to provide users with accurate and reliable mapping data. Additionally, we have incorporated AWS cloud technology and SQL database to enhance the scalability and performance of our application. We have also used Colab for data analysis and visualization to ensure that the information we provide is relevant and useful for our users.

1.2 Need and Difference:

The need for our web application arises from the difficulties that students face when it comes to selecting the right college, finding the best route to their college, and accessing detailed information about colleges. Our application provides a solution to these problems by providing students with advanced college search features, optimal routing, detailed college information, and the ability to compare colleges. What sets us apart from other college search platforms is our focus on providing optimal routing to college, which is a unique feature not commonly found in other applications. Additionally, our web application is designed to be user-friendly and intuitive, providing a seamless experience for students looking to find the right college for them.

Objective

1. The web application described in this report is aimed at addressing common problems faced by students while traveling to their college. Students often face various challenges when it comes to choosing the right college, finding the best route to their college, and accessing detailed information about colleges. In response to these problems, the developers have created a user-friendly web application that provides advanced college search features, optimal routing from home to college, detailed information about specific colleges, and the ability to compare two colleges based on various factors such as course duration, travel time, placement, etc.

2. The application is designed to assist students in making informed decisions about their college education and to simplify the process of finding the right college. With the advanced search features, students can easily find colleges that match their specific requirements and preferences, such as location, course duration, and placement opportunities. The routing feature helps students to find the most optimal route from their home to the college, considering factors such as traffic, public transportation, and distance.

3. In addition, the application provides detailed information about specific colleges, including their history, facilities, faculty, courses, and placement records. This information is essential for students to make informed decisions about the college they want to attend. Moreover, the comparison feature allows students to compare two colleges based on various factors, making it easier to choose the right college.

4. Overall, the web application is designed to simplify the process of finding the right college for students. It provides a comprehensive solution to the common problems that students face while searching for colleges, such as finding the right college, optimal routing, and accessing detailed information about colleges. The application is user-friendly and provides advanced search features that make it easy for students to find the college that best fits their needs and preferences.

Chapter 2: Literature Review

In today's digital age, most colleges have their own websites that provide information about their academic programs, admission requirements, campus life, and other relevant details. However, the amount of information provided by these websites can be overwhelming for the average student, especially if they are trying to compare multiple colleges at once.

Additionally, many college search engines are available that allow students to search for colleges based on various criteria such as location, majors, size, and other factors. However, these search engines may not always provide the necessary options for filtering and comparing colleges effectively.

Furthermore, the user interface of some of these websites and search engines can be subpar, making it difficult for students to navigate and find the information they need.

Our application aims to simplify the college search process by providing a user-friendly interface that allows students to easily search for colleges based on their preferences and compare them side-by-side. By offering advanced filtering options and an intuitive user interface, our application helps students make informed decisions about the best-fit college for them.

In summary, while many resources are available for college search, they can be overwhelming and limited in functionality. Our application addresses these issues by providing a streamlined and comprehensive approach to college search.

Table 1. Summary of Literature review

Sr. No.	**Title**	**Year**	**Algorithm and Methodology**	**Conclusion**
1.	Hopper	2019	AI	Hopper provides price forecasting and making predictions about user intent. While it has focused on airline travel, they may be well-positioned to expand into other types of travel and lodging. For example, a user's air travel query is highly correlated with the type of lodging they will purchase. The company will need to make additional investments to learn customer preferences and provide personalized recommendations about additional services or upgrades

2.	Mobile Application Development for Tourist Guide in Pekanbaru City	2020	KNN	The purpose of this research is to design an application on an Android smartphone for Tour Guide Pekanbaru. The system will use GPS to find the user point and can provide information about tourist attractions, hotels, restaurants, and shopping so that they can use the time in Pekanbaru effectively.
3.	Mobile Smart Travelling Application For Indonesia Tourism	2017	image recognition using landmark detection feature from Google Cloud Vision Application Program Interface (API)	The application is named Smart Travelling and has many features in order to facilitate the tourists who visit Indonesia. Based on the conducted study, the smart travelling application is successfully published in Google Play Store and evaluated by 35 selected participants. It shows the smart travelling application is really useful

				for tourists either local or foreigner, especially the image recognition feature which can automatically recognize the tourist attractions with great performance, i.e., 86% accuracy. Even though the application already launched, there are several points which can be considered for the improvement in the future, such as use our own algorithm for the image recognition instead of Google Cloud Vision API, reconstruct the application for other platforms, redesign the interface, etc.
4.	A Travel guide android Application	2021	Android Studio, Firebase	The motive of this android application, is to manage a tour for the users through mobile computing with complete

				requirements and needs in location.
5.	Travel with us mobile application	2021	Android studio, Java programming	This research paper explains the evolution of tourism industry and applications based on tourism and what is the impact of corona virus on tourism industry. What features and problems are associated with tourist applications? And how our application is going to fill these research gaps.
6.	Advanced Tour Guide Android App	2022		Tourism App Help User to Search Best location For the Holiday. Also Book nearest Hotels. Tourism motivations include relaxation, strengthening family. In addition whether for casting is main point of our project, tourists are also motivated to travel by other factors.

7.	Mobile Application Based Intelligent Role of Information Technologies in Tourism Sector	2021	React Native, Apollo GraphQL, Redux	Tourism App Help User to Search Best location For the Holiday. Also Book nearest Hotels. Tourism motivations include relaxation, strengthening family. In addition whether for casting is main point of our project, to.
8.	Virtual reality and modern tourism	2020	VR technology,3D visualizations	The purpose of this paper is to provide an insight as to how recent trends in virtual reality (VR) have changed the way tourism and hospitality industry communicates their offerings and meets the tourists' needs
9.	Digital business models in cultural tourism	2021		Digitalization had a relevant impact on the cultural tourism sector, both demand and supply. If, on the one hand, advances in digital technologies provided

				tourists with new mobile services able to amplify the cultural experience, on the other hand, they catalyzed the development of new business models by digital enterprises. This paper has a twofold purpose: to detect business models and key characteristics of mobile apps for cultural tourism and to analyze the offering of app-based services in this sector.
10.	College Raptor	2014	Predictive Analytics	College Raptor is a college search website that provides information on colleges and scholarships. Their college search tool allows students to filter by factors such as location, majors, and cost.
11.	CollegeSimply	2011	Search Engine	CollegeSimply is a college search website

				that provides information on colleges and scholarships. Their college search tool allows students to filter by factors such as location, majors, and test scores.
12.	The College Board	2011	BigFuture is a college planning website that provides information on colleges and financial aid. Their college search tool allows students to filter by factors such as location, majors, and test scores.	BigFuture is a college planning website that provides information on colleges and financial aid. Their college search tool allows students to filter by factors such as location, majors, and test scores.
13.	Unigo	2008	Search engine	Unigo is a website that provides reviews and ratings of colleges and universities. Their college search tool allows students to filter by

				factors such as location, majors, and campus life.
14.	"Exploring the Factors Influencing the Adoption of Travel Apps: A Study of Indonesian Tourists"	2021	Online survey of 272 Indonesian tourists	Perceived usefulness, ease of use, and trust are the main factors influencing the adoption of travel apps among Indonesian tourists.
15.	"Enhancing the Student Experience through the Use of Mobile Applications: A Study of Students in a UK University"	2020	Online survey of 350 UK university students	Mobile apps can enhance student engagement, communication, and learning experiences in higher education, but require careful consideration of user needs and preferences.
16.	"Usability and User Experience Evaluation of	2020	Evaluation of 10 tourism mobile apps using usability heuristics and user	Many tourism mobile apps have poor usability and user experience, with common issues such as

	Tourism Mobile Applications"		experience questionnaire	poor navigation, slow loading, and lack of relevant content.
17.	"Enhancing Visitors' Experience through Personalized Tourism Websites"	2019	Online survey of 193 visitors to a Greek island	Personalization of tourism websites can enhance visitors' experience by providing customized recommendations, personalized content, and interactive features.
18.	"Designing an Interactive Mobile Application for Tourists to Enhance Their Travel Experience"	2019	Case study of a mobile app development project for tourists	An interactive mobile app can enhance tourists' travel experience by providing personalized recommendations, real-time information, and social networking features.
19.	"The Effect of E-Word-of-Mouth on Online Travel Purchase	2019	Online survey of 353 Chinese online travel consumers	E-word-of-mouth has a significant positive effect on online travel purchase intention, mediated by

	Intention: An Application of the Extended Technology Acceptance Model"			perceived usefulness and trust.
20.	"A Study on the Usability of College Websites"	2019	Evaluation of 100 college websites using usability heuristics	Many college websites have poor usability, with common issues such as poor navigation, cluttered layout, and lack of clear information.
21.	"A Study of Online Review Analysis and Sentiment Classification of Travel Websites"	2018	Analysis of online reviews of 6 travel websites using sentiment analysis	Sentiment analysis can help to identify the strengths and weaknesses of travel websites based on user feedback, and inform website design and improvement.

Chapter 3: Materials and Methods

3.1 Materials and Methods

3.1.1 Materials:

1. **Programming Languages:** The web application was developed using programming languages such as HTML, CSS, JavaScript, and TypeScript.
2. **Frameworks and Libraries:** Several frameworks and libraries were used in the development of the web application, including React.js, Express.js, and Prisma.
3. **API and Databases:** To fetch and store data, API endpoints and databases were utilized. The web application uses the Google Maps API to fetch data related to optimal routing from home to college, and Prisma is used to connect to and query the database.
4. **Development Tools:** Various development tools were used, including Visual Studio Code, Git for version control, and NPM for package management.

3.2.2 Methods:

1. Requirements Analysis: A detailed analysis of the requirements was conducted to identify the problems faced by students while traveling to their college.

2. Design: Based on the analysis, the web application's design was created, keeping in mind the user interface, usability, and ease of use.

3. Development: After finalizing the design, the web application was developed using the materials mentioned above.

4. Testing: The web application was tested thoroughly to ensure that it met the requirements and provided accurate results.

5. Deployment: After the testing phase, the web application was deployed to a web server to make it accessible to users.

6. **Maintenance**: The web application will be regularly maintained and updated to ensure that it stays up-to-date with the latest technologies and functionalities.

3.2 Data Pre-Processing

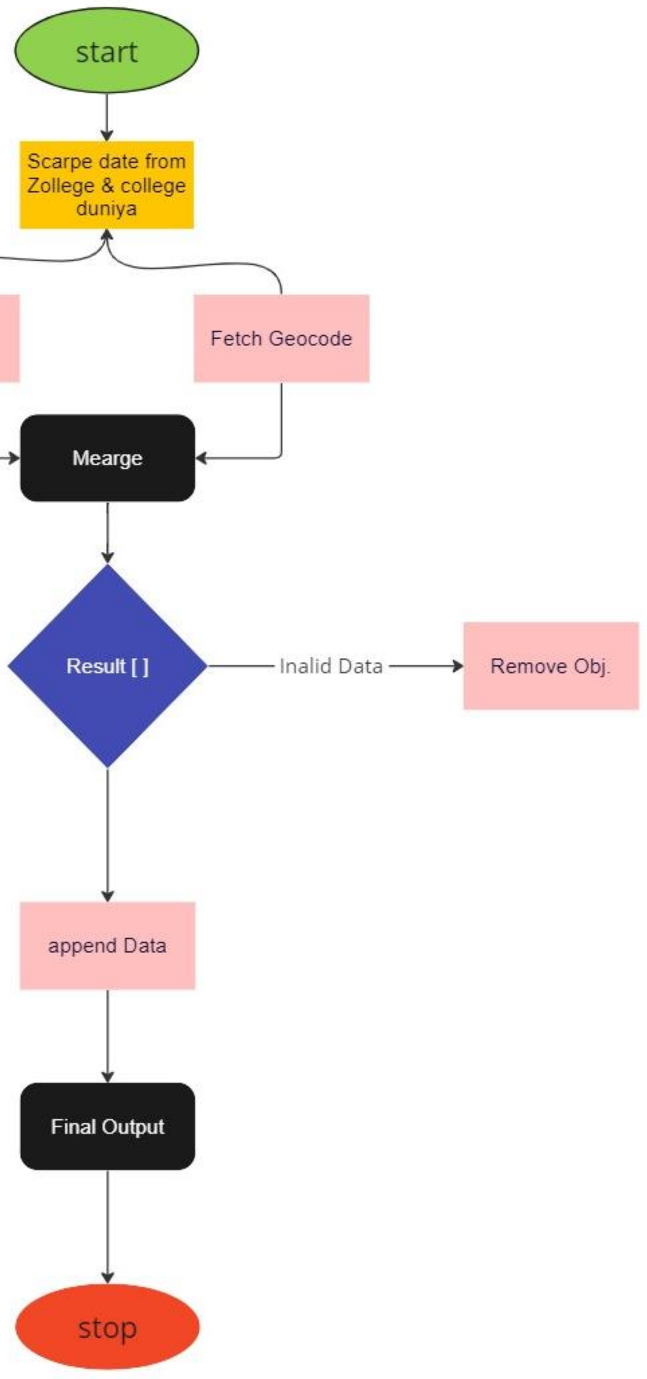


Fig. 1 Data Pre-Processing

1. **Scraping Data**: You start by scraping data from websites like Zollege and College Duniya. This involves extracting information about 202 colleges from their websites.
2. **Code in Colab**: You have written the code for scraping and processing the data in Colab. Colab is a cloud-based platform that allows you to write and execute Python code.
3. **Filtering Data:** After scraping the data, you filter it based on the desired data structure. This step helps you organize and extract the relevant information from the scraped data.
4. **Geocode API:** You use the Google Geocode API to fetch the geocode (geographical coordinates) of each college. This information allows you to determine the precise location of the colleges.
5. **Appending Geocode:** Once you obtain the geocode for each college, you append this data to the respective JSON file. This step helps associate the geographical information with the college data.
6. **Data Validation:** You perform data validation to ensure the accuracy and integrity of the collected information. This step helps identify any inconsistencies or errors in the data and allows you to rectify them.
7. **Merge Objects:** After validating the data, you merge all the college objects into a single array. This consolidation simplifies further processing and analysis of the data.
8. **Store in Data.json:** The merged college data is then stored in a JSON file called Data.json. JSON is a popular file format for storing structured data, and it allows easy access and manipulation of the data.

9. **Fetching Additional Data:** The Data.json file contains URLs that provide additional information about the colleges. You use these URLs to fetch more data, such as ratings, placement companies, courses, faculty, and facilities.
10. **Append Additional Data:** The fetched additional data is appended to the respective college JSON files. This step enriches the existing college data with more detailed information.
11. **Final JSON Creation:** Once all the additional data has been appended, you create a final JSON file that includes the complete information for each college.
12. **Local Export:** Finally, you export the final JSON file locally, which means saving it on your local machine or a specified location. This allows you to access and use the data conveniently.

3.2 Backend

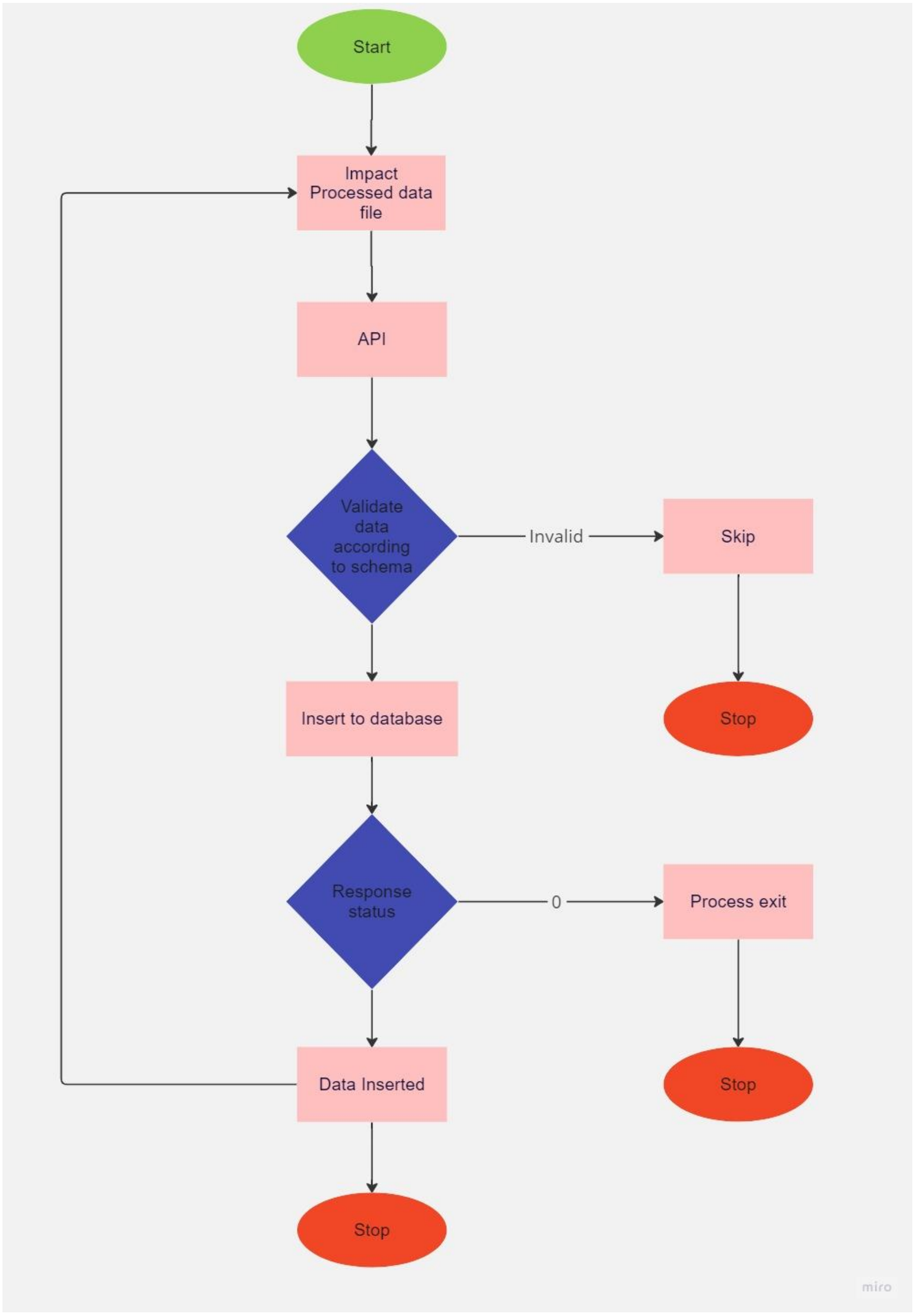


Fig.2 Backend

1. **Export.json Input:** We start with an Export.json file obtained from the previous data processing task. This file contains the collected and processed data of colleges.
2. **Creating APIs:** We create six APIs using Node.js and Express.js, following the MVC (Model-View-Controller) pattern. These APIs provide different functionalities to interact with the college data.

   a. **getListOfColleges**: This API retrieves a list of colleges from the database.
   b. **groupByPlacement**: This API groups colleges based on their placement information, allowing us to retrieve colleges with similar placement records.
   c. **groupByCourse**: This API groups colleges based on the courses they offer, enabling us to fetch colleges that provide specific courses.
   d. **groupByCompanies**: This API groups colleges based on the companies that participate in their placement drives, helping us find colleges associated with particular companies.
   e. **getOneCollege**: This API retrieves detailed information about a specific college based on its unique identifier.
   f. **deleteCollege**: This API allows us to delete a college from the database using its unique identifier.

3. **Prisma ORM and SQL Server:** We use the Prisma ORM (Object-Relational Mapping) along with a SQL Server database. Prisma simplifies database

operations by providing an interface to interact with the database using JavaScript/TypeScript.

4. **Schema Creation:** We create a schema using Prisma ORM, which defines the structure and relationships of the data entities (such as colleges) in the database.
5. **College Specific Controller:** We create a specific controller for handling college-related operations. This controller contains the logic and functions required to process and manage college data.
6. **API Implementation:** Using the Express.js framework, we implement the APIs by defining the routes, request handling functions, and connecting them to the controller. These APIs handle incoming requests, perform the necessary operations on the data, and generate appropriate responses.
7. **Data Validation:** During API execution, we validate the incoming data to ensure it is in the correct format and contains all the required fields. If the JSON data is invalid or lacks essential information, it is discarded.
8. **Custom Logger:** We implement a custom logger that helps us track the progress of API requests. It logs messages in the terminal, indicating whether an API request was successful or failed, providing insights into the execution flow.
9. **Data Insertion:** The processed data is inserted into the SQL Server database using Prisma ORM. The database acts as the persistent storage for our college data.
10. **Serving Data to Frontend:** With the data inserted into the database, our backend is now ready to serve the data to the frontend. The frontend can make requests to the implemented APIs to retrieve and display college information as required.

3.3 Frontend

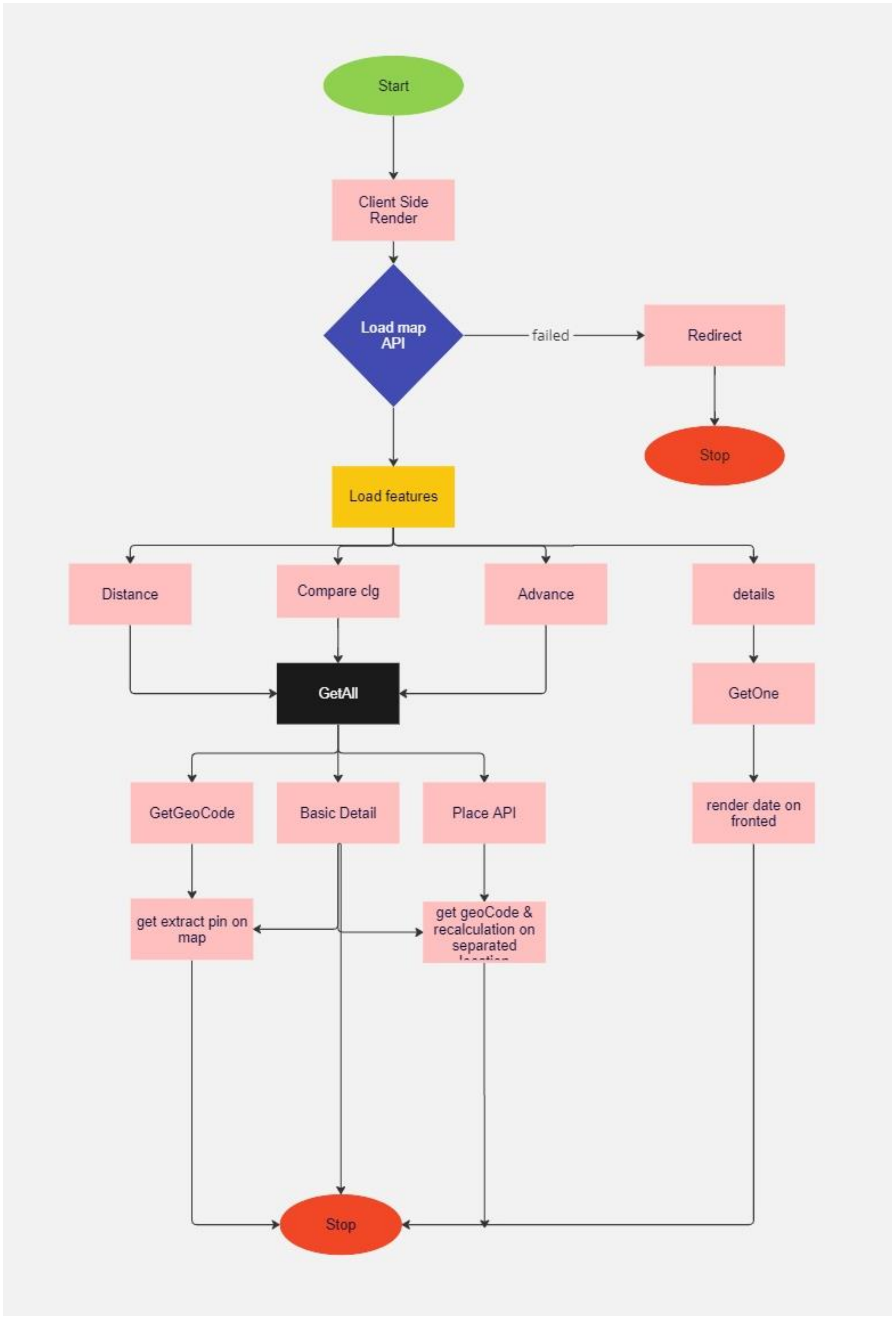


Fig. 3 Frontend

1. **Data Fetching:** Your application starts by fetching data from a data source, such as an API or a database. This data likely contains information about colleges, including details like fees, courses, faculty, and placement.
2. **State Management:** Once the data is fetched, you store it in the application's state. Since you mentioned using Redux, it's likely that you're using Redux's store to manage the application state. The fetched data is stored in the Redux store so that it can be accessed by different components throughout your application.
3. **Data Structuring:** After fetching and storing the data, you structure it according to the needs of your frontend components. This step involves organizing the data in a way that makes it easier to work with and display in your UI. For example, you might filter the data based on certain criteria or transform it into a format that is more suitable for rendering.
4. **Rendering Components:** Once the data is structured, you render the necessary components in your application. You mentioned using functional components, so you would likely create functional components for each feature (Advance, Detail, Distance, Compare) as well as smaller features. These components will receive the necessary data from the Redux store through props and display it in the UI.
5. **User Interaction:** Users can then interact with the different features of your application. For example, in the Advance feature, they can use multiple filters to filter the data and view the filtered results on the page. In the Distance feature, they can view colleges within a specific circle on a map and see their details below. In the Detail feature, they can view the exact location of a college on a map along with all the details about that college. And in the Compare feature, they can compare colleges based on various criteria like fees, courses, faculty, and placement.

6. **Advance Feature:**
   - This feature allows users to apply multiple filters to the data and view the filtered results.
   - When a user applies filters, the frontend sends a request to the backend API, passing the filter parameters.
   - The backend API processes the request, applies the filters to the data, and returns the filtered results.
   - The frontend receives the filtered results from the API response and updates the UI to display the filtered data.
7. **Detail Feature:**
   - This feature displays detailed information about a specific college.
   - When a user selects a college, the frontend sends a request to the backend API, providing the college's unique identifier.
   - The backend API retrieves the details of the selected college from the database and returns them as a response.
   - The frontend receives the college details from the API response and renders them in the UI, including the location on a map.
8. **Distance Feature:**
   - This feature uses a circle to indicate which colleges are within a specific distance.
   - When a user defines the circle's parameters (center and radius), the frontend calculates the boundaries of the circle.
   - The frontend then sends a request to the backend API, passing the circle's boundaries.
   - The backend API queries the database to find colleges that fall within the circle's boundaries and returns them as a response.

- The frontend receives the colleges within the circle from the API response and updates the map and the list of colleges in the UI accordingly.

9. **Compare Feature:**
   - This feature enables users to compare colleges based on different criteria.
   - When a user selects multiple colleges to compare, the frontend sends a request to the backend API, providing the college identifiers.
   - The backend API retrieves the details of the selected colleges from the database and returns them as a response.
   - The frontend receives the college details from the API response and displays them in a comparison view, allowing users to compare fees, courses, faculty, placement, and other criteria.

3.4 Performance Measure

As a web application, the performance of our college search tool can be measured in several aspects.

1. Time Required in API Calls:

The time required in API calls is an important performance metric for any web application. The industry standard for API call response time is between 50ms to 250ms. In our college search tool, we aim to keep the API call response time within this range to provide a fast and responsive user experience.

To achieve this, we can optimize our code for performance, use efficient data structures, and optimize the database queries. We can also implement caching techniques to reduce the response time of frequently requested data.

2. Error/Exception Handling:

Error/Exception handling is an important aspect of any web application as it ensures that the application continues to function smoothly even in the presence of errors. In our college search tool, we use best practices for error handling such as try-catch blocks, maintaining error logs, and displaying meaningful error messages to users.

On the frontend, we use error boundaries to catch errors that occur during rendering and prevent the entire application from crashing. We also use Higher-Order Components (HOCs) to wrap components with error-handling logic.

3. Caching Data:

Caching data is an effective way to improve the performance of a web application by reducing the response time of frequently requested data. In our college search tool, we use Redis as a caching mechanism to store frequently requested data such as college search results and user preferences.

By caching data, we can reduce the number of API calls required, thereby improving the overall performance of the application.

4. Making Use of Best Practices:

To further improve the performance of our college search tool, we can make use of best practices such as debounce for search, avoiding uncontrolled component use, and testing.

Debouncing the search input can improve the performance of the search functionality by reducing the number of API calls required. We can also avoid using uncontrolled components, which can cause performance issues due to excessive re-rendering.

Finally, testing is an important aspect of any web application as it ensures that the application is functioning as expected and can help identify performance bottlenecks. By implementing automated testing, we can ensure that our college search tool is performing optimally and providing the best user experience possible.

3.5 Hardware and Software use

a. **Hardware:**

The hardware used for developing the college search tool is a computer. A computer is an essential tool for any software development as it provides the necessary hardware resources to run the software and tools required for development.

b. **Cloud:**

We have also used cloud services such as GitPod and AWS for development and deployment. GitPod provides a cloud-based development environment that allows developers to easily collaborate on projects and provides a pre-configured development environment for the college search tool. AWS is used for deployment, hosting the application in the cloud to ensure high availability and scalability.

c. Software Used:

a. **Editor:** We have used several code editors for development, including VSCode, GitPod, Docker, and Wordbeanch. These editors provide essential features such as code highlighting, debugging, and code completion, making development faster and more efficient.

b. **Backend:** For the backend, we have used several technologies, including Prisma, ExpressJS, NodeJS, Redis, MVC, and SQL Server. Prisma is used as an ORM to map the database schema to the application code. ExpressJS is a popular web framework for NodeJS used to build web applications. NodeJS is a runtime environment used to execute JavaScript code on the server-side. Redis is used as a caching mechanism to improve application performance. MVC (Model-View-Controller) is an architectural pattern used to separate the application logic into three separate components. SQL Server is used as a database management system to store and retrieve data.

c. **Frontend:** For the frontend, we have used several technologies, including React18, ViteJS, GoogleAPI, Jest, RTL (React Testing Library), DaisyUI, and Tailwind. React18 is a popular JavaScript library used for building user interfaces. ViteJS is a build tool used to bundle the application code for deployment. GoogleAPI is used to integrate the Google Maps API into the application. Jest and RTL are used for testing, providing an efficient and easy-to-use testing framework. DaisyUI and Tailwind are CSS frameworks used to style the user interface of the application.

Overall, the hardware and software used for the development of the college search tool provide a robust and efficient platform for developing and deploying web applications.

3.8 Implementation details

The college travel website created using React JS is a web application that allows users to search for colleges based on their location and travel options. It comprises four different components, each with its unique features.

The Advance component allows users to customize their search results based on specific requirements such as the type of college, tuition fees, and other filters. The Distance component calculates the distance between the user's location and the selected college. The Detail component displays the relevant details about the college, such as the name, address, and other relevant information. Lastly, the Compare component compares different colleges based on specific criteria, such as tuition fees, location, and other factors.

The website allows users to search for college names and locations using a search box feature. Users can input the college name, and the website will display relevant results based on the search query. Additionally, the advanced feature allows users to customize their search results based on their specific requirements. They can apply filters to refine the search results based on various parameters such as the type of college, location, and tuition fees. The sorting feature helps users navigate the search results more easily, making it easier to find what they're looking for.

After applying the desired filters and customization, the website generates a map that displays the most efficient travel routes using different modes of transportation such as walking, car, train, and bus. To implement this feature, the website likely utilized APIs for mapping and transportation information.

The website was likely developed using React JS, along with other technologies such as HTML, CSS, and JavaScript. Additionally, the developer likely wrote code for the different components and features, integrating them together to create a seamless user experience. Overall, the college travel website created using React JS provides an intuitive and user-friendly platform for students to search for colleges based on specific criteria and make informed decisions.

To implement this project, the developer likely started by defining the requirements and functionalities of the college travel website. They then selected the appropriate technologies to use, including React JS for the front-end, and potentially Node.js and Express.js for the back-end.

The developer would have likely used HTML and CSS to create the website layout and design, while utilizing JavaScript for the interactive features such as the search and filter options.

For the mapping and transportation information, the developer likely integrated APIs such as Google Maps API or OpenStreetMap API. These APIs allow for the display of maps, route calculations, and transportation information.

To create the advanced, distance, detail, and compare components, the developer would have likely written code to define the functionality of each component. For example, the distance component would have required code to calculate the distance between two locations, while the advanced component would have required code to handle the filtering and customization of search results.

Finally, the developer would have integrated all the components and features together to create a seamless user experience. This would have included testing the application for bugs and errors, and making adjustments as necessary.

Overall, the implementation of this project would have required a strong understanding of web development technologies and best practices, as well as familiarity with APIs and integrating various components together to create a functional and user-friendly website.

Chapter 4: Experimental Setup and Results

4.1 Setup

To set up the experimental environment, we first need to set up a database. Here, we are using Docker to run a container that will host our database.

Docker is a platform that allows us to easily create, deploy, and run applications in containers. Containers are lightweight, portable, and self-contained environments that can run on any machine with Docker installed.

To start the database container, we first need to download the appropriate image from Docker Hub. We can do this by running the following command in the terminal:

- **Code:**

```
docker pull <database_image_name>
```

Once the image is downloaded, we can start the container using the following command:

- **Code:**

```
docker run --name <container_name> -p <port_number>:<container_port_number> -d <database_image_name>
```

Here, we are giving our container a name, mapping the container's port to a port on our machine, and running the container in detached mode (in the background). We can then check the status of the container using the following command:

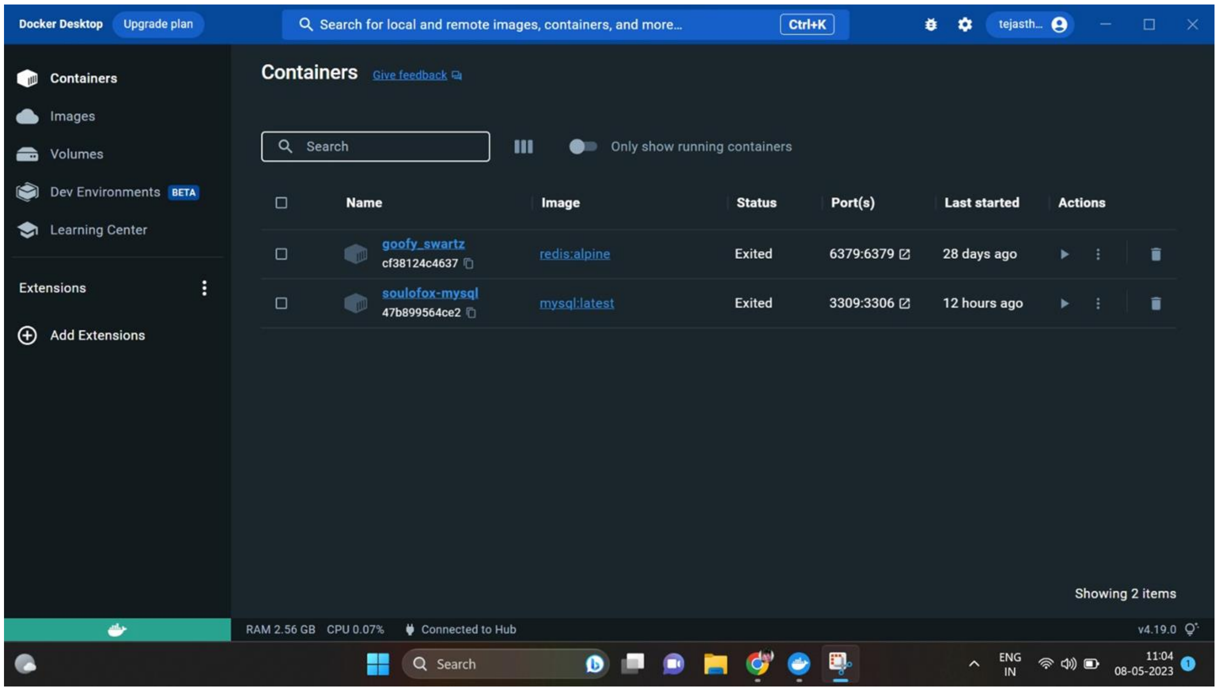
- **<u>Code:</u>**

```
docker ps
```

This should display a list of running containers, including the one we just started.

Once the container is up and running, we can connect to it using our database client (e.g. SQL Server Management Studio for SQL Server). We can then create our database tables, seed them with data, and perform any necessary setup for our application.

Overall, setting up the database using Docker allows us to easily manage our database environment and ensure consistency across different machines and environments.



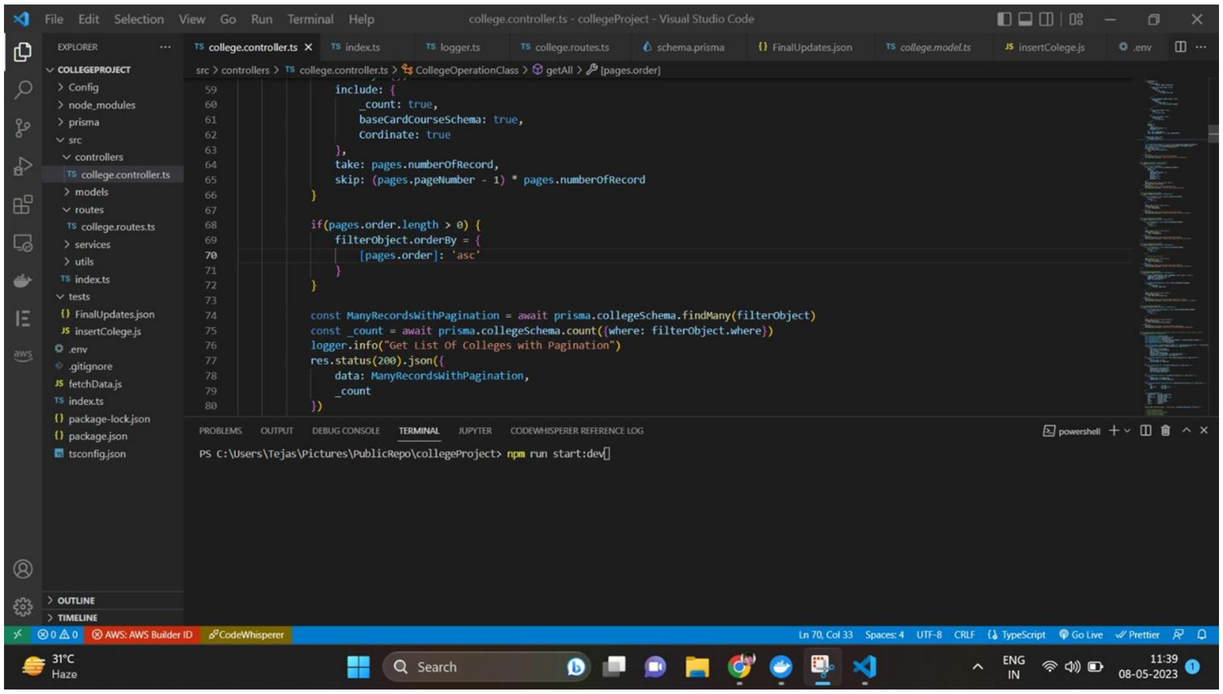
Containers

Name	Image	Status	Port(s)	Last started
goofy_swartz cf38124c4637	redis:alpine	Exited	6379:6379	28 days ago
soulofox-mysql 47b899564ce2	mysql:latest	Exited	3309:3306	12 hours ago

Showing 2 items

Fig 4 Using Docker to run a container

2. Open vs code and run (npm run start:dev) to start development build of backend this will automatically connect to database and show server is running message in terminal

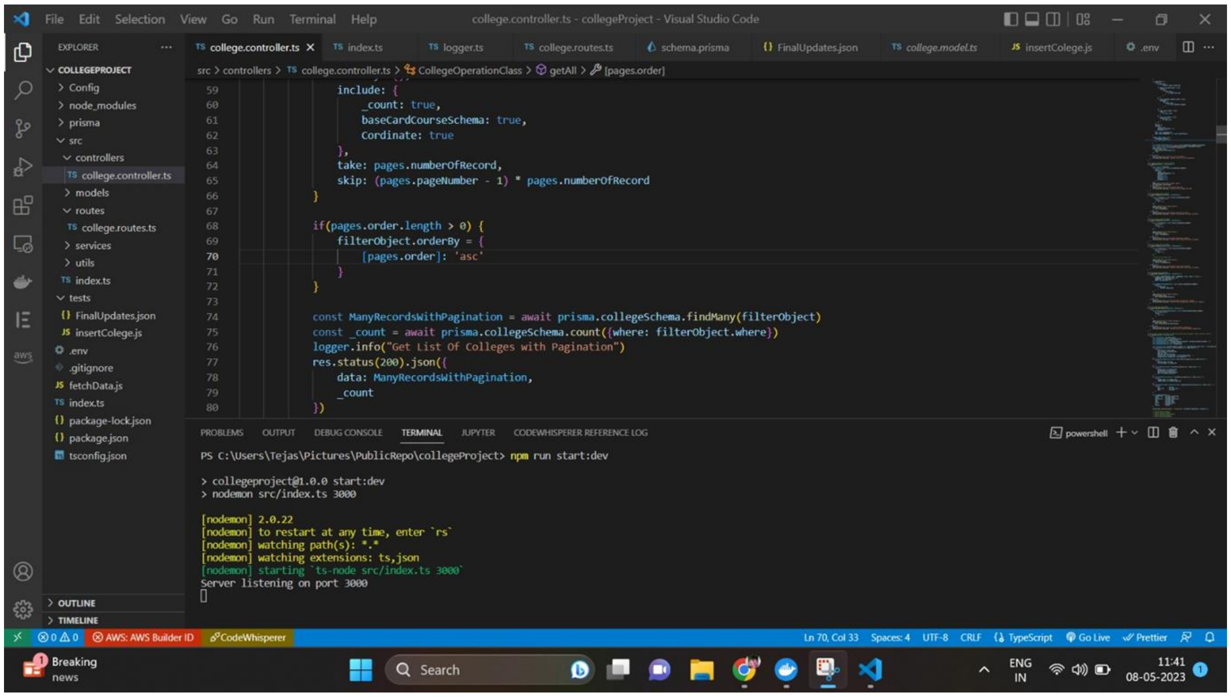


```
            include: {
                _count: true,
                baseCardCourseSchema: true,
                Cordinate: true
            },
            take: pages.numberOfRecord,
            skip: (pages.pageNumber - 1) * pages.numberOfRecord
        }

        if(pages.order.length > 0) {
            filterObject.orderBy = {
                [pages.order]: 'asc'
            }
        }

        const ManyRecordsWithPagination = await prisma.collegeSchema.findMany(filterObject)
        const _count = await prisma.collegeSchema.count({where: filterObject.where})
        logger.info("Get List Of Colleges with Pagination")
        res.status(200).json({
            data: ManyRecordsWithPagination,
            _count
        })
```

```
PS C:\Users\Tejas\Pictures\PublicRepo\collegeProject> npm run start:dev
```

Fig.5 Start development build of backend

3. This message will show in backend terminal

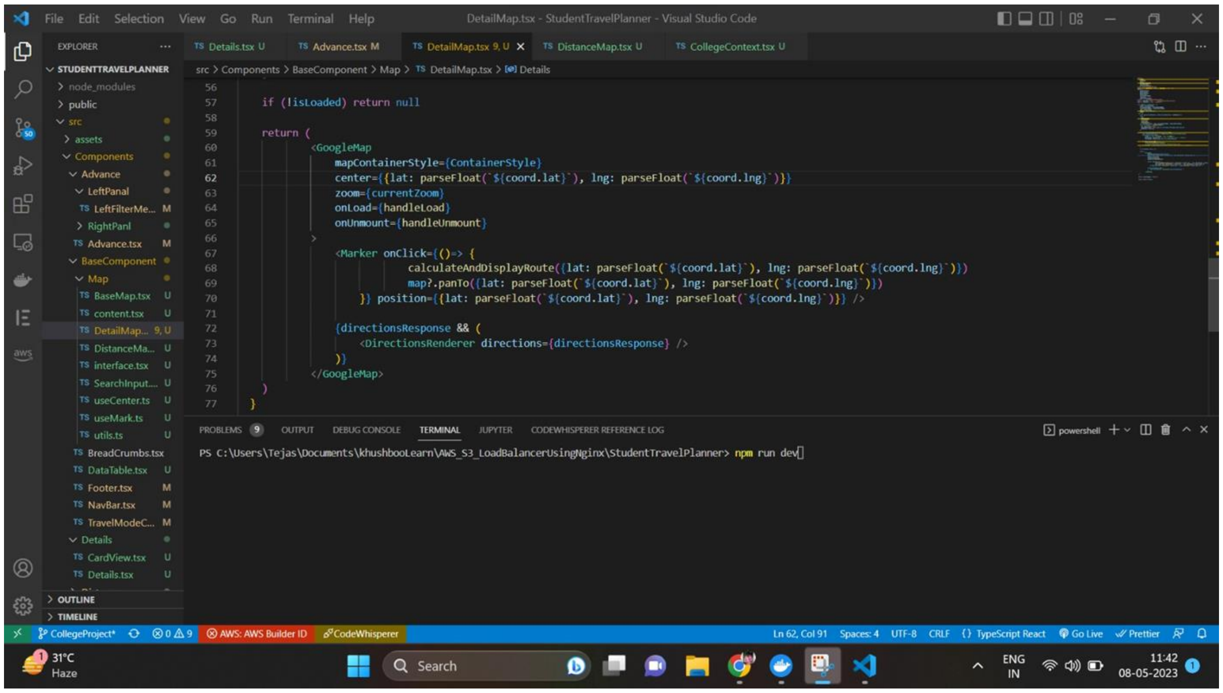


```
            include: {
                _count: true,
                baseCardCourseSchema: true,
                Cordinate: true
            },
            take: pages.numberOfRecord,
            skip: (pages.pageNumber - 1) * pages.numberOfRecord
        }

        if(pages.order.length > 0) {
            filterObject.orderBy = {
                [pages.order]: 'asc'
            }
        }

        const ManyRecordsWithPagination = await prisma.collegeSchema.findMany(filterObject)
        const _count = await prisma.collegeSchema.count({where: filterObject.where})
        logger.info("Get List Of Colleges with Pagination")
        res.status(200).json({
            data: ManyRecordsWithPagination,
            _count
        })
```

```
PS C:\Users\Tejas\Pictures\PublicRepo\collegeProject> npm run start:dev

> collegeproject@1.0.0 start:dev
> nodemon src/index.ts 3000

[nodemon] 2.0.22
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: ts,json
[nodemon] starting `ts-node src/index.ts 3000`
Server listening on port 3000
```

Fig. 6 Backend Terminal

4. Open another vscode and run (npm run dev) to start development build of frontend (development build mean it's for developer not ready yet for production)

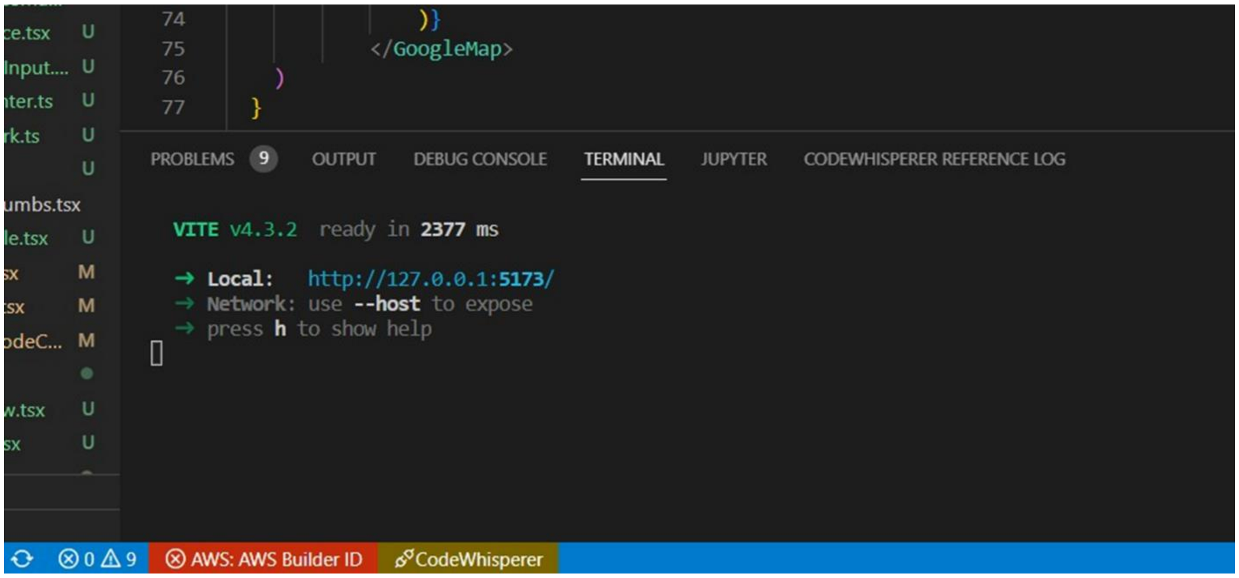


```
    if (!isLoaded) return null

    return (
            <GoogleMap
                mapContainerStyle={ContainerStyle}
                center={{lat: parseFloat(`${coord.lat}`), lng: parseFloat(`${coord.lng}`)}}
                zoom={currentZoom}
                onLoad={handleLoad}
                onUnmount={handleUnmount}
            >
                <Marker onClick={()=> {
                            calculateAndDisplayRoute({lat: parseFloat(`${coord.lat}`), lng: parseFloat(`${coord.lng}`)})
                            map?.panTo({lat: parseFloat(`${coord.lat}`), lng: parseFloat(`${coord.lng}`)})
                    }} position={{lat: parseFloat(`${coord.lat}`), lng: parseFloat(`${coord.lng}`)}} />

                {directionsResponse && (
                    <DirectionsRenderer directions={directionsResponse} />
                )}
            </GoogleMap>
    )
}
```

```
PS C:\Users\Tejas\Documents\khushbooLearn\AWS_S3_LoadBalancerUsingNginx\StudentTravelPlanner> npm run dev
```

Fig. 7 Start development build of Frontend

5. After that this Message will prompt server ip address (address where project is running) and some extra details like (VITE v4.3.2) which mean project bundle will manage by VITE in production



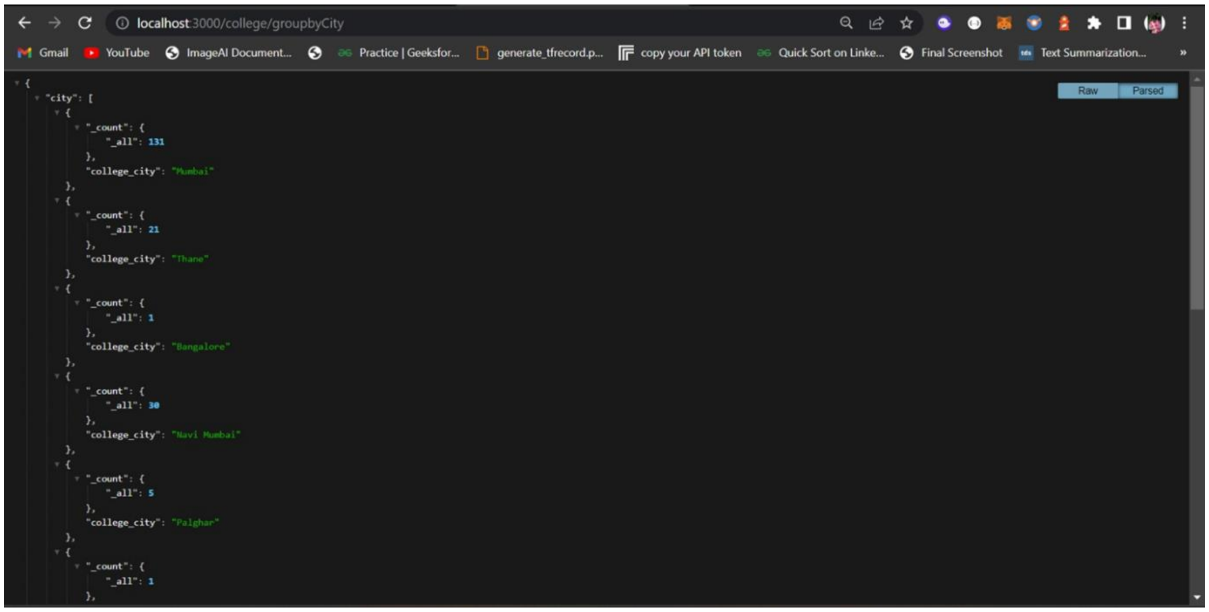
```
  VITE v4.3.2  ready in 2377 ms

  ➜  Local:   http://127.0.0.1:5173/
  ➜  Network: use --host to expose
  ➜  press h to show help
```

Fig. 8 Frontend Terminal

6. Open browser and hit both backend and frontend ip's and check if both are working perfectly

backend: http://localhost:3000

frontend: http://127.0.0.1:5173/



```
{
  "city": [
    {
      "_count": {
        "_all": 131
      },
      "college_city": "Mumbai"
    },
    {
      "_count": {
        "_all": 21
      },
      "college_city": "Thane"
    },
    {
      "_count": {
        "_all": 1
      },
      "college_city": "Bangalore"
    },
    {
      "_count": {
        "_all": 30
      },
      "college_city": "Navi Mumbai"
    },
    {
      "_count": {
        "_all": 5
      },
      "college_city": "Palghar"
    },
    {
      "_count": {
        "_all": 1
      },
```

Fig. 9 open Browser and run backend IP **http://localhost:3000**

7. If frontend running properly that you will see this result

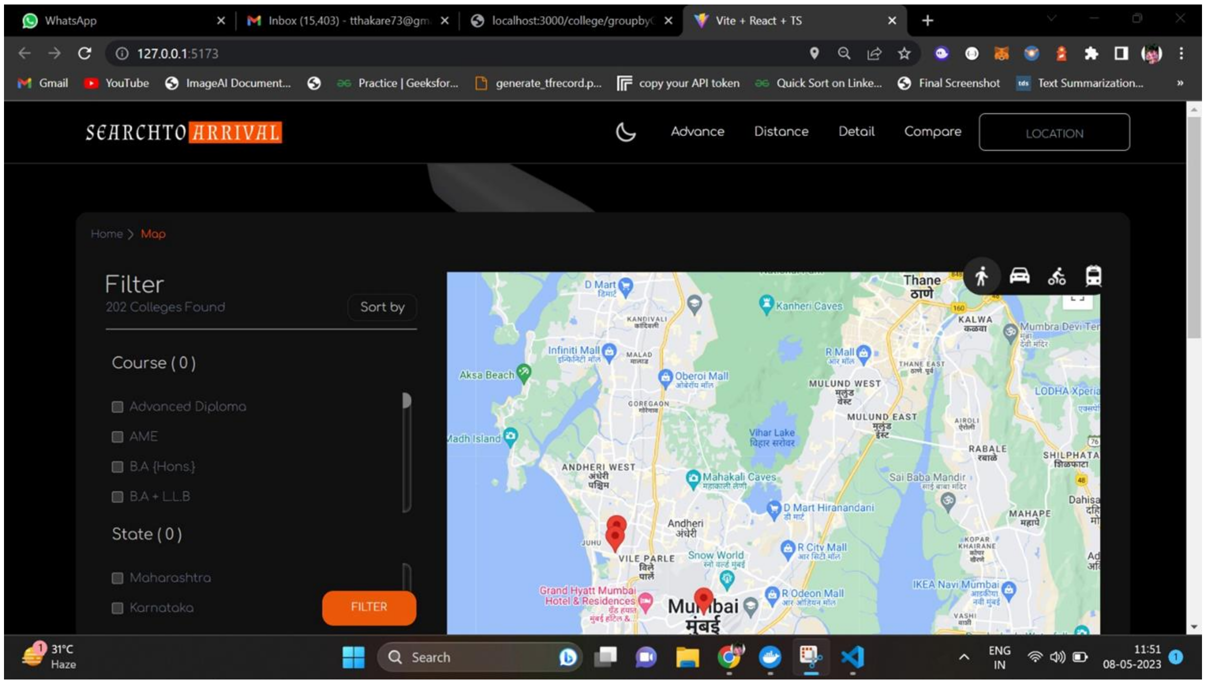
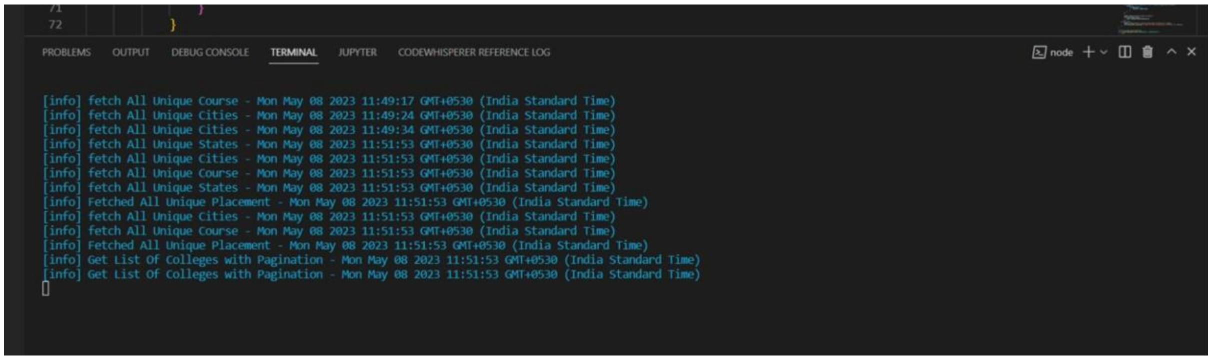


Fig. 10 open Browser and run Frontend IP http://127.0.0.1:5173/

8. Backend result after running perfectly

while in every api hit in from frontend backend terminal show which api is being hit and logging server will let you know the status of api, "[info]":- in above result mean api is successfully executed with status code 200 and there is no error

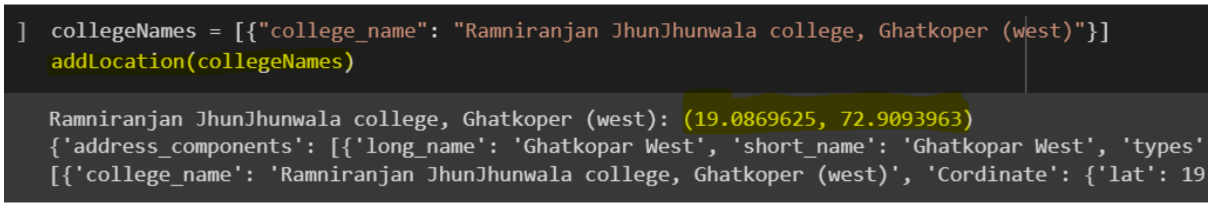


```
[info] fetch All Unique Course - Mon May 08 2023 11:49:17 GMT+0530 (India Standard Time)
[info] fetch All Unique Cities - Mon May 08 2023 11:49:24 GMT+0530 (India Standard Time)
[info] fetch All Unique Cities - Mon May 08 2023 11:49:34 GMT+0530 (India Standard Time)
[info] fetch All Unique States - Mon May 08 2023 11:51:53 GMT+0530 (India Standard Time)
[info] fetch All Unique Cities - Mon May 08 2023 11:51:53 GMT+0530 (India Standard Time)
[info] fetch All Unique Course - Mon May 08 2023 11:51:53 GMT+0530 (India Standard Time)
[info] fetch All Unique States - Mon May 08 2023 11:51:53 GMT+0530 (India Standard Time)
[info] Fetched All Unique Placement - Mon May 08 2023 11:51:53 GMT+0530 (India Standard Time)
[info] fetch All Unique Cities - Mon May 08 2023 11:51:53 GMT+0530 (India Standard Time)
[info] fetch All Unique Course - Mon May 08 2023 11:51:53 GMT+0530 (India Standard Time)
[info] Fetched All Unique Placement - Mon May 08 2023 11:51:53 GMT+0530 (India Standard Time)
[info] Get List Of Colleges with Pagination - Mon May 08 2023 11:51:53 GMT+0530 (India Standard Time)
[info] Get List Of Colleges with Pagination - Mon May 08 2023 11:51:53 GMT+0530 (India Standard Time)
▯
```

Fig. 11 Backend Results

4.2 Result's Screenshots

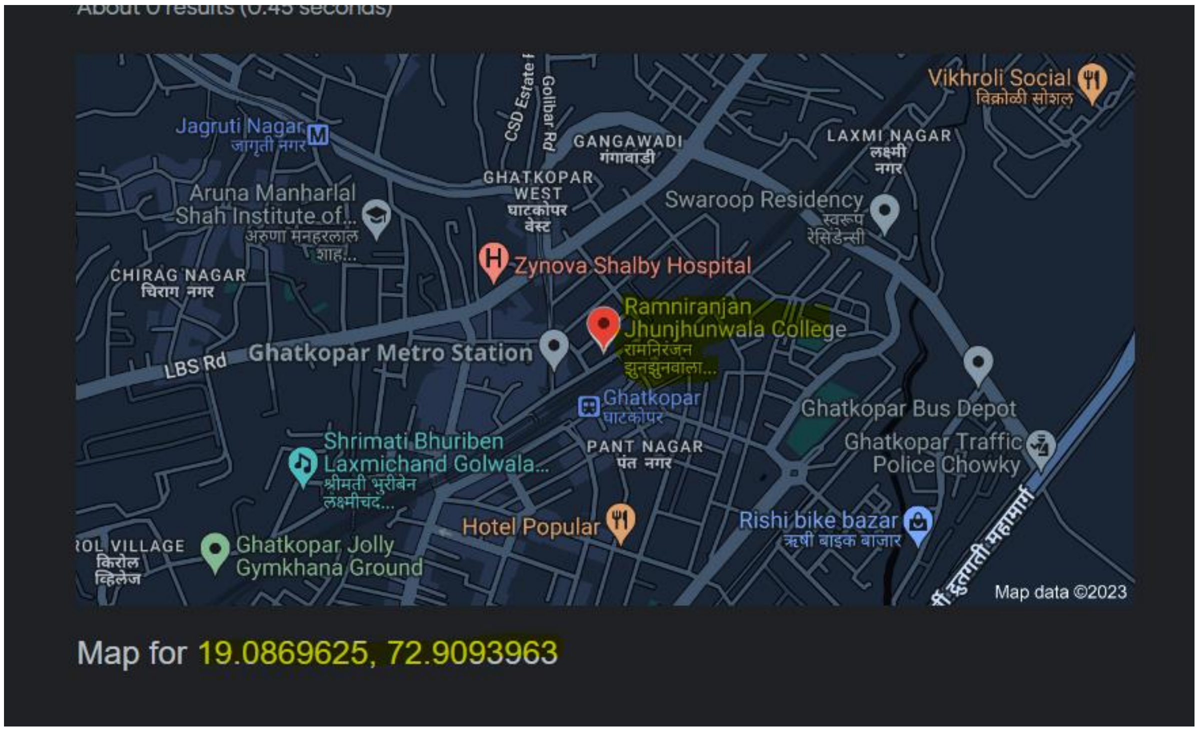
1. Data processing for extracting **GEO-CODE**



```
] collegeNames = [{"college_name": "Ramniranjan JhunJhunwala college, Ghatkoper (west)"}]
  addLocation(collegeNames)

  Ramniranjan JhunJhunwala college, Ghatkoper (west): (19.0869625, 72.9093963)
  {'address_components': [{'long_name': 'Ghatkopar West', 'short_name': 'Ghatkopar West', 'types'
  [{'college_name': 'Ramniranjan JhunJhunwala college, Ghatkoper (west)', 'Cordinate': {'lat': 19
```

Fig. 12 Extracting Geocode

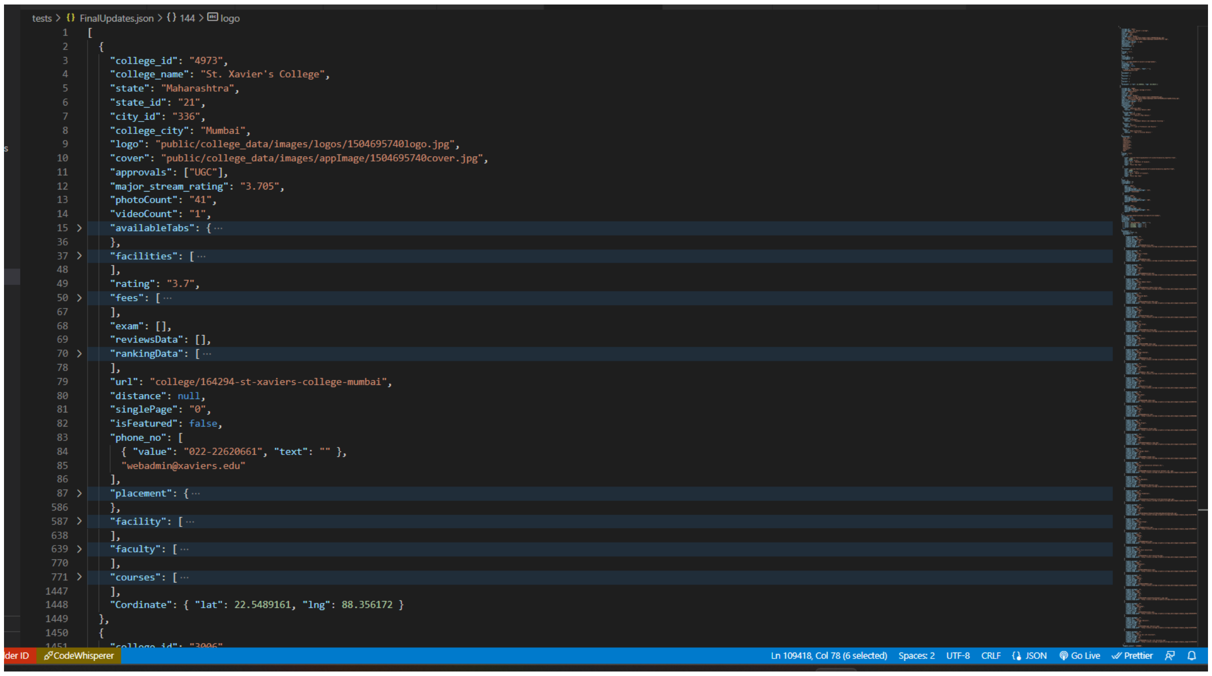
2. Verifying extracted **GEO-CODE**



Map for 19.0869625, 72.9093963

Fig. 13 Verification of Geocode

3. Data processing for extracting All Required data



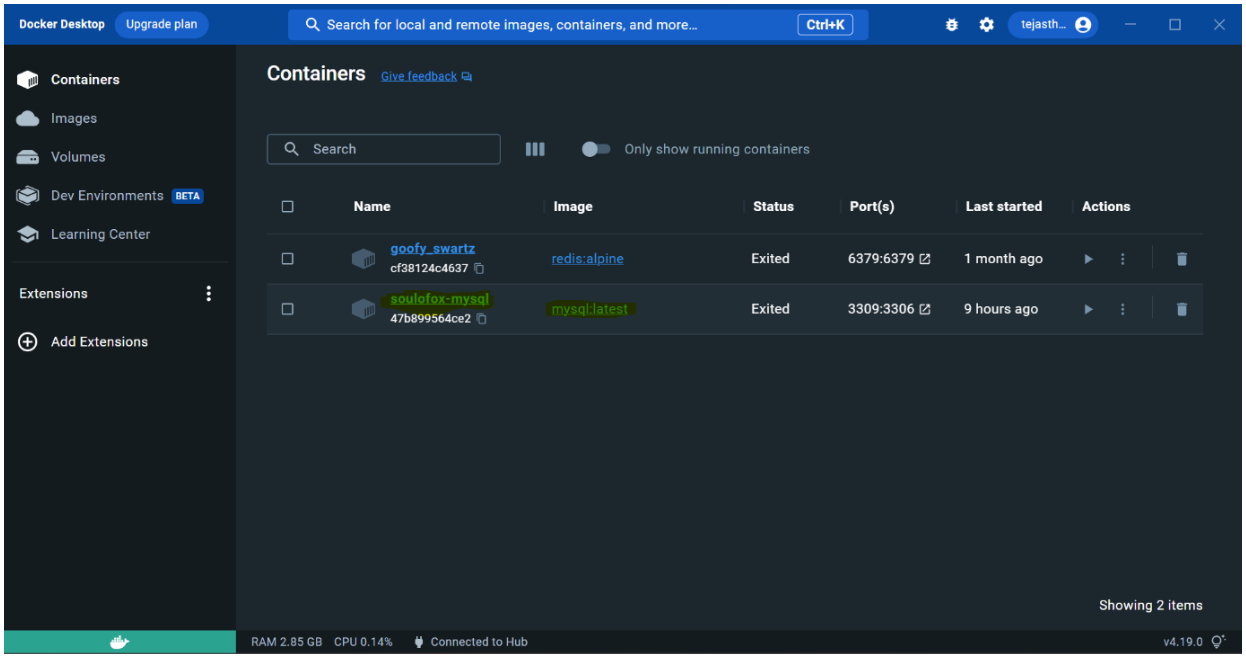
tests > {} FinalUpdates.json > {} 144 > logo

```
[
  {
    "college_id": "4973",
    "college_name": "St. Xavier's College",
    "state": "Maharashtra",
    "state_id": "21",
    "city_id": "336",
    "college_city": "Mumbai",
    "logo": "public/college_data/images/logos/1504695740logo.jpg",
    "cover": "public/college_data/images/appImage/1504695740cover.jpg",
    "approvals": ["UGC"],
    "major_stream_rating": "3.705",
    "photoCount": "41",
    "videoCount": "1",
    "availableTabs": {···
    },
    "facilities": [···
    ],
    "rating": "3.7",
    "fees": [···
    ],
    "exam": [],
    "reviewsData": [],
    "rankingData": [···
    ],
    "url": "college/164294-st-xaviers-college-mumbai",
    "distance": null,
    "singlePage": "0",
    "isFeatured": false,
    "phone_no": [
      { "value": "022-22620661", "text": "" },
      "webadmin@xaviers.edu"
    ],
    "placement": {···
    },
    "facility": [···
    ],
    "faculty": [···
    ],
    "courses": [···
    ],
    "Cordinate": { "lat": 22.5489161, "lng": 88.356172 }
  },
  {
```

Ln 109418, Col 78 (6 selected)   Spaces: 2   UTF-8   CRLF   {} JSON   Go Live   Prettier

Fig. 14 Snapshot of 202 colleges in final json

4. Create Sql database using Docker



Containers

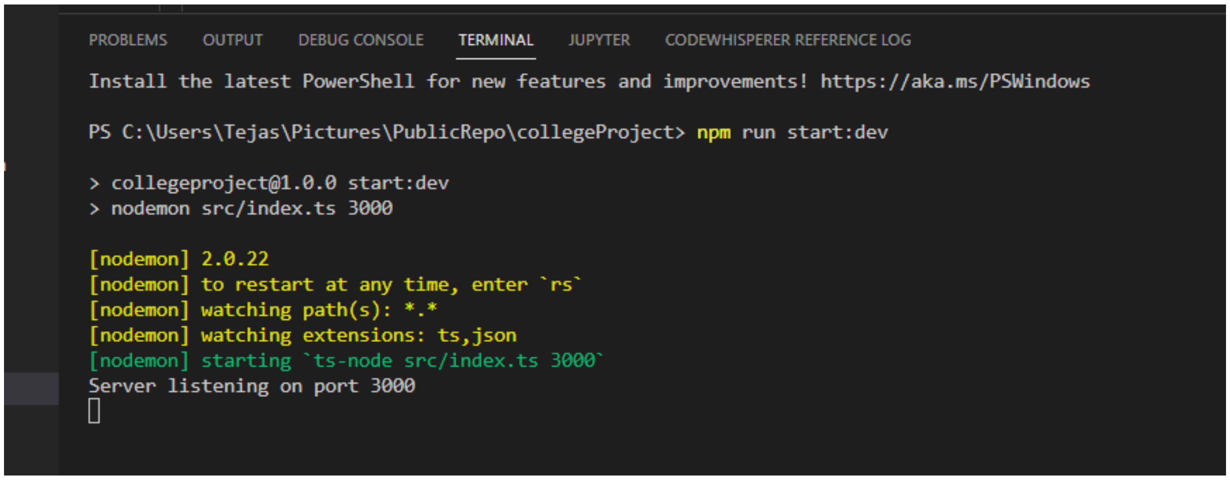
	Name	Image	Status	Port(s)	Last started	Actions
☐	goofy_swartz cf38124c4637	redis:alpine	Exited	6379:6379	1 month ago	
☐	soulofox-mysql 47b899564ce2	mysql:latest	Exited	3309:3306	9 hours ago	

Showing 2 items

RAM 2.85 GB   CPU 0.14%   Connected to Hub   v4.19.0

Fig. 15 Create Sql database using Docker on port 3309

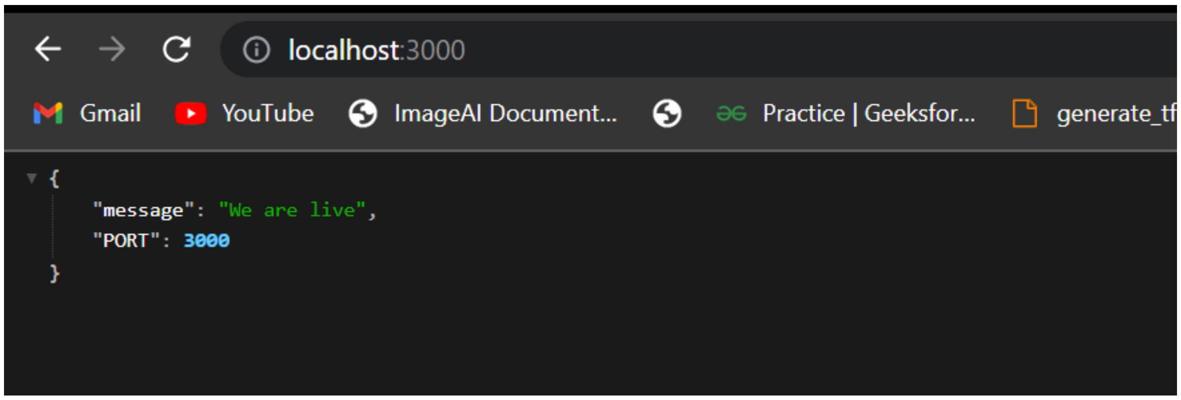
5. Connecting database and server



```
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Tejas\Pictures\PublicRepo\collegeProject> npm run start:dev

> collegeproject@1.0.0 start:dev
> nodemon src/index.ts 3000

[nodemon] 2.0.22
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: ts,json
[nodemon] starting `ts-node src/index.ts 3000`
Server listening on port 3000
```

Fig. 16 Start Script on port 3000

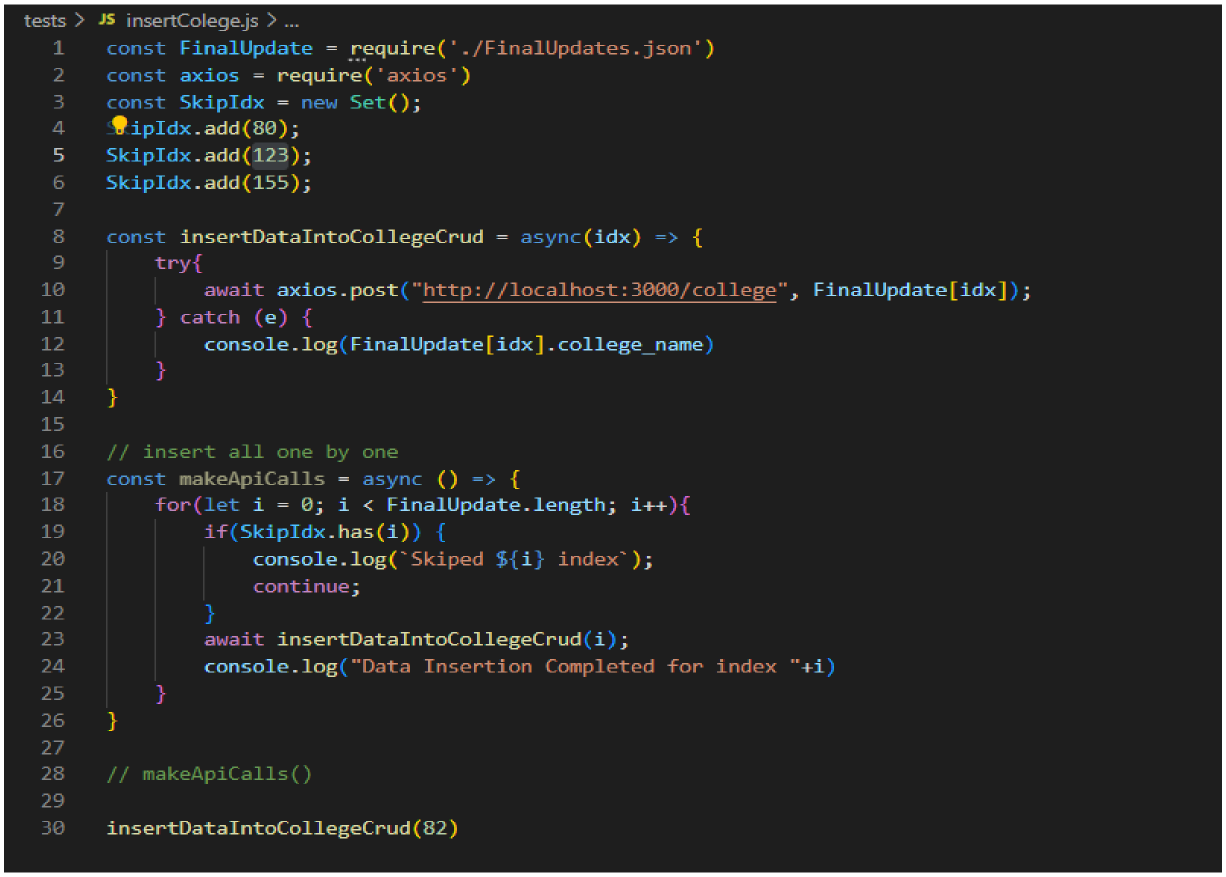
6. Checking Server is connected or not



```
{
    "message": "We are live",
    "PORT": 3000
}
```

Fig 17. Check if server is live

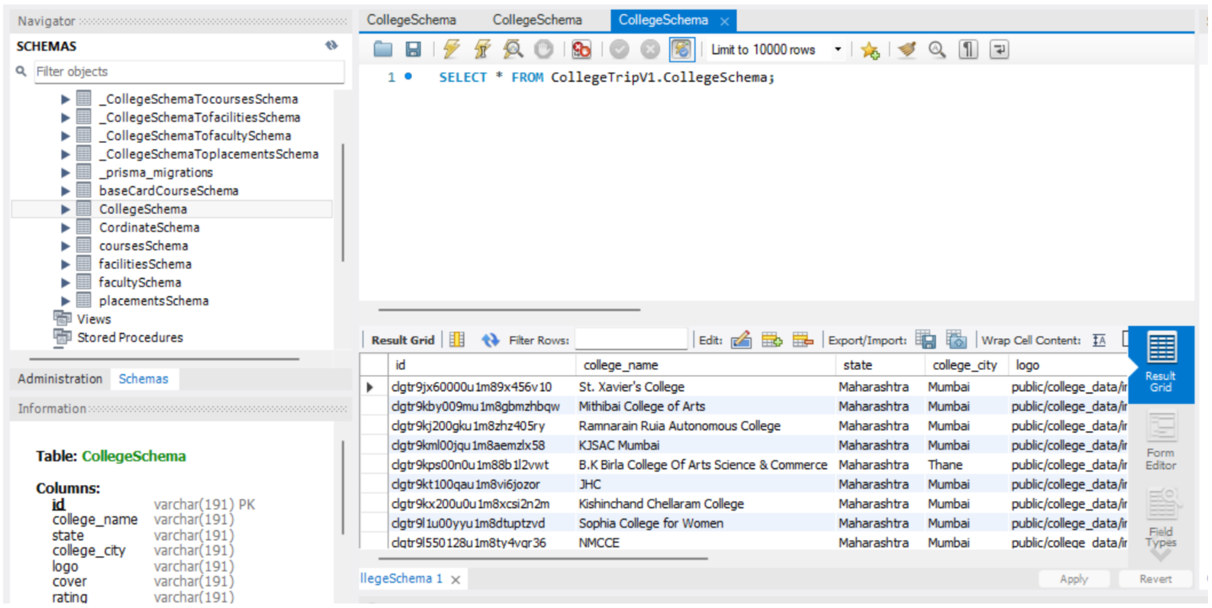
7. Writing Module for data Insertion


tests > JS insertColege.js > ...

```javascript
const FinalUpdate = require('./FinalUpdates.json')
const axios = require('axios')
const SkipIdx = new Set();
SkipIdx.add(80);
SkipIdx.add(123);
SkipIdx.add(155);

const insertDataIntoCollegeCrud = async(idx) => {
    try{
        await axios.post("http://localhost:3000/college", FinalUpdate[idx]);
    } catch (e) {
        console.log(FinalUpdate[idx].college_name)
    }
}

// insert all one by one
const makeApiCalls = async () => {
    for(let i = 0; i < FinalUpdate.length; i++){
        if(SkipIdx.has(i)) {
            console.log(`Skiped ${i} index`);
            continue;
        }
        await insertDataIntoCollegeCrud(i);
        console.log("Data Insertion Completed for index "+i)
    }
}

// makeApiCalls()

insertDataIntoCollegeCrud(82)
```

Fig.18 Write Module to Insert Data into Database

8. Checking if data is inserted or not



```
SELECT * FROM CollegeTripV1.CollegeSchema;
```

id	college_name	state	college_city	logo
clgtr9jx60000u1m89x456v10	St. Xavier's College	Maharashtra	Mumbai	public/college_data/ir
clgtr9kby009mu1m8gbmzhbqw	Mithibai College of Arts	Maharashtra	Mumbai	public/college_data/ir
clgtr9kj200gku1m8zhz405ry	Ramnarain Ruia Autonomous College	Maharashtra	Mumbai	public/college_data/ir
clgtr9kml00jqu1m8aemzlx58	KJSAC Mumbai	Maharashtra	Mumbai	public/college_data/ir
clgtr9kps00n0u1m88b1l2vwt	B.K Birla College Of Arts Science & Commerce	Maharashtra	Thane	public/college_data/ir
clgtr9kt100qau1m8vi6jozor	JHC	Maharashtra	Mumbai	public/college_data/ir
clgtr9kx200u0u1m8xcsi2n2m	Kishinchand Chellaram College	Maharashtra	Mumbai	public/college_data/ir
clgtr9l1u00yyu1m8dtuptzvd	Sophia College for Women	Maharashtra	Mumbai	public/college_data/ir
clgtr9l550128u1m8ty4vqr36	NMCCE	Maharashtra	Mumbai	public/college_data/ir

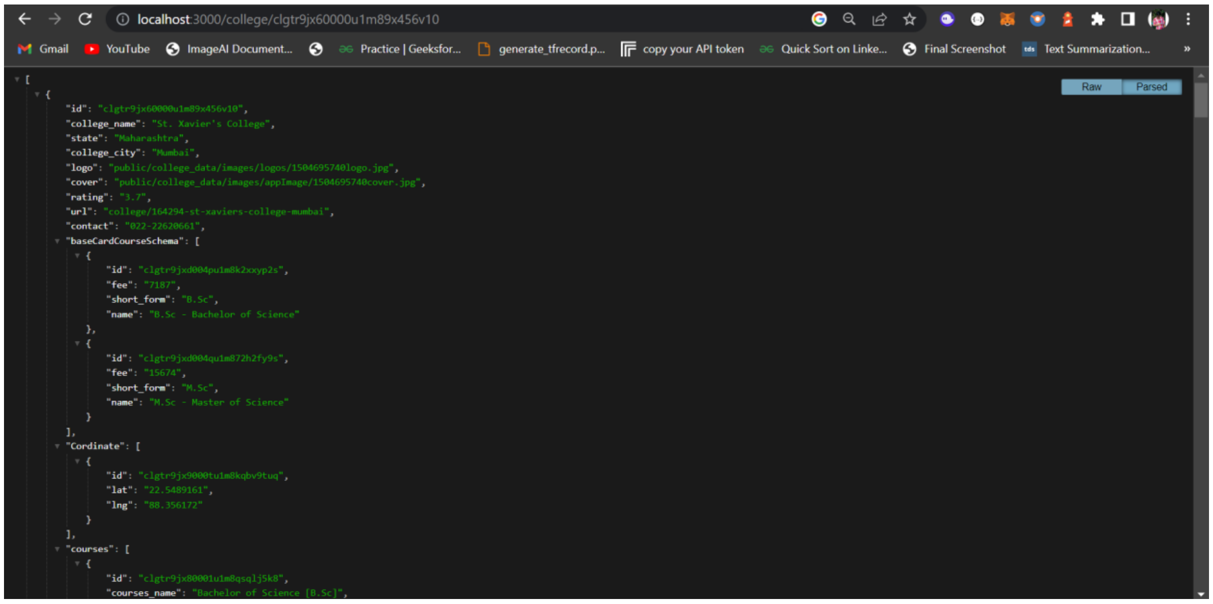
Table: CollegeSchema

Columns:

id	varchar(191) PK
college_name	varchar(191)
state	varchar(191)
college_city	varchar(191)
logo	varchar(191)
cover	varchar(191)
rating	varchar(191)

Fig.19 Check if Data is inserted using SQL workbench

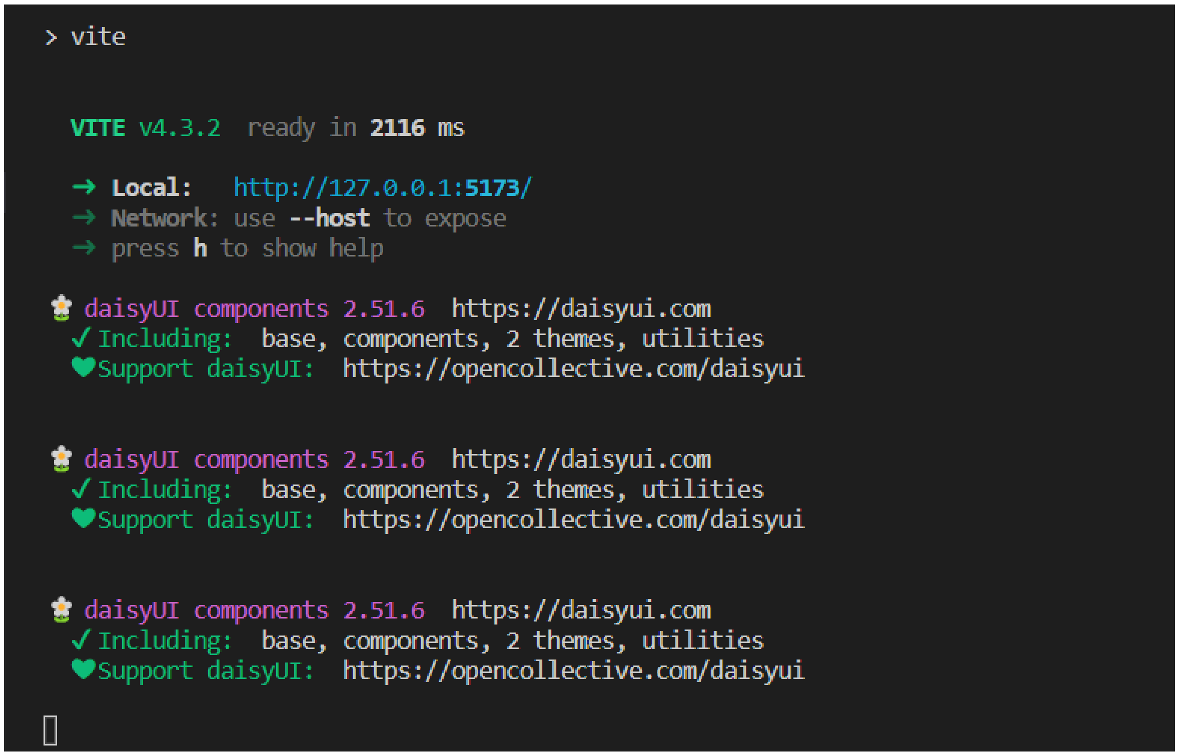
9. Checking if API's is Working



```
[
  {
    "id": "clgtr9jx60000u1m89x456v10",
    "college_name": "St. Xavier's College",
    "state": "Maharashtra",
    "college_city": "Mumbai",
    "logo": "public/college_data/images/logos/1504695740logo.jpg",
    "cover": "public/college_data/images/appImage/1504695740cover.jpg",
    "rating": "3.7",
    "url": "college/164294-st-xaviers-college-mumbai",
    "contact": "022-22620661",
    "baseCardCourseSchema": [
      {
        "id": "clgtr9jxd004pu1m8k2xxyp2s",
        "fee": "7187",
        "short_form": "B.Sc",
        "name": "B.Sc - Bachelor of Science"
      },
      {
        "id": "clgtr9jxd004qu1m872h2fy9s",
        "fee": "15674",
        "short_form": "M.Sc",
        "name": "M.Sc - Master of Science"
      }
    ],
    "Cordinate": [
      {
        "id": "clgtr9jx9000tu1m8kqbv9tuq",
        "lat": "22.5489161",
        "lng": "88.356172"
      }
    ],
    "courses": [
      {
        "id": "clgtr9jx80001u1m8qsqlj5k8",
        "courses_name": "Bachelor of Science [B.Sc]",
```

Fig. 20 Check if apis are also working perfectly

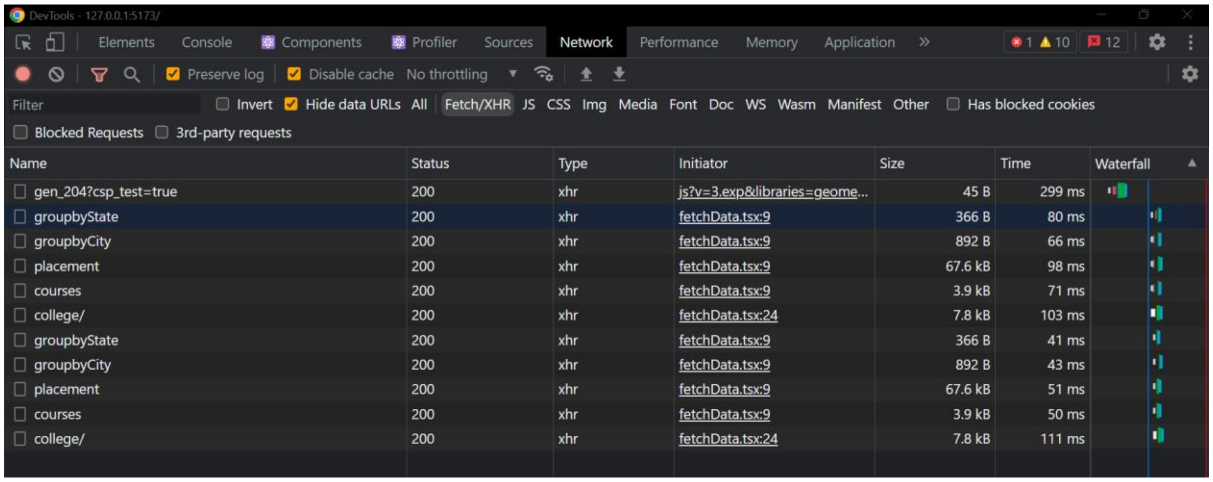
10. Start the Fronted File



```
> vite


  VITE v4.3.2  ready in 2116 ms

  ➜  Local:   http://127.0.0.1:5173/
  ➜  Network: use --host to expose
  ➜  press h to show help

🌼 daisyUI components 2.51.6  https://daisyui.com
  ✓ Including:  base, components, 2 themes, utilities
  💚Support daisyUI:  https://opencollective.com/daisyui


🌼 daisyUI components 2.51.6  https://daisyui.com
  ✓ Including:  base, components, 2 themes, utilities
  💚Support daisyUI:  https://opencollective.com/daisyui


🌼 daisyUI components 2.51.6  https://daisyui.com
  ✓ Including:  base, components, 2 themes, utilities
  💚Support daisyUI:  https://opencollective.com/daisyui

▯
```

Fig. 21 Frontend Start with npm run dev Command

11. Checking in Network, API is called or not

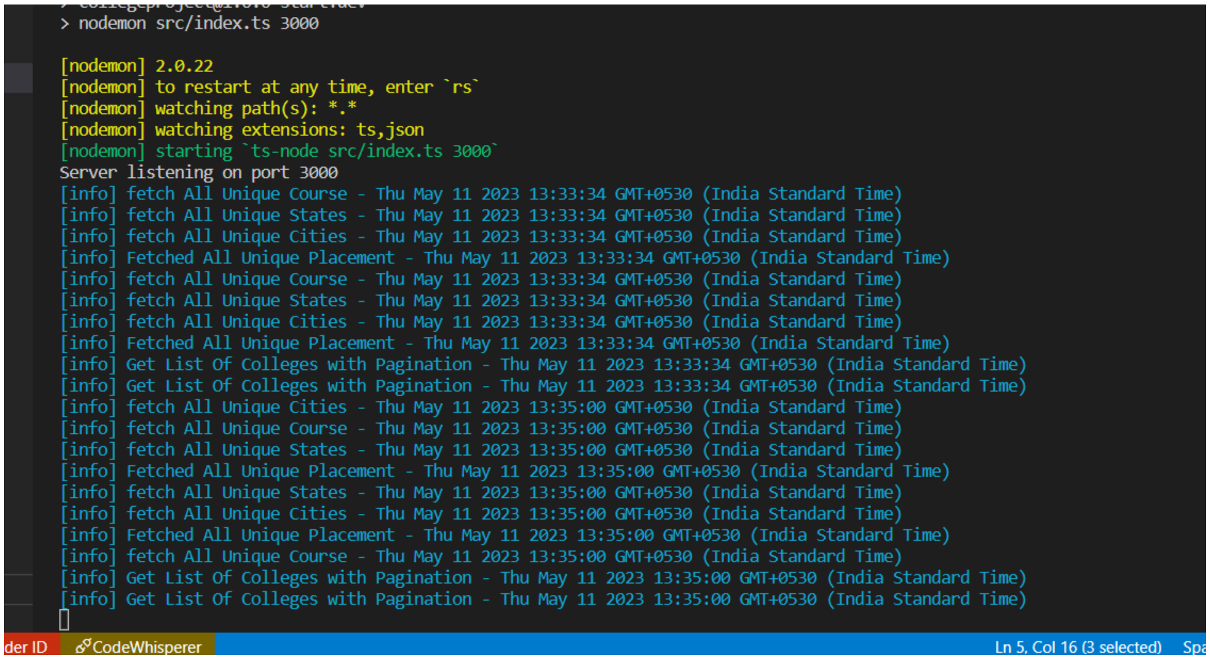


Preserve log ✓ Disable cache ✓ No throttling Hide data URLs ✓

Name	Status	Type	Initiator	Size	Time	Waterfall
gen_204?csp_test=true	200	xhr	js?v=3.exp&libraries=geome...	45 B	299 ms	
groupbyState	200	xhr	fetchData.tsx:9	366 B	80 ms	
groupbyCity	200	xhr	fetchData.tsx:9	892 B	66 ms	
placement	200	xhr	fetchData.tsx:9	67.6 kB	98 ms	
courses	200	xhr	fetchData.tsx:9	3.9 kB	71 ms	
college/	200	xhr	fetchData.tsx:24	7.8 kB	103 ms	
groupbyState	200	xhr	fetchData.tsx:9	366 B	41 ms	
groupbyCity	200	xhr	fetchData.tsx:9	892 B	43 ms	
placement	200	xhr	fetchData.tsx:9	67.6 kB	51 ms	
courses	200	xhr	fetchData.tsx:9	3.9 kB	50 ms	
college/	200	xhr	fetchData.tsx:24	7.8 kB	111 ms	

Fig. 22 Make Api Calls

12 Logger will log the message of each api calls in backend terminal



```
> nodemon src/index.ts 3000

[nodemon] 2.0.22
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: ts,json
[nodemon] starting `ts-node src/index.ts 3000`
Server listening on port 3000
[info] fetch All Unique Course - Thu May 11 2023 13:33:34 GMT+0530 (India Standard Time)
[info] fetch All Unique States - Thu May 11 2023 13:33:34 GMT+0530 (India Standard Time)
[info] fetch All Unique Cities - Thu May 11 2023 13:33:34 GMT+0530 (India Standard Time)
[info] Fetched All Unique Placement - Thu May 11 2023 13:33:34 GMT+0530 (India Standard Time)
[info] fetch All Unique Course - Thu May 11 2023 13:33:34 GMT+0530 (India Standard Time)
[info] fetch All Unique States - Thu May 11 2023 13:33:34 GMT+0530 (India Standard Time)
[info] fetch All Unique Cities - Thu May 11 2023 13:33:34 GMT+0530 (India Standard Time)
[info] Fetched All Unique Placement - Thu May 11 2023 13:33:34 GMT+0530 (India Standard Time)
[info] Get List Of Colleges with Pagination - Thu May 11 2023 13:33:34 GMT+0530 (India Standard Time)
[info] Get List Of Colleges with Pagination - Thu May 11 2023 13:33:34 GMT+0530 (India Standard Time)
[info] fetch All Unique Cities - Thu May 11 2023 13:35:00 GMT+0530 (India Standard Time)
[info] fetch All Unique Course - Thu May 11 2023 13:35:00 GMT+0530 (India Standard Time)
[info] fetch All Unique States - Thu May 11 2023 13:35:00 GMT+0530 (India Standard Time)
[info] Fetched All Unique Placement - Thu May 11 2023 13:35:00 GMT+0530 (India Standard Time)
[info] fetch All Unique States - Thu May 11 2023 13:35:00 GMT+0530 (India Standard Time)
[info] fetch All Unique Cities - Thu May 11 2023 13:35:00 GMT+0530 (India Standard Time)
[info] Fetched All Unique Placement - Thu May 11 2023 13:35:00 GMT+0530 (India Standard Time)
[info] fetch All Unique Course - Thu May 11 2023 13:35:00 GMT+0530 (India Standard Time)
[info] Get List Of Colleges with Pagination - Thu May 11 2023 13:35:00 GMT+0530 (India Standard Time)
[info] Get List Of Colleges with Pagination - Thu May 11 2023 13:35:00 GMT+0530 (India Standard Time)
```

Fig. 23 Logger will log the message of each api calls in backend terminal

13 Fill and hydrate the dom with data and show result to user

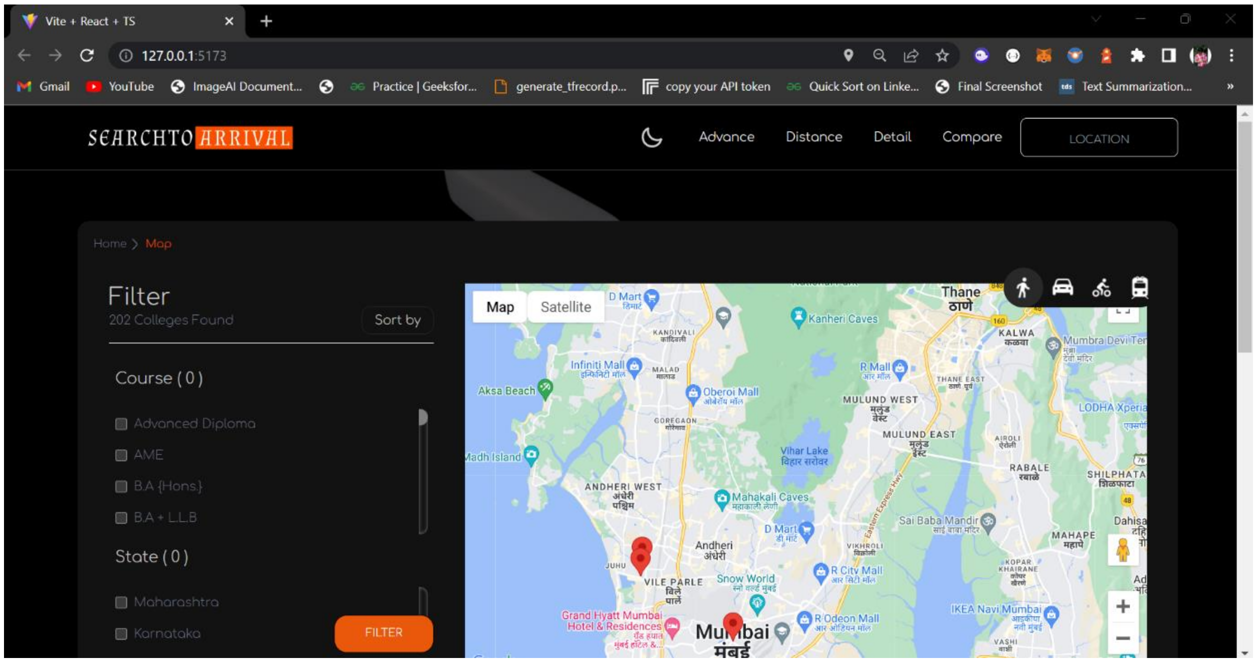
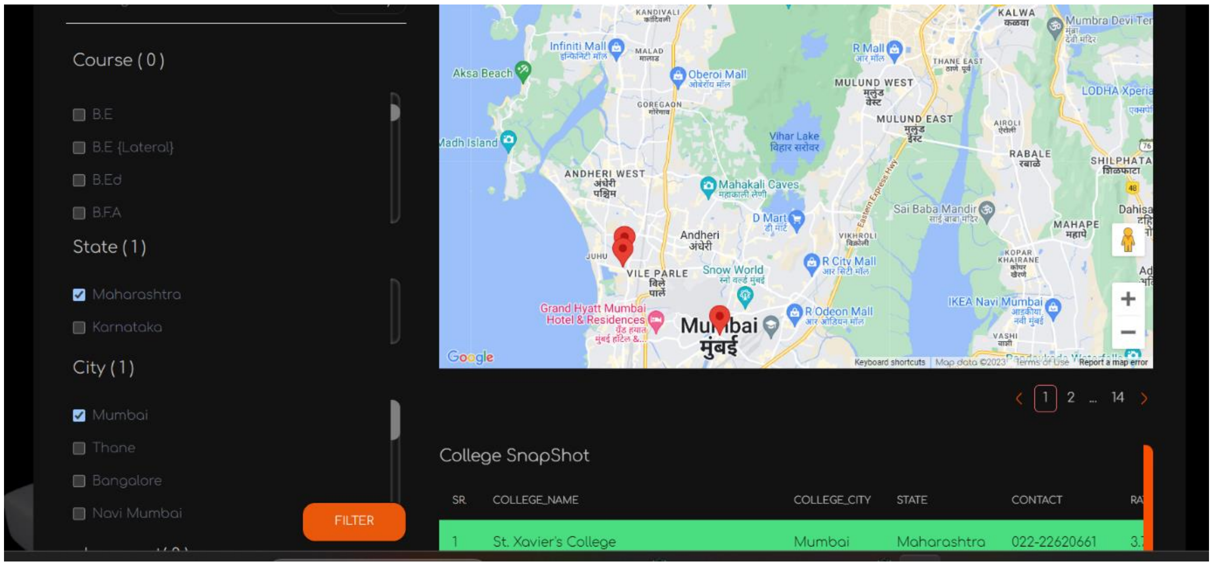


Fig. 24 Fill and hydrate the dom with data and show result to user

14 Filter Data And Check if everything working properly



College SnapShot

SR.	COLLEGE_NAME	COLLEGE_CITY	STATE	CONTACT
1	St. Xavier's College	Mumbai	Maharashtra	022-22620661

Fig. 25 Filter Data And Check if everything working properly

15 Distance Feature

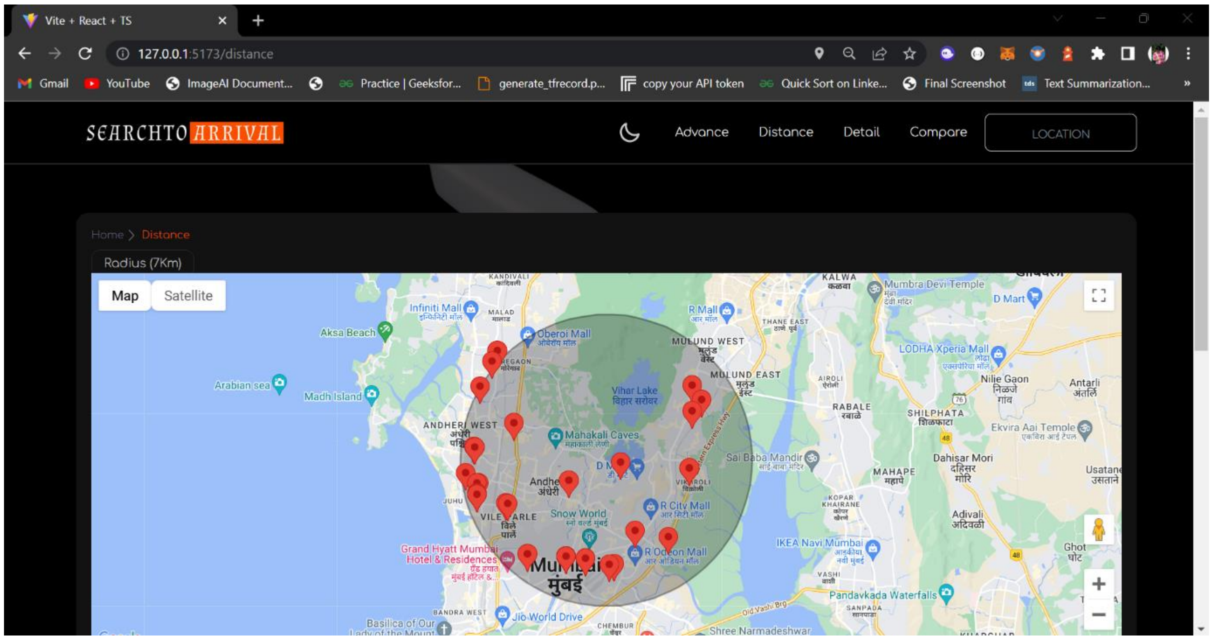


Fig. 26 Distance Feature

16 Routing

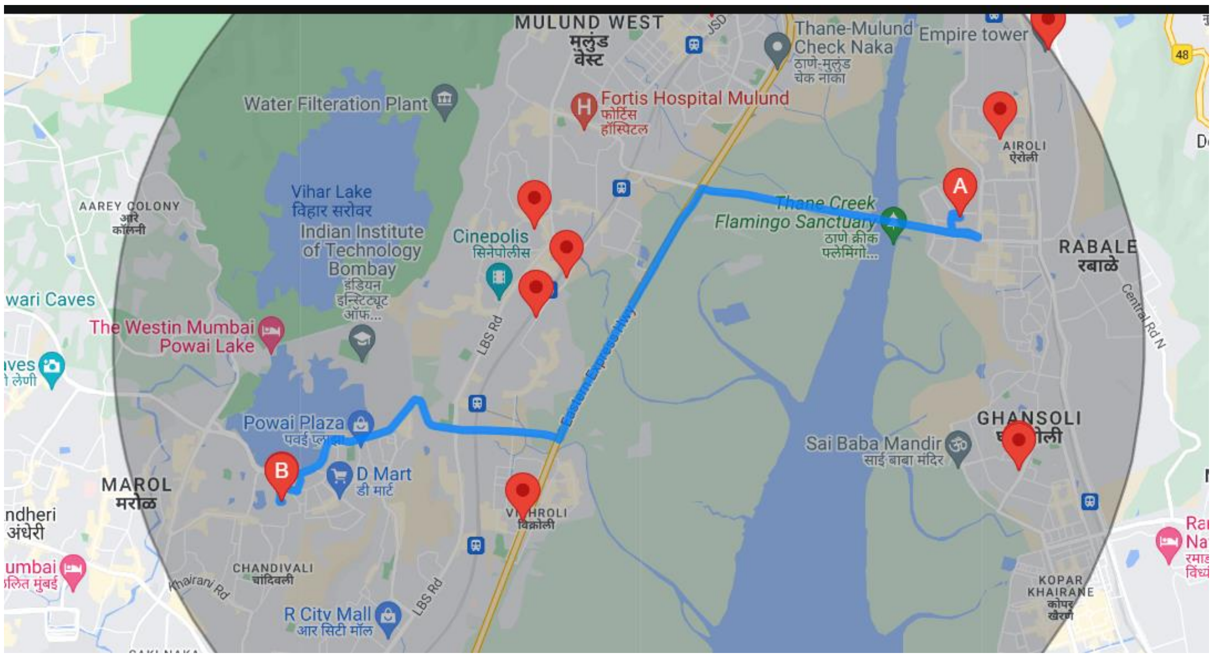
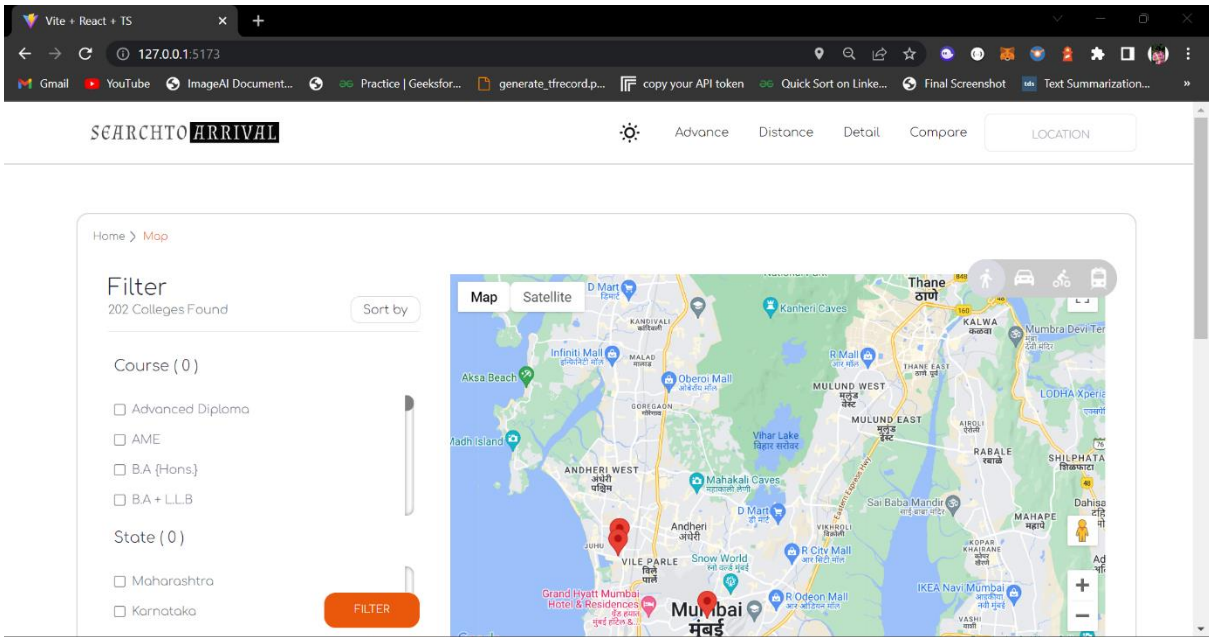


Fig. 27 Routing

17 Mode Light And Dark



Filter
202 Colleges Found
Sort by

Course (0)

- Advanced Diploma
- AME
- B.A {Hons}
- B.A + L.L.B

State (0)

- Maharashtra
- Karnataka

Fig. 28 Mode light and dark

Chapter 5: Discussion

1. **User-Friendly Interface**: Our website offers a user-friendly interface that allows students to easily navigate and explore college options. The advanced search, locality search, and comparison features are intuitive and help users find relevant information efficiently.

2. **Real-Time Results:** The advanced search feature of our website displays results in real-time on a map, providing students with instant insights into colleges that meet their criteria. This dynamic feature saves time and enhances the user experience.

3. **Customization and Filtering:** Our website offers multiple filters and sorting methods, enabling students to customize their search based on specific preferences such as courses, facilities, faculty, and fees. The ability to apply multiple filters enhances the accuracy of search results.

4. **Locality-based Search:** The locality search feature allows students to specify a desired scan area, making it convenient to explore colleges in a particular location of interest. This targeted approach helps students find colleges within their preferred geographic range.

5. **Comparative Analysis:** The comparison feature of our website enables students to compare colleges side by side. The ability to evaluate colleges based on parameters like distance, placement companies, courses, and fees assists students in making informed decisions and saves them time in the research process.

6. **Integration of Google Maps:** We leverage Google Maps services, providing a familiar and reliable platform for users. It offers interactive maps for visualizing college locations, which enhances the overall user experience and aids in decision-making.

7. **Efficiency and Time-saving:** Our website streamlines the college search process by providing comprehensive information and a range of search options. This saves students time by presenting relevant results quickly and enabling them to compare and evaluate colleges efficiently.

Chapter 6: Conclusion & Future Work

6.1 Conclusion

In conclusion, the college travel website created using React JS with four different components - advance, distance, detail, and compare - provides a comprehensive solution to the common challenges faced by students while traveling to college. The website offers a user-friendly interface that makes it easy for students to search for college names and locations using the search box feature. The advanced feature allows students to customize their search results based on specific requirements, and the sorting feature makes it easier to navigate the search results.

The website generates a map that displays the most efficient travel routes using different modes of transportation such as walking, car, train, and bus. This helps students to plan their travel in advance and make informed decisions about the best transportation mode to use.

The use of advanced technologies and tools, including React JS, HTML, CSS, and JavaScript, ensures the efficiency and scalability of the web application. The integration of APIs for mapping and transportation information enhances the user experience and provides accurate and real-time information.

Overall, the web application provides a simplified college selection process, enabling students to make informed decisions about their education. The user-friendly interface, combined with the advanced features and use of cutting-edge technologies, makes the website a valuable resource for students searching for colleges.

6.2 Future enhancement

The potential enhancements listed for the college travel website provide a glimpse into the possibilities for improving the user experience and functionality of the site. Let's take a closer look at each of these possible improvements:

6.2.1. Integration of real-time data:

By incorporating real-time data into the website, users could receive more accurate and up-to-date information about transportation schedules, traffic updates, and other relevant information. This could enhance the reliability of the travel recommendations and help users plan their trips more efficiently.

6.2.2. Personalization features:

Adding more customization features to the website could improve the user experience and allow users to further personalize their travel plans. For instance, users could choose preferred modes of transportation or specific landmarks they want to visit along the way.

6.2.3. Mobile optimization:

With the increasing use of smartphones and tablets, optimizing the website for mobile devices is becoming increasingly important. This could help users access the site and plan their travels on-the-go, from wherever they are.

6.2.4. Improved user interface and design:

Improving the user interface and design of the website could enhance the user experience and make the site more visually appealing and user-friendly. This could include updating the color scheme, layout, and typography to create a more modern and professional look.

6.2.5. Expanded coverage:

Expanding the coverage of the website to include additional colleges and universities, as well as additional locations, could make the site more useful to a broader range of users. This could also help to attract more traffic to the site, which could potentially generate more revenue through advertising or other means.

In conclusion, these potential enhancements highlight the many possibilities for improving the college travel website. By incorporating real-time data, adding more customization features, optimizing for mobile devices, improving the user interface and design, and expanding the coverage of the site, the web application could become even more valuable to users and more successful in achieving its goals.

Chapter 7: References

1. https://github.com/narukashu/Travel_Jaipur
2. https://github.com/DhrumilShah98/FifthProject-TourGuideApp
3. https://github.com/coding-catie/Tour-Guide-of-Tomodachi-Life
4. https://github.com/RB-93/TourGuideApp
5. https://ieeexplore.ieee.org/document/9491136
6. https://ieeexplore.ieee.org/document/9103380
7. collegeboard.org - https://www.collegeboard.org/
8. naviance.com - https://www.naviance.com/
9. cappex.com - https://www.cappex.com/
10. niche.com - https://www.niche.com/
11. chegg.com - https://www.chegg.com/
12. unigo.com - https://www.unigo.com/
13. petersons.com - https://www.petersons.com/
14. collegeconfidential.com - https://www.collegeconfidential.com/
15. bigfuture.collegeboard.org - https://bigfuture.collegeboard.org/
16. nces.ed.gov/collegenavigator - https://nces.ed.gov/collegenavigator/
17. scoir.com - https://scoir.com/
18. raise.me - https://www.raise.me/
19. noodle.com - https://www.noodle.com/
20. youvisit.com - https://www.youvisit.com/
21. collegeraptor.com - https://www.collegeraptor.com/
22. schoold.co - https://schoold.co/
23. methodtestprep.com - https://www.methodtestprep.com/
24. bridge-u.com - https://bridge-u.com/
25. mymajors.com - https://www.mymajors.com/

26.maps.google.com - https://www.google.com/maps/

27.tripadvisor.com - https://www.tripadvisor.com/

28.airbnb.com - https://www.airbnb.com/

29.booking.com - https://www.booking.com/

30.expedia.com - https://www.expedia.com/

31.kayak.com - https://www.kayak.com/

32.skyscanner.com - https://www.skyscanner.com/

33.hopper.com - https://www.hopper.com/

34.tripit.com - https://www.tripit.com/

35.rome2rio.com - https://www.rome2rio.com/

36.momondo.com - https://www.momondo.com/

37.travelocity.com - https://www.travelocity.com/

38.hostelworld.com - https://www.hostelworld.com/

39.roadtrippers.com - https://www.roadtrippers.com/

40.hoteltonight.com - https://www.hoteltonight.com/

41.flights.google.com - https://www.google.com/flights/

42.trip.com - https://www.trip.com/

43.priceline.com - https://www.priceline.com/

44.waze.com - https://www.waze.com/

45.triphobo.com - https://www.triphobo.com/

46.travel.sygic.com - https://travel.sygic.com/

47.loungebuddy.com - https://www.loungebuddy.com/

48.timeshifter.com - https://www.timeshifter.com/

49.translate.google.com - https://translate.google.com/

50.grab.com - https://www.grab.com/

51.theculturetrip.com - https://theculturetrip.com/

52.virtuo.io - https://www.virtuo.io/

53.appintheair.mobi - https://www.appintheair.mobi/

54.expensify.com - https://www.expensify.com/

55.tripscout.co - https://www.tripscout.co/

56.utrip.com - https://utrip.com/

57.tripactions.com - https://tripactions.com/

58.omio.com - https://www.omio.com/

59.hitlistapp.com - https://hitlistapp.com/

60.collegesimply.com - https://www.collegesimply.com/

61.collegemapper.com - https://www.collegemapper.com/

62.campusreel.org - https://www.campusreel.org/

63.https://www.emerald.com/insight/content/doi/10.1108/IJEBR-01-2021-0070/full/html#sec005

Chapter 8: Program Code

8.1 Schema

CollegeSchema		
Column	***Type***	***Attributes***
id	String	@id @default(cuid())
college_name	String	
state	String	
college_city	String	
logo	String	
cover	String	
rating	String	
url	String	
contact	String	
Cordinate	CordinateSchema[]	
placements	placementsSchema[]	
Facilities	facilitiesSchema[]	
Faculty	facultySchema[]	
courses	coursesSchema[]	
baseCardCourse	baseCardCourseSchema[]	

Table: CordinateSchema		
Column	***Type***	***Attributes***
id	String	@id @default(cuid( ))
lat	String	
lng	String	
college	CollegeSchema[]	
Table: placementsSchema		
Column	***Type***	***Attributes***
id	String	@id @default(cuid( ))
company_name	String	@unique
company_image	String	
college	CollegeSchema[]	
Table: facilitiesSchema		
Column	***Type***	***Attributes***

id	String	@id @default(cuid( ))
facility_name	String	@unique
facility_image	String	
college	CollegeSchema[]	
Table: facultySchema		
Column	***Type***	***Attributes***
id	*String*	*@id @default(cuid( ))*
faculty_name	*String*	
designation	*String*	
exp	*String*	
qualification	*String*	
college	*CollegeSchema[]*	
Table: coursesSchema		
Column	***Type***	***Attributes***

id	String	@id @default(cuid( ))
courses_name	String	
short_head	String	@unique
type	String	
duration	String	
fees_per_year	String	
eligibility	String	
college	CollegeSchema[]	
Table: baseCardCourseSchema		
Column	***Type***	***Attributes***
id	String	@id @default(cuid( ))
fee	String	
short_form	String	
name	String	
college	CollegeSchema[]	

8.2 Data processing

1. Imports

```
# all required Imports
import time
import json
import requests
```

2. Variables

```
# array where we are storing final result
result = []
```

3. Start Point of application

```
def StartPoint():
  clgDetail = "https://zollege.in/web-api/"
  # replace with your own API key # Replace YOUR_API_KEY with your actual
API key from OpenCage Geocoder # geocoder =
opencage.geocoder.OpenCageGeocode("36a32331c66f48b6b2240b71adf1d544
")
  # List of college names and addresses
  with open('/content/sample_data/affCol.json') as f:
     data = json.load(f)
  colleges = data
  print(len(colleges))
  result = []
  Fetch_Colleges_and_filter(colleges, clgDetail)
  updateResult("University", "University of Mumbai")
  addLocation(result)
  writeOutputOnfile(result)
```

4. Scrape Data module

```
# fetch data from zollege website to get row data and than pre process that to use
in my application
def Fetch_Colleges_and_filter(modCollegeData, baseUrl):
  for index, college in enumerate(modCollegeData):
    clgDetailUrl = college["url"]
    college_name = college["college_name"]
    time.sleep(1.5)
    response = requests.get(f"{baseUrl}{clgDetailUrl}")
    data = response.json()
    phone_no = data["basic_info"]["phone_no"]
    placement = data["college_facilities"]["placements"] #["placement"]
    facility = data["college_facilities"]["facility"]
    faculty = data["faculty"]["faculty_list"] if "faculty_list" in data["faculty"] else
data["faculty"]
    courses = data["course_data"]["courses"] if "courses" in data["course_data"]
else data["course_data"]
    college["phone_no"] = phone_no
    college["placement"] = placement
    college["facility"] = facility
    college["faculty"] = faculty
    college["courses"] = courses
    print(205 - index)
    print(college_name)
    result.append(college)
```

5. Add GeoCode module

```
# Add location of every college in the list
def addLocation(colleges):
  api_key = "AIzaSyCtAtFTYTpduPIXVcR4eMAnkSI_6hTnc7M"
  for college in colleges:
    college_name = college["college_name"]
    time.sleep(1.5)
    response                                                        =
requests.get(f"https://maps.googleapis.com/maps/api/geocode/json?address={c
ollege_name}&key={api_key}")
    data = response.json()

    if data["status"] == "OK":
        # Extract the latitude and longitude from the first result
        lat = data["results"][0]["geometry"]["location"]["lat"]
        lng = data["results"][0]["geometry"]["location"]["lng"]
        college['Cordinate'] = {'lat': lat, 'lng': lng}
        result.append(college)
        print(f"{college_name}: ({lat}, {lng})")
        print(data["results"][0])
    else:
        college['Cordinate'] = {'lat': 0, 'lng': 0}
        result.append(college)
        print(f"{college_name}: Unable to obtain location.")

  print(result)
```

6. Filter and update module

```
# update result with specific key and value
def updateResult(key, value, data):
  for college in data:
    college[key] = value
```

7. Output of performed action in json file

```
# create output in output.js file
def writeOutputOnfile(outputData):
  # Write the JSON string to a file
  json_string = json.dumps(outputData)
  with open('/content/sample_data/output.json', 'w') as file:
      file.write(json_string)
```

8.3 Backend

8.3.1 Start Point of code

This code sets up an Express server that listens on a specified port. It creates an instance of the `**TriptoServer**` class which has methods for configuring middleware and routes. Middleware is used to parse incoming requests and enable Cross-Origin Resource Sharing (CORS). The server responds to the root path with a message indicating that it is live, and all requests to the `**/college**` path are handled by the `**collegeRoutes**` router. Finally, the server is started by calling the `**start**()` method on the `**TripToServerInstance**` object.

Code:

```
import collegeRoutes from "./routes/college.routes";
import cors from 'cors'
const PORT:number = process.argv[2] ? parseInt(process.argv[2]+"") :
parseInt(8000+"");
import express, { Application, Request, Response } from "express";

class TriptoServer {
  private app: Application;
  private port: number;

  constructor(port: number) {
    this.app = express();
    this.port = port;
    this.configureMiddleware();
    this.configureRoutes();
  }

  private configureMiddleware(): void {
    this.app.use(express.json());
    this.app.use(express.urlencoded({ extended: true }));
    this.app.use(cors())
  }

  private configureRoutes(): void {
    this.app.get("/", (req: Request, res: Response) => {
      res.status(200).json({ message: "We are live", PORT });
    });

    this.app.use("/college", collegeRoutes)
  }

  public start(): void {
    this.app.listen(this.port, () => {
      console.log(`Server listening on port ${this.port}`);
    });
  }
}

const TripToServerInstance = new TriptoServer(PORT);
TripToServerInstance.start();
```

8.3.2 Routes of all apis

This code exports a class named `**OpenAiRoutesClass**` which creates an instance of the `**Router**` class from the `**express**` module. The constructor of this class initializes various API routes using the methods of `**CollegeController**`, which handles all the logic for these API routes. The API routes include options to retrieve college information based on different filters such as city, state, courses, placement, and to get details of a single college based on its ID. Additionally, there is a route to create a new college and another one to delete an existing college. Finally, the `**OpenAiRoutes**` constant exports an instance of the `**OpenAiRoutesClass**`.

Code:

```
import express, { Router } from "express";
import CollegeController from "../controllers/college.controller";

class OpenAiRoutesClass {
   router: Router;
   constructor(router: Router){
      this.router = router;
      this.router.post("/", CollegeController.getAll); //main api with all filters
      this.router.get("/groupbyCity",   CollegeController.getAllGroupByCity);
// filter option city
      this.router.get("/groupbyState", CollegeController.getAllGroupByState);
//filter option state
      this.router.get("/placement",
CollegeController.getCollegesByPlacement); //filter option companies
```

```
        this.router.get("/college",     CollegeController.getOneByCollegeName);
//single college search
        this.router.get("/courses",      CollegeController.getAllGroupByCourse);
//filter option course
        this.router.get("/:id", CollegeController.getById); // Details of single
college
        this.router.post("/", CollegeController.create); // create one college with
all required details
        this.router.delete("/", CollegeController.delete);
    }
}
const OpenAiRoutes = new OpenAiRoutesClass(express.Router());
export default OpenAiRoutes.router;
```

8.3.3 Utilities

8.3.3.1 Logger module

The Below code defines a custom logger module using the Node.js `**EventEmitter**` class. It defines a `**logger**` object that emits events for different logging levels, such as `**info**`, `**warn**`, `**error**`, and `**Update_Request**`. Each logging level has a specific color associated with it that is printed to the console along with the log message and a timestamp. The module also exports three functions `**logAuthenticationSuccess**`, `**logAuthorizationSuccess**`, and `**logAuthenticationFailure**` that emit custom log events for successful authentication, authorization, and authentication failure, respectively. The purpose of this logger module is to enable logging and monitoring of events within the application and to provide a simple mechanism for custom logging with different logging levels**.**

Code:

```
import EventEmitter from 'events'
const logger:any = new EventEmitter();

const pick_color:any = {
    info: "\x1b[36m",
    error: "\x1b[31m",
    warn:  "\x1b[33m",
    Update_Request: "\x1b[40m",
}

logger.on('log', ({ level, message }:any) => {
    const timestamp = new Date()
    console.log(`${pick_color[level]}%s\x1b[0m`,                `[${level}]
${message} - ${timestamp}`);
});

logger.info = function (message:any) {
    this.emit('log', { level: "info", message });
};

logger.update = function() {
    this.emit('log',  {  level:  "Update_Request",  message:  "triggering
special update message log"})
}
```

```
logger.warn = function (message:any) {
   this.emit('log', { level: "warn", message });
};

logger.error = function(message:any) {
   this.emit('log', { level: "error", message})
}

export function logAuthenticationSuccess(email:any) {
   logger.emit('log', { level: "info", message: `User ${email}
successfully logged in` });
}

export function logAuthorizationSuccess(userID:any) {
   logger.emit('log', { level: "info", message: `User ${userID} is
Authorize.`});
}

export function logAuthenticationFailure(email:any, reason:any) {
   logger.emit('log', { level: "warn", message: `User ${email} failed to
log in (${reason})`});
}

export default logger;
```

8.3.3.2 Helper Module

The `**formatDate**` function is a helper function that takes a `Date` object as input and returns a formatted string in the format of `YYYY-MM-DD`. It uses the `**getFullYear**()`, `**getMonth**()`, and `**getDate**()` methods of the `Date` object to extract the year, month, and day values, respectively. The month and day values are also padded with leading zeros using the `**padStart**()` method to ensure they are two digits. This function can be useful when working with dates in a specific format, such as when formatting dates for use in a database or API.

```
export function formatDate(date: Date) {
   const year = date.getFullYear();
   const month = date.getMonth() + 1;
   const day = date.getDate();
   return `${year}-${month.toString().padStart(2, '0')}-
${day.toString().padStart(2, '0')}`;
}
```

8.3.4 Abstract Interfaces

The `User` interface defines the shape of user data, including an ID, name, email, password, createdAt and updatedAt timestamps.

The `**Cordinate**` interface is used to represent latitude and longitude coordinates as strings.

The `**Faculty**` interface defines the shape of faculty data, including their name, designation, experience, and qualification.

The `**Placement**` interface is used to represent placement data, including the name of the company and an associated image.

The `**Facilitie**` interface defines the shape of facilities data, including the name of the facility and an associated image.

The `**BaseData**` interface defines a base set of data for a college, including its name, state, city, logo, cover image, rating, URL, and contact information.

The `**BaseCardCourse**` interface defines a base set of data for a course card, including its fee, short form, and name.

The `**CourseData**` interface defines additional data for a course, including its short header, type, duration, fees per year, and eligibility criteria.

Code:

```
export interface User {
   id: number;
   name: string;
   email: string;
   password: string;
   createdAt: Date;
   updatedAt: Date;
}

export interface Cordinate { lat:string, lng:string }
export interface Faculty {
   faculty_name : string
   designation  : string
   exp          : string
   qualification: string
}

export interface Placement {
   company_name : string
   company_image: string
}

export interface Facilitie {
   facility_name:  string
   facility_image: string
}

export interface BaseData {
   college_name  : string
   state         : string
   college_city  : string
   logo          : string
   cover         : string
   rating        : string
   url           : string
   contact       : string
}

export interface BaseCardCourse {
    fee            : string
    short_form     : string
    name           : string
}

export interface CourseData {
    short_head     : string
    courses_name   : string
    type           : string
    duration       : string
    fees_per_year  : string
    eligibility    : string
}
```

8.3.5 Controller snip

This is a TypeScript file that defines a class called **`CollegeOperationClass`** with methods that handle various operations related to colleges. The **`getAll`** method retrieves a list of colleges with basic details, feature course details, and location data, filtered by different parameters such as college name, course name, placement company, state, and city. The result is paginated and sorted based on a specified order. The other methods in the class handle other operations such as getting colleges by ID, grouping colleges by city, state, or course, and creating or deleting colleges. The **`CollegeOperation`** constant creates an instance of the **`CollegeOperationClass`** to be exported for use in other files. The file imports necessary dependencies such as Prisma, logger, and Express.

Code:

```
import { Prisma, PrismaClient, baseCardCourseSchema } from
'@prisma/client';
import logger from '../utils/logger'
import { Request, Response, query } from 'express';
import { BaseCardCourse, BaseData, Cordinate, CourseData, Facilitie,
Faculty, Placement } from '../models/college.model';
const prisma = new PrismaClient();

type orderBy = "college_name" | "rating"

class CollegeOperationClass {
  //get all list of colleges with basic college details and feture course details
and location data
  async getAll(req:Request, res:Response) {
    try {
        const pages: {numberOfRecord: number, pageNumber: number,
college_name: string, short_head: string[], placement_comapany: string[],
state: string[], city: string[], order: orderBy} = {
            numberOfRecord: req.body.numberOfRecord || 10,
            pageNumber: req.body.pageNumber || 1,
            college_name: req.body.college_name || "",
            short_head: req.body.short_head || [],
            placement_comapany: req.body.placement_comapany || [],
            state: req.body.state || [],
            city: req.body.city || [],
            order: req.body.order || ""
        }

        let filterObject = {
            where: {
                college_name: {
                        contains: pages.college_name
                },
                ...(pages.short_head.length > 0 && {
                    courses: {
                        some: {
                            short_head: {
                                in: pages.short_head
                            }
                        }
        }
    }),
    ...(pages.placement_comapany.length > 0 && {
        placements: {
            some: {
                company_name: {
                    in: pages.placement_comapany
                }
            }
        }
    }),
    ...(pages.state.length > 0 && {
        state: {
            in: pages.state
        }
    }),
    ...(pages.city.length > 0 && {
        college_city: {
            in: pages.city
        }
    })
},
orderBy: {},
include: {
    _count: true,
    baseCardCourseSchema: true,
                Cordinate: true
            },
            take: pages.numberOfRecord,
            skip: (pages.pageNumber - 1) * pages.numberOfRecord
        }

        if(pages.order.length > 0) {
            filterObject.orderBy = {
                [pages.order]: 'asc'
            }
        }

        const ManyRecordsWithPagination = await
prisma.collegeSchema.findMany(filterObject)
        const _count = await prisma.collegeSchema.count({where:
filterObject.where})
        logger.info("Get List Of Colleges with Pagination")
        res.status(200).json({
            data: ManyRecordsWithPagination,
            _count
        })

    } catch (e) {
        logger.error("Failed to Get All List Of Colleges");
        res.status(500).json({msg: "Failed to Get All List Of Colleges"});
    }
    }
    async getById(req:Request, res:Response) {}
    async getAllGroupByCity(req:Request, res:Response) {}
    async getAllGroupByState(req:Request, res:Response) {}
    async getAllGroupByCourse(req:Request, res:Response) {}
    async getOneByCollegeName(req:Request, res:Response) {}
    async getCollegesByPlacement(req:Request, res:Response) {}
    async create(req:Request, res:Response) {}
    async delete(req:Request, res:Response) {}
}

const CollegeOperation = new CollegeOperationClass();

export default CollegeOperation;
```

8.3.6 Dependencies and code start script

This is a basic `package.json` file for a Node.js project named `collegeproject`. It lists the project name, version, description, entry point, scripts, dependencies, and devDependencies. The devDependencies include TypeScript, ts-node, and nodemon for development purposes. The dependencies include Express and cors for building the API, dotenv for handling environment variables, and Prisma and axios for database operations and making HTTP requests, respectively.

Code:

```
{
  "name": "collegeproject",
  "version": "1.0.0",
"description": "",
"main": "src/index.ts",
"scripts": {
  "start": "node src/index.ts",
  "start:dev": "nodemon src/index.ts 3000",
  "test": "node tests/insertColege.js"
},
"keywords": [],
"author": "",
"license": "ISC",
"devDependencies": {
  "@types/cors": "^2.8.13",
  "@types/express": "^4.17.17",
  "@types/node": "^18.15.13",
  "express": "^4.18.2",
  "nodemon": "^2.0.22",
  "prisma": "^4.13.0",
  "ts-node": "^10.9.1",
  "typescript": "^5.0.4"
},
"dependencies": {
  "@prisma/client": "^4.13.0",
  "axios": "^1.3.6",
  "cors": "^2.8.5",
  "dotenv": "^16.0.3"
} }
```

8.4 Frontend

8.4.1 Main entry point

This code is a basic entry point for a React application. It imports various modules and components, sets up the necessary contexts, and renders the main application component (App) inside a hierarchy of context providers

Code:

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App'
import './index.css'
import ThemProvider from './context/ThemeContext'
import "./output.css"
import NavigationProvider from './context/NavigationContext'
import CollegeProvider from './context/CollegeContext'
import MapContextProvider from './context/MapContext'

ReactDOM.createRoot(document.getElementById('root') as
HTMLElement).render(
  <React.StrictMode>
    <ThemProvider>
      <NavigationProvider>
        <CollegeProvider>
          <MapContextProvider>
              <App />
          </MapContextProvider>
        </CollegeProvider>
      </NavigationProvider>
    </ThemProvider>
  </React.StrictMode>,
)
```

8.4.2 Features navigation

This code is a React component called App. It imports various dependencies and components, and it uses the useContext and useEffect hooks from React.

The useContext hook is used to access the values from the ThemeContext and NavigationContext contexts. The ThemeContext provides the current theme, and the NavigationContext provides the current navigation state.

Inside the useEffect hook, there is a function that updates the theme of the document's body whenever the theame.theme value (obtained from the ThemeContext) changes. This ensures that the UI reflects the current theme.

The component then uses the nav value from the NavigationContext to conditionally render different components based on the navigation state. If nav is equal to 0, it renders the Advance component. If nav is equal to 1, it renders the Distence component. If nav is equal to 2, it renders the Details component. Lastly, if nav is equal to 4, it also renders the Distence component

Code:

```
import { useContext, useEffect, useState } from 'react'
import './App.css'
import Advance from './Components/Advance/Advance'
import Distence from './Components/Distence/Distence';
import { NavigationContext } from './context/NavigationContext';
import { ThemeContext } from './context/ThemeContext';
import Details from './Components/Details/Details';

function App() {
  const theame = useContext(ThemeContext);
  useEffect(() => {
    const body = document.body;
    body.setAttribute("data-theme", theame.theme);
  }, [theame.theme]);
  const {nav} = useContext(NavigationContext);
  return (
    <>
      {
        nav === 0 && (<Advance />)
      }

      {
        nav === 1 && (<Distence />)
      }

      {
        nav === 2 && (<Details />)
      }
```

```
        {
          nav === 4 && (<Distence />)
        }
      </>
    )
  }
export default App
```

8.4.3 create context here i am given example of one of the context

This code is implementing a theme provider in React using the context API. It creates a context called ThemeContext with a default theme of "light" and an empty function for changing the theme. The ThemProvider component sets the initial theme state to "dark" using the useState hook. It also defines a changeTheme function that toggles the theme between "light" and "dark" when called. The ThemProvider component wraps its children with the ThemeContext.Provider, providing the current theme and the changeTheme function as the context value. This allows any child components that consume the ThemeContext to access and update the theme as needed.

Code:

```
import React, { createContext, ReactNode, useState } from "react";

type ThemeType = "light" | "dark";

type ThemeContextType = {
   theme: ThemeType,
   changeTheme: ()=> void
}
```

```
export const ThemeContext = createContext<ThemeContextType>({
   theme: "light",
   changeTheme: ()=> {
      //
   }
});

type Themeprops = {
   children: ReactNode
}
const ThemProvider = (props:Themeprops) => {
   const [theme, setTheme] = useState<ThemeType>("dark");
   const changeTheme = ():void => {
      setTheme((pre)=> {
         return pre == "light" ? "dark" : "light"
      })
   }
   return (
      <ThemeContext.Provider value={{
         theme,
         changeTheme
      }}>{props.children}</ThemeContext.Provider>
   )
}
export default ThemProvider
```

8.4.4 Map Object to create Map Every time we render the UI to use with our Features

This code defines a React component called MapBox that utilizes the @react-google-maps/api library to render a Google Map.

The component accepts several props such as center, latitudeFieldName, longitudeFieldName, and zoom, which can be customized to control the initial center position and zoom level of the map.

Within the component, the useContext hook is used to access the colleges data from the CollegeContext context.

The component also uses the useJsApiLoader hook from @react-google-maps/api to load the Google Maps JavaScript API and ensure that it is available before rendering the map. The API key required for loading the map is obtained from an environment variable.

Code:

```
import {
  GoogleMap,
  Marker,
  useJsApiLoader,
 } from "@react-google-maps/api"
 import { FC, useContext } from "react"
 import {
  ContainerStyle,
  DefaultCenter,
  DefaultZoom,
  LatitudeFieldName,
  Libraries,
  LongitudeFieldName,
} from "../../BaseComponent/Map/content"
import { MapProps } from "../../BaseComponent/Map/interface"
import { useCenter } from "./useCenter"
import { CollegeContext } from "../../../context/CollegeContext"

const MapBox: FC<MapProps> = ({
  center = DefaultCenter,
  latitudeFieldName = LatitudeFieldName,
  longitudeFieldName = LongitudeFieldName,
  zoom = DefaultZoom,
}) => {
  const { colleges } = useContext(CollegeContext)

  const {
    map,
    currentCenter,
    currentZoom,
    handleLoad,
    handleUnmount,
  } = useCenter(center, zoom, latitudeFieldName, longitudeFieldName)

  const { isLoaded } = useJsApiLoader({
    id: "google-map-script",
    googleMapsApiKey:
import.meta.env.VITE_REACT_APP_GOOGLE_MAPS_API_KEY,
    libraries: Libraries,
  })
  if (!isLoaded) return null
  return (
    <GoogleMap
      mapContainerStyle={ContainerStyle}
      center={currentCenter}
      zoom={currentZoom}
      onLoad={handleLoad}
      onUnmount={handleUnmount}>
        {
          colleges.length > 0 && colleges.map((val:any, index) => {
            return (
                <Marker key={index} onClick={()=> {
                  map?.panTo({lat: parseFloat(val.Cordinate[0].lat), lng:
parseFloat(val.Cordinate[0].lng)})
                }} position={{lat: parseFloat(val.Cordinate[0].lat), lng:
parseFloat(val.Cordinate[0].lng)}} />
              )
          })
        }
    </GoogleMap>
  )
 }
  MapBox.displayName = "MapBox"
  export default MapBox
```

8.4.5 some Untidily functions to standardize api calls

The getData function takes two parameters: params (a string) and query (an object). It returns a Promise that resolves to the response data from the API call. The function builds a URL using the buildUrl function (presumably defined in a separate file) and appends the query object as query parameters to the URL if it is provided. Then it makes a GET request using Axios and returns the response data.

The postData function is similar to getData but makes a POST request instead. It takes the same parameters, params and query, and returns a Promise that resolves to the response data from the API call. It builds the URL using the buildUrl function and sends a POST request using Axios with the query object as the request body.

Code:

```
import axios from 'axios';
import buildUrl from './urlBuilder';
export const getData = (params:string, query:any) => {
  return new Promise((resolve:any, reject:any) => {
    (async () => {
      try {
        const url = buildUrl(params)+(query? query: "")
        const data = await axios.get(url, query? query : "");
        resolve(data);
      } catch (e) {
        console.log('Error occurred in API call: ', e);
        reject(e);
      }
    })();
  }); };
```

```
export const postData = (params:string, query:any) => {
  return new Promise((resolve:any, reject:any) => {
    (async () => {
      try {
        const data = await axios.post(buildUrl(params), query? query : "");
        resolve(data);
      } catch (e) {
        console.log('Error occurred in API call: ', e);
        reject(e);
      }
    })();
  });
}
```

8.4.6 Layout to have basic standered structure

This code defines a React functional component called Layout. It takes several props, including children, className, fullWidth, id, transparent, and noBorder.

Within the component, it first imports the necessary dependencies from React and other custom components. It also imports the ThemeContext from a context file.

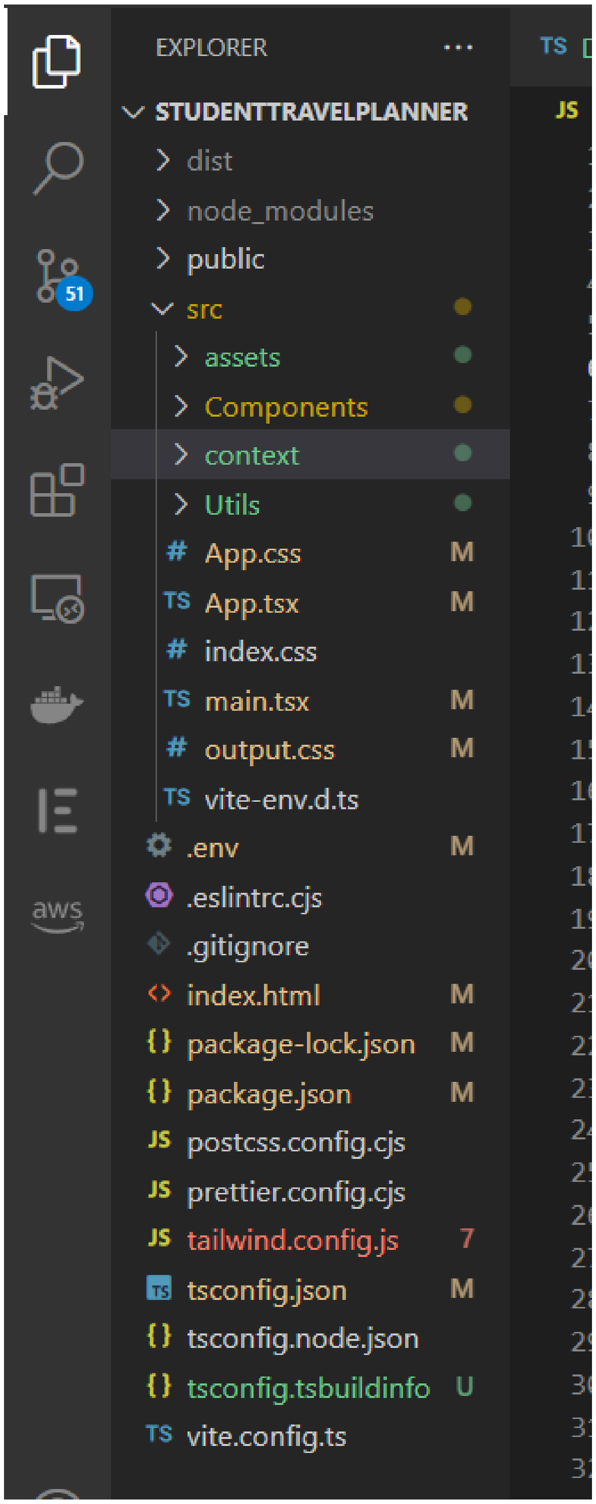
The component renders a main container (<main>) with a flex layout and minimum height of the screen. It has a column layout and aligns its items at the start. The background color of the container depends on the transparent prop value.

Inside the main container, it renders a NavBar component at the top and a Footer component at the bottom.

Code:

```
import React, { useContext, useEffect } from "react";
// import Footer from "./Footer";
// import NavBar from "./NavBar";
import Footer from "../BaseComponent/Footer";
import NavBar from "../BaseComponent/NavBar";
import { ThemeContext } from "../../context/ThemeContext";

type Props = {
  children: JSX.Element;
  fullWidth?: boolean;
  className?: string;
  id?: string;
  transparent?: boolean;
  noBorder?: boolean;
};

export default function Layout({ children, className, fullWidth, id,
transparent, noBorder }: Props) {
  const theame = useContext(ThemeContext);
  return (

    <main className={`flex min-h-screen w-screen flex-col items-center
justify-start ${!transparent && "bg-lightBg2"} `}>
      <NavBar />
    <div className="flex w-full flex-col justify-between md:px-24 md:py-16
rounded-3xl">
      <div className="pb-16">
        <div id={id}
          className={`${className} ${
            fullWidth ? 'w-full' : 'max-w-screen-2xl'
          } mx-auto ${theame.theme == "light" ? `border-base-300 ${!noBorder
&& "md:border-2"}`: ""} ${transparent ? "bg-base-100" : "bg-base-200"}
rounded-2xl`}>

          {children}
        </div>
      </div>
      <Footer />
    </div>
  </main>

 );
}
```

Folder Structure:



```
EXPLORER ...
∨ STUDENTTRAVELPLANNER
  > dist
  > node_modules
  > public
  ∨ src                         ●
    > assets                    ●
    > Components                ●
    > context                   ●
    > Utils                     ●
    App.css                     M
    App.tsx                     M
    index.css
    main.tsx                    M
    output.css                  M
    vite-env.d.ts
  .env                          M
  .eslintrc.cjs
  .gitignore
  index.html                    M
  package-lock.json             M
  package.json                  M
  postcss.config.cjs
  prettier.config.cjs
  tailwind.config.js            7
  tsconfig.json                 M
  tsconfig.node.json
  tsconfig.tsbuildinfo          U
  vite.config.ts
```

Fig. 29 Front-end Folder Structure in vscode Editor

1. `dist`: This folder is created when the production build of the application is generated. It contains all the files needed to deploy the application.

2. `node_modules`: This folder contains all the dependencies required by the application. When you run `npm install`, it installs all the dependencies in this folder.

3. `public`: This folder contains all the publicly accessible files, such as `index.html`, which is the entry point of the application.

4. `src`: This folder contains all the source code of the application, including components, assets, and styles.

5. `.env`: This file is used to store environment variables, such as API keys or database credentials. It is usually excluded from version control.

6. `.eslintrc.cjs`: This file contains the rules for linting the code. ESLint is a popular JavaScript linting tool that helps identify and fix errors in the code.

7. `.gitignore`: This file specifies which files and folders should be ignored by Git when committing changes. For example, `node_modules` should be ignored because it is generated by `npm install` and is not needed in the repository.

8. `index.html`: This file is the entry point of the application. It contains the root element where the React components are rendered.

9. `package-lock.json`: This file is automatically generated by `npm install` and is used to lock the versions of the dependencies. This ensures that the same versions of the dependencies are installed on different machines.

10. `package.json`: This file contains the metadata of the application, such as its name, version, description, and dependencies. It also contains scripts that can be run using `npm run <script>`.

11. `postcss.config.cjs`: This file is used to configure PostCSS, a tool that helps transform CSS with JavaScript plugins. It is used to apply styles such as autoprefixing or minification.

12. `prettier.config.cjs`: This file contains the configuration for Prettier, a code formatter that automatically formats the code according to certain rules. It is used to enforce consistent formatting across the codebase.

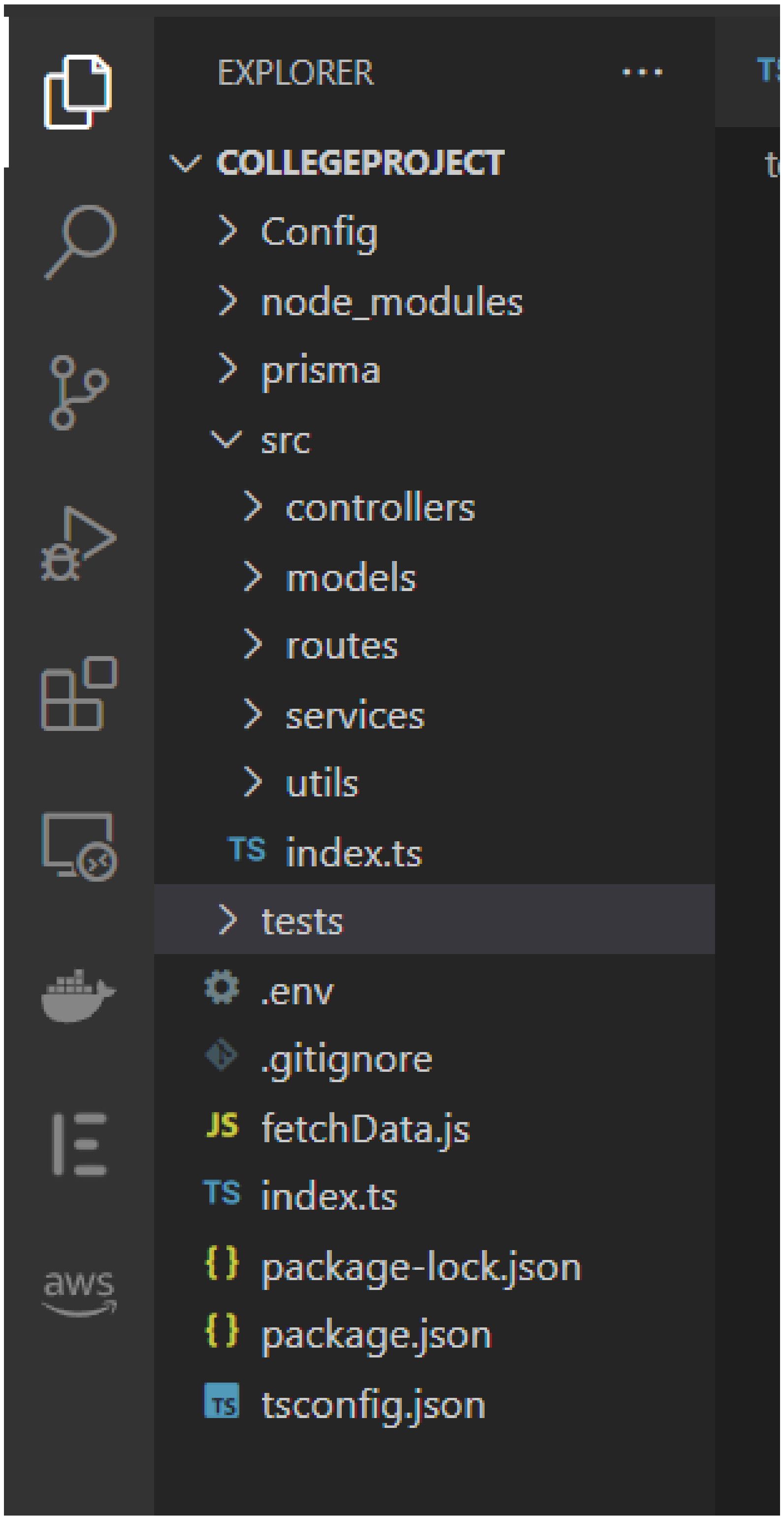
13. `tailwind.config.js`: This file is used to configure Tailwind CSS, a utility-first CSS framework that makes it easy to create custom designs. It is used to apply specific styles to the components.

14. `tsconfig.json`: This file is used to configure TypeScript, a typed superset of JavaScript that adds features such as interfaces, enums, and type checking. It specifies the compiler options and the files to include in the build.

15. `tsconfig.node.json`: This file is a helper file for TypeScript that specifies the compiler options for running the application in Node.js.

16. `tsconfig.tsbuildinfo`: This file is automatically generated by TypeScript and contains information about the build process. It is used to speed up the subsequent builds.

17. `vite.config.ts`: This file is used to configure Vite, a build tool that provides fast and optimized builds for React applications. It specifies the entry point, the output directory, and the plugins to use.



```
EXPLORER ...
v COLLEGEPROJECT
  > Config
  > node_modules
  > prisma
  v src
    > controllers
    > models
    > routes
    > services
    > utils
    TS index.ts
  > tests
  .env
  .gitignore
  JS fetchData.js
  TS index.ts
  {} package-lock.json
  {} package.json
  tsconfig.json
```

Fig. 30 Backend Folder Structure in vscode Editor

1. `Config`: This folder contains all the configuration files required for the project.

2. `Node_modules`: This folder contains all the project dependencies. When you install any package using npm, it is installed in this folder.

3. `Prisma`: This folder contains all the Prisma-related files, including the schema and the database configuration.

4. `Src`: This folder is the main folder of the project and contains all the application's source code.

a. `Controllers`: This folder contains all the controller functions of the project. These functions take the incoming requests, process them, and return the response.

b. `Models`: This folder contains all the abstract classes that represent the data structure of the application.

c. `Routes`: This folder contains all the API routes of the project. Each route maps to a controller function.

d. `Services`: This folder contains all the services required by the project. A service is a set of functions that interact with the database or other external APIs.

e. `Utils`: This folder contains small utility functions that are used throughout the project.

f. `Index.ts`: This file is the entry point of the application. It sets up the server and starts listening to incoming requests.

5. `Test`: This folder holds all the testing files and test cases for the project.

6. `.env`: This file contains all the secret keys and environmental variables that the project uses. This file is not committed to version control.

7. `.gitignore`: This file lists all the files and folders that should be ignored when committing the code to Git.

8. `fetchData.js`: This is a normal JavaScript file that contains dummy data. It can be used to test the API routes of the project.

9. `Package-lock.json`: This file keeps track of all the modules that are installed in the project, along with their dependencies.

10. `Package.json`: This file contains all the project details, including the dependencies, scripts, and project name.

11. `Tsconfig.json`: This file is used to configure the TypeScript compiler. It specifies the TypeScript version, the output directory, and the root directory of the project.