**BUS PASS GO: A USER-FRIENDLY PORTAL FOR ONLINE BUS PASS MANAGEMENT**

**Abstract**

The development of bus pass management systems has evolved significantly with the advancement of technology. Traditionally, bus passes were handled manually through physical applications, where users would submit paper forms to bus operators or local government offices. These forms were reviewed manually, and the approval process was time-consuming and prone to errors. In many cases, applicants would have to visit the offices multiple times, leading to delays in receiving the bus passes. Several systems, such as manual ticketing in local bus services or government-operated public transport systems, attempted to solve this issue but often lacked automation and efficiency. The need for a more streamlined and efficient system became apparent as urbanization increased and the demand for public transport grew. The traditional approach to managing bus passes posed numerous challenges, including long queues, paperwork errors, and lack of centralized access to information. The absence of digital records made it difficult for transport authorities to track and manage bus pass renewals, approvals, or application statuses in real-time. The inspiration for developing an automated web-based bus pass management system arose from the need to address these inefficiencies. A web-based system allows users to apply for bus passes online, check the status of their applications, and request renewals without needing to visit the bus service offices. This system aims to reduce processing time, eliminate paperwork, and provide a more accessible, user-friendly interface for both applicants and administrators. It offers real-time status updates, reduces human errors, and ensures that bus pass applications are processed faster.The primary problem in the traditional system was the inefficiency in processing and tracking applications, which led to delays, mismanagement of applications, and a lack of transparency. Furthermore, the absence of an online platform meant users had limited access to information regarding the status of their bus pass applications, leading to confusion and frustration. Additionally, renewal processes were often cumbersome and required repeated visits to the bus office.The proposed system utilizes a web application to address these challenges by digitizing the application process. Applicants can submit their details, including necessary documents, through an online form. The application status can be tracked in real time, and the system sends automated notifications regarding approvals or rejections.

**CHAPTER 1**

**INTRODUCTION**

**1.1 Overview**

In many urban cities, the management of public transport systems, particularly bus passes, has been an ongoing challenge. With increasing populations and higher demand for public transportation, traditional methods of issuing bus passes, which relied heavily on paper forms and physical submission, became inefficient and cumbersome. This inefficiency led to long wait times, human errors, delays in approvals, and a lack of real-time updates for users. The growing need for a more streamlined, transparent, and efficient system has driven the shift toward digital solutions. A web-based bus pass management system eliminates these inefficiencies by automating the entire process of application, approval, and renewal. It provides an online platform where users can submit applications, check the status of their bus passes, and manage renewals without ever having to visit a physical office. For transport authorities, it simplifies the approval process, ensures quick verification of user data, and centralizes all records for easy access and management. The web-based system is designed to provide real-time updates to users, allowing them to track their application status and receive notifications regarding approvals or rejections.

Such a system empowers both the users and administrators by making the process more transparent and user-friendly. Additionally, it reduces the workload on administrative staff, minimizes human errors, and improves the overall user experience by offering an easy-to-navigate, digital solution. The integration of Django and React for the development of this system ensures that the platform is both robust and scalable, capable of handling increasing traffic and providing a seamless experience for users.

**1.2 Research Motivation**

The motivation for developing a web-based bus pass management system stems from the challenges that traditional, manual systems pose to both transport authorities and commuters. Public transportation plays a vital role in modern urban life, and any inefficiency in its management directly affects the convenience and accessibility for users. Historically, the manual process of applying for, renewing, and managing bus passes was slow, opaque, and error-prone.

The key motivation for the research is to create a system that streamlines these processes and provides a more efficient, reliable, and transparent solution. The need to reduce human intervention in the process is critical to minimizing errors and delays. Furthermore, offering an online platform where users can directly interact with the system without the need to visit transport offices helps in saving time and resources. The research is driven by the objective of enhancing the user experience and simplifying the work of administrators, making it easier to track applications and manage renewals. The technological shift towards web applications opens up new opportunities for automating these manual processes, improving the accuracy of the system, and ensuring real-time accessibility for all stakeholders.

**1.3 Problem Definition**

Before the introduction of software applications for bus pass management, the entire process was handled manually, which led to several problems. One of the most prominent issues was the inefficiency of handling a large volume of applications. With no digital platform, each applicant had to physically submit forms, wait in long queues, and repeatedly visit offices for status updates. The paper-based system also posed a risk of data loss or misplacement, making it difficult to track applicants or provide real-time updates.

**1.4 Significance**

The significance of transitioning from a manual to a digital bus pass management system is clear. By replacing traditional paper-based methods, the digital system enhances efficiency, saves time, and reduces human errors. Transport authorities can process a much larger volume of applications without the need for physical paperwork, which leads to faster approvals and renewals. The elimination of physical forms also reduces administrative costs, making the entire process more cost-effective.

For users, the digital system offers a user-friendly interface that allows them to apply for and manage their bus passes online. This ensures accessibility and convenience, as users can access the system at any time and from anywhere. Real-time status updates and notifications ensure that users are always informed about the progress of their applications. The digitalization of the process also improves transparency, as both users and administrators can track and verify the status of each application instantly.

Moreover, the ability to track and manage bus passes digitally allows transport authorities to optimize their operations, such as identifying trends in demand and addressing bottlenecks. This results in a smoother and more reliable public transport system, benefiting both the authorities and the commuters. The integration of Django and React in the development ensures scalability and robustness, making the system capable of handling increasing traffic and evolving user needs.

**1.5 Applications**

1. **Simplified Application Process**: Users can apply for bus passes directly through the web platform, eliminating the need for physical forms and visits to offices.
2. **Real-Time Status Updates**: Applicants can track the progress of their bus pass application at any time, receiving updates on approval or rejection.
3. **Automated Renewals**: The system can automatically send reminders to users for bus pass renewals, making the process seamless and reducing manual intervention.
4. **Centralized Data Management**: Transport authorities can manage all bus pass applications in a single, centralized database, improving efficiency and reducing errors.
5. **Time and Cost Savings**: The system reduces the administrative burden on staff and the time spent by users waiting for approvals or updates.
6. **Transparency and Accessibility**: The digital platform provides transparency to both users and administrators, ensuring everyone is informed in real time.

**CHAPTER 2**

**LITERATURE SURVEY**

Li, Cheng, et al. [1] suggested an RFID-based attendance system based on the use of passive tags with the purpose of automating the attendance-tracking process in schools. The design they present illustrates how passive RFID tags to which each student is allocated can be read from RFID readers mounted at entrance points in the classroom. The data gathered is automatically recorded and maintained by a backend database, removing human error and administrative time. The research defines system performance, cost-effectiveness, and efficiency when compared to manual attendance systems as a key concept for transport systems where authentication of the user is vital.

Wang, Hua, et al. [2] specifically dealt with the design and implementation of an RFID-based metro pass system. It enables passengers to utilize RFID cards for convenient and contactless travel across metro stations. The structure entails central monitoring, fare deduction systems, and real-time logging of data. Their use redresses problems such as slow queues, fraudulent ticketing, and delayed manual processing, highlighting how RFID systems make public transport more efficient and convenient for commuters, a principle that can be extended to bus pass systems.

Chen, Wei, et al. [3] investigated the design of an RFID-based bus pass system and outlined its implementation and performance analysis. It uses RFID tags for student or commuter identification and fare management. It enables users to swipe RFID cards at bus entry points, which lowers boarding time considerably. Performance analysis indicated high reliability and rapid data processing, demonstrating its feasibility in large-scale implementations such as city-wide transportation services.

Liu, Shuai, et al. [4] presented an RFID-based real-time locating system for tracking assets in hospital settings. The focus of their research was on real-time tracking and automated data capture. While the target application was healthcare, the same paradigm of real-time monitoring of data and automation can be applied to transit systems where monitoring pass usage of buses and user behavior is needed for optimization and control.

Xu, Xiaofeng, et al. [5] developed an RFID-based library management system that facilitates automatic check-in and check-out for books. The system eliminates the use of conventional barcode systems and offers much faster processing and accuracy in inventory information. This research justifies using RFID in organized systems where user authentication and real-time logging of transactions are necessary, similar to being used in public transport systems where authenticating user passes and tracking usage is essential.

Wang, Lei, et al. [6] used an RFID-based highway automatic toll collection system to minimize delays in manual collection and improve traffic flow. In this study, they identified secure data exchange, quick tag reading, and vehicle classification, all controlled by a central control platform. The automated fare processing and monitoring concept has direct applicability to RFID-based bus pass systems with the same traffic management requirements.

Sharma, Rajesh, et al. [7] have presented an extensive overview of RFID-based supply chain management systems emphasizing automation, tracking, and data integrity. Their evidence indicates the significance of RFID in managing flow, inventory, and user actions among different nodes. These aspects are vital in transit systems that have to support high numbers of users and dynamic operating flows efficiently.

Zhang, Ming, et al. [8] used RFID for wildlife conservation by implementing an animal tracking system. Although the application is different, the principles of real-time location tracking, user-specific data logging, and remote access to the records apply to the requirements of transport authorities to monitor bus usage, user history, and route planning optimization.

Lee, Seung-Hwan, et al. [9] proposed an RFID-based access control system for securing buildings and showed how RFID tags can be employed to control and record entry and exit activity. The secure and scalable approach is applicable in the case of controlling access to public transport systems, where preventing unauthorized access and tracking usage in real-time are paramount.

Chen, Jian, et al. [10] created an RFID-based asset management system for a manufacturing context. Their work illustrates the benefits of centralized storage of data and automatic tag reading in optimizing tracking efficiency, minimizing loss, and assisting with operational planning. This architecture has strong affinity with public transport systems looking to streamline the management of users, passes issued, and service tracking.

Yang, Dong, et al. [11] summarized RFID applications in intelligent agriculture, specifying its application in tracking, monitoring, and automation of field operations. Though the sector is agriculture, the fundamental concepts of automated data capture, system intelligence, and operational efficacy can be directly applied to RFID-based bus pass systems that involve user tracking and service optimization.

Park, Min-Jeong, et al. [12] proposed an RFID-based airport passenger identification system for security improvement and passenger flow simplification. Their system identifies users, records entry points, and facilitates centralized monitoring. Such capabilities are critical for public transport systems aiming to provide safe access, reduce fraud, and keep up-to-date travel records.

**CHAPTER 3**

**EXISTING SYSTEM**

**3.1 Manual Application Process**

In traditional bus pass management systems, applicants were required to physically visit the office to submit their applications. The entire process involved filling out physical forms, attaching necessary documents, and waiting in long queues for manual verification. This system often led to delays due to manual data entry and the sheer volume of paperwork that needed to be processed by administrative staff. Furthermore, applicants had to frequently follow up with authorities to check the status of their applications, which was not always timely or accurate. The lack of a digital interface meant that applicants had no way of tracking their application status online, resulting in frequent frustration. Additionally, the process was prone to human error, including misplacing applications or incorrect data entry, which further complicated the approval or renewal process.

**3.2 Paper-Based Record Keeping**

In the traditional system, all records, including applications, approvals, and renewals, were maintained on paper. This method required significant physical storage space, making it difficult for administrators to access or retrieve specific records quickly. Over time, the sheer volume of paperwork became overwhelming, and searching for a particular document became a time-consuming task. Moreover, physical records were highly susceptible to damage, loss, or misplacement. Without a centralized digital database, there was no easy way to backup the data, leaving it vulnerable to incidents like natural disasters or administrative errors. The lack of an organized system also made it challenging to maintain consistency and accuracy across records, often leading to duplication or incomplete information.

**3.3 Manual Approval and Renewal System**

The approval and renewal process in traditional systems involved significant manual intervention. Once an application was submitted, it would be reviewed by administrative staff, who manually verified the information provided. This process was both time-consuming and labor-intensive, as it involved checking documents, verifying applicant details, and often requiring multiple rounds of communication with applicants. Renewal processes were similarly cumbersome; applicants had to re-submit physical forms and documents each time their bus pass expired. Without a system for automatic reminders, many applicants missed renewal deadlines, leading to disruptions in their access to public transportation. Additionally, since each renewal required manual processing, it further increased the workload on administrative staff and delayed the overall process.

**3.4 Lack of Real-Time Information**

Traditional bus pass systems often lacked the ability to provide real-time information to both the applicants and the transport authorities. Applicants had to rely on periodic updates or phone calls to inquire about the status of their application or renewal, leading to uncertainty and anxiety. Since there was no centralized digital platform, administrative staff struggled to maintain consistent communication with applicants, often resulting in delayed responses. Moreover, the lack of real-time data meant that transport authorities could not efficiently track the demand for bus passes or identify any trends in application patterns. This inefficiency not only caused delays in processing applications but also made it difficult for authorities to plan for future needs, leading to resource mismanagement.

**3.5 Limited Accessibility and Convenience**

Traditional bus pass management systems were highly reliant on physical presence, making them inconvenient for many users. Applicants had to visit the transport office in person during specific working hours, which often conflicted with their own schedules. Additionally, people living in distant areas had to travel long distances to submit their applications, creating unnecessary hardship. The lack of an online platform meant that users could not access or update their information at their convenience. This inconvenience disproportionately affected those with time constraints, such as working professionals or students, further contributing to the inefficiency of the system.

**Overall Drawbacks of the Traditional System**

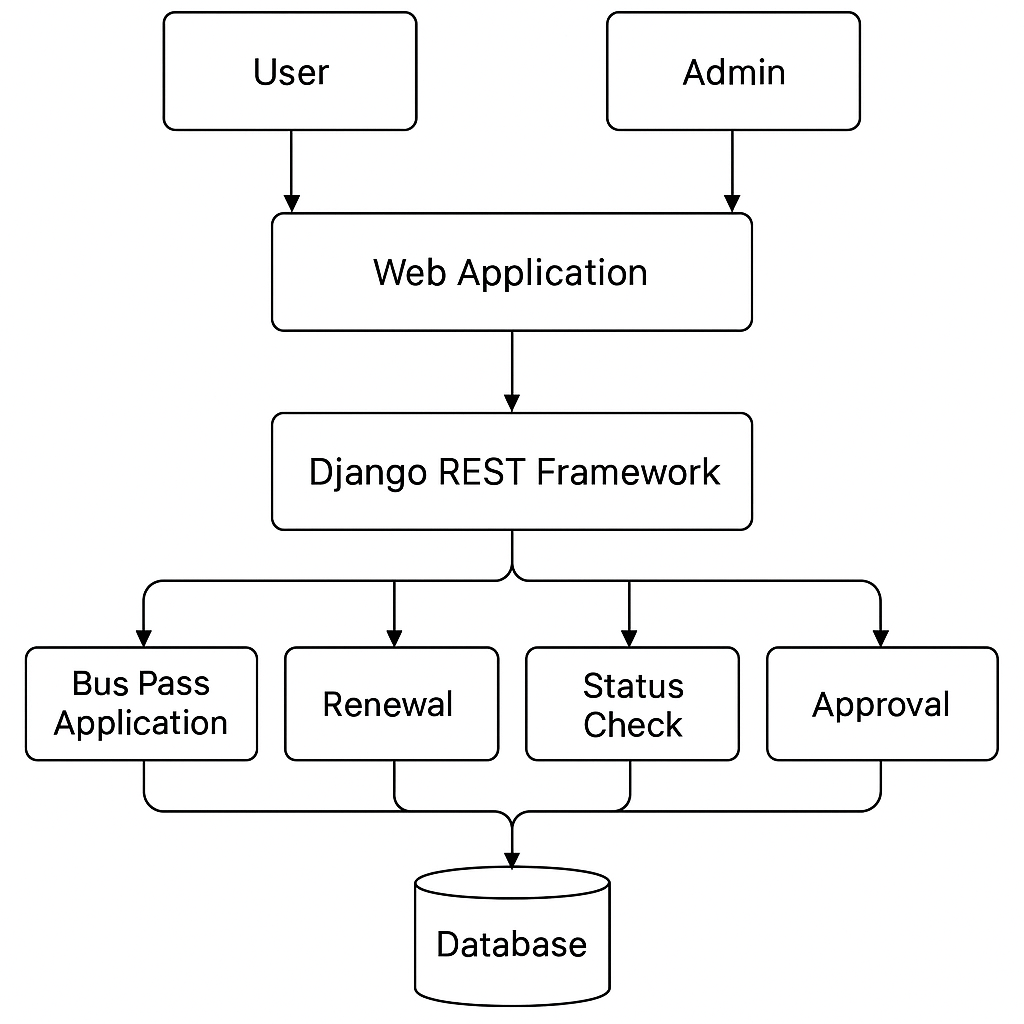
1. **Inefficiency**: The manual process of submitting applications and renewals resulted in long waiting times, causing significant delays in processing.
2. **Increased Administrative Burden**: With paper-based records and manual data entry, administrative staff had to manage a large volume of paperwork, leading to overload and errors.
3. **Lack of Transparency**: Applicants had no visibility into the status of their applications, creating frustration and uncertainty.
4. **Human Error**: Manual data entry and record-keeping introduced the risk of mistakes, such as misplacing applications or entering incorrect details.
5. **Limited Access**: The traditional system required physical visits to the office, restricting accessibility for applicants, particularly those with time constraints or residing in remote areas.
6. **Vulnerability to Data Loss**: Paper records were susceptible to damage, loss, or misplacement, and there was no easy way to backup or restore data.

**CHAPTER 4**

**PROPOSED SYSTEM**

**4.1 Overview**

The proposed system aims to revolutionize the traditional bus pass management process by introducing a comprehensive web-based platform built using Django and React. The new system eliminates the need for physical paperwork, manual data entry, and in-person visits to the transport office. Through a seamless online interface, applicants can register, submit required documents, apply for new bus passes, and renew existing ones without any administrative delays. The platform offers real-time updates on application status, allowing users to track their progress efficiently. Administrative tasks such as approval, verification, and management of applications are streamlined through an automated backend system, reducing the workload on staff and minimizing human errors. Additionally, the system integrates secure user authentication, ensuring that only authorized individuals can access sensitive information. With enhanced accessibility, users can apply for or renew their bus passes at their convenience, reducing the burden on both applicants and administrative authorities. The solution is designed to be user-friendly, fast, and reliable, offering a significant improvement over traditional manual methods.



**4.2 Key Features and Functionalities**

1. **User Registration and Authentication**The system allows users to create accounts with secure authentication processes, ensuring that only registered individuals can access the platform. The user registration includes essential information such as personal details, contact information, and documents for bus pass application.
2. **Bus Pass Application**Users can easily apply for new bus passes by filling out online forms, uploading necessary documents (such as proof of identity and residency), and submitting the application electronically. The system automatically validates the entered data for completeness and correctness.
3. **Online Bus Pass Renewal**The system offers a streamlined process for renewing existing bus passes. Users are notified of upcoming expiry dates and can renew their passes with minimal effort, reducing the need for physical visits to the office.
4. **Document Upload and Verification**Applicants can upload supporting documents for verification through the platform. The system allows the admin to review the documents, verify their authenticity, and approve or reject the application based on the provided data.
5. **Application Status Tracking**Once an application is submitted, users can track its progress in real time. Notifications are sent to users as their applications move through different stages, such as "Approved," or "Rejected."
6. **Admin Dashboard for Application Management**The system features an intuitive dashboard for administrators to manage all bus pass applications. Admins can view, approve, or reject applications, check document validity, and ensure the timely processing of requests.

Django REST Framework (DRF) plays a crucial role in developing the bus pass management system by providing a robust, flexible, and easy-to-use framework for building the backend API. DRF facilitates the creation of RESTful web services that can be consumed by the frontend (React) or any other client application. The following outlines how DRF is utilized in the development of the bus pass management system:

1. **API Design and Endpoints**DRF is used to design RESTful APIs that handle various operations in the system, such as user registration, login, bus pass application, status updates, and payment processing. These APIs are structured around resources, with each resource representing an entity like a user, application, or payment. For example, /api/register/, /api/apply\_pass/, and /api/payment\_status/ are examples of endpoints that users interact with. Each of these endpoints handles specific requests and returns data in a structured JSON format.
2. **Serialization of Data**DRF provides serializers that convert complex data types (like Django models) into JSON, which can be easily sent over the web to the client. For instance, the bus pass application and user registration data are serialized using DRF serializers. These serializers ensure that data is properly validated and structured before being sent to the frontend or database. For example, UserSerializer and BusPassSerializer are used to convert user and bus pass application data into JSON.
3. **Authentication and Permissions** DRF facilitates secure authentication methods, ensuring that only authorized users can access certain endpoints. The system uses token-based authentication where users must provide a valid token (received after logging in) to access specific resources. Additionally, DRF’s permission classes ensure that only users with the appropriate roles can perform certain actions. For example, only an admin user can approve or reject bus pass applications, while regular users can only view their own application status.
4. **Handling CRUD Operations**The bus pass management system requires the handling of Create, Read, Update, and Delete (CRUD) operations. DRF simplifies this by using APIView or ViewSets, which abstract away much of the boilerplate code for these operations. For example, when a user submits a bus pass application, the system creates a new entry in the database using the create() method. Similarly, users can view their application status using the retrieve() method, and admins can update or delete applications as necessary using the update() and destroy() methods.
5. **Validation and Error Handling** DRF provides built-in support for data validation, ensuring that all inputs meet the required format and constraints. For instance, when a user submits their application form, the system checks that the provided documents are valid and that all required fields are filled. If there is an error or invalid data, DRF automatically handles the error and returns appropriate HTTP status codes (e.g., 400 for bad requests) along with error messages in JSON format. This helps the frontend handle errors and display them to users in a user-friendly way.
6. **File Uploads** Since bus pass applications often require document uploads (e.g., proof of identity), DRF is used to handle file uploads. Using FileUploadParser, DRF facilitates the uploading of files through API endpoints. Once the files are uploaded, they are saved to the server or cloud storage, and their paths are stored in the database for later retrieval.
7. **Pagination and Filtering** DRF allows for efficient data retrieval by supporting pagination and filtering. For example, when an admin needs to view all bus pass applications, the system can paginate the results to avoid overwhelming the UI with too many records at once. Additionally, filtering allows for sorting applications based on criteria such as application status, user, or date.
8. **Integration with Frontend (React)** The APIs developed using DRF seamlessly integrate with the frontend React application. React makes requests to the DRF API, retrieves the required data, and updates the user interface dynamically. The asynchronous nature of React’s HTTP requests ensures a smooth user experience, with data being updated in real-time without needing to reload the page. DRF and React work together to provide a responsive and interactive experience for both users and administrators.
9. **Security and Rate Limiting** To prevent abuse and ensure the security of the system, DRF provides built-in features for securing APIs, including token authentication, permissions, and rate limiting. The system is designed to handle high traffic, ensuring that API requests are limited to a reasonable rate to prevent overloading the server and maintain system stability.

**4.4 React Developments and Components**

React plays a pivotal role in the development of the bus pass management system’s frontend by providing an efficient and dynamic user interface. React’s component-based architecture enables the development of modular, reusable, and maintainable components. Here’s an overview of the key developments and components in the system:

1. **Component-Based Architecture** React’s component-based structure divides the user interface into smaller, self-contained components. Each component is responsible for rendering a specific part of the UI and managing its own state. This modular approach allows for better maintainability and reusability. For example, components like UserLogin, BusPassForm, ApplicationStatus, and AdminDashboard are created to manage specific UI elements, ensuring that the code is organized and easy to manage.
2. **State Management** React’s state management allows components to dynamically update the UI based on user interactions and data fetched from the backend (via the Django API). For instance, when a user submits a bus pass application, the BusPassForm component stores the application data in the component’s state and, once submitted, triggers a re-render of the UI to reflect the changes. State management also helps in handling forms, validations, and displaying messages (like success or error) based on the response from the API.
3. **React Hooks for Functional Components** React Hooks, such as useState and useEffect, are used to manage component states and side effects. The useState hook is used to store and manage data within components, while useEffect handles side effects like fetching data from the backend when a component is mounted or updated. For example, the useEffect hook is employed to fetch the current user’s bus pass application status from the backend when the ApplicationStatus component is rendered.
4. **Form Handling and Validation** React components are responsible for rendering and handling form inputs. These forms are essential for tasks like user registration, bus pass application, and payment submissions. The BusPassForm component, for instance, captures user input (such as name, email, and documents) and validates the data before submission. Form validation ensures that users provide valid information (e.g., required fields are filled out), and any errors are displayed on the form in real-time.
5. **API Integration and Data Fetching** React interacts with the Django REST APIs to retrieve and display data to users. Using JavaScript’s fetch API or libraries like Axios, React components send requests to the Django backend and handle the responses. For instance, the AdminDashboard component makes API calls to retrieve all bus pass applications, displaying them in a paginated table. Data fetched from the backend is stored in React state and rendered on the UI.
6. **Routing with React Router** React Router is used for navigating between different views in the bus pass management system. It enables the creation of a single-page application (SPA) where users can seamlessly navigate between pages like the home page, registration form, bus pass application form, and admin dashboard without reloading the entire page. React Router manages the browser’s history and allows for dynamic route matching based on URL paths.
7. **User Authentication and Session Management** React manages the frontend authentication flow by storing user session data, such as authentication tokens, in local storage or session storage. When a user logs in, React stores the authentication token returned by the Django backend and includes it in the headers of subsequent API requests. This ensures that only authenticated users can access protected routes, such as viewing or updating their bus pass application or accessing the admin dashboard.
8. **Component Communication** React components communicate with each other through the passing of props (data) and the use of context for more complex state management. For instance, the ApplicationStatus component might receive data as props from its parent component, UserDashboard, which holds the user’s personal information and application status. Context can be used to manage global state, such as the current logged-in user, and make this data accessible to any component in the application.
9. **UI Design and Responsiveness** The bus pass management system uses responsive design principles to ensure that the UI adapts to various screen sizes. This is achieved through CSS frameworks like Bootstrap or Material-UI, which provide pre-built components that are optimized for different devices. The React components are styled with CSS or CSS-in-JS libraries like styled-components, ensuring that the layout and design are consistent across all devices, from desktop to mobile.

**4.5 Working**

The working of the bus pass management system combines the power of Django for backend development with React for the frontend, resulting in a dynamic and efficient web application. The Django backend is responsible for handling the core logic, including user authentication, bus pass application management, and API communication. Django REST Framework is utilized to expose API endpoints that allow the React frontend to interact with the system. The API handles requests such as user registration, bus pass application submission, status updates, and administrative actions like approving or rejecting applications.

On the frontend, React components are used to create an intuitive, interactive user interface. React’s state management system handles dynamic updates of data such as application statuses or user details without the need for page reloads. Each form submission, whether for user registration or bus pass application, triggers API calls to the Django backend, which processes the data and responds with appropriate results, such as confirmation of successful application or error messages. React components like ApplicationStatus, BusPassForm, and AdminDashboard manage the interaction between the user and the system, ensuring that users can easily apply for bus passes, track the progress of their applications, and administrators can approve or reject applications. The React frontend also handles user authentication by storing session tokens after login, enabling secure access to protected routes. The data fetched from the Django backend is dynamically rendered in React, ensuring that users have real-time visibility of their application status. The entire system is designed to be responsive, providing a seamless experience across different devices. By integrating Django and React, the system effectively bridges the gap between the user interface and the backend, providing an efficient, scalable, and user-friendly bus pass management solution.

**CHAPTER 5**

**UML DIAGRAMS**

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group. The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

**GOALS:** The Primary goals in the design of the UML are as follows:

· Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.

· Provide extendibility and specialization mechanisms to extend the core concepts.

· Be independent of particular programming languages and development process.

· Provide a formal basis for understanding the modeling language.

· Encourage the growth of the OO tools market.

· Support higher level development concepts such as collaborations, frameworks, patterns and components.

· Integrate best practices.

**Class diagram**

The class diagram is used to refine the use case diagram and define a detailed design of the system. The class diagram classifies the actors defined in the use case diagram into a set of interrelated classes. The relationship or association between the classes can be either an "is-a" or "has-a" relationship. Each class in the class diagram was capable of providing certain functionalities. These functionalities provided by the class are termed "methods" of the class. Apart from this, each class may have certain "attributes" that uniquely identify the class.

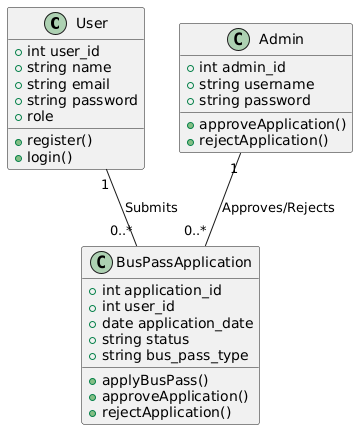


Figure-5.1: Class Diagram

**Sequence Diagram**

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows, as parallel vertical lines (“lifelines”), different processes or objects that live simultaneously, and as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

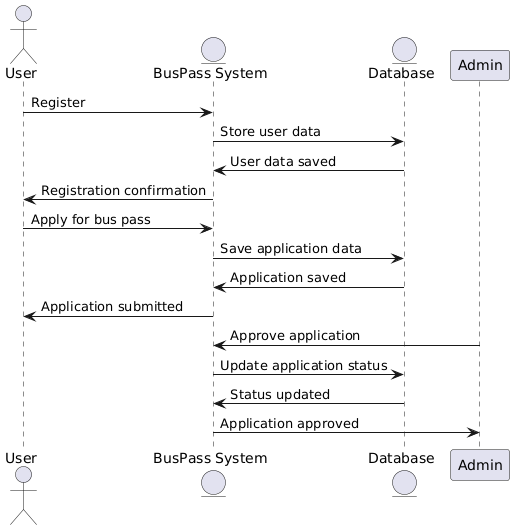


Figure-5.2: Sequence Diagram

**Activity diagram**

Activity diagrams are graphical representations of Workflows of stepwise activities and actions with support for choice, iteration, and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

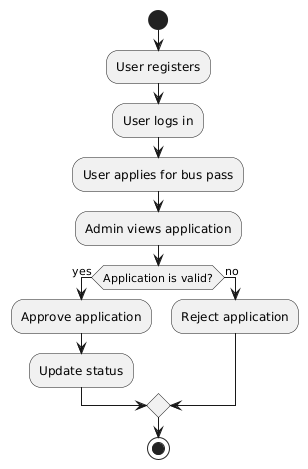


Figure-5.3: Activity Diagram

**Data flow diagram**

A data flow diagram (DFD) is a graphical representation of how data moves within an information system. It is a modeling technique used in system analysis and design to illustrate the flow of data between various processes, data stores, data sources, and data destinations within a system or between systems. Data flow diagrams are often used to depict the structure and behavior of a system, emphasizing the flow of data and the transformations it undergoes as it moves through the system.

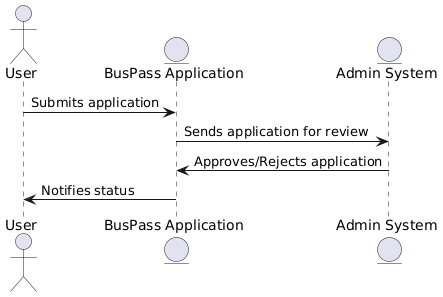


Figure-5.4: Data Flow Diagram

**Component diagram:** Component diagram describes the organization and wiring of the physical components in a system.

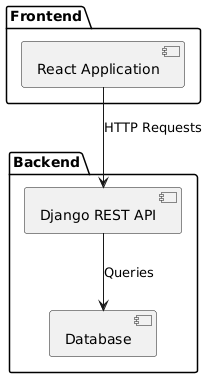


Figure-5.5: Component Diagram

**Use Case diagram:** A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

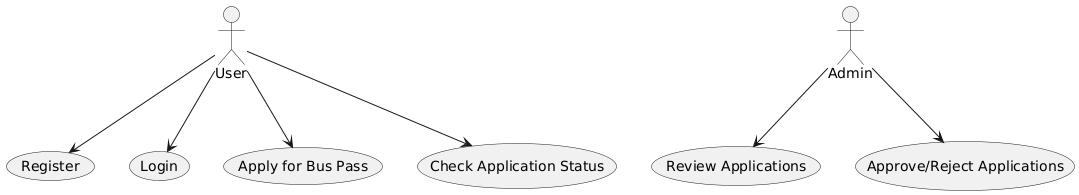


Figure-5.6: Use Case Diagram

**Deployment Diagram**

The Deployment Diagram illustrates the physical architecture of the online cooking classes system, showing how software components are deployed on hardware. It includes the user device, which interacts with the system through a browser, the web server hosting the Django application, and the database server running SQLite. Interactions flow as the user's browser sends HTTP requests to the web server, which processes them, interacts with the SQLite database for data operations, and sends back appropriate responses to the user's browser.

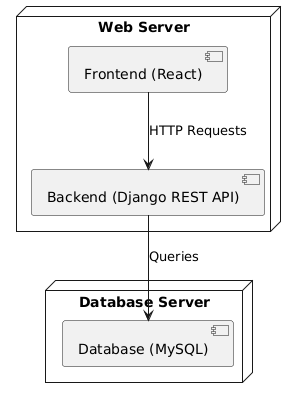


Figure-5.7: Deployment Diagram

**Architectural Block Diagram**

The Architectural Block Diagram complements this by providing a high-level overview of the system's structure, highlighting the client (browser) interfacing with the server (Django app) and the database (SQLite). The browser communicates with the Django app via HTTP requests for various user actions, while the Django app manages business logic and interacts with the SQLite database for data management, ensuring a seamless user experience across the system.

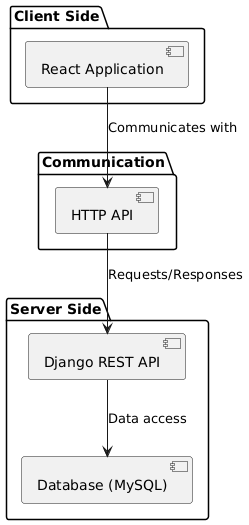


Figure-5.8: Architectural Diagram

**CHAPTER 6**

**SOFTWARE ENVIRONMENT**

**What is Python?**

Below are some facts about Python.

· Python is currently the most widely used multi-purpose, high-level programming language.

· Python allows programming in Object-Oriented and Procedural paradigms. Python programs generally are smaller than other programming languages like Java.

· Programmers have to type relatively less and indentation requirement of the language, makes them readable all the time.

· Python language is being used by almost all tech-giant companies like – Google, Amazon, Facebook, Instagram, Dropbox, Uber… etc.

The biggest strength of Python is huge collection of standard library which can be used for the following –

· Machine Learning

· GUI Applications (like Kivy, Tkinter, PyQt etc. )

· Web frameworks like Django (used by YouTube, Instagram, Dropbox)

· Image processing (like Opencv, Pillow)

· Web scraping (like Scrapy, BeautifulSoup, Selenium)

· Test frameworks

· Multimedia

**Advantages of Python**

Let’s see how Python dominates over other languages.

**1.Extensive Libraries**

Python downloads with an extensive library and it contain code for various purposes like regular expressions, documentation-generation, unit-testing, web browsers, threading, databases, CGI, email, image manipulation, and more. So, we don’t have to write the complete code for that manually.

**2. Extensible**

As we have seen earlier, Python can be extended to other languages. You can write some of your code in languages like C++ or C. This comes in handy, especially in projects.

**3. Embeddable**

Complimentary to extensibility, Python is embeddable as well. You can put your Python code in your source code of a different language, like C++. This lets us add scripting capabilities to our code in the other language.

**4. Improved Productivity**

The language’s simplicity and extensive libraries render programmers more productive than languages like Java and C++ do. Also, the fact that you need to write less and get more things done.

**5. IOT Opportunities**

Since Python forms the basis of new platforms like Raspberry Pi, it finds the future bright for the Internet Of Things. This is a way to connect the language with the real world.

**6. Simple and Easy**

When working with Java, you may have to create a class to print ‘Hello World’. But in Python, just a print statement will do. It is also quite easy to learn, understand, and code. This is why when people pick up Python, they have a hard time adjusting to other more verbose languages like Java.

**7. Readable**

Because it is not such a verbose language, reading Python is much like reading English. This is the reason why it is so easy to learn, understand, and code. It also does not need curly braces to define blocks, and indentation is mandatory. This further aids the readability of the code.

**8. Object-Oriented**

This language supports both the procedural and object-oriented programming paradigms. While functions help us with code reusability, classes and objects let us model the real world. A class allows the encapsulation of data and functions into one.

**9. Free and Open-Source**

Like we said earlier, Python is freely available. But not only can you download Python for free, but you can also download its source code, make changes to it, and even distribute it. It downloads with an extensive collection of libraries to help you with your tasks.

**10. Portable**

When you code your project in a language like C++, you may need to make some changes to it if you want to run it on another platform. But it isn’t the same with Python. Here, you need to code only once, and you can run it anywhere. This is called Write Once Run Anywhere (WORA). However, you need to be careful enough not to include any system-dependent features.

**11. Interpreted**

Lastly, we will say that it is an interpreted language. Since statements are executed one by one, debugging is easier than in compiled languages.

Any doubts till now on the advantages of Python? Mention in the comment section.

**Advantages of Python Over Other Languages**

10. **Less Coding**

Almost all of the tasks done in Python requires less coding when the same task is done in other languages. Python also has an awesome standard library support, so you don’t have to search for any third-party libraries to get your job done. This is the reason that many people suggest learning Python to beginners.

**2. Affordable**

Python is free therefore individuals, small companies or big organizations can leverage the free available resources to build applications. Python is popular and widely used so it gives you better community support.

The 2019 Github annual survey showed us that Python has overtaken Java in the most popular programming language category.

**3. Python is for Everyone**

Python code can run on any machine whether it is Linux, Mac or Windows. Programmers need to learn different languages for different jobs but with Python, you can professionally build web apps, perform data analysis and machine learning, automate things, do web scraping and also build games and powerful visualizations. It is an all-rounder programming language.

**Disadvantages of Python**

So far, we’ve seen why Python is a great choice for your project. But if you choose it, you should be aware of its consequences as well. Let’s now see the downsides of choosing Python over another language.

10. **Speed Limitations**

We have seen that Python code is executed line by line. But since Python is interpreted, it often results in slow execution. This, however, isn’t a problem unless speed is a focal point for the project. In other words, unless high speed is a requirement, the benefits offered by Python are enough to distract us from its speed limitations.

**2. Weak in Mobile Computing and Browsers**

While it serves as an excellent server-side language, Python is much rarely seen on the client-side. Besides that, it is rarely ever used to implement smartphone-based applications. One such application is called Carbonnelle.

The reason it is not so famous despite the existence of Brython is that it isn’t that secure.

**3. Design Restrictions**

As you know, Python is dynamically typed. This means that you don’t need to declare the type of variable while writing the code. It uses duck-typing. But wait, what’s that? Well, it just means that if it looks like a duck, it must be a duck. While this is easy on the programmers during coding, it can raise run-time errors.

**4. Underdeveloped Database Access Layers**

Compared to more widely used technologies like JDBC (Java DataBase Connectivity) and ODBC (Open DataBase Connectivity), Python’s database access layers are a bit underdeveloped. Consequently, it is less often applied in huge enterprises.

**5. Simple**

No, we’re not kidding. Python’s simplicity can indeed be a problem. Take my example. I don’t do Java, I’m more of a Python person. To me, its syntax is so simple that the verbosity of Java code seems unnecessary.

This was all about the Advantages and Disadvantages of Python Programming Language.

**Modules**

### **1. Django Rest Framework**

* **Description**: djangorestframework (DRF) is a powerful and flexible toolkit for building Web APIs in Django. It provides a set of tools for creating RESTful APIs, including serializers, viewsets, routers, and authentication mechanisms. DRF simplifies the process of converting Django models into JSON and handling HTTP requests.  
   **Key Features**:
  + **Serialization**: Convert complex Django models and querysets into JSON format.
  + **Viewsets**: Create views for CRUD operations (Create, Read, Update, Delete) with minimal code.
  + **Authentication & Permissions**: Provides out-of-the-box authentication schemes (e.g., token-based, session-based) and customizable permissions.
  + **Pagination**: Automatically handles pagination for large sets of data.
  + **Browsable API**: It has a built-in web interface for interacting with your API, making it easy to test API endpoints.

### **2. Django-cors-headers**

* **Description**: django-cors-headers is a Django app that allows Cross-Origin Resource Sharing (CORS) headers to be added to responses. CORS is a mechanism that allows a web application to request resources from a different domain. This is often used when you have a frontend hosted on a different domain than the Django backend (for example, React app calling a Django API).  
   **Key Features**:
  + **Add CORS headers**: Automatically add CORS headers to HTTP responses based on configurable rules.
  + **Customizable**: You can configure which domains are allowed to access the API, specify which HTTP methods (GET, POST, etc.) are allowed, and manage credentials (cookies, authorization headers).
  + **Support for OPTIONS requests**: Preflight OPTIONS requests can be handled automatically.

### **3. Django rest framework-simple jwt**

* **Description**: djangorestframework-simplejwt is an authentication library for Django REST Framework that provides JSON Web Token (JWT) authentication. JWT is a compact and self-contained way to represent claims between two parties. This package allows you to secure your API with token-based authentication, where the token is sent with each request to authenticate the user.  
   **Key Features**:
  + **Token-based Authentication**: Users authenticate once and receive a JWT, which can be used for subsequent requests.
  + **Support for Token Refresh**: It supports both access and refresh tokens, ensuring that users can keep their session active without needing to log in repeatedly.
  + **Customizable Claims**: You can include custom claims in the JWT, like user roles or other metadata.
  + **Easy to Integrate**: Simple integration with DRF’s authentication system, and you can configure token expiration and other settings.

**Install Python Step-by-Step in Windows and Mac**

Python, a versatile programming language, doesn't come pre-installed on your computer devices. Python was first released in the year 1991 and until today it is a very popular high-level programming language. Its style philosophy emphasizes code readability with its notable use of great whitespace.

The object-oriented approach and language construct provided by Python enables programmers to write both clear and logical code for projects. This software does not come pre-packaged with Windows.

**How to Install Python on Windows and Mac**

There have been several updates in the Python version over the years. The question is how to install Python? It might be confusing for the beginner who is willing to start learning Python but this tutorial will solve your query. The latest or the newest version of Python is version 3.7.4 or in other words, it is Python 3.

Note: The python version 3.7.4 cannot be used on Windows XP or earlier devices.

Before you start with the installation process of Python. First, you need to know about your System Requirements. Based on your system type i.e. operating system and based processor, you must download the python version. My system type is a Windows 64-bit operating system. So the steps below are to install python version 3.7.4 on Windows 7 device or to install Python 3. Download the Python Cheatsheet here.The steps on how to install Python on Windows 10, 8 and 7 are divided into 4 parts to help understand better.

**Download the Correct version into the system**

Step 1: Go to the official site to download and install python using Google Chrome or any other web browser. OR Click on the following link: [https://www](https://www/).python.org

Now, check for the latest and the correct version for your operating system.

Step 2: Click on the Download Tab.

Step 3: You can either select the Download Python for windows 3.7.4 button in Yellow Color or you can scroll further down and click on download with respective to their version. Here, we are downloading the most recent python version for windows 3.7.4

Step 4: Scroll down the page until you find the Files option.

Step 5: Here you see a different version of python along with the operating system.

· To download Windows 32-bit python, you can select any one from the three options: Windows x86 embeddable zip file, Windows x86 executable installer or Windows x86 web-based installer.

· To download Windows 64-bit python, you can select any one from the three options: Windows x86-64 embeddable zip file, Windows x86-64 executable installer or Windows x86-64 web-based installer.

Here we will install Windows x86-64 web-based installer. Here your first part regarding which version of python is to be downloaded is completed. Now we move ahead with the second part in installing python i.e. Installation

Note: To know the changes or updates that are made in the version you can click on the Release Note Option.

**Installation of Python**

Step 1: Go to Download and Open the downloaded python version to carry out the installation process.

Step 2: Before you click on Install Now, Make sure to put a tick on Add Python 3.7 to PATH.

Step 3: Click on Install NOW After the installation is successful. Click on Close.

With these above three steps on python installation, you have successfully and correctly installed Python. Now is the time to verify the installation.

Note: The installation process might take a couple of minutes.

**Verify the Python Installation**

Step 1: Click on Start

Step 2: In the Windows Run Command, type “cmd”.

Step 3: Open the Command prompt option.

Step 4: Let us test whether the python is correctly installed. Type python –V and press Enter.

Step 5: You will get the answer as 3.7.4

Note: If you have any of the earlier versions of Python already installed. You must first uninstall the earlier version and then install the new one.

**Check how the Python IDLE works**

Step 1: Click on Start

Step 2: In the Windows Run command, type “python idle”.

Step 3: Click on IDLE (Python 3.7 64-bit) and launch the program

Step 4: To go ahead with working in IDLE you must first save the file. Click on File > Click on Save

Step 5: Name the file and save as type should be Python files. Click on SAVE. Here I have named the files as Hey World.

Step 6: Now for e.g. enter print (“Hey World”) and Press Enter.

You will see that the command given is launched. With this, we end our tutorial on how to install Python. You have learned how to download python for windows into your respective operating system.

Note: Unlike Java, Python does not need semicolons at the end of the statements otherwise it won’t work.

**How to Install Node.js on Windows and Mac**

**Node.js** is an open-source, cross-platform runtime environment for executing JavaScript code outside of a browser. It allows you to run JavaScript on the server side, which is typically used for building scalable network applications, including web servers, real-time applications, REST APIs, and more. Node.js uses the V8 JavaScript engine (the same engine used by Google Chrome) and is designed to be lightweight and efficient, especially for handling asynchronous I/O tasks.

#### **Key Features:**

* **Asynchronous & Event-Driven**: Node.js operates on a non-blocking, event-driven architecture, which makes it suitable for building real-time applications like chat applications or streaming services.
* **Single-Threaded**: It uses a single-threaded event loop, making it memory efficient while handling many connections concurrently.
* **Fast Execution**: Thanks to V8, Node.js is fast in executing JavaScript code.
* **Package Manager**: Node.js uses **npm** (Node Package Manager) for handling libraries and dependencies, allowing you to easily install third-party libraries and modules.

#### **Common Use Cases:**

* **Web Servers**: Node.js is frequently used to build web servers and APIs (often with frameworks like Express).
* **Real-Time Applications**: Applications like online gaming, chat systems, and live updates benefit from Node.js' real-time capabilities.
* **Microservices**: It is well-suited for building microservices-based architectures because of its lightweight, non-blocking I/O operations.

### **Installing Node.js**

You can install Node.js on Windows, macOS, and Linux. Here’s how:

#### **Windows or macOS Installation:**

1. **Download the Node.js Installer**:
   * Go to the official Node.js website:<https://nodejs.org>.
   * Download the installer that is appropriate for your operating system.
     + **LTS (Long Term Support)** version is recommended for most users.
     + **Current version** is the latest stable version with the newest features.
2. **Run the Installer**:
   * After the download completes, run the installer and follow the instructions. The installer will guide you through the installation process, including adding Node.js to your system’s PATH.
3. **Verify Installation**:
   * After installation, open your terminal (Command Prompt on Windows, Terminal on macOS), and type the following commands to verify the installation:

**CHAPTER 7**

**SYSTEM REQUIREMENTS**

**Software Requirements**

The functional requirements or the overall description documents include the product perspective and features, operating system and operating environment, graphics requirements, design constraints and user documentation.

The appropriation of requirements and implementation constraints gives the general overview of the project in regard to what the areas of strength and deficit are and how to tackle them.

· Python IDLE 3.7 version (or)

· Anaconda 3.7 (or)

· Jupiter (or)

· Google colab

**Hardware Requirements**

Minimum hardware requirements are very dependent on the particular software being developed by a given Enthought Python / Canopy / VS Code user. Applications that need to store large arrays/objects in memory will require more RAM, whereas applications that need to perform numerous calculations or tasks more quickly will require a faster processor.

· Operating system : Windows, Linux

· Processor : minimum intel i3

· Ram : minimum 4 GB

· Hard disk : minimum 250GB

**CHAPTER 8**

**FUNCTIONAL REQUIREMENTS**

**Frontend (React)**

1. **React Router Setup**: The app uses react-router-dom to handle routing. Different routes are defined for home, login, registration, admin login, and other related pages like bus pass applications, checking status, and viewing renewals.
2. **State Management**: The application maintains two state variables: isAuthenticated (which tracks if the user is logged in) and admin (which determines if the logged-in user is an admin). This state is persisted through localStorage.
3. **Logout Logic**: The Logout component handles logging out the user by clearing the localStorage and redirecting to the login page.
4. **Header Component**: The Header component is displayed across the app and shows different UI elements based on whether the user is authenticated and whether they are an admin.
5. **User Routes**: Various routes are set up for the users, such as:
   * **Login** and **Register** pages for user authentication and registration.
   * **BusPassForm** for applying for a bus pass.
   * **CheckStatus** to view the status of a bus pass.
   * **RenewalList** to view past bus pass renewals.
   * **PendingBusPasses** to show bus passes pending approval for admins.
   * **Logout** for logging out.

### **Backend (Django)**

1. **User Authentication and Token Management**:
   * **CustomTokenObtainPairSerializer**: A custom serializer to extend the JWT token generation. It adds additional user information (like id, username, email, etc.) to the JWT token response.
   * **UserModelViewSet**: This is a viewset for handling user-related API actions, such as creating users while ensuring no duplicate usernames exist.
   * **UserViewSet**: A viewset for fetching users, with specific logic to allow only admins to view non-staff users.
2. **Bus Pass Management**:
   * **Apply Bus Pass**: The apply\_bus\_pass view handles the process of applying for a bus pass. It checks if the user already has an active bus pass and prevents new applications if one is still valid.
   * **Check Bus Pass Status**: The check\_bus\_pass\_status view returns the status and expiry date of the latest approved bus pass for the authenticated user.
   * **Renew Bus Pass**: This view allows users to renew their bus passes, ensuring that the pass has expired before renewing and not allowing multiple renewals within an active period.
   * **Get Renewals**: It fetches the renewal requests for the authenticated user.
3. **Admin Controls**:
   * **Get Pending Bus Passes**: The get\_pending\_bus\_passes view lists all bus passes that are pending approval, intended for admin use.
   * **Approve Bus Pass**: The approve\_bus\_pass view allows an admin to approve a pending bus pass application.
4. **User Registration**: The register\_user view handles user registration by validating that the username and email are unique and creating a new user in the database.

### **URL Routing and API Endpoints**

* The backend defines various API endpoints for managing users, bus pass applications, renewals, and approvals:  
  + **JWT Token Routes**: api/token/ for getting a JWT token, api/token/refresh/ for refreshing the token.
  + **User Management**: Includes API routes for registering users and managing bus pass applications and renewals.
  + **Admin Endpoints**: Admins can approve bus passes and view pending applications via specific routes.

**CHAPTER 9**

**SOURCE CODE**

**DJANGO**

from django.contrib import admin

from django.urls import path,include

from rest\_framework\_simplejwt.views import TokenObtainPairView,TokenRefreshView

# main url routes developments

from application.views import CustomTokenObtainPairView

from rest\_framework.routers import DefaultRouter

from rest\_framework.routers import DefaultRouter

from application.views import UserViewSet

from application import views

router = DefaultRouter()

router.register(r'users', UserViewSet)

urlpatterns = [

path('admin/', admin.site.urls),

path('api/token/',CustomTokenObtainPairView.as\_view(),name='get\_token'),

path('api/token/refresh/',TokenRefreshView.as\_view(),name='refresh\_token'),

path('api/', include(router.urls)),

path('api/apply-bus-pass/',views.apply\_bus\_pass, name='apply\_bus\_pass'),

path('api/register/',views.register\_user, name='register\_user'),

path("api/renew-bus-pass/",views.renew\_bus\_pass, name="renew-bus-pass"),

path('api/bus-pass/status/',views.check\_bus\_pass\_status),

path('api/renewals/',views.get\_renewals),

path('api/pending-passes/', views.get\_pending\_bus\_passes, name='pending\_passes'),

path('api/approve-pass/<int:pass\_id>/', views.approve\_bus\_pass, name='approve\_pass'),

]

from django.shortcuts import render

# Create your views here.

from django.contrib.auth.models import User

from rest\_framework import viewsets

from rest\_framework.response import Response

from rest\_framework\_simplejwt.views import TokenObtainPairView,TokenRefreshView

from rest\_framework\_simplejwt.serializers import TokenObtainPairSerializer

from .serializers import UserSerializer

from .models import BusPass, RenewalRequest

from .serializers import UserSerializer,BusPassSerializer, RenewalRequestSerializer

from rest\_framework import generics, permissions

from rest\_framework.views import APIView

from rest\_framework.decorators import api\_view,permission\_classes,parser\_classes

class UserModelViewSet(viewsets.ModelViewSet):

queryset = User.objects.all()

serializer\_class = UserSerializer

permission\_classes = [permissions.AllowAny]

def create(self, request, \*args, \*\*kwargs):

username = request.data.get('username')

password = request.data.get('password')

email = request.data.get('email')

if User.objects.filter(username=username).exists():

return Response({'message': 'Username already exists'})

user = User.objects.create\_user(username=username, password=password, email=email)

user.save()

return Response({'message': 'User created successfully'})

class CustomTokenObtainPairSerializer(TokenObtainPairSerializer):

def validate(self, attrs):

data = super().validate(attrs)

user = self.user

data["user"] = {

"id": user.id,

"username": user.username,

"email": user.email,

"is\_staff": user.is\_staff,

"is\_superuser": user.is\_superuser,

}

return data

class CustomTokenObtainPairView(TokenObtainPairView):

serializer\_class = CustomTokenObtainPairSerializer

class UserViewSet(viewsets.ModelViewSet):

queryset = User.objects.all()

serializer\_class = UserSerializer

permission\_classes = [permissions.IsAuthenticated]

def get\_queryset(self):

if self.request.user.is\_staff:

return User.objects.filter(is\_staff=False,is\_superuser=False)

from datetime import timedelta

from django.utils import timezone

from rest\_framework.parsers import MultiPartParser, FormParser

@api\_view(['POST'])

@parser\_classes([MultiPartParser, FormParser])

@permission\_classes([permissions.IsAuthenticated])

def apply\_bus\_pass(request):

full\_name = request.data.get("full\_name")

date\_of\_birth = request.data.get("date\_of\_birth")

gender = request.data.get("gender")

email = request.data.get("email")

phone\_number = request.data.get("phone\_number")

address = request.data.get("address")

aadhaar\_number = request.data.get("aadhaar\_number")

pass\_type = request.data.get("pass\_type")

source\_location = request.data.get("source\_location")

destination\_location = request.data.get("destination\_location")

pass\_duration = request.data.get("pass\_duration")

address\_proof = request.FILES.get("address\_proof")

photo = request.FILES.get("photo")

current\_date = timezone.now().date()

last\_pass = BusPass.objects.filter(user=request.user, application\_status="Approved").order\_by('-applied\_date').first()

if last\_pass:

expiry\_date = last\_pass.applied\_date.date() + timedelta(days=last\_pass.pass\_duration \* 30)

if current\_date < expiry\_date:

return Response(

{"message": f"You already have an active bus pass valid until {expiry\_date.strftime('%Y-%m-%d')}"}

)

bus\_pass = BusPass.objects.create(

user=request.user,

full\_name=full\_name,

date\_of\_birth=date\_of\_birth,

gender=gender,

email=email,

phone\_number=phone\_number,

address=address,

aadhaar\_number=aadhaar\_number,

pass\_type=pass\_type,

source\_location=source\_location,

destination\_location=destination\_location,

pass\_duration=pass\_duration,

address\_proof=address\_proof,

photo=photo,

application\_status="Pending", # Default status

applied\_date=timezone.now(), # Automatically set application date

)

bus\_pass.save()

return Response({"message": "Application submitted successfully!"})

@api\_view(['GET'])

@permission\_classes([permissions.IsAuthenticated])

def check\_bus\_pass\_status(request):

user = request.user

bus\_pass = BusPass.objects.filter(user=user).order\_by('-applied\_date').first()

if not bus\_pass:

return Response({"message": "No active bus pass application found."}, status=404)

expiry\_date = bus\_pass.applied\_date + timedelta(days=bus\_pass.pass\_duration \* 30)

return Response({

"full\_name": bus\_pass.full\_name,

"pass\_type": bus\_pass.pass\_type,

"application\_status": bus\_pass.application\_status,

"applied\_date": bus\_pass.applied\_date,

"expiry\_date": expiry\_date,

})

@api\_view(['GET']) # ✅ Now it supports GET requests

@permission\_classes([permissions.IsAuthenticated])

def renew\_bus\_pass(request):

user = request.user

current\_date = timezone.now().date()

bus\_pass = BusPass.objects.filter(user=user, application\_status="Approved").order\_by('-applied\_date').first()

if not bus\_pass:

return Response({"message": "No active bus pass found."}, status=404)

expiry\_date = bus\_pass.applied\_date.date() + timedelta(days=bus\_pass.pass\_duration \* 30)

last\_renewal = RenewalRequest.objects.filter(bus\_pass=bus\_pass).order\_by('-generated\_date').first()

if last\_renewal and last\_renewal.new\_expiry\_date >= current\_date:

return Response({"message": f"Your bus pass is already renewed until {last\_renewal.new\_expiry\_date}"}, status=400)

if expiry\_date > current\_date:

new\_expiry\_date = current\_date + timedelta(days=bus\_pass.pass\_duration \* 30)

RenewalRequest.objects.create(bus\_pass=bus\_pass, new\_expiry\_date=new\_expiry\_date)

return Response({"message": f"Bus pass renewed successfully! New expiry date: {new\_expiry\_date}"}, status=200)

return Response({"message": f"Bus pass is still active until {expiry\_date}. Renewal is not needed yet."}, status=400)

@api\_view(['GET'])

@permission\_classes([permissions.IsAuthenticated])

def get\_renewals(request):

user = request.user

renewals = RenewalRequest.objects.filter(bus\_pass\_\_user=user).order\_by('-generated\_date')

serializer = RenewalRequestSerializer(renewals, many=True)

return Response(serializer.data)

from rest\_framework.decorators import api\_view, permission\_classes

from rest\_framework.permissions import AllowAny

from rest\_framework.response import Response

@api\_view(['POST'])

@permission\_classes([permissions.AllowAny])

def register\_user(request):

username = request.data.get("username")

email = request.data.get("email")

password = request.data.get("password")

first\_name = request.data.get("name")

if not username or not email or not password:

return Response({"error": "All fields are required"}, status=400)

if User.objects.filter(username=username).exists():

return Response({"error": "Username already exists"}, status=400)

if User.objects.filter(email=email).exists():

return Response({"error": "Email already exists"}, status=400)

user = User.objects.create\_user(username=username,first\_name=first\_name, email=email, password=password)

return Response({"message": "User registered successfully", "user\_id": user.id}, status=201)

# views.py

from django.shortcuts import render, get\_object\_or\_404

from django.http import JsonResponse

from .models import BusPass

from django.views.decorators.csrf import csrf\_exempt

import json

def get\_pending\_bus\_passes(request):

if request.method == "GET":

pending\_passes = BusPass.objects.filter(application\_status='Pending')

data = [

{

"id": pass\_obj.id,

"full\_name": pass\_obj.full\_name,

"pass\_type": pass\_obj.pass\_type,

"email": pass\_obj.email,

"phone\_number": pass\_obj.phone\_number,

"source": pass\_obj.source\_location,

"destination": pass\_obj.destination\_location,

"duration": pass\_obj.pass\_duration

}

for pass\_obj in pending\_passes

]

return JsonResponse(data, safe=False)

@csrf\_exempt

def approve\_bus\_pass(request, pass\_id):

if request.method == "POST":

bus\_pass = get\_object\_or\_404(BusPass, id=pass\_id)

bus\_pass.application\_status = "Approved"

bus\_pass.save()

return JsonResponse({"message": "Bus pass approved successfully"})

REACT

import { BrowserRouter as Router, Route, Routes} from 'react-router-dom';

import './App.css';

import Header from './components/Header';

import { Home } from './components/Home';

import { Login } from './components/Login';

import { Register } from './components/Register';

import { useState,useEffect } from 'react';

import { ACCESS\_TOKEN } from './constants';

import { useNavigate } from 'react-router-dom';

import UserList from './components/UserList';

import { Admin } from './constants';

import BusPassForm from './components/BusPassForm';

import CheckStatus from './components/CheckStatus';

import RenewalList from './components/RenewalList';

import PendingBusPasses from './components/PendingBusPasses';

function Logout({setisAuthenticated,setadmin}){

const navigate=useNavigate();

useEffect(()=>{

localStorage.clear();

setisAuthenticated(false);

setadmin(false);

},[setisAuthenticated,setadmin])

navigate('/login')

return null;

}

function App() {

const [isAuthenticated, setisAuthenticated] = useState(!!localStorage.getItem(ACCESS\_TOKEN));

const [admin,setadmin]=useState(!!localStorage.getItem(Admin));

return(

<Router>

<div className='flex'>

<Header isAuthenticated={isAuthenticated} admin={admin} />

<Routes>

<Route path='' element={<Home/>}/>

<Route path='login' element={<Login setisAuthenticated={setisAuthenticated} setadmin={setadmin}/>}/>

<Route path='admin-login' element={<Login setisAuthenticated={setisAuthenticated} setadmin={setadmin}/>}/>

<Route path='register' element={<Register/>}/>

<Route path='admin/users' element={<UserList />}/>

<Route path='apply-pass' element={<BusPassForm />}/>

<Route path='status' element={<CheckStatus />}/>

<Route path='admin/applications' element={<PendingBusPasses />}/>

<Route path='RenewalList' element={<RenewalList/>}/>

<Route path='logout' element={< Logout setisAuthenticated={setisAuthenticated} setadmin={setadmin}/>}/>

</Routes>

</div>

</Router>

);

}

export default App;

**CHAPTER 10**

**RESULT AND DISCUSSIONS**

**10.1 Implementation and Description**

The implementation of the bus pass management system involves the integration of a robust backend using Django REST Framework and a dynamic, user-friendly frontend built with React. The system is designed to streamline the entire process of bus pass applications, approvals, and status tracking for both users and administrators. The Django backend manages all server-side operations such as user registration, authentication, application submission, and status updates. It provides a set of RESTful API endpoints that serve as the communication bridge between the frontend and the database. All application logic, business rules, and data validations are handled on the server side, ensuring secure and reliable operations.

On the frontend, React is used to build a responsive interface with components such as login and registration forms, bus pass application forms, dashboards for users and admins, and real-time status updates. Axios is used to perform API calls to the backend. Form inputs are validated at both client and server ends to prevent erroneous data submission. React Router is employed to manage multiple views such as home, apply, status check, and admin approval pages.The admin dashboard allows authorized personnel to review applications, approve or reject them, and view analytics related to user submissions. The application data is stored in a structured MySQL database with defined relations for users, applications, and their statuses. Token-based authentication (using JWT or Django’s built-in authentication) ensures only authenticated users can access sensitive parts of the system.

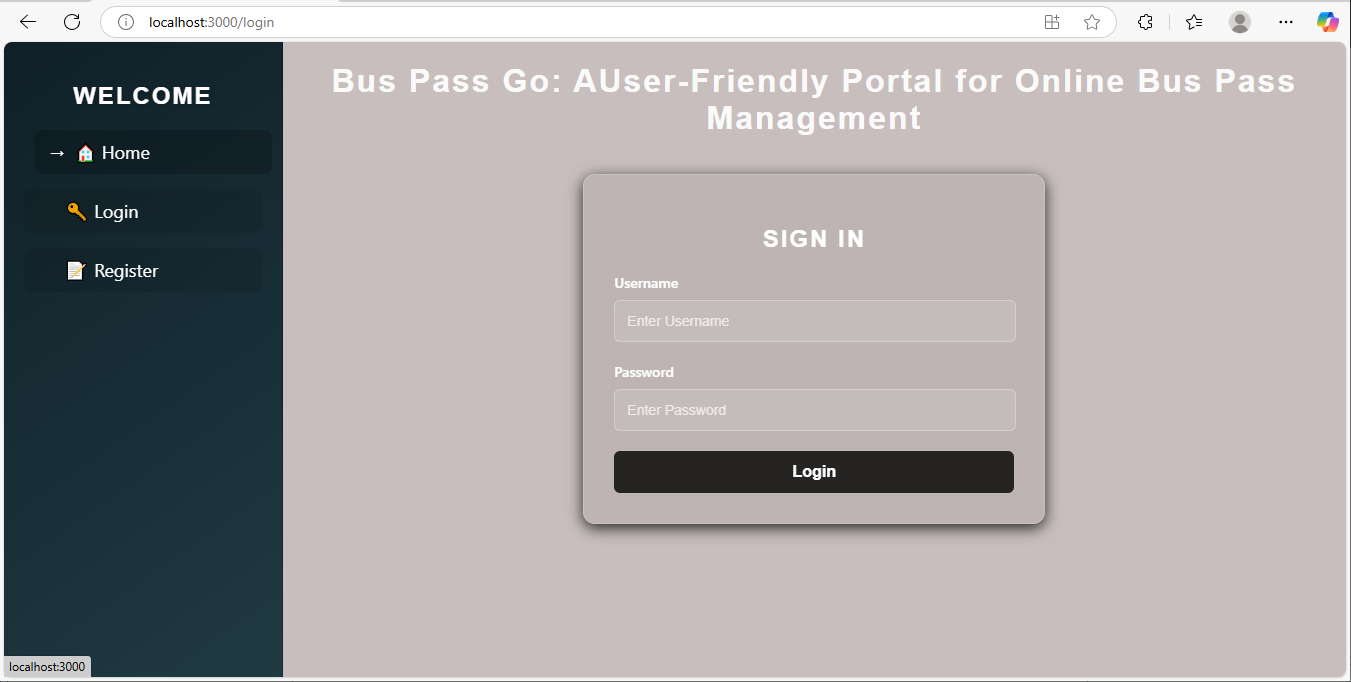
All operations from form submission to backend validation, data persistence, and frontend rendering are performed asynchronously to maintain responsiveness. The deployment is done on a web server with proper routing, static file handling, and secure database connection configurations. The entire implementation supports modularity, scalability, and maintainability, ensuring future enhancements and role-based access controls can be integrated seamlessly.

**10.2 Results**



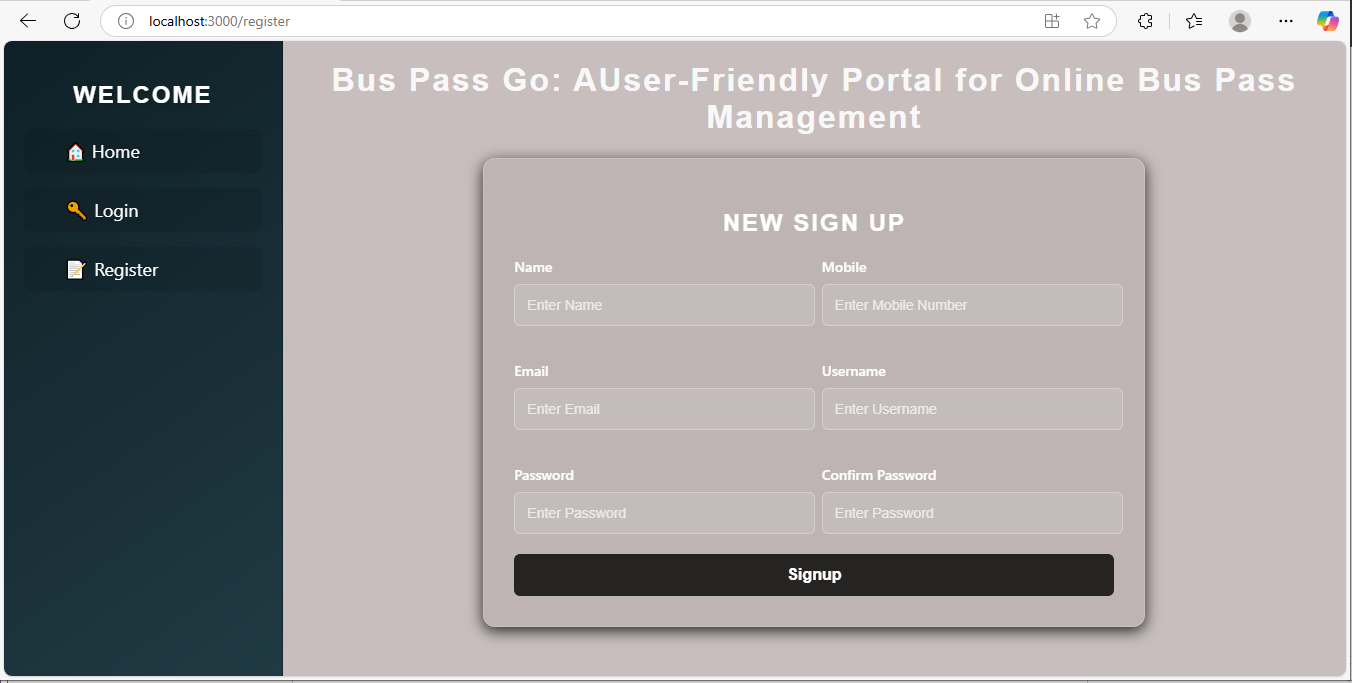
10.1 Home Page

The figure displays a "WELCOME" section on the left side of the screen, serving as a navigation panel. This panel offers three basic options: "Home," "Login," and "Register," each accompanied by a small icon. This simple layout suggests a landing page with clear pathways for users to navigate to the main content, sign in to an existing account, or create a new one.



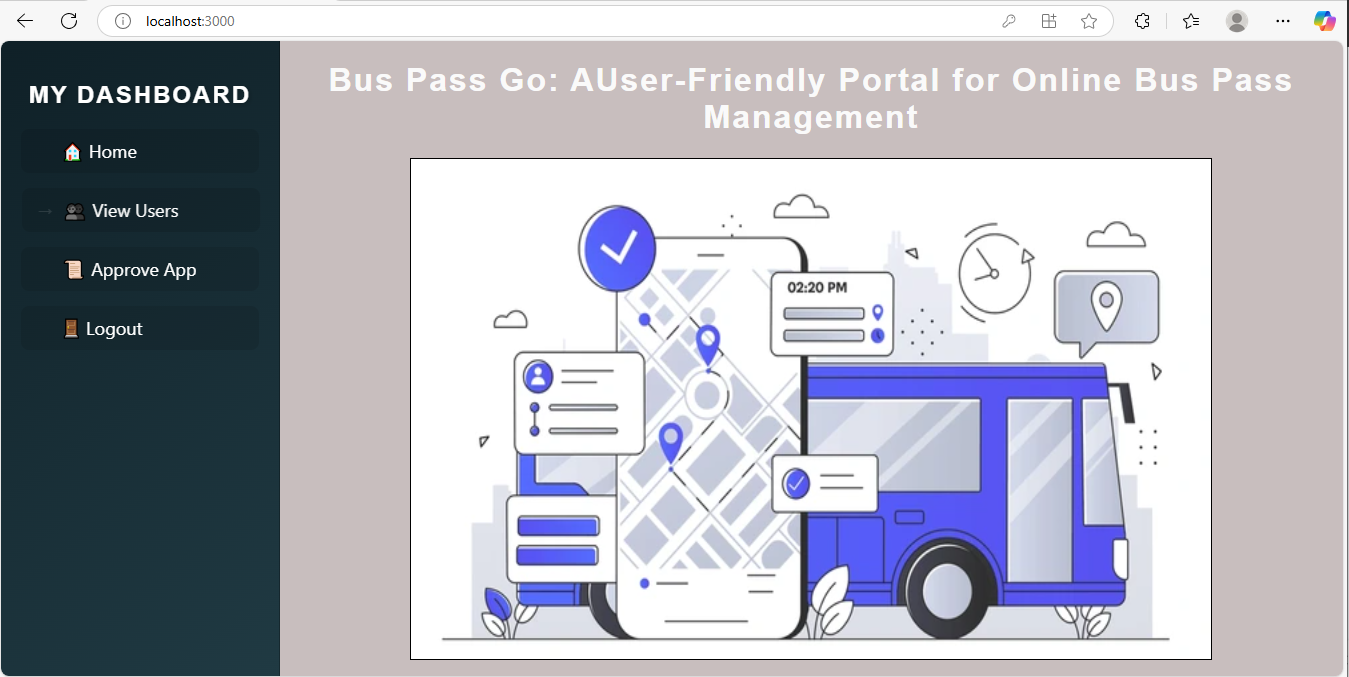
10.2 Login Page

The figure displays a "SIGN IN" login interface. It features two input fields for user credentials: "Username" and "Password," each with placeholder text prompting the user for input. A "Login" button is positioned at the bottom, which users would click to authenticate and proceed. The overall design provides a clear and straightforward method for user access.

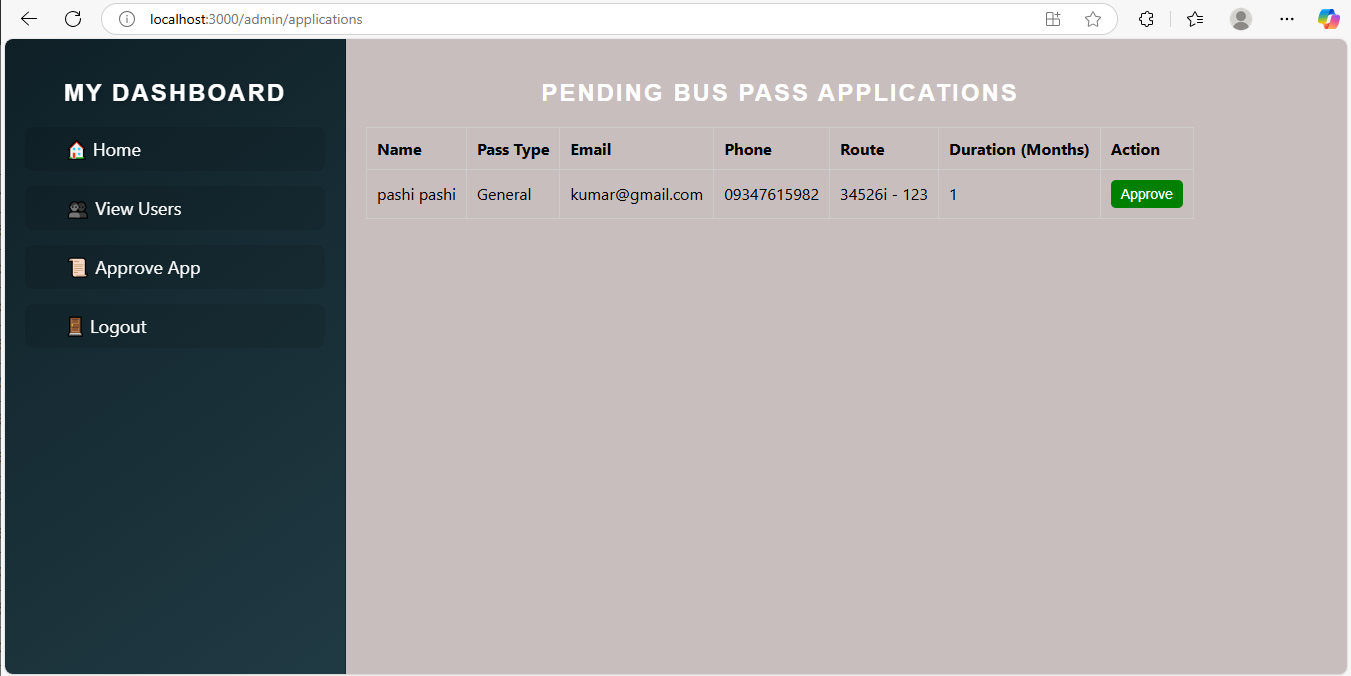


10.3 Register page

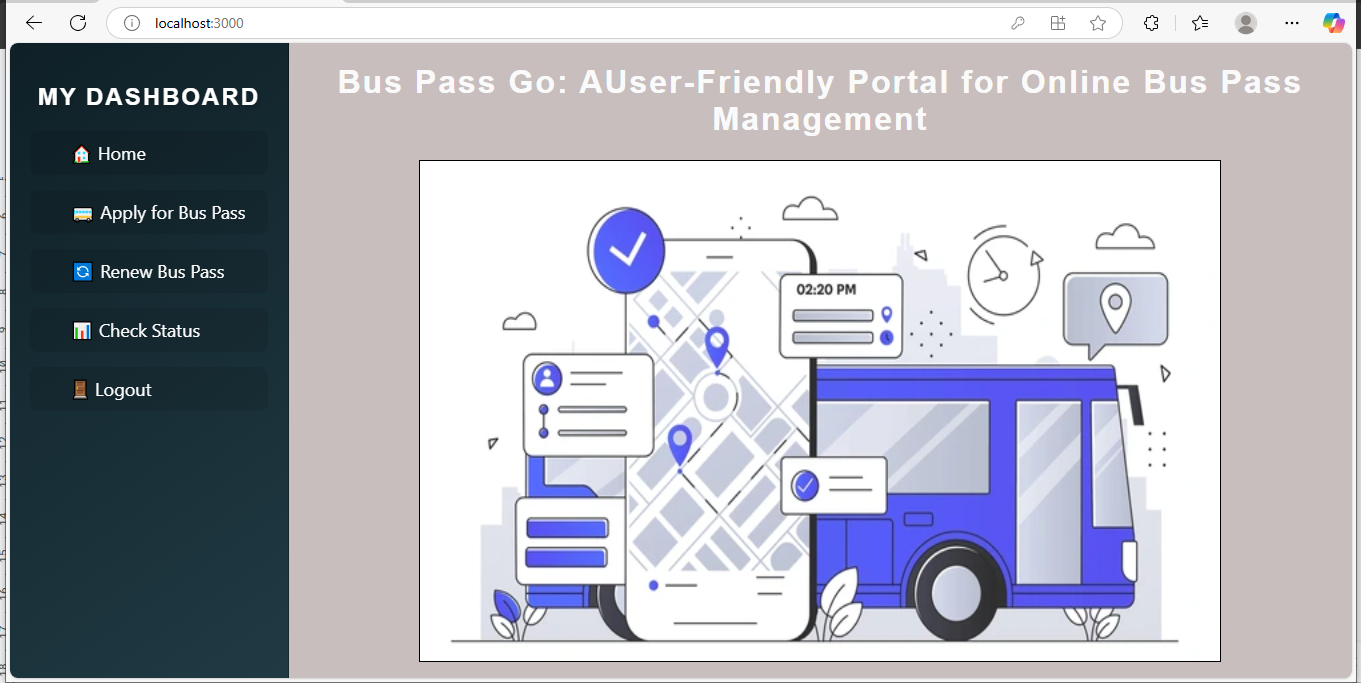
The figure displays a "NEW SIGN UP" form, serving as a registration page for new users. It includes input fields for "Name," "Mobile," "Email," "Username," "Password," and "Confirm Password," allowing users to enter their personal and account details. A "Signup" button is present at the bottom, which users would click to complete their registration. This screen provides a clear and organized way for individuals to create a new account within the system.



10.4 Admin Dashboard

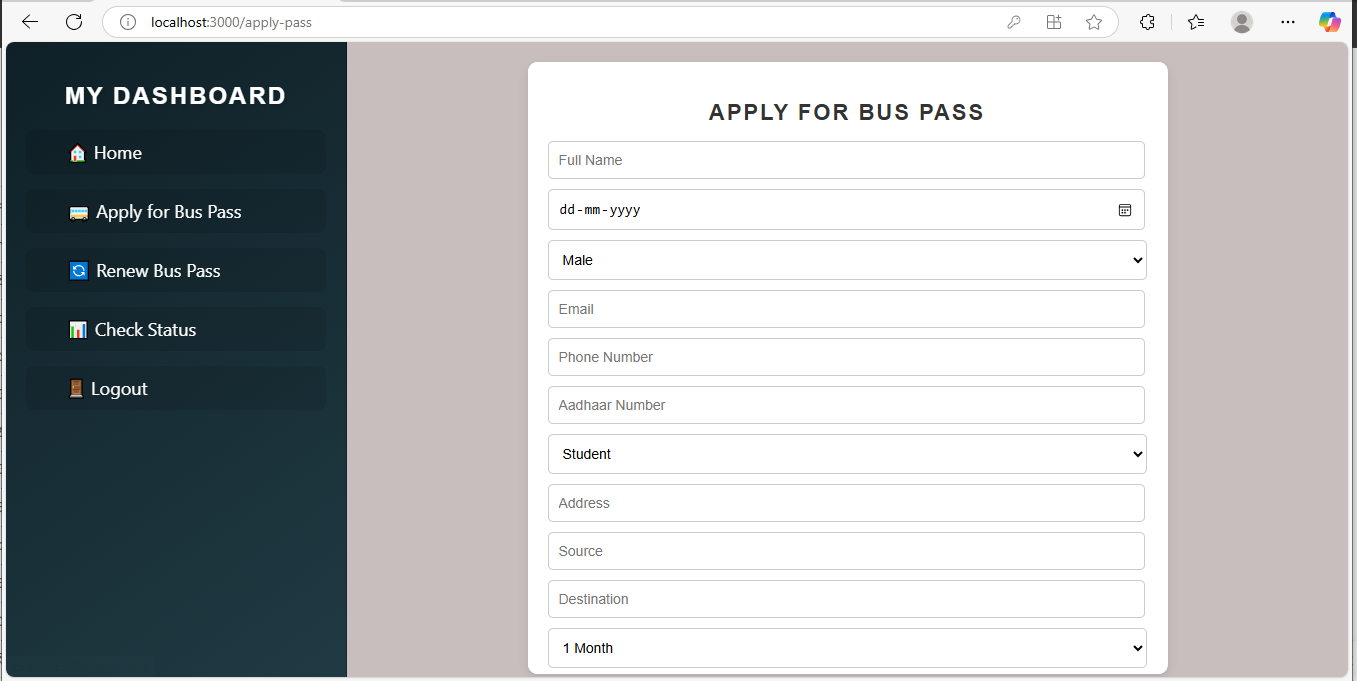


10.5 Approve Applications



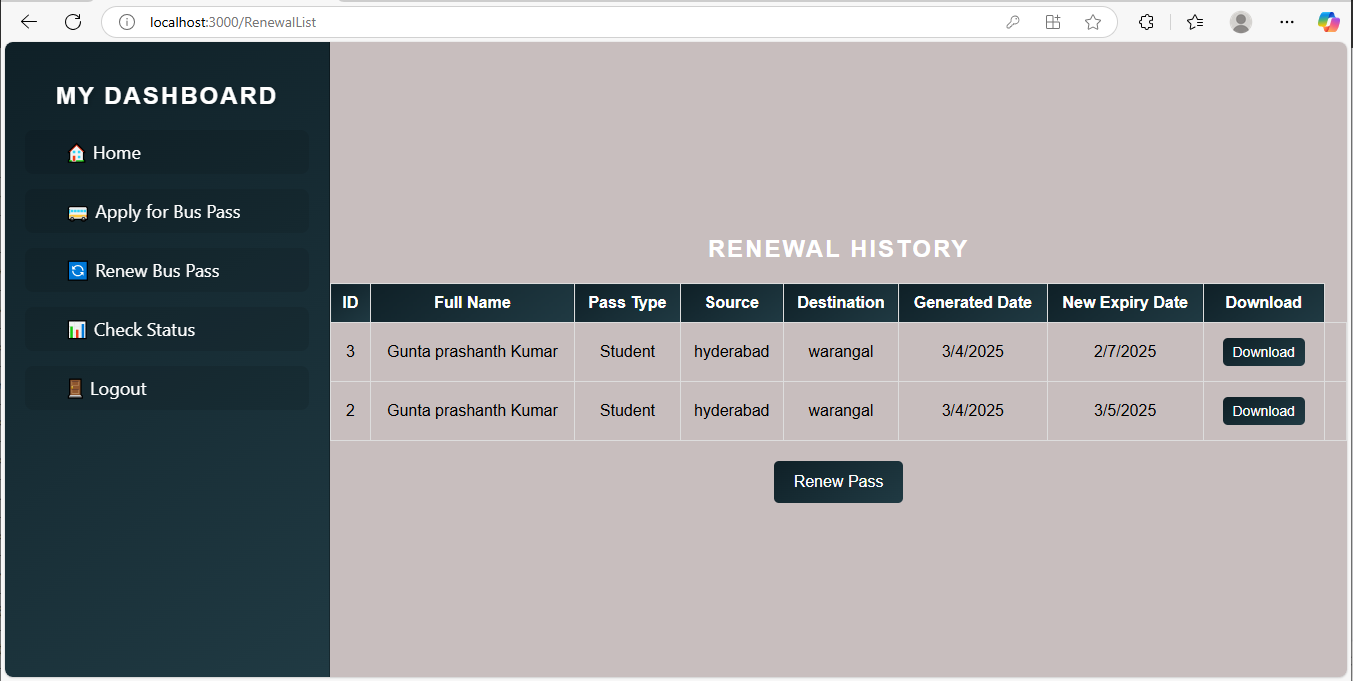
10.6 User Dashboard

The figure displays a "MY DASHBOARD" navigation panel, providing several options for a user. These options include "Home," "Apply for Bus Pass," "Renew Bus Pass," "Check Status," and "Logout." This set of links suggests a system focused on managing bus pass related activities, allowing users to apply for new passes, renew existing ones, track their application status, and securely log out.



10.7 apply bus pass

The figure displays an "APPLY FOR BUS PASS" form, which is designed for users to submit an application for a bus pass. The form includes various input fields for personal details such as "Full Name," a date field (dd-mm-yyyy), "Gender" (with a dropdown), "Email," "Phone Number," "Aadhaar Number," and "Student" (with a dropdown). Additionally, it requests location information with fields for "Address," "Source," and "Destination." Finally, there's a dropdown for the duration of the pass, showing "1 Month." This comprehensive form allows users to provide all necessary information for a bus pass application.



10.8 renewal pass

The figure displays a "RENEWAL HISTORY" table, which provides a record of past bus pass renewals. The table includes columns for "ID," "Full Name," "Pass Type," "Source," "Destination," "Generated Date," and "New Expiry Date." Each row represents a previous renewal, detailing the specific pass information and its new expiry date. Additionally, there's a "Download" button for each entry, suggesting that a digital copy of the renewed pass can be obtained. At the bottom of the table, a "Renew Pass" button is present, allowing users to initiate a new renewal process.

**CHAPTER 11**

**CONCLUSION AND FUTURE SCOPE**

**11.1 Conclusion**

The bus pass management system developed using Django and React successfully addresses the inefficiencies, delays, and manual errors associated with traditional bus pass issuance processes. It provides a seamless and transparent mechanism for students and commuters to apply for, track, and manage their bus passes online. By integrating Django REST Framework as the backend, the system ensures secure data handling, structured API responses, and reliable user authentication. The use of React in the frontend offers a dynamic and responsive user interface, allowing smooth interaction across devices. Together, this full-stack approach bridges the gap between users and transport authorities, reducing physical queues, paperwork, and time spent on administrative approvals. The system supports user registration, login/logout functionalities, bus pass application submission, and a detailed status tracking mechanism. Users can view their application status in real-time, and administrators are provided with a clean dashboard to manage applications, approve or reject requests, and maintain pass issuance records. Security is enforced through role-based access, input validation, and secure storage of sensitive data. The modular structure of both frontend and backend allows for easy updates, maintenance, and future scalability. The user-centric design improves overall experience by ensuring fast access to services and reducing the dependency on in-person visits or intermediaries.

**11.2 Future Scope**

The bus pass management system has the potential for several future enhancements. Integration with online payment gateways can enable users to pay fees digitally, removing the need for manual payment handling. The system can be extended to mobile platforms via React Native, providing a native app experience. Features like QR code-based digital passes, automatic expiration alerts, integration with GPS tracking systems, and real-time bus schedules can further improve usability. Additionally, linking the system with institutional databases can automate student verification, making the process faster and more secure. As smart city initiatives expand, such systems can be scaled to support other modes of transportation like metro, train, and ferry services under a unified transport management platform.

**REFERENCE**

1. Li, Cheng, et al. "An RFID-Based Attendance System Using Passive Tags." IEEE Transactions on Education, vol. 60, no. 4, pp. 296-302, 2017.
2. Wang, Hua, et al. "Design and Implementation of an RFID-Based Metro Pass System." IEEE Transactions on Vehicular Technology, vol. 66, no. 11, pp. 10347-10357, 2017.
3. Chen, Wei, et al. "RFID-Based Bus Pass System: Implementation and Performance Evaluation." Transportation Research Part C: Emerging Technologies, vol. 88, pp. 67-81, 2018.
4. Liu, Shuai, et al. "RFID-Based Real-Time Locating System for Asset Tracking in Hospitals." Journal of Medical Systems, vol. 41, no. 2, pp. 25-32, 2017.
5. Xu, Xiaofeng, et al. "Design and Implementation of an RFID-Based Library Management System." Journal of Librarianship and Information Science, vol. 50, no. 1, pp. 45-57, 2018. Wang, Lei, et al. "RFID-Based Automated Toll Collection System for Highways." IEEE Transactions on Intelligent Transportation Systems, vol. 20, no. 4, pp. 1305-1317, 2019.
6. Sharma, Rajesh, et al. "RFID-Based Supply Chain Management: A Comprehensive Review and Future Directions." Computers & Industrial Engineering, vol. 128, pp. 1067-1087, 2019.
7. Zhang, Ming, et al. "RFID-Based Animal Tracking System: A Case Study in Wildlife Conservation." Journal of Wildlife Management, vol. 82, no. 6, pp. 1203-1215, 2018.
8. Lee, Seung-Hwan, et al. "RFID-Based Access Control System for Secure Building Management." IEEE Access, vol. 6, pp. 78921-78934, 2018.
9. Chen, Jian, et al. "RFID-Based Asset Management System for Manufacturing Facilities." International Journal of Advanced Manufacturing Technology, vol. 94, no. 5-8, pp. 1955-1967, 2018.
10. Yang, Dong, et al. "RFID-Based Smart Agriculture: A Comprehensive Review." Computers and Electronics in Agriculture, vol. 161, pp. 272-289, 2019. [15] Park, Min-Jeong, et al. "RFID-Based Passenger Identification System for Improved Security in Airports." IEEE Transactions on Intelligent Transportation Systems, vol. 20, no. 7, pp. 2521-2534, 2019.