Project Report

# Service Navigator

## By Sudipto Patra

### Under the Guidance of Prof. Sujoy Dutta

------------------------------

## Problem Statement

**Integrated Common Services for Common People**

Although a wealth of information is available on the internet, people often struggle to find relevant, complete, and accurate information. The goal of this project is to develop a unified platform that integrates various essential services, making them easily accessible and beneficial for the general public.

------------------------------

## Introduction

Service Navigator is a web application that serves as a central hub for accessing various services, including transportation, healthcare, financial services, and more, all through a user-friendly interface. The application gathers data from various sources and provides users with information tailored to their needs. Additionally, service providers can register and showcase their services, facilitating greater visibility and easier management through a dedicated dashboard.

------------------------------

## Features

### For Service Consumers:

#### Healthcare:

- Provides information about nearby healthcare facilities (hospitals, pharmacies, labs) including basic details and reviews.  
- Offers additional information such as opening hours, pricing range, and reviews for selected facilities.  
- Highlights the best healthcare facilities in your area.  
- Lists nearby emergency services and elder care options.

#### Finance:

- Provides information about nearby banks, including details on interest rates for various accounts and loans, and shows the nearest banks.  
- Displays information about nearby ATMs and their current operational status.

#### Education:

- Offers information about schools in your state, region, or block, including details on labs, playgrounds, libraries, and internet availability.  
- Lists tutors available in your locality.  
- Provides information on educational aids such as government policies and second-hand study materials.

#### Transport:

- Shows a visual representation of your route on a map with details like distance, estimated time, and price comparisons between Ola, Uber, and local services.  
- Lists different local pick-and-drop service providers.  
- Provides information on vehicle rental options in your area.  
- Offers details on the cheapest flight options based on your preferences.

### For Service Providers:

#### Service Registration:

- Allows service providers to register and list their services, making them visible to a large consumer base.  
- Supports customized forms for different services to ensure accurate data collection.

#### Dashboard:

- Includes a comprehensive dashboard for easy management and overview of all services provided.  
- Enables service providers to update information, manage availability, and respond to user reviews and inquiries.

#### Additional Features:

- Facilitates showcasing services to a broad audience, enhancing visibility and reach.  
- Ensures secure authentication for both consumers and service providers, using Auth0 and auth tokens in Django.

------------------------------

# Working Principles

## Service Consumer

### Authentication

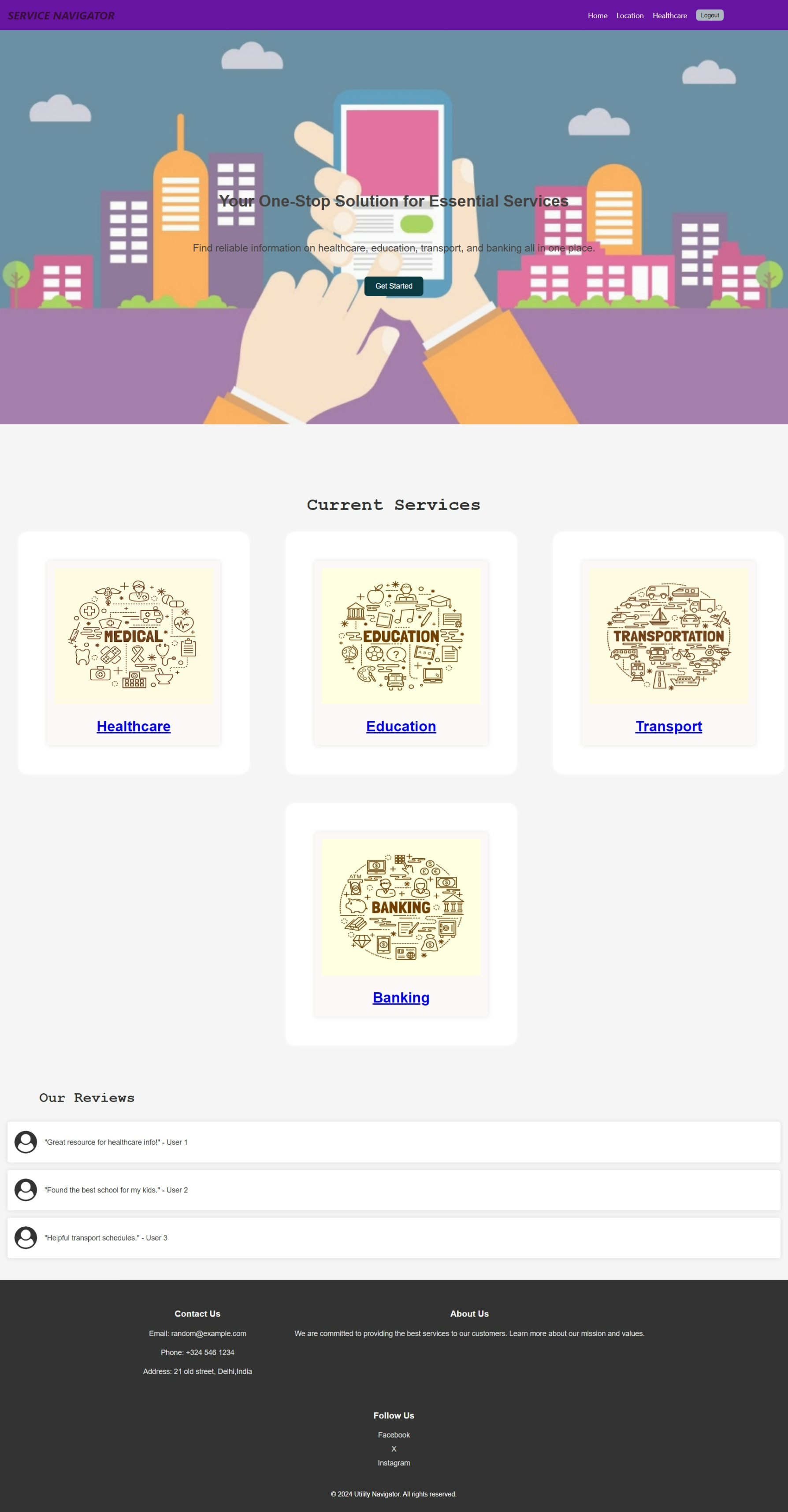
**Landing Page**: Users encounter a landing page with options to log in or register, managed through Auth0's streamlined authentication flow. There is also an option to redirect to the service provider page.



**Successful Authentication**: Upon successful authentication, users are redirected to the homepage. Before loading the homepage, authentication is verified using the isAuthenticated function for security. This check is applied to all pages.

### Home Page

**Service Selection**: Users can choose from the current pool of services. Each option is linked to its respective page using the Link React component.



**Location Handling**: Upon opening a new page, the system checks for the presence of location data in the user's local storage. If not present, a pop-up for location entry appears.

**Automatic Location Retrieval**: Utilizes the Geolocation API to get the current location of the user. This is a built-in feature of modern web browsers. If the browser does not support this feature, an error response is generated.

**Manual Location Entry**: Facilitated by Google Autocomplete for better location accuracy.

**Initialization**: Upon component mount, the Google Maps Places Autocomplete service is initialized on the input field (id="autocomplete").

**User Interaction**: Users can type in the input field to receive location suggestions based on Google's extensive database of places.

**Selection Handling**: When a user selects a place from the autocomplete dropdown, the place\_changed event triggers. This event retrieves the selected place’s details and updates the component’s state with its latitude and longitude coordinates.



### Fetching Data

**Data Request**: Based on the selected service type, the assigned component makes a request to the corresponding backend endpoint. The component sends a POST request with the user's location and the selected filter.

**Backend Processing**:

**Error Handling**: The backend checks for any missing or invalid parameters and returns an appropriate error response if needed.

**Data Collection**: Data is collected in various ways depending on the service, including API requests, database queries, and CSV file reading.

**External API Requests**:

* + - * Google Places API: Fetches nearby places such as hospitals, pharmacies, labs, banks, and ATMs.
      * Google Distance Matrix API: Calculates the distance between the user's location and the services' locations.
      * Google Geocode API: Retrieves state, district, and block information from latitude and longitude.
      * Amadeus API: Fetches IATA codes and flight offers.
      * Exchange Rate API: Fetches the latest exchange rates.
      * Ola and Uber API: Fetches price estimates for cab rides.

**Database Queries**: Django ORM queries fetch educational, healthcare, and transport services.

**CSV File Reading**: Local CSV files are read to get interest rates and school information.

**Cached Data**: Cached results optimize performance and reduce redundant API calls. The cache is designed to store recent 100 data and for 1 hour.

**Request Data**: Collected directly from the incoming request payload in JSON format.

**Fuzzy Matching**: The FuzzyWuzzy library is used to match and clean bank names against a list of known bank names.

### Response Generation

The processed and aggregated data is encapsulated in a Response object formatted as JSON, the standard format for APIs. The Response object is returned to the frontend, including the status code (e.g., 200 for success, 400 for bad request) and the JSON data payload.

The frontend receives the JSON response and processes it as needed. Users can interact with the list using pagination controls to view more or fewer items.

### Example of Data Displayed After Retrieval



## Service Provider

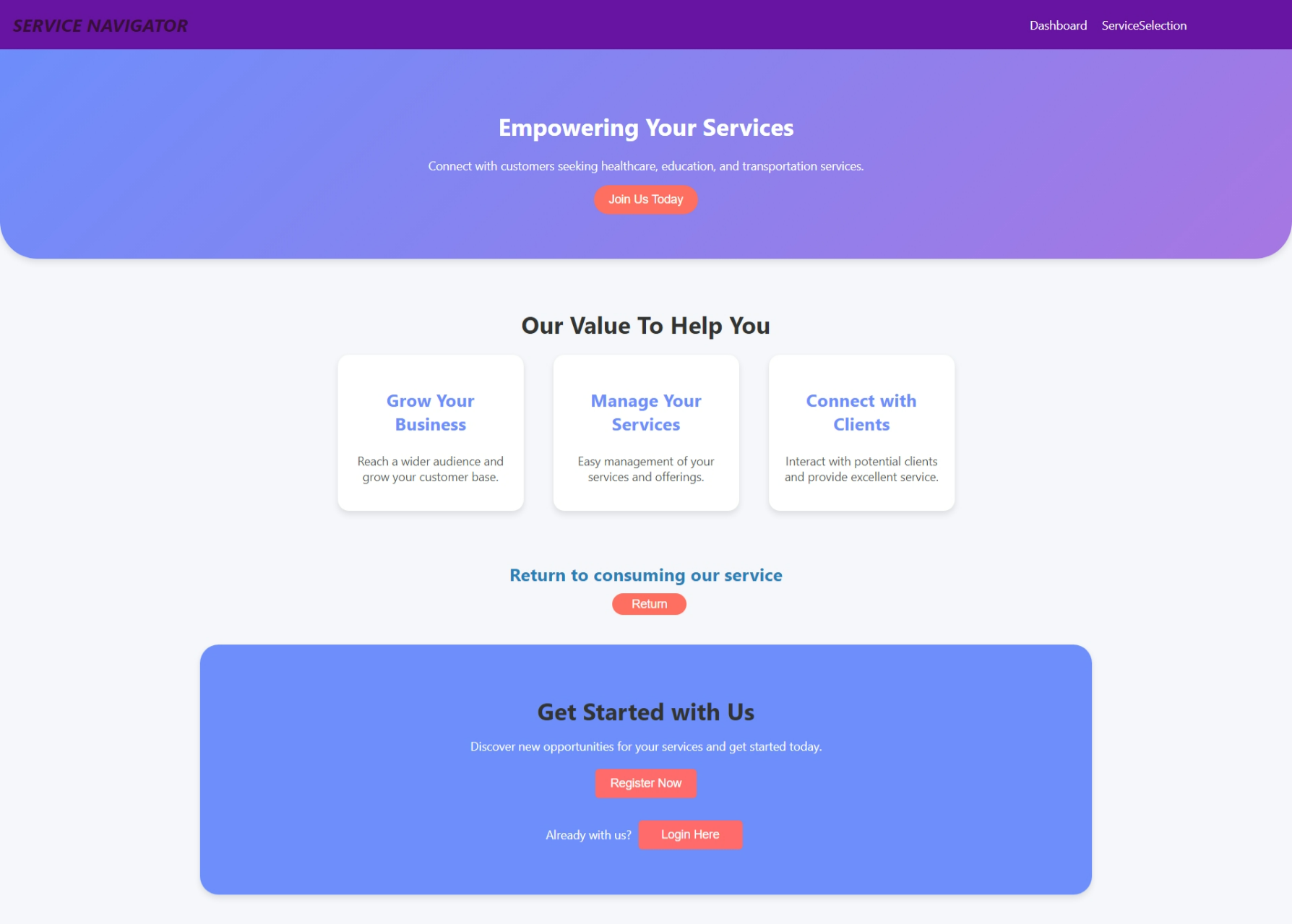
### Authentication and Navigation

**Service Provider Home Page**: Users can register, log in, or return to the landing page. The registration and authentication are handled as follows:

**User Registration**: Users register via the RegistrationForm, which sends a POST request to /register/.

**Login**: Upon registration, users can log in using the LoginForm, sending a POST request to the authentication endpoint provided by dj\_rest\_auth. Successful login saves a token in localStorage used for authenticated requests.

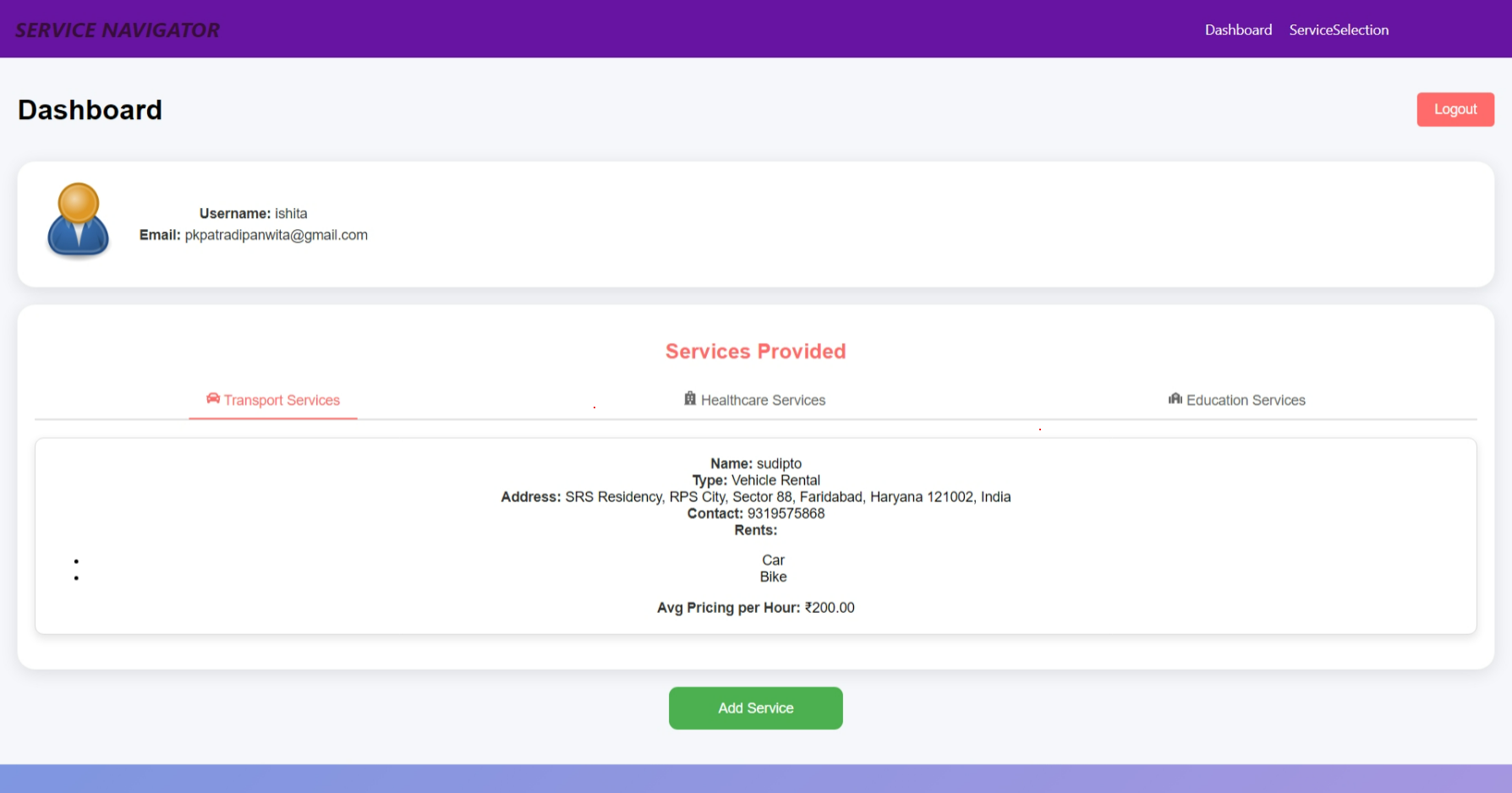
After successful login and authentication, the dashboard is loaded.



### Fetching User Data and Services

The Dashboard component fetches user data from /profile/ and user services from /profile/services/ on mount. Both endpoints require the authentication token for access.

The Dashboard component includes an option for logout, which handles logout by redirecting to the home page and removing the token from localStorage.



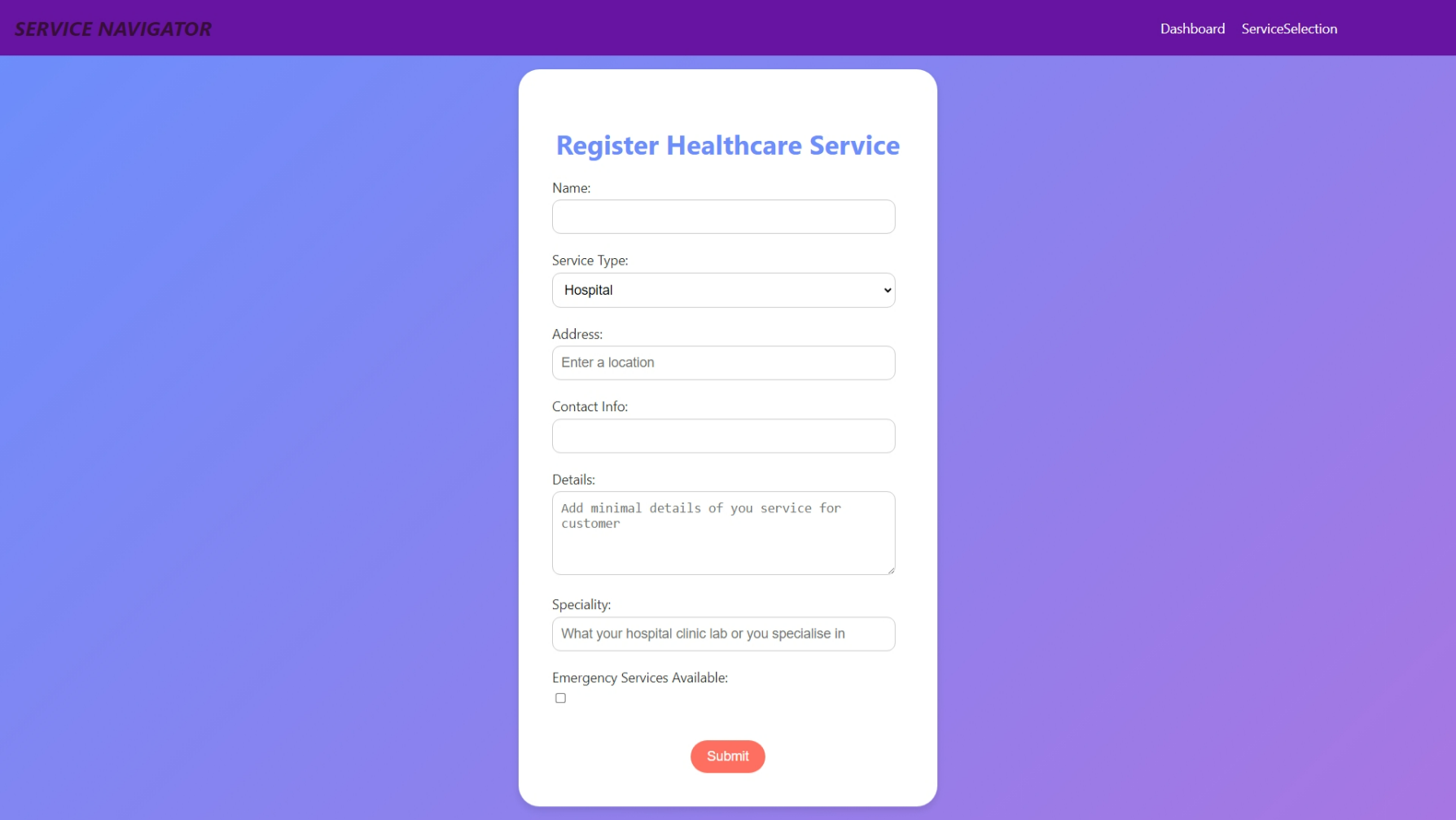
### Service Registration

After selecting a service type in the ServiceSelection component, users are navigated to the corresponding service registration form. Service registration forms send a POST request to the appropriate endpoint (/register-transport/, /register-healthcare/, /register-educational/) with the service data.

The endpoint checks for duplicate data and generates an error if duplicate data is entered, helping to reduce redundancies.

After successful registration of the service, the user is redirected to the dashboard.

### Example of a Service Registration Form



### Handling User State

* The frontend components use React state and hooks to manage user data, services, and loading/error states. Navigation is handled using useNavigate from react-router-dom.

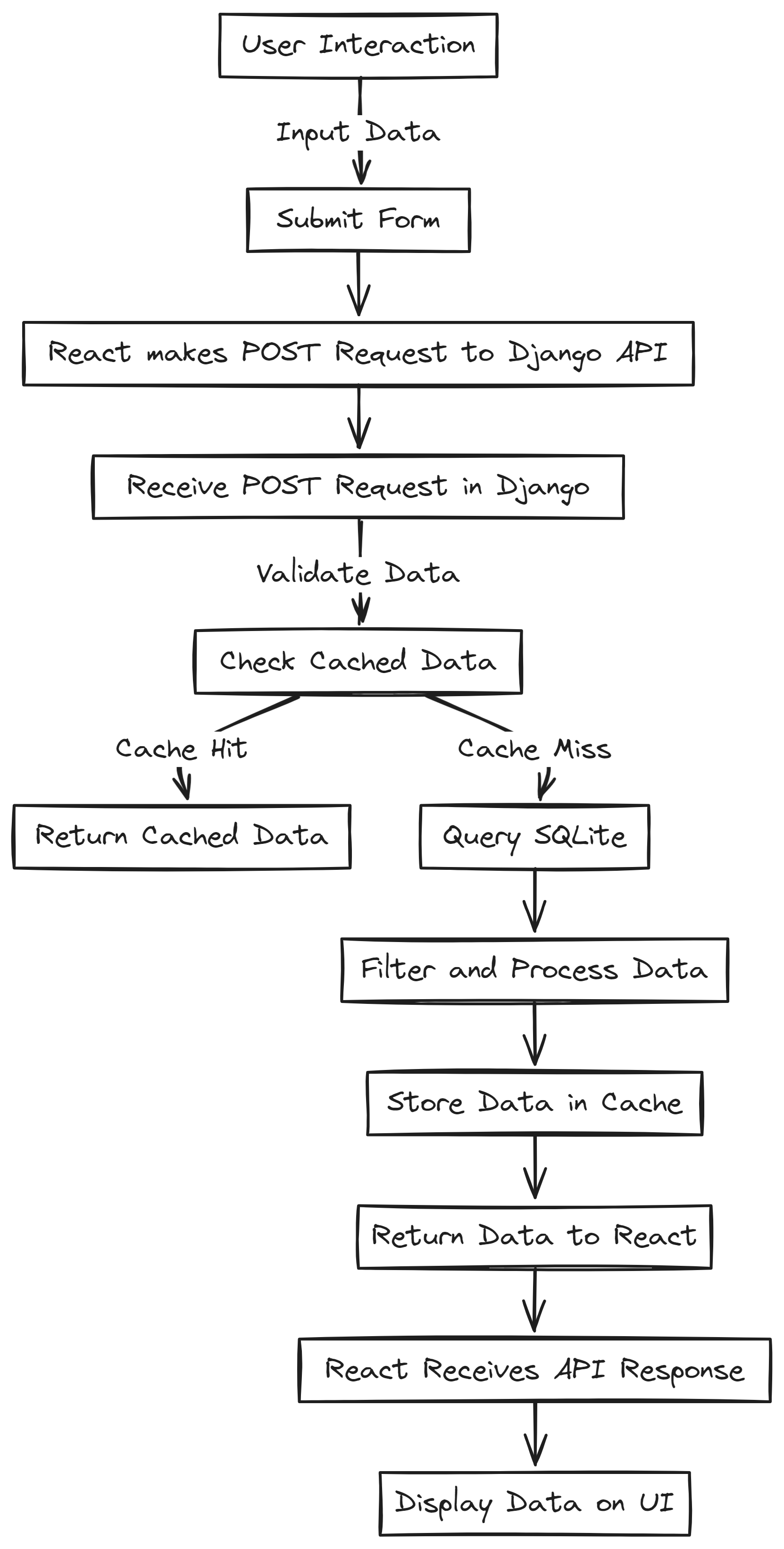
------------------------------

**Architecture Diagram**



------------------------------

## Process Flow



------------------------------

## Major Challenges and Solutions

1. Incomplete Data for School Facilities:  
 - Created a personal data file using web scraping and processed it with Pandas.  
  
2. Redundant API Calls:  
 - Implemented caching using TTLCache from Cachetools to reduce unnecessary multiple API calls.  
  
3. Backend Response Delays Due to Distance Calculations:  
 - Implemented asynchronous calls from the frontend to retrieve facilities first, followed by distance calculations, allowing users to view other data in the meantime.  
  
4. User Authentication and Route Protection:  
 - Utilized Auth0 for user authentication and auth tokens in Django for service provider authentication, securing routes by checking authentication before page loads.  
  
5. Matching Interest Rates of Nearby Bank Branches:  
 - Created a file with bank interest rates and used FuzzyWuzzy for fuzzy matching to align nearby bank branch data with bank information, displaying accurate results.

------------------------------

## Technologies Used

### Frontend:

- React: Used throughout the frontend to create a dynamic and responsive user interface.  
- React Router: For handling navigation and routing.  
- Axios: For making API requests to the Django backend and external services like Google APIs and Auth0.  
- HTML (JSX): For the structure of components.  
- CSS: For adding styles to components.

### Backend:

- Python: Core programming language for backend logic, API integration, data processing, and more.  
- Django: Provides a robust backend to manage multiple functionalities efficiently.  
- Django REST Framework: For creating API views, serializers, and managing HTTP requests and responses.  
- Geopy: For distance calculations between geographical points.  
- Cachetools: Utilized TTLCache for caching API responses and frequently accessed data.  
- Pandas: For reading and processing CSV files.  
- FuzzyWuzzy: For string matching.

### APIs and External Services Used:

- Google APIs  
- Amadeus APIs  
- Ola APIs  
- Uber APIs  
- Auth0

Note: For installation of all dependencies and code execution, please refer to the README file in the repository.

------------------------------

## Advantages

1. Caters to Both Service Providers and Consumers:

The platform is designed to serve both service providers and service consumers, ensuring that both parties benefit from its use.

2. Multiple Services Integrated in One Place:

By offering a variety of services in one location, the platform provides convenience and efficiency for users.

3. Industry-Level Authentication:

The platform employs advanced authentication methods to ensure security and trustworthiness.

4. Data from Trusted Sources:

The data used on the platform is sourced from reliable government websites and large companies, enhancing credibility.

5. Easy Organization of Services:

Service providers find it easy to manage and organize their offerings on the platform.

6. User-Friendly Interface:

The platform features a simple, intuitive user interface, making it accessible and easy to navigate for all users.

7. Robust Error Handling System:

A strong error handling system is in place to manage and resolve issues efficiently.

8. Large Data Pool:

The platform has access to a substantial amount of data, providing users with a wealth of information.

## Disadvantages

1. Limited to India:

Major services offered by the platform are not available to users outside of India, limiting its global reach.

2. Dependence on Third-Party Data:

The platform relies heavily on data from third-party sources, which may affect its reliability and accuracy.

3. Low Scale of Operation:

Currently operating at a low scale, the platform may struggle to handle high traffic olumes, potentially impacting performance.

4. Lack of Local or In-House Data:

The platform does not have access to local or in-house data, which could limit the depth and specificity of the information provided and increase its dependence on third-party data.

------------------------------

## Future Scope

- Expand globally to cater to users outside India.  
- Enhance the existing dataset to provide more comprehensive and detailed information.  
- Include additional services beyond the current offerings.  
- Implement a custom review and rating system.

------------------------------

## Conclusion

Service Navigator is a well-structured React application designed to provide users with easy access to various services. By focusing on performance, reliability, scalability, usability, security, and maintainability, the application delivers a high-quality user experience. Future enhancements will broaden the project's impact, improve user experience, and maintain its competitive edge in the market.