ANUDIP FOUNDATION

**Project Title**

**“Notes App”**

**By**

|  |  |
| --- | --- |
| **Name** | **Enrollment No** |
| **Sarves Munde** | **AF0482936** |

**Under Guidance**

**Of**

**Rajshri Thete**

**ABSTRACT**

The **Notes App** is a web-based application designed to simplify the creation, organization, and retrieval of personal and categorized notes. This system replaces traditional methods such as handwritten notebooks or scattered digital files with a secure, responsive, and user-friendly digital platform. It allows users to efficiently add, update, search, and manage notes under various categories like *Important*, *Business*, and *Personal*.

The Notes App provides authenticated access to individual users, ensuring that their data is securely isolated and protected. Key functionalities include note creation and editing, keyword-based search, category-based filtering, a real-time dashboard, and a detailed view for each note. Users can access their content anytime through a centralized system designed for ease of use and cross-device compatibility.

By integrating a secure backend using Django and responsive frontend technologies like Next.js and Tailwind CSS, the application improves note accessibility, enhances user productivity, reduces the risk of data loss, and promotes better information organization. The system ensures privacy, seamless data handling, and an improved user experience, making it an ideal solution for students, professionals, and anyone needing an efficient digital note management tool.

**Index**

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Contents** | **Page No.** |
| 1 | **INTRODUCTION** | 5-11 |
| 1.1 Company Profile | 6 |
| 1.2 Motivation | 7 |
| 1.3 problem statement | 8 |
| 1.4 purpose/objective and goals | 9 |
| 1.5 literature survey | 10 |
| 1.6 project scope and limitations | 10 |
| 2 | **SYSTEM ANALYSIS** | 12-16 |
| 2.1 Comparative study of Existing systems | 12 |
| 2.2 scope and limitations of existing systems | 12 |
| 2.3 project perspective, features | 13 |
|  | 2.4 stakeholders | 14 |
|  | 2.5 Requirement analysis - Functional requirements, performance requirements, security requirements etc. | 16 |
| 3 | **SYSTEM DESIGN** | 17-33 |
| 3.1 Design constraints | 17 |
| 3.2 Sequence diagram | 19 |
| 3.3 Activity diagram | 20 |
| 34 Entity-Relationship diagram | 20 |
| 3.5 Class diagram | 21 |
| 3.6 Use case diagram3 | 22 |
| 3.7 Component Diagram | 23 |
| 3.8 Data dictionary | 24 |
| 3.9 Test procedure & implementation | 25 |
| 3.10 Screen shot | 29 |
| 3.11 Reports | 32 |
| 4 | **IMPLEMENTATION DETAILS**  4.1 Software/hardware specifications, etc | 34 |
| 5 | **CODING** | 35-49 |
| 6 | **TESTING**  Test Plan, Black Box Testing or Data Validation Test Cases, White Box Testing or Functional Validation Test cases and results | 50 |
| 7 | **LIMITATIONS** | 53 |
| 8 | **CONCLUSION** | 54 |
| 9 | **BIBLIOGRAPHY** | 55 |
| 10 | **REFERENCES** | 56 |

1. **INTRODUCTION**

A **Notes Management System** is an essential productivity tool designed to help users digitally capture, organize, and manage their thoughts, tasks, and information across various categories. In an increasingly digital world, such applications replace traditional notebooks and scattered documents with centralized, searchable, and secure digital notes.

This **Notes App**, developed using **Next.js**, **React Query**, **Tailwind CSS**, and powered by a **Django backend**, exemplifies a full-stack modern web application. It combines robust backend logic with a responsive and intuitive frontend interface, offering users a seamless experience for managing their personal, academic, or professional notes.

The application allows users to **register and log in securely**, **create**, **edit**, **delete**, and **view notes**, and organize them into predefined categories like *Personal*, *Business*, and *Important*. With features such as **real-time data fetching via React Query**, **search functionality**, and **category-based filtering**, the app supports fast retrieval and efficient organization of notes. Backend validations ensure data accuracy, while frontend feedback provides an interactive user experience.

**Objectives**

The main objectives of the Notes App project are:

* To develop a user-friendly application for creating, organizing, and managing personal and professional notes.
* To implement CRUD functionality (Create, Read, Update, Delete) for notes with a clean and responsive UI.
* To demonstrate full-stack development using Next.js for the frontend and Django for the backend.
* To incorporate React Query for efficient data fetching, caching, and UI synchronization.
* To enable search and category-based filtering for easy access and note organization.
* To apply robust backend validations and real-time frontend feedback for a reliable and interactive experience.
* To provide data persistence and secure user session management.

**1.1 Company Profile**

* **Organization Name:** Anudip Foundation for Social Welfare
* **Year of Establishment:** 2007
* **Type:** Non-Profit Organization
* **Headquarters:** Kolkata, Mumbai, Pune

**Overview:**

Anudip Foundation is a globally recognized non-profit organization dedicated to transforming lives through technology-enabled education and livelihood programs. Established in 2007, Anudip empowers youth, women, and marginalized communities by equipping them with essential digital, technical, and professional skills. By merging technology with social innovation, the organization creates pathways to sustainable employment and economic independence.

Operating in over 20 states across India and with a growing global presence, Anudip collaborates with government institutions, corporate partners, and international NGOs to implement high-impact programs. These initiatives aim to bridge the digital divide, promote gender equality, and enhance the employability of underserved populations.

**Relevance to the Project:**

The **Notes Management System**, developed using **Next.js**, **React Query**, **Tailwind CSS**, and **Django**, was created as part of Anudip Foundation’s **IT and Coding Bootcamp** initiative. The project was developed under expert mentorship and with technical resources provided by Anudip, offering students hands-on experience in building real-world, full-stack web applications.

**Mission Statement:**

“To transform lives by creating digitally-enabled, self-reliant communities.”

**Vision Statement:**

“To be a global leader in delivering digital skills and career transformation to underserved populations.”

**1.2 Motivation**

The idea for the **Notes Management System** arose from the increasing demand for a structured, accessible, and efficient way to capture and organize personal and professional thoughts, tasks, and important information. In a fast-paced, information-driven world, relying on scattered documents, paper notebooks, or unsynchronized note-taking tools often leads to disorganization, reduced productivity, and lost insights. This project addresses that challenge by offering a centralized, responsive, and user-friendly web application to help users manage their notes with clarity and convenience.

**Key Motivations:**

* **Simplifying Information Organization**  
  Whether users are students, professionals, or individuals managing daily tasks, keeping notes scattered across different files or platforms can be overwhelming. This system offers a clean, categorized way to store notes, ensuring easy access and better mental clarity.
* **Enhancing Productivity**  
  A well-designed note-taking system boosts productivity by reducing the time spent searching for information. The app's ability to filter notes by categories such as *Personal*, *Business*, or *Important*, along with an intuitive search feature, helps users focus on what matters most.
* **Reducing Data Redundancy and Loss**  
  Traditional note-keeping methods like notebooks or unsaved text files are prone to loss, duplication, or inaccessibility. The Notes App ensures secure, persistent storage of data with support for CRUD operations and real-time feedback, significantly reducing such risks.
* **Improving Accessibility and Searchability**  
  Users often need to retrieve specific content quickly—be it a to-do list, project idea, or reminder. The app’s full-text search and category filtering capabilities allow quick and precise access to relevant information.
* **Supporting Personal and Academic Use Cases**  
  From daily journaling and academic note-taking to work-related task tracking and meeting summaries, the system is designed to be flexible and adaptable to a wide range of user needs.
* **Promoting Digital Record-Keeping and Skill Development**  
  This project not only supports users in transitioning from paper-based note-taking to a structured digital platform, but also showcases how modern full-stack technologies (Next.js, React Query, Django) can be used to create real-world solutions with social and educational impact.

**1.3 Problem Statement**

In today’s fast-moving, information-driven world, both individuals and professionals are constantly generating and consuming large amounts of information—ranging from task lists and reminders to study notes, ideas, and project plans. Despite the availability of digital tools, many users still rely on scattered and unstructured methods such as paper notebooks, mobile notepads, sticky notes, or disorganized digital documents to record and retrieve information. These approaches are often inefficient, hard to maintain, and prone to data loss.

As academic, professional, and personal responsibilities grow, managing notes in a meaningful and accessible way becomes increasingly important. Users often struggle to retrieve specific notes quickly, forget where information is stored, or duplicate effort due to a lack of a centralized system. Moreover, existing note-taking tools are either too simplistic for deep organization or too complex for casual use, lacking features like categorization, filtering, and smart organization.

A modern **Notes Management System** should go beyond simple text entry. It should empower users to efficiently record, categorize, search, and manage notes in a way that boosts productivity and ensures long-term accessibility.

**Identified Problems:**

* **Lack of a Centralized Platform**: Users lack a unified, cloud-based system to securely store and manage their notes, making access across devices or locations inconsistent and unreliable.
* **Unstructured and Scattered Storage**: Relying on text files, notepad apps, or paper notebooks leads to disorganized information, redundancy, and difficulty in retrieving important content when needed.
* **Limited Search and Filter Capabilities**:Most traditional or basic apps lack powerful search or filtering options, making it hard to quickly find notes based on keywords, categories, or tags.
* **No Categorization or Labeling System**: Without the ability to organize notes into categories such as *Personal*, *Business*, or *Important*, users face difficulty in managing their content efficiently.
* **No Real-Time Updates or Feedback**: Changes to notes may not reflect immediately in basic tools, reducing collaboration potential and disrupting workflow in real-time usage scenarios.
* **No Analytics or Usage Insights**: Users cannot view metrics such as the number of notes per category, note frequency, or trends over time, which could help them better manage their time and content.

**1.4 Purpose / Objective and Goals**

The purpose of the Notes App is to offer an intuitive and effective solution for managing personal and professional notes. Users can create, edit, categorize, and search through their notes with ease, all within a clean and accessible interface. The main goal is to help users store their thoughts, ideas, and important information in one place—improving memory retention, organization, and productivity. By offering offline access, the app ensures that users can work without interruption, regardless of their internet connectivity. make this small

### ****Objectives****:

### To develop a full-stack web-based Notes App using Next.js (frontend), React Query (for efficient data fetching), and Django (backend).

### To implement secure user authentication through login, registration, and session management.

### To enable complete CRUD (Create, Read, Update, Delete) operations for notes with efficient handling via Django’s ORM.

### To provide a search feature for quickly locating specific notes.

### To implement category-based sorting for notes, such as Important, Business, and Personal.

### To design a clean, user-friendly, and responsive interface using modern frontend tools.

### To ensure offline access and seamless user experience through optimized data fetching and caching strategies.

### To support scalability and ease of maintenance by following modular development principles.

### ****Goals****:

* User Authentication: Only registered users can create, view, and manage their notes.
* Create/Edit/Delete Notes: Users can easily manage note details including title, content, and category.
* Search and Filter: Users can search notes by keywords and filter them by predefined categories.
* Note Detail View: Users can view a detailed note page with options to update or delete.
* Dashboard View: Provides a centralized view of all notes created by the user.
* Category Sorting: Notes can be categorized as Important, Business, or Personal for better organization.
* Responsive UI: Ensures a smooth experience across devices, including desktops, tablets, and smartphones.
* Data Integrity: Maintains consistency in note data and prevents duplication.
* Future-Ready Architecture: Allows easy integration of features such as reminders, cloud backup, or collaboration.

**1.5 Literature Survey**

A literature survey was conducted to analyze existing digital note-taking applications and explore their strengths and limitations. Popular tools like Google Keep, Microsoft OneNote, and Evernote provide advanced features such as cloud sync, collaboration, and multimedia support. However, these platforms often require continuous internet connectivity, user data sharing, and can overwhelm users with complex interfaces and excessive features.

Open-source alternatives and minimalist note apps do exist, but they typically demand significant technical setup or lack customization options. Additionally, many lack category-based organization or offline functionality, which are crucial for focused personal and professional use.

**1.6 Project Scope and Limitations**

**Scope**

**User Authentication & Authorization**

* Users can securely register, log in, and manage their own notes.
* Each user’s notes are private and accessible only to them.

**Contact Management**

* Full CRUD (Create, Read, Update, Delete) operations for notes.
* Each note can contain a title, content, and category (Important, Business, Personal).

**Search & Filter Functionality**

* Users can quickly search through their notes using keywords.
* Filtering is available based on note categories for easy organization.

**Web-based Interface**

* The system is accessible via any modern web browser.
* Built using Next.js and styled with responsive design principles for a clean and intuitive UI.

**Responsive Design**

* Ensures smooth performance and compatibility across desktops, tablets, and mobile devices.

**Backend Integration**

* Utilizes Django as the backend with Django REST Framework for API development.
* Data fetching and caching are handled efficiently using React Query.

**Security & Data Privacy**

#### Authentication and authorization are implemented to ensure only the rightful user can access or modify their notes.

#### Django’s built-in security features (e.g., CSRF protection, hashed passwords) help protect user data.

**Extendability**

* The system architecture supports future enhancements like reminders, cloud backup, or collaborative note sharing.

#### ****Limitations****

* **Single User Role**

No role-based access (e.g., admin, moderator); all users have the same access level.

* **No Collaboration or Sharing Features**

Notes are limited to individual users and cannot be shared or collaboratively edited.

* **Basic Note Structure**

Notes include only a title, content, and category; no support for rich text, file attachments, or checklists.

* **No Export/Backup Options**

Currently, there is no feature to export or back up notes to external formats (e.g., PDF, CSV).

* **No Real-time Features**

The system does not support real-time collaboration, live updates, or notifications.

* **No Offline Editing or Syncing**

While data can be cached, full offline editing and syncing capabilities are not implemented.

* **Scalability Untested for Large User Bases**

The app is optimized for individual users or small groups; performance under heavy concurrent usage is unverified.

* **Minimal Error Feedback**

Error messages are functional but basic; no advanced user-facing logging or system diagnostics are included.

1. **SYSTEM ANALYSIS**

**2.1 Comparative Study of Existing Systems**

**Manual Note-Keeping (Paper-Based)**

* **Description**: Notes are written in physical notebooks, diaries, or sticky notes.
* **Advantages**:
  + Simple, no technical setup required
  + Low cost and easily accessible for everyone
* **Disadvantages**:
  + High risk of loss or damage
  + Not searchable or sortable
  + No backup or recovery
  + Inconvenient for managing multiple categories or topics

**Spreadsheet-Based Note Management (e.g., MS Excel, Google Sheets)**

* **Description**: Users store notes in structured spreadsheets, often organized by rows and categories.
* **Advantages**:
  + Easier to search, sort, and filter than paper
  + No programming knowledge required
* **Disadvantages**:
  + No authentication or access control
  + Not designed for rich-text or long notes
  + Risk of accidental deletion or data corruption
  + Not optimized for multi-user collaboration or scalability

**Legacy Desktop Applications (e.g., Simple Note Apps, Notepad, Word Processors)**

* **Description**: Standalone apps like Notepad, WordPad, or older desktop-based note software
* **Advantages**:
  + Basic text editing features
  + Offline accessibility
* **Disadvantages**:
  + No synchronization across devices
  + No categorization or search functionality
  + No user authentication or role management
  + Difficult to maintain or customize

**2.2** **Scope and Limitations of Existing Systems**

Most existing note apps cater to a wide range of use cases—from academic to business environments—making them **feature-rich but bloated** for users who need a straightforward tool. Their **dependence on cloud infrastructure** limits usage in offline scenarios. Additionally, closed-source systems limit customization, while some open-source ones lack user-friendliness or modern UI/UX design.

Limitations of Existing System

#### No User-Based Data Isolation Many simple note apps or shared tools don't isolate user data, resulting in potential privacy issues.

#### Lack of Secure Authentication Several systems do not provide built-in login/registration or secure authentication mechanisms.

#### Limited Search and Categorization Searching and filtering notes by keyword or type (e.g., personal, business) is often absent or inefficient.

#### Unresponsive User Interfaces Some tools are not optimized for different screen sizes, resulting in a poor experience on mobile or tablet devices.

#### No Centralized Dashboard Users cannot get an overview of all their notes, recent activity, or quick access features from a single page.

#### ****Our System’s Improvements:****

* **Secure Authentication System**Users can register, log in, and manage their notes securely, with Django handling authentication and session management.
* **User-Specific Data Isolation**Each user has access only to their personal notes; data is securely scoped per user account.
* **Categorized Notes and Filters**Notes can be tagged as *Important*, *Business*, or *Personal*, and users can filter or search notes easily via keywords.
* **Modern, Responsive Interface**Built with Next.js and styled for responsiveness, the app provides a consistent experience across devices.
* **Centralized Dashboard**

Users can view all notes in one place, create new notes, and navigate to detailed pages for editing or deleting existing ones.

**2.3 Project Perspective, Features**

The Notes Management System is a secure, web-based platform designed to help users efficiently create, organize, and manage personal or professional notes. Built using Django (backend) and Next.js with React Query (frontend), the application delivers a seamless user experience focused on simplicity, security, and productivity

**Key Features by Role:**

* **Add & Manage Notes**Users can create, update, and delete notes with fields like title, content, and category (e.g., Personal, Important, Business).
* **Search & Filter Notes**A fast and intuitive search bar allows users to quickly locate notes based on keywords or categories.
* **Update & Delete Notes**Each note can be edited or removed easily from the detail view or dashboard.
* **Category-Based Organization**Notes are grouped by category, helping users stay organized and focused.
* **View Note Details**Each note has a dedicated detail page showing the full content with options for editing and deleting.

#### ****Additional Platform Features:****

* **Secure User Authentication**  
  User registration, login, and session management are handled using Django’s authentication system, ensuring that each user’s notes remain private.
* **Responsive UI**  
  The interface is mobile-friendly and fully responsive, built using **Tailwind CSS and Next.js**, ensuring smooth usability across devices.
* **Interactive Dashboard**  
  The dashboard provides a summary of total notes, categorized note views, recent updates, and quick links to add or manage notes.
* **Real-Time Updates**  
  Thanks to React Query, all changes (create/update/delete) are reflected instantly without reloading the page.
* **Data Privacy & Isolation**  
  Notes are user-specific and securely isolated, ensuring no cross-user access.
* **Database Integration**  
  The system uses Django ORM to interface with secure databases like **SQLite** or **MYSQL** for efficient note storage and retrieval

**2.4 Stakeholders**

The Notes App project involves several key stakeholders, each playing a vital role in the development, usage, and future scalability of the platform. These stakeholders contribute to the success of the application from different perspectives—ranging from end-user experience to system development and organizational support.

* **End Users**
  + These are the primary consumers of the application. They include:
    - **Students** using the app to manage academic notes, assignments, and study materials.
    - **Professionals** organizing work-related ideas, meeting notes, and task lists.
    - **General Users** managing personal reflections, reminders, or creative writing.
  + **Their needs influence:**
    - UI/UX design choices
    - Accessibility and responsiveness
    - Features like search, categories, tagging, and secure login
* **Developers**The development team consists of **two full-stack developers** who are jointly responsible for the end-to-end creation and maintenance of the platform. They handle both frontend (Next.js, React Query, Tailwind CSS) and backend (Django, MySQL) development.
  + **Their responsibilities include:**
    - Designing and developing all features and interfaces
    - Managing database schemas and API endpoints
    - Implementing authentication, validations, and security
    - Writing test cases (manual and automated)
    - Debugging and performance optimization
    - Deploying and maintaining the system

This dual-role model ensures hands-on learning and deep collaboration, with both developers equally invested in all aspects of the stack.

* **Anudip Foundation**
  + Organizations like the **Anudip Foundation** play a key stakeholder role in providing a **real-world learning environment** and **project-based evaluation**. They may use this application to:
    - Train students in full-stack development
    - Demonstrate use cases in digital productivity
    - Evaluate student projects based on practical implementation
    - Encourage collaborative work and documentation practices
  + **Institutional influence includes:**
    - Setting expectations for documentation, testing, and feature completion
    - Guiding adherence to development best practices
    - Encouraging deployment readiness and usability
* **Future Contributors & Maintainers**
  + Though currently managed by two developers, the application has the potential to grow. Future contributors—students, interns, or open-source volunteers—may:
* Extend the app’s functionality (e.g., add dark mode, mobile app, or export/import options)
* Help scale the backend or integrate new features like AI suggestions

**2.5 Requirement Analysis**

To ensure the application aligns with user needs, performance goals, and security standards, a clear understanding of functional and non-functional requirements is essential.

**Functional Requirements**:

Users should be able to:

* Register and log in securely.
* Create, edit, delete, and view notes.
* Assign categories (Important, Business, Personal) to each note.
* Search and filter notes using keywords or categories.
* View a summary/dashboard of total and recent notes.

**Performance Requirements**:

* **Fast Response Time**  
  CRUD operations (Create, Read, Update, Delete) should execute quickly and reflect changes in real time using React Query’s state synchronization.
* **Lightweight Execution**  
  The system is optimized for performance and can run on low-end devices with limited processing power and memory.
* **Efficient Resource Usage**  
  Frontend and backend components are designed to minimize memory and network usage, ensuring smooth performance across different environments.

**Security Requirements**:

* **Secure Local and Server-Side Data Handling**  
  Notes and user credentials are stored securely using Django’s ORM and authentication system, preventing unauthorized access.
* **Input Validation**  
  The system validates user input both on the frontend and backend to prevent invalid data, injection attacks, or app crashes.
* **User-Specific Data Access**  
  Each user's notes are isolated in the backend, ensuring that no other user can access, modify, or view another user's data

**3. SYSTEM DESIGN**

**3.1 Design Constraints**

The Notes App is designed with a focus on simplicity, usability, and open-source accessibility. However, several design limitations have been considered to maintain cost-effectiveness and practicality in real-world usage. These constraints are outlined below:

**Platform Limitations**:

* The application is developed as a **web-based platform only**.
* **No native mobile app** (Android/iOS) is available.
* Users without regular access to a **laptop or desktop browser** may face usability challenges.

**Internet Dependency**:

* As a **web application**, the Notes App requires a **stable internet connection** for:
  + User authentication.
  + Data fetching and submission.
  + Accessing dashboard or individual note details.
* This may be restrictive in areas with **low or intermittent internet connectivity**.

**Scalability**:

* The system is currently optimized for **individual usage** or light use scenarios.
* Handling **large volumes of notes** or **multi-user support** (e.g., teams or enterprise) would require:
  + Database optimization.
  + Backend scaling.
  + Possibly switching to more scalable cloud infrastructure.

**Budget Constraints**:

* Built entirely with **open-source technologies** such as:
  + Django (Backend)
  + Next.js + React Query (Frontend)
  + SQLite/MySQL (Database)
* Designed to be **cost-effective** and easily deployable by students or small teams without the need for premium tools or cloud subscriptions.

**Technology Choice**:

* Django is used for its simplicity, security, and rapid development capability.
* **Real-time features (like live sync or collaboration)** are not

Supported unless extended with tools like **Django Channels or WebSockets**.

* React Query enables efficient data fetching but doesn’t include offline support by default.

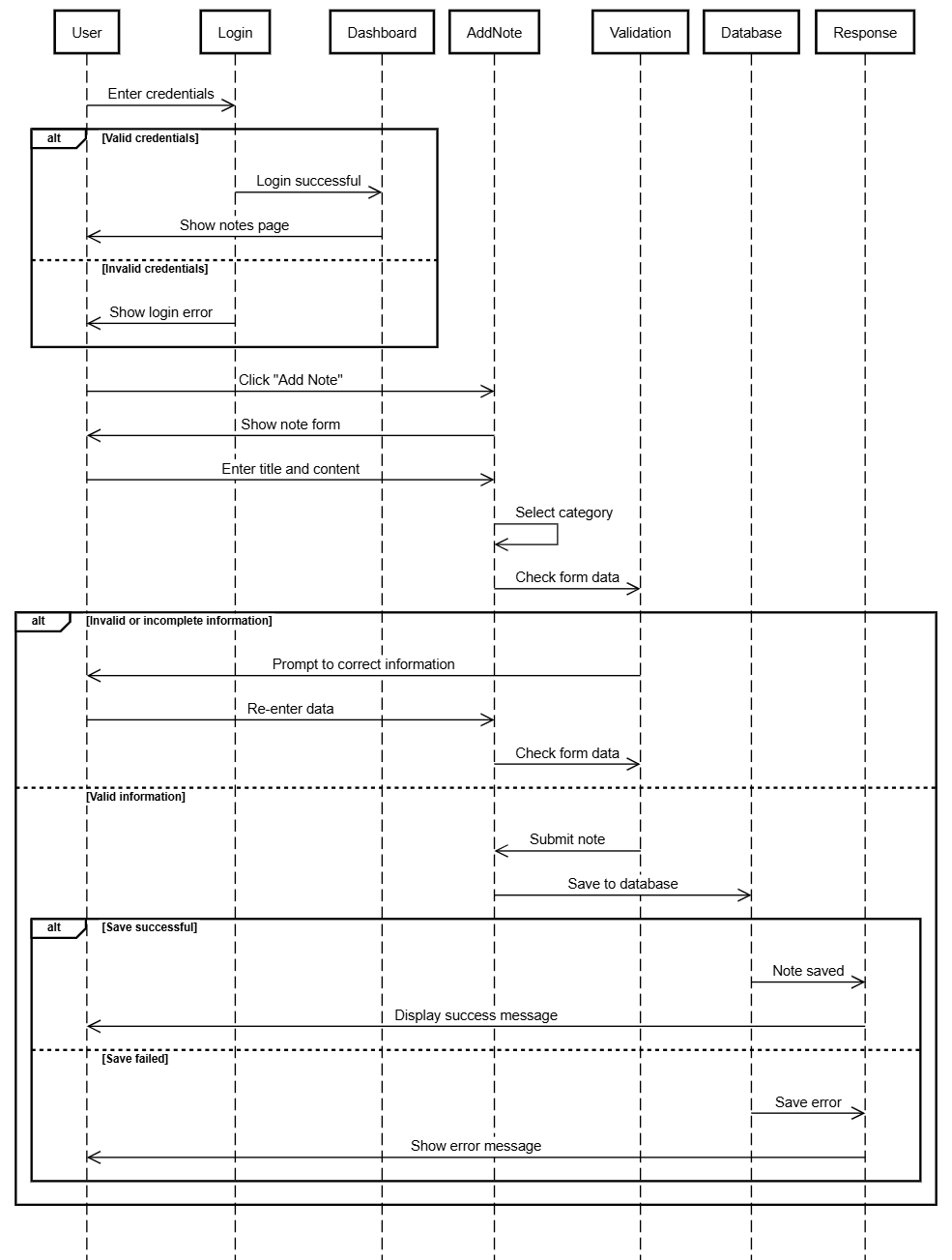
• **User Skills**:

* Assumes users have **basic digital literacy**, including:
  + Creating an account
  + Logging in
  + Creating and editing notes
  + Navigating through dashboards and lists
* Users unfamiliar with web platforms may require a short **intro or guide**.

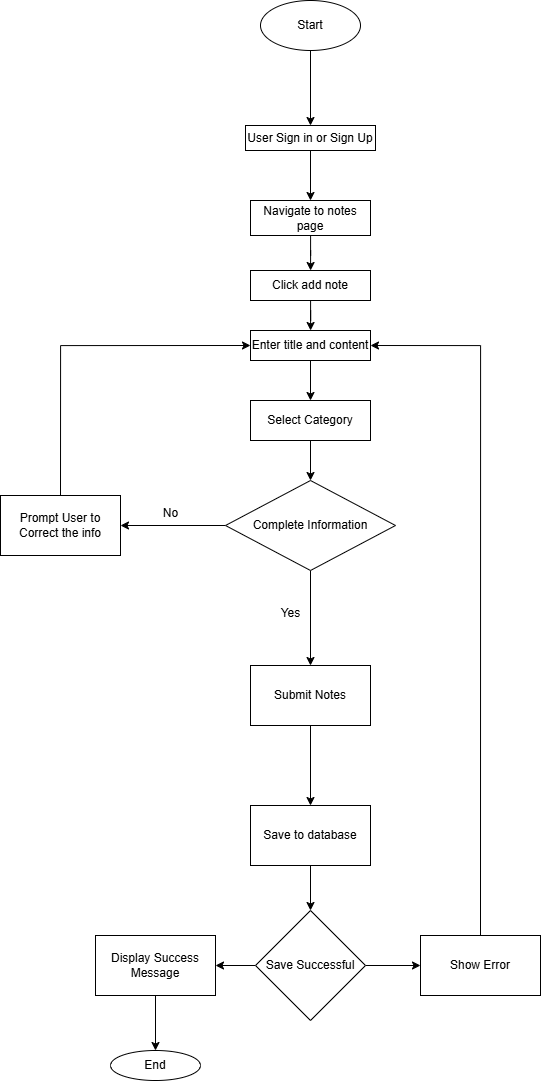
• **Device Compatibility**:

* The UI is designed to be **responsive**, supporting various screen sizes.
* However, performance might be limited on:
  + Low-end mobile devices
  + Outdated web browsers
* To ensure compatibility, **complex animations and advanced frontend features are minimized**.

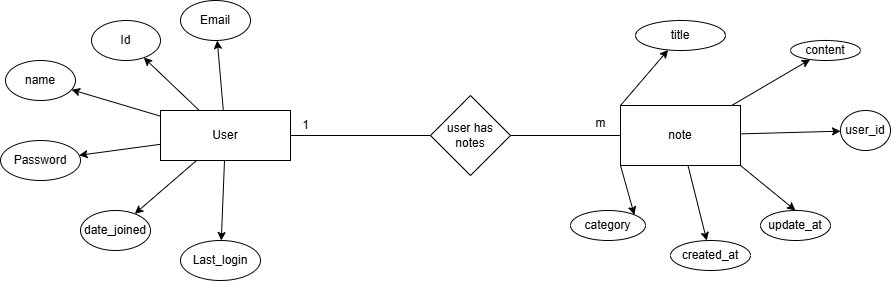
**3.2 Sequence Diagram**



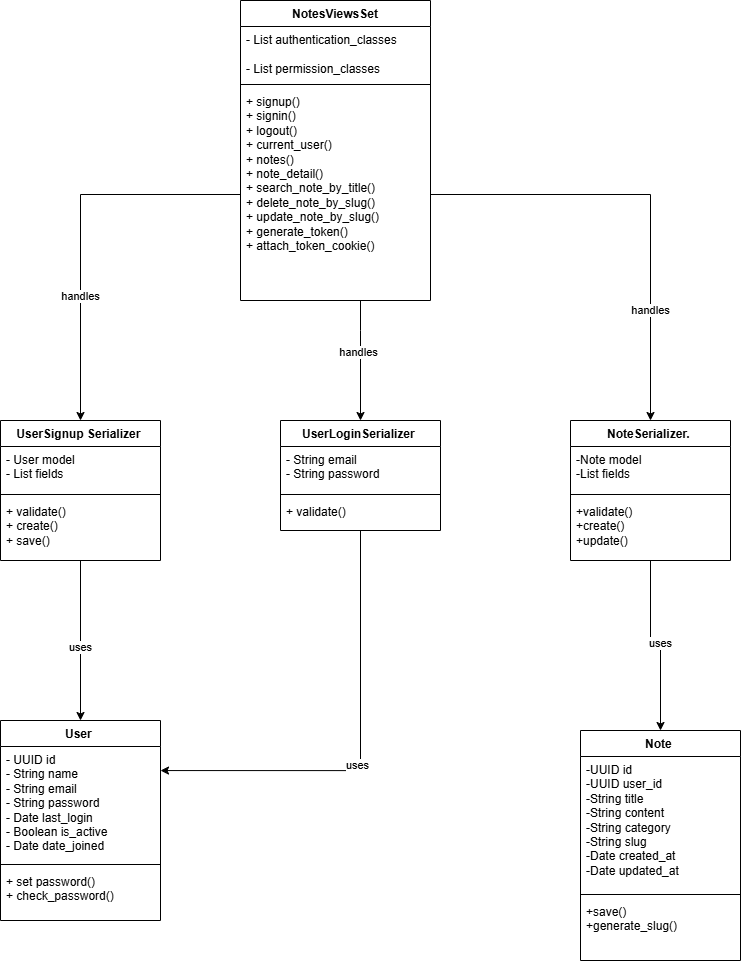
**3.3 Activity Diagram**



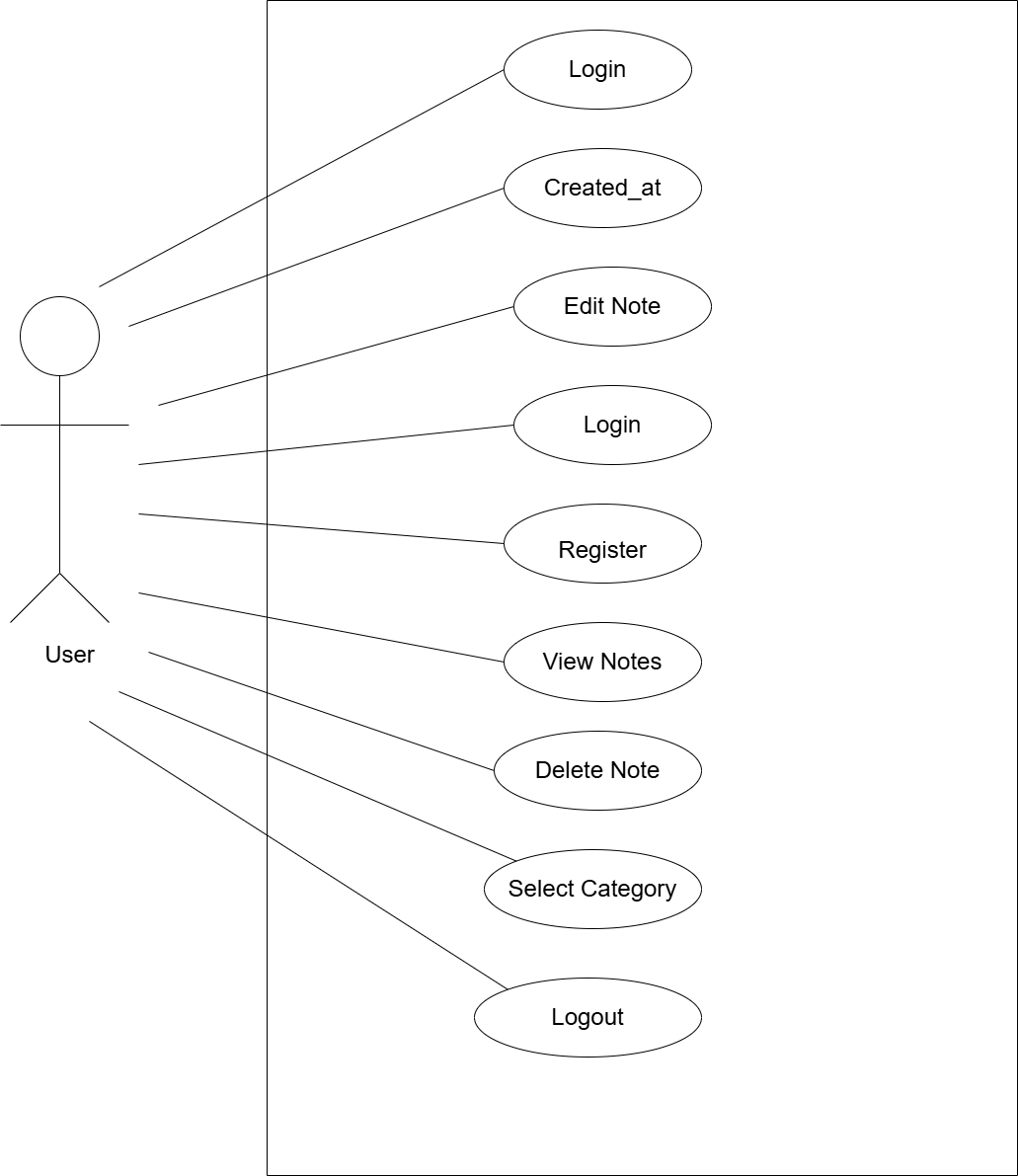
**3.4 Entity-Relationship Diagram**

****

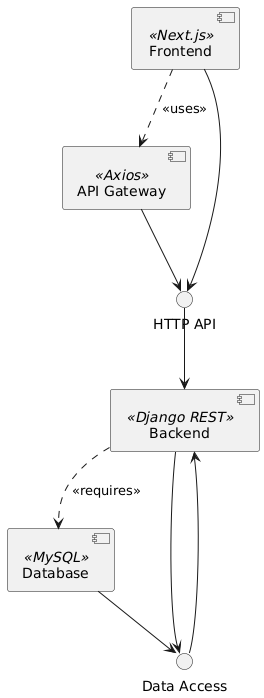
**3.5 Class Diagram**

****

**3.6 Use Case Diagram**



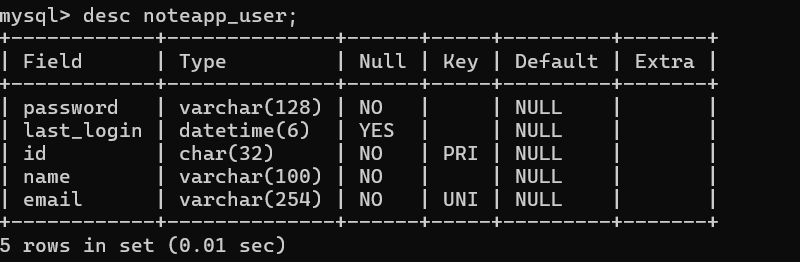
**3.7 Component Diagram**



**3.9 Data Dictionary**

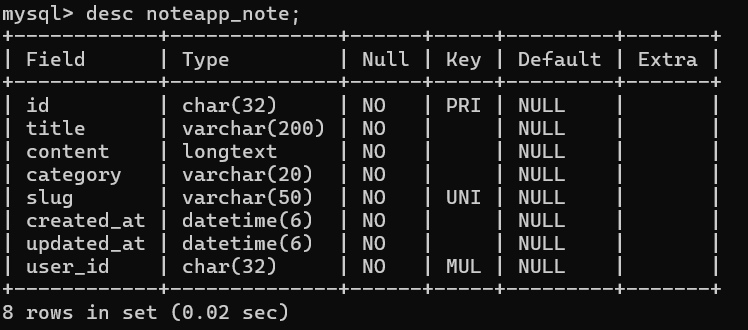
### 1. User

This store user personal and login details.

****

### ****2. Note****

This table store the details of user notes

****

**3.8 Test Procedure & Implementation**

## ****1. Manual Testing –**** Notes App

Manual testing involves testers verifying that the application behaves as expected by interacting with the UI and observing outcomes. Below are test scenarios tailored for major components of the Notes App:

**Forms (User Registration, Login, Note Creation/Update)**

* Test Case 1: Ensure all required fields (e.g., name, email, password, note title, content) are present and enabled.
* Test Case 2: Submit forms with valid inputs and verify that data is correctly saved to the database.
* Test Case 3: Enter invalid data (e.g., invalid email format, short password, empty required fields) and confirm proper error messages are shown.
* Test Case 4: Verify that field-level validations are enforced (e.g., unique email, required title, character limits).

### Notes Management (Create, View, Edit, Delete)

### Test Case 1: Create a new note and verify it appears in the dashboard with correct content.

### Test Case 2: Edit an existing note and ensure updates are saved and reflected instantly.

### Test Case 3: Delete a note and confirm it is removed from the UI and backend database.

### Test Case 4: View individual note details and check if content displays correctly with timestamps and category.

### ****Login/Logout & Session Management****

* Test Case 1: Log in with valid credentials and verify redirection to the dashboard with user-specific notes.
* Test Case 2: Attempt login with incorrect credentials and ensure appropriate error feedback is shown.
* Test Case 3: Logout and verify the session ends; user should be redirected to the login screen.
* Test Case 4: Try accessing dashboard or note pages without being logged in and confirm redirection to login page.

**2. Automated Testing**

### Automated testing was used to ensure the reliability of key functionalities such as authentication, note operations, and form validation. Tools such as Django’s TestCase, PyTest, and optionally Selenium for frontend simulation were employed.

### ****Forms (Contact Creation, User Registration, Profile Update)****

* + Test Case 1: Simulate form submissions with valid inputs and assert successful user registration, login, and note creation.

### Test Case 2: Submit forms with invalid inputs and check that:

### Proper error messages are shown.

### Data is not saved.

### Uniqueness constraints (e.g., duplicate email) are enforced.

### Note Operations (Create/Edit/Delete)

### Test Case 1: Automatically create notes using test data and verify database entries.

### Test Case 2: Update notes using automated scripts and confirm the changes are correctly reflected via assertions.

### Test Case 3: Delete notes and verify they are removed from both frontend and backend.

### Test Case 4: Filter or view notes by category and assert that results match expected output.

### Authentication Flow (Login/Logout)

* **Test Case 1:** Log in using valid credentials and assert redirection to the user dashboard.
* **Test Case 2:** Attempt login with invalid credentials and ensure errors are raised.
* **Test Case 3:** Perform logout and check that session is cleared and redirection to login page occurs.
* **Test Case 4:** Try accessing protected views after logout and assert redirection to login.

**3. Handling Invalid Inputs & Backend Validations**

### ****Forms & User Input****

* **Frontend Validation (Next.js/HTML/JavaScript/React):**
  + Basic constraints are enforced using native HTML5 input attributes and controlled React form validation:
  + Fields like **email**, **password**, **note title**, and **note content** use:
* required attribute to ensure inputs are not left blank.
* type="email" to validate email format.
* minLength/maxLength and character count restrictions.
* Real-time feedback using client-side validation for better user experience (e.g., React Hook Form or custom validation logic).
* **Backend Validation (Django Models/Forms):**

Django models and forms are leveraged to ensure robust server-side validation:

* User Registration:
* Valid email format using EmailField.
* Password strength and confirmation logic.
* Unique constraint on the email field.

Note Creation/Update:

* Title and content fields must not be empty.
* Optional fields (like category) must match predefined choices.
* Enforce user-ownership relationship for access control.

### Notes Management Validation

**Validation Rules:**

* **Note Title**: Cannot be empty and must meet a minimum length requirement.
* **Note Content:** Required and must be meaningful (non-empty, non-whitespace).
* **Category (e.g., Personal, Business, Important):** Must match predefined choices to prevent invalid category entries.
* **User Association**: Notes are user-specific and cannot be accessed or edited by unauthorized users.

**Error Handling:**

### Users receive specific error messages such as:

### “Title is required.”

### “Note content cannot be empty.”

### “You do not have permission to edit this note.”

### Validation errors from Django are rendered cleanly in the frontend forms via the API response (e.g., 400 Bad Request with error details).

### ****Login/Logout & Authorization****

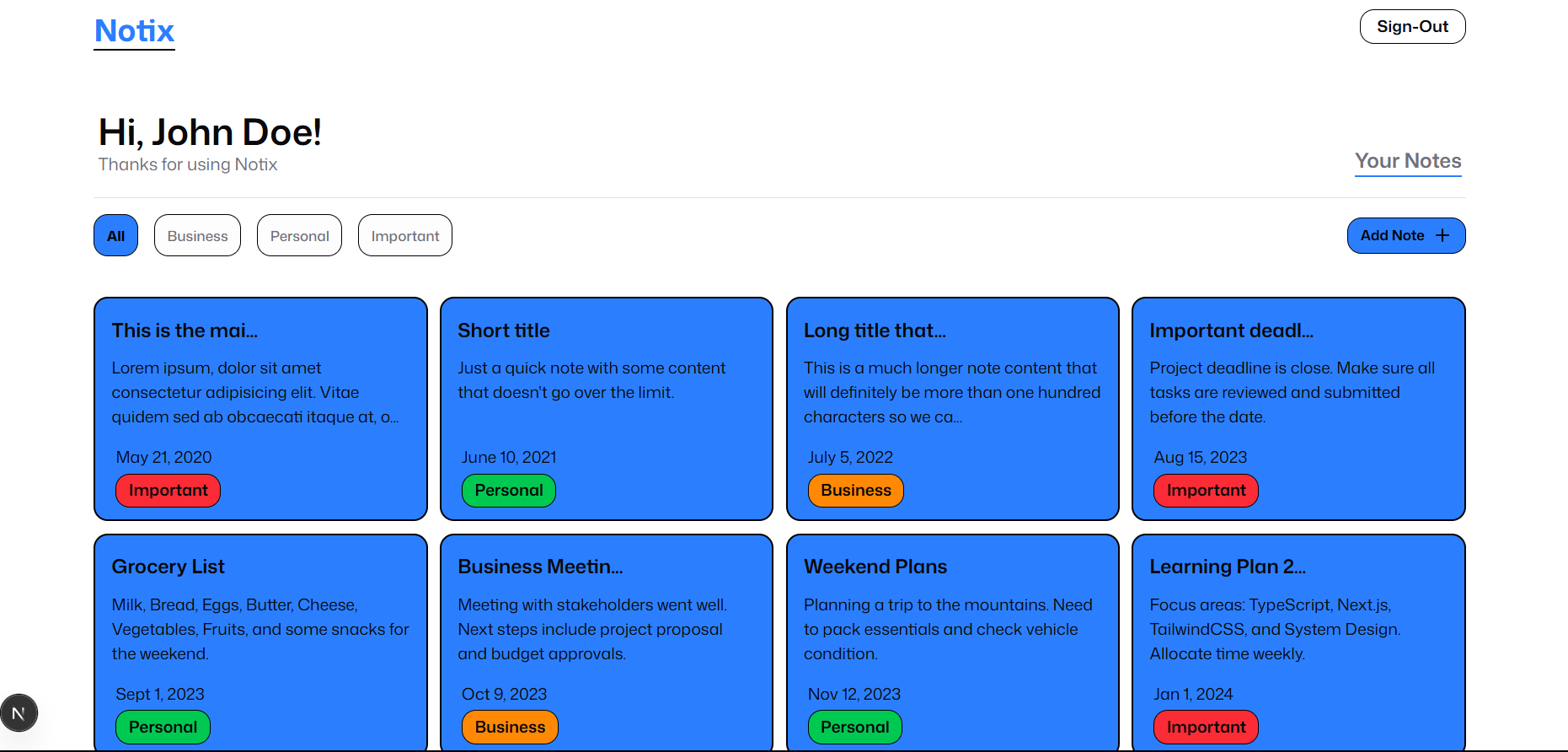
* **Login:**
  + Backend checks that:
  + The email and password combination exists and matches a registered user.
  + Incorrect credentials return clear feedback like:  
    **“Invalid email or password.”**
  + Frontend ensures email format is valid before sending a request to the server
* **Session & Permissions:**

Django sessions or token-based auth (if using JWT or session cookies) are validated on each request:

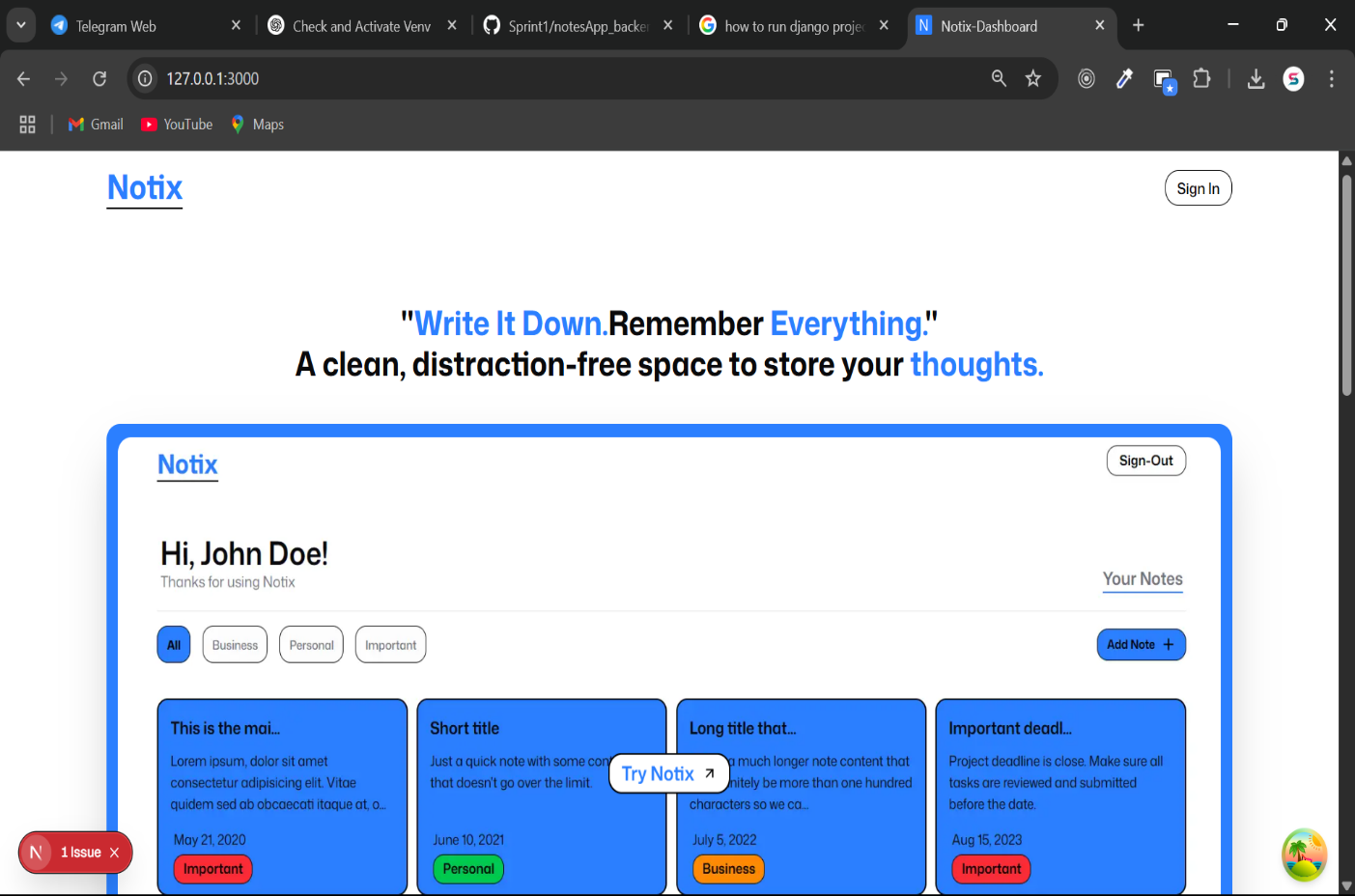
* Unauthorized access attempts to dashboard, note views, or creation endpoints are blocked with a **401 Unauthorized** response.
* Role-based or user-based access controls:
  + Notes are restricted to the currently logged-in user.
  + Attempts to edit or delete another user’s notes are denied with proper error messages.

**3.9 Screenshots**

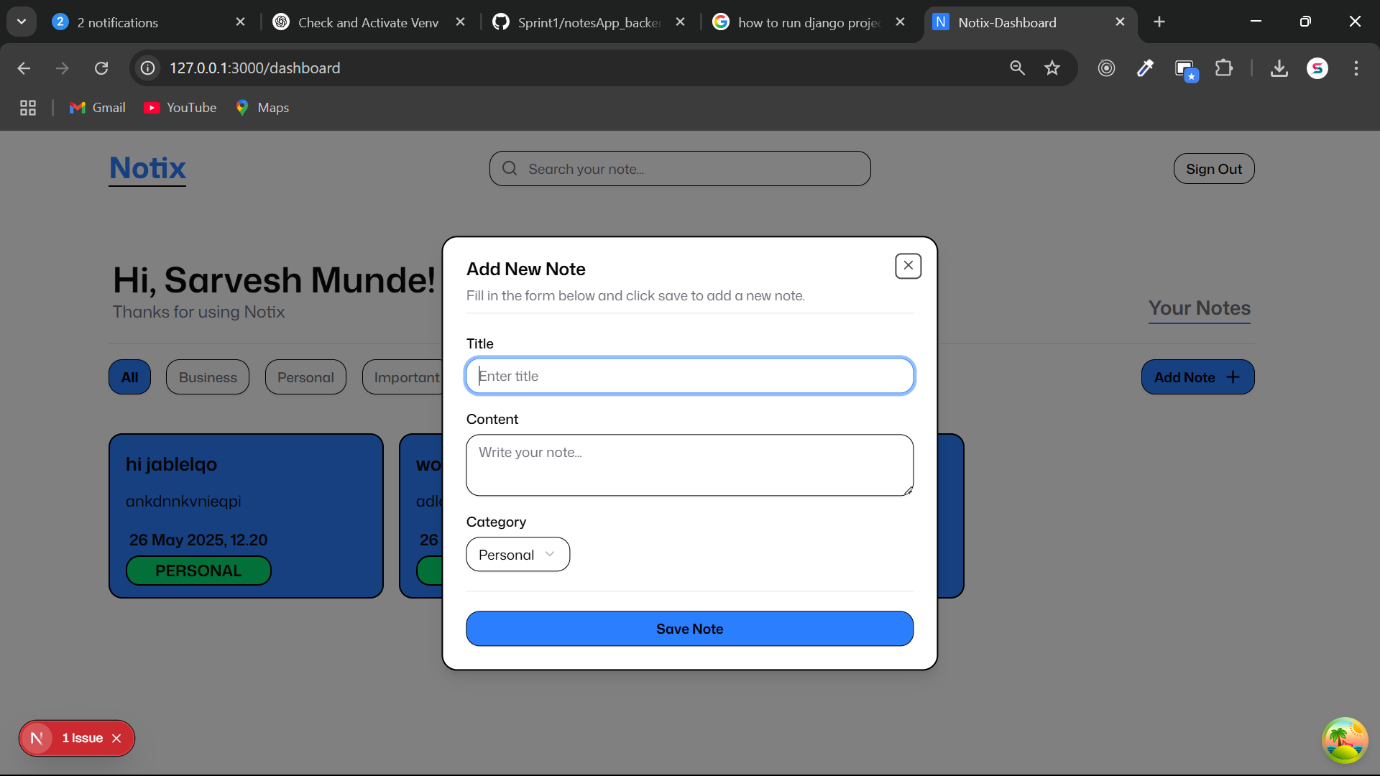
**1. Dashboard Page**

****

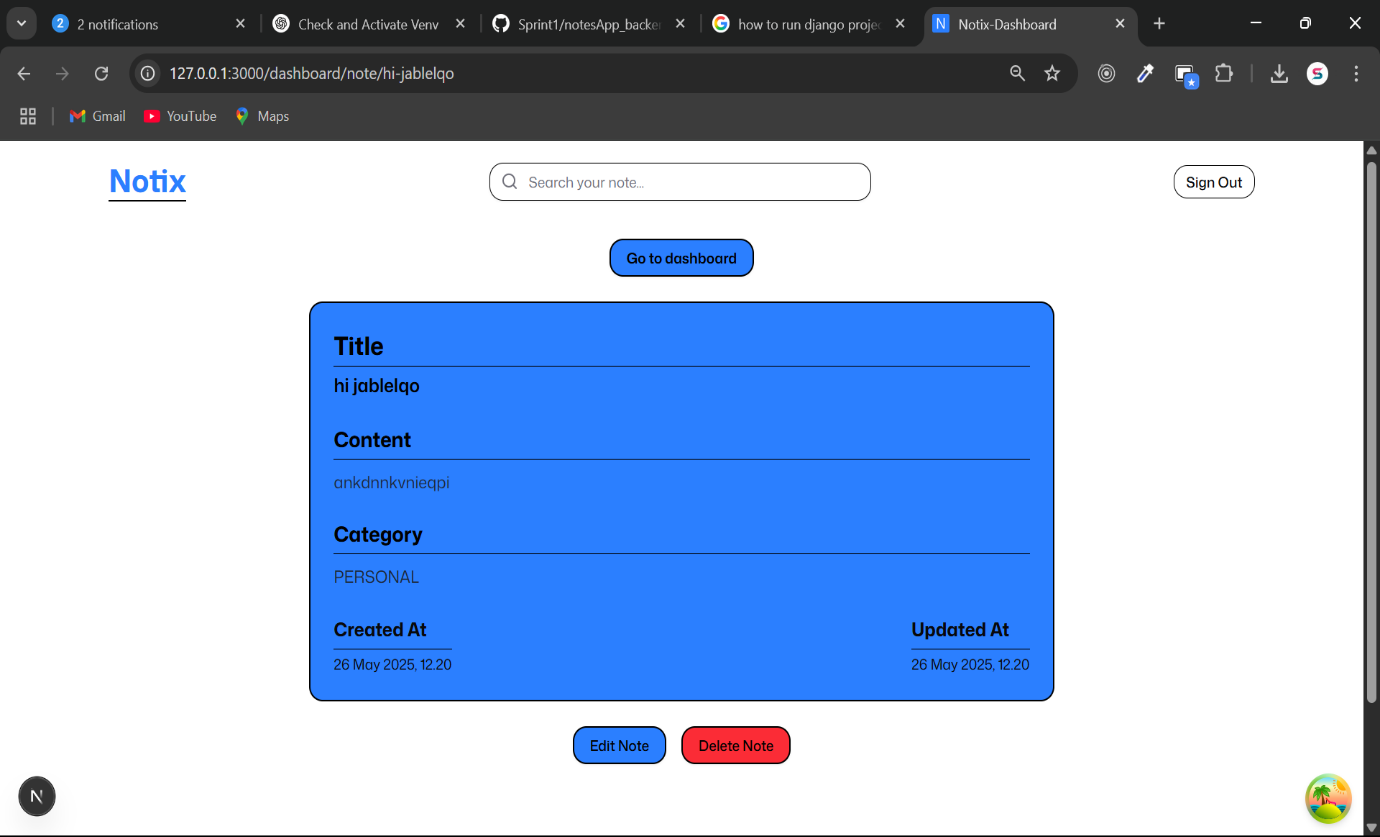
**2. Landing Page**

****

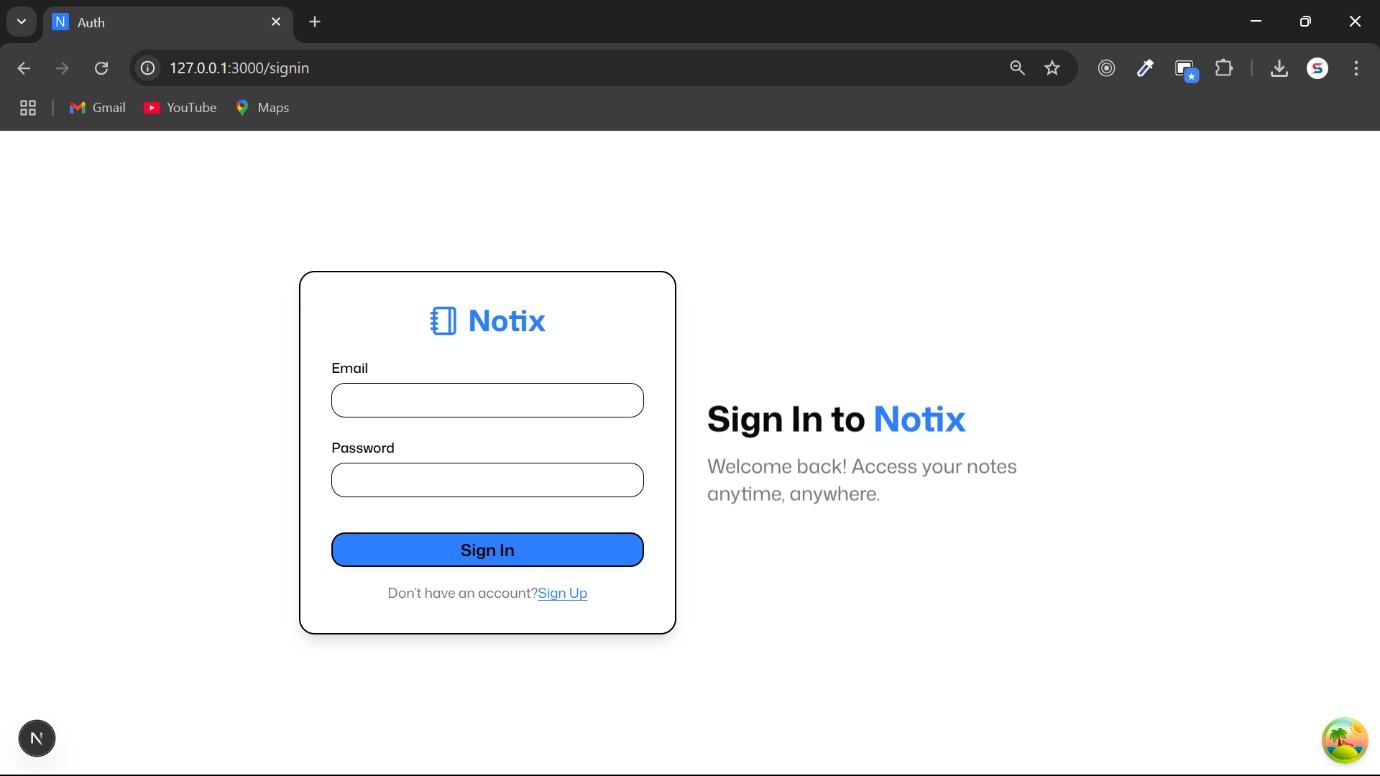
**3. Add Note**

****

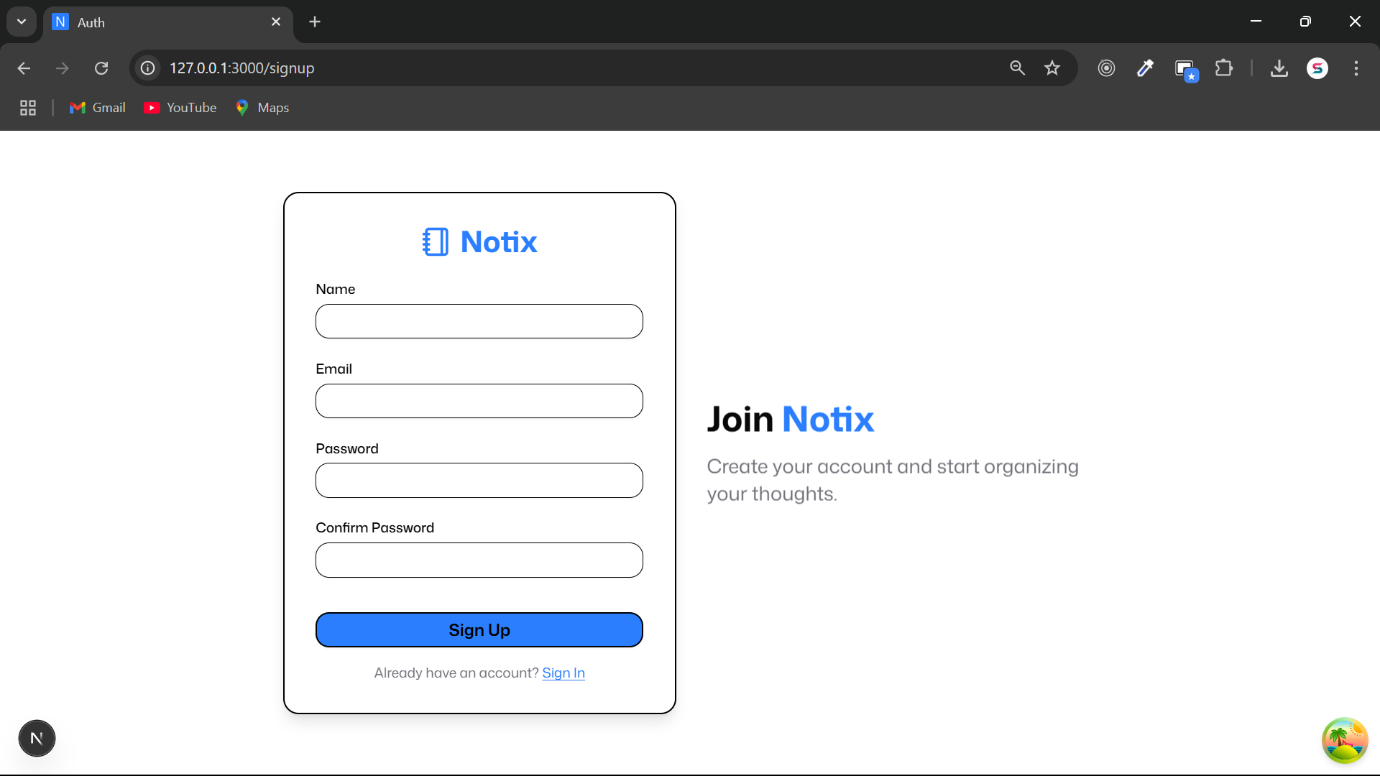
**4. Note Detail Page :**

****

**5. Sign In**

****

**6. Sign Up**

****

**3.10 Reports**

While the Notes App does not include a traditional admin dashboard or advanced reporting tools, it provides essential user-focused insights and organizational tools that function as lightweight, dynamic reports. These features empower users to monitor their note-taking habits, organize their content effectively, and maintain clarity in managing their digital notes.

**1. Category-Based Note Organization**

* Users can categorize notes under predefined or custom labels such as:
  + Personal – For daily journaling, personal goals, or reminders.
  + Business – For work-related documents, meeting summaries, and tasks.
  + Important – For critical notes, deadlines, or priority items.

Users can filter and view notes under each category, enabling structured access and improved focus.

**2. Real-Time Note Statistics**

* Although minimalistic, the app provides valuable insights through automatic count displays:
  + Total number of notes
  + Number of notes per category
  + Recently created notes
  + Recently updated notes

These metrics serve as quick summaries, helping users stay aware of their activity levels and engagement with the app.

**3. Intelligent Search Functionality**

* The integrated search system allows users to:
  + Search notes by **title**, **tags**, or **content** keywords
  + Retrieve relevant results in real-time
  + Filter results by category or creation date (optional feature)

**4. IMPLEMENTATION DETAILS**

**4.1 Software Specifications**

**Frontend:**

* **Framework:** Next.js 14+
* **Styling:** Tailwind CSS
* **State Management / Data Fetching:** React Query
* **Form Handling:** React Hook Form
* **Language:** JavaScript / TypeScript (optional)

**Backend:**

* **Language:** Python 3.9+
* **Framework:** Django 3.x (or higher)
* **Database:** MySQL
* **REST API:** Django REST Framework (DRF)

**Development Tools:**

* **IDE:** Visual Studio Code
* **Package Manager:** npm / yarn
* **API Testing:** Postman

### ****Hardware Requirements****

|  |  |
| --- | --- |
| **Component** | **Minimum Requirement** |
| **Processor:** | Intel i3 or equivalent (dual-core CPU) |
| **RAM:** | 4 GB (8 GB recommended) |
| **Hard Drive** | 5 GB free space minimum |
| **Operating System** | Windows 10 / macOS / Linux (Ubuntu preferred) |

**5. CODING**

**Frontend Main Pages :**

**1.Login /Sign-In Page**

**"use client";**

**import { signInApiCall } from "@/apiCalls/auth";**

**import { Button } from "@/components/ui/button";**

**import { Form, FormControl, FormField,FormItem,FormLabel, FormMessage,} from "@/components/ui/form";**

**import { Input } from "@/components/ui/input";**

**import useAuth from "@/hooks/useAuth";**

**import { signInSchema, SignInSchemaType } from "@/lib/zod-schemas";**

**import { zodResolver } from "@hookform/resolvers/zod";**

**import { LoaderCircle } from "lucide-react";**

**import Link from "next/link";**

**import { useState } from "react";**

**import { useForm } from "react-hook-form";**

**export default function SignInForm() {**

**const [errorMessage, setErrorMessage] = useState("");**

**const form = useForm<SignInSchemaType>({**

**resolver: zodResolver(signInSchema),**

**defaultValues: { email: "", password: "", }, });**

**const { isPending, mutate } = useAuth(signInApiCall);**

**const onSubmit = (data: SignInSchemaType) => {**

**const url = "signin/";**

**const signInData = { url, data: {**

**email: data.email,**

**password: data.password, }, };**

**mutate(signInData, {**

**onSuccess: () => {**

**form.reset(); },**

**onError: (error) => {**

**setErrorMessage(error.message); }, }); };**

**return (**

**<div className="max-w-md mx-auto">**

**<Form {...form}>**

**<form onSubmit={form.handleSubmit(onSubmit)} className="space-y-6">**

**<FormField**

**control={form.control}**

**name="email"**

**render={({ field }) => (**

**<FormItem>**

**<FormLabel>Email</FormLabel>**

**<FormControl>**

**<Input type="email" {...field} />**

**</FormControl>**

**<FormMessage />**

**</FormItem> )} />**

**<FormField**

**control={form.control}**

**name="password"**

**render={({ field }) => (**

**<FormItem>**

**<FormLabel>Password</FormLabel>**

**<FormControl>**

**<Input type="password" {...field} />**

**</FormControl>**

**<FormMessage />**

**</FormItem> )} />**

**{errorMessage && (**

**<p className="text-red-500 text-center">{errorMessage}</p> )}**

**<Button**

**type="submit"**

**className="w-full mt-3 cursor-pointer font-semibold text-[17px] p-3 text-black"**

**disabled={isPending}>**

**Sign In {isPending && <LoaderCircle className="animate-spin" />}**

**</Button>**

**</form>**

**</Form>**

**<p className="mt-4 text-center text-sm text-muted-foreground">**

**Don’t have an account?**

**<Link**

**href="/signup"**

**className="text-primary underline underline-offset-2">**

**Sign Up**

**</Link> </p> </div> );}**

**2.SignUp**

**"use client";**

**import { signUpApiCall } from "@/apiCalls/auth";**

**import { Button } from "@/components/ui/button";**

**import { Form, FormControl, FormField, FormItem, FormLabel, FormMessage,} from "@/components/ui/form";**

**import { Input } from "@/components/ui/input";**

**import useAuth from "@/hooks/useAuth";**

**import { signUpSchema, SignUpSchemaType } from "@/lib/zod-schemas";**

**import { zodResolver } from "@hookform/resolvers/zod";**

**import { LoaderCircle } from "lucide-react";**

**import Link from "next/link";**

**import { useRouter } from "next/navigation";**

**import { useState } from "react";**

**import { useForm } from "react-hook-form";**

**export default function SignUpForm() {**

**const [errorMessage, setErrorMessage] = useState("");**

**const Router = useRouter();**

**const form = useForm<SignUpSchemaType>({**

**resolver: zodResolver(signUpSchema),**

**defaultValues: { name: "", email: "", password: "", confirmPassword: "" }, });**

**const { isPending, mutate: signUp } = useAuth(signUpApiCall);**

**const onSubmit = (data: SignUpSchemaType) => {**

**const { name, email, password } = data;**

**signUp( {**

**url: "signup/",**

**data: { name, email, password }, },{**

**onSuccess: () => { form.reset(); },**

**onError: (error) => { setErrorMessage(error.message); }, });**

**form.reset(); };**

**return (**

**<div className="max-w-md mx-auto">**

**<Form {...form}>**

**<form onSubmit={form.handleSubmit(onSubmit)} className="space-y-6">**

**<FormField**

**control={form.control}**

**name="name"**

**render={({ field }) => (**

**<FormItem>**

**<FormLabel>Name</FormLabel>**

**<FormControl>**

**<Input {...field} />**

**</FormControl>**

**<FormMessage />**

**</FormItem> )} />**

**<FormField**

**control={form.control}**

**name="email"**

**render={({ field }) => (**

**<FormItem>**

**<FormLabel>Email</FormLabel>**

**<FormControl>**

**<Input type="email" {...field} />**

**</FormControl>**

**<FormMessage />**

**</FormItem> )} />**

**<FormField**

**control={form.control}**

**name="password"**

**render={({ field }) => (**

**<FormItem>**

**<FormLabel>Password</FormLabel>**

**<FormControl>**

**<Input type="password" {...field} />**

**</FormControl>**

**<FormMessage />**

**</FormItem> )} >**

**<FormField**

**control={form.control}**

**name="confirmPassword"**

**render={({ field }) => (**

**<FormItem>**

**<FormLabel>Confirm Password</FormLabel>**

**<FormControl>**

**<Input type="password" {...field} />**

**</FormControl>**

**<FormMessage />**

**</FormItem> )} />**

**{errorMessage && ( <p className="text-red-500 text-center">{errorMessage}</p> )}**

**<Button type="submit"**

**className="w-full mt-3 cursor-pointer font-semibold text-[17px] p-3 text-black"**

**disabled={isPending} >**

**Sign Up {isPending && <LoaderCircle className="animate-spin" />}**

**</Button> </form> </Form>**

**<p className="mt-4 text-center text-sm text-muted-foreground">**

**Already have an account?{" "}**

**<Link href="/signin"**

**className="text-primary underline underline-offset-2" >**

**Sign In**

**</Link> </p> </div> ); }**

**3. Notes Component :**

**"use client";**

**import { Dialog, DialogContent, DialogTrigger } from "@/components/ui/dialog";**

**import { Pencil, Plus } from "lucide-react";**

**import { memo, useMemo, useState } from "react";**

**import Loader from "./Loader";**

**import NewNoteDialog from "./NewNoteDialog";**

**import { Button } from "./ui/button";**

**import { formatDate } from "@/lib/utils";**

**import { useRouter } from "next/navigation";**

**const NOTES\_CATEGORIES = [**

**{ id: "all", label: "All" },**

**{ id: "business", label: "Business" },**

**{ id: "personal", label: "Personal" },**

**{ id: "important", label: "Important" },**

**];**

**export type Note = {**

**id: string;**

**title: string;**

**content: string;**

**created\_at: Date;**

**updated\_at: Date;**

**category: string;**

**slug: string;**

**};**

**type NotesProps = { notes: Note[]; isPending: boolean; };**

**const truncate = (text: string, limit: number): string =>**

**text.length > limit ? text.slice(0, limit) + "..." : text;**

**const getCategoryColor = (category: string): string => {**

**const map: Record<string, string> = {**

**important: "bg-red-400",**

**personal: "bg-green-400",**

**business: "bg-orange-400", };**

**return map[category.toLowerCase()] || "bg-gray-400";};**

**const NotesItem = memo( ({ label, active, onClick, disabled, }: {**

**label: string;**

**active: boolean;**

**onClick: () => void;**

**disabled: boolean; }) => (**

**<Button onClick={onClick}**

**className={`px-3 md:py-2 py-1 rounded-md border-[1.4px] border-black cursor-pointer transition-colors ${ active**

**? "bg-primary text-black font-bold"**

**: "bg-muted/20 hover:bg-muted text-muted-foreground font-medium" }`}**

**disabled={disabled} >**

**<span className="text-sm">{label}</span> </Button> ) );**

**export const NoteCard = memo(({ note }: { note: Note }) => {**

**const router = useRouter();**

**const handleClick = () => {**

**router.push(`/dashboard/note/${note.slug}`); };**

**return ( <div onClick={handleClick}**

**className="border-2 cursor-pointer transition-transform duration-300 hover:scale-[1.02] border-black p-3 bg-primary rounded-md group relative overflow-hidden flex flex-col justify-between hover:shadow-md" >**

**<div className="p-1">**

**<h3 className="font-semibold text-[1.2rem]">**

**{truncate(note.title, 15)} </h3>**

**<p className="mt-3">{truncate(note.content, 100)}</p> </div>**

**<div className="mt-3 px-1 space-x-6 flex justify-between items-center">**

**<div className="space-y-1">**

**<div className="font-semibold px-1">**

**{formatDate(note.created\_at)} </div>**

**<div className={`${getCategoryColor( note.category**

**)} font-semibold text-center rounded-md border-[1.5px] border-black px-2 py-0.5`} >**

**{note.category} </div> </div>**

**<div className="rounded-full bg-white p-2 border-[1.8px] border-black opacity-0 group-hover:opacity-100 translate-y-2 group-hover:translate-y-0 transition-all duration-300 ease-in-out">**

**<Pencil className="size-6" /> </div> </div> </div> ); });**

**const Notes = ({ notes, isPending }: NotesProps) => {**

**const [activeCategory, setActiveCategory] = useState("all");**

**const [isDialogOpen, setIsDialogOpen] = useState(false);**

**const filteredNotes = useMemo(() => {**

**return activeCategory === "all" ? notes : notes.filter(**

**(note) => note.category.toLowerCase() === activeCategory.toLowerCase() );**

**}, [notes, activeCategory]);**

**if (isPending) return <Loader />;**

**return ( <main>**

**<div className="mt-4 flex justify-between md:items-center gap-3">**

**<div className="flex flex-wrap gap-2 md:gap-4">**

**{NOTES\_CATEGORIES.map((item) => ( <NotesItem**

**key={item.id}**

**label={item.label}**

**active={item.id === activeCategory.toLowerCase()}**

**onClick={() => setActiveCategory(item.id)}**

**disabled={notes.length === 0} />))} </div>**

**<Dialog open={isDialogOpen} onOpenChange={setIsDialogOpen}>**

**<DialogTrigger asChild>**

**<Button className="gap-2 border-[1.4px] border-black font-semibold text-black">**

**<span className="hidden md:inline">Add Note</span>**

**<Plus className="size-5" /> </Button>**

**</DialogTrigger>**

**<DialogContent className="max-w-[500px] max-h-[95%] overflow-auto border-2 border-black">**

**<NewNoteDialog**

**title="Add New Note"**

**description="Fill in the form below and click save to add a new note."**

**onSuccess={() => setIsDialogOpen(false)} />**

**</DialogContent>**

**</Dialog> </div>**

**{filteredNotes.length === 0 ? (**

**<div className="text-2xl w-full text-center pt-20 text-muted-foreground">**

**{activeCategory === "all"**

**? "There are no notes created by you."**

**: `There are no notes under ${activeCategory}.`}**

**</div> ) : (**

**<div className="mt-10 grid gap-4 grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 xl:grid-cols-4">**

**{filteredNotes.map((note) => (**

**<NoteCard key={note.id} note={note} /> ))}**

**</div> )} </main> ); };**

**export default Notes;**

**Django Files :**

**Models.py:**

**# models.py**

**from django.db import models**

**from django.utils.text import slugify**

**from django.utils.crypto import get\_random\_string**

**import uuid**

**from django.contrib.auth.models import AbstractBaseUser, BaseUserManager**

**from django.contrib.auth.models import User**

**class CustomUserManager(BaseUserManager):**

**def create\_user(self, name, email, password=None):**

**if not email: raise ValueError("Email is required")**

**email = self.normalize\_email(email)**

**user = self.model(name=name, email=email)**

**user.set\_password(password)**

**user.save(using=self.\_db)**

**return user**

**class User(AbstractBaseUser):**

**id = models.UUIDField(primary\_key=True, default=uuid.uuid4, editable=False)**

**name = models.CharField(max\_length=100)**

**email = models.EmailField(unique=True)**

**objects = CustomUserManager()**

**USERNAME\_FIELD = 'email'**

**REQUIRED\_FIELDS = ['name']**

**def \_\_str\_\_(self): return self.email**

**class Note(models.Model):**

**id = models.UUIDField(primary\_key=True, default=uuid.uuid4, editable=False)**

**CATEGORY\_CHOICES = [**

**('BUSINESS', 'Business'),**

**('PERSONAL', 'Personal'),**

**('IMPORTANT', 'Important'), ]**

**user = models.ForeignKey(User, on\_delete=models.CASCADE, related\_name='notes')**

**title = models.CharField(max\_length=200)**

**content = models.TextField()**

**category = models.CharField(max\_length=20, choices=CATEGORY\_CHOICES, default='PERSONAL')**

**slug = models.SlugField(unique=True, blank=True)**

**created\_at = models.DateTimeField(auto\_now\_add=True)**

**updated\_at = models.DateTimeField(auto\_now=True)**

**def \_\_str\_\_(self): return self.title**

**def save(self, \*args, \*\*kwargs):**

**slug\_base = slugify(self.title)**

**# Generate a new slug only if it's not already set (for new note creation)**

**if not self.slug: new\_slug = slug\_base**

**while Note.objects.filter(slug=new\_slug).exists():**

**random\_suffix = get\_random\_string(5)**

**new\_slug = f'{slug\_base}-{random\_suffix}'**

**self.slug = new\_slug**

**# Call the parent class's save method to persist the note**

**super().save(\*args, \*\*kwargs)**

**urls.py**

**from django.urls import path**

**from . import views**

**urlpatterns = [**

**path('notes/search/', views.search\_note\_by\_title, name="search"),**

**path('notes/', views.notes, name='notes'),**

**path('notes/<slug:slug>/', views.note\_detail, name='note\_detail'),**

**path('signup/', views.signup, name='signup'),**

**path('signin/', views.signin, name='signin'),**

**path('logout/', views.logout, name='logout'),**

**path('profile/', views.current\_user, name='current\_user'),**

**path('notes/delete/<slug:slug>/', views.delete\_note\_by\_slug, name='delete-note-by-slug'),**

**path('notes/update/<slug:slug>/', views.update\_note\_by\_slug, name='update-note'),**

**]**

**Views.py**

**from django.conf import settings**

**from datetime import timedelta**

**from django.contrib.auth import authenticate**

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

**from rest\_framework import status**

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

**from rest\_framework.permissions import IsAuthenticated**

**from rest\_framework.response import Response**

**from rest\_framework\_simplejwt.tokens import AccessToken**

**from noteapp.models import Note**

**from noteapp.serializers import NoteSerializer**

**from django.shortcuts import get\_object\_or\_404**

**from .serializers import UserSignupSerializer, UserLoginSerializer**

**def generate\_token(user):**

**access = AccessToken.for\_user(user)**

**access.set\_exp(lifetime=timedelta(days=7))**

**return str(access)**

**def attach\_token\_cookie(response, token):**

**response.set\_cookie(**

**key='notes\_app\_token',**

**value=token,**

**httponly=True,**

**max\_age=7 \* 24 \* 60 \* 60,**

**secure=False,**

**samesite='Lax',**

**)**

**return response**

**@csrf\_exempt**

**@api\_view(['POST'])**

**@authentication\_classes([])**

**@permission\_classes([])**

**def signup(request):**

**serializer = UserSignupSerializer(data=request.data)**

**if serializer.is\_valid():**

**user = serializer.save()**

**token = generate\_token(user)**

**response = Response({**

**'user': {**

**'name': user.name,**

**'email': user.email,**

**}**

**}, status=status.HTTP\_201\_CREATED)**

**return attach\_token\_cookie(response, token)**

**return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)**

**@csrf\_exempt**

**@api\_view(['POST'])**

**@authentication\_classes([])**

**@permission\_classes([])**

**def signin(request):**

**serializer = UserLoginSerializer(data=request.data)**

**if serializer.is\_valid():**

**user = authenticate(**

**email=serializer.validated\_data['email'],**

**password=serializer.validated\_data['password']**

**)**

**if user:**

**token = generate\_token(user)**

**response = Response({**

**'user': {**

**'name': user.name,**

**'email': user.email,**

**}**

**}, status=status.HTTP\_200\_OK)**

**return attach\_token\_cookie(response, token)**

**return Response({'detail': 'Invalid credentials'}, status=status.HTTP\_401\_UNAUTHORIZED)**

**return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)**

**@api\_view(['POST'])**

**@permission\_classes([IsAuthenticated])**

**def logout(request):**

**response = Response({'detail': 'Logged out successfully'}, status=status.HTTP\_200\_OK)**

**response.delete\_cookie('notes\_app\_token')**

**return response**

**@api\_view(['GET'])**

**@permission\_classes([IsAuthenticated])**

**def current\_user(request):**

**user = request.user**

**return Response({**

**'name': user.name,**

**'email': user.email**

**})**

**@api\_view(['GET', 'POST'])**

**@permission\_classes([IsAuthenticated])**

**def notes(request):**

**if request.method == 'GET':**

**notes = Note.objects.filter(user=request.user).order\_by('-created\_at') # optional: ordering**

**serializer = NoteSerializer(notes, many=True)**

**return Response(serializer.data)**

**elif request.method == 'POST':**

**serializer = NoteSerializer(data=request.data)**

**if serializer.is\_valid():**

**serializer.save(user=request.user)**

**return Response(serializer.data, status=status.HTTP\_201\_CREATED)**

**return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)**

**@api\_view(['GET', 'PUT'])**

**@permission\_classes([IsAuthenticated])**

**def note\_detail(request, slug):**

**try:**

**note = Note.objects.get(slug=slug, user=request.user)**

**except Note.DoesNotExist:**

**return Response({'detail': 'Note not found'}, status=status.HTTP\_404\_NOT\_FOUND)**

**if request.method == 'GET':**

**return Response(NoteSerializer(note).data)**

**@api\_view(['GET'])**

**@permission\_classes([IsAuthenticated])**

**def search\_note\_by\_title(request):**

**print("SEARCH VIEW HIT")**

**query = request.query\_params.get('q')**

**print("QUERY:", query)**

**if not query:**

**return Response(**

**{"detail": "Search query parameter 'q' is required"},**

**status=status.HTTP\_400\_BAD\_REQUEST**

**)**

**notes = Note.objects.filter(title\_\_icontains=query, user=request.user)**

**if not notes.exists():**

**return Response({"detail": "No notes found"}, status=status.HTTP\_404\_NOT\_FOUND)**

**serializer = NoteSerializer(notes, many=True)**

**return Response(serializer.data, status=status.HTTP\_200\_OK)**

**@api\_view(['DELETE'])**

**@permission\_classes([IsAuthenticated])**

**def delete\_note\_by\_slug(request, slug):**

**note = get\_object\_or\_404(Note, slug=slug, user=request.user)**

**note.delete()**

**return Response({'detail': 'Note deleted successfully'}, status=status.HTTP\_204\_NO\_CONTENT)**

**@api\_view(['PATCH'])**

**@permission\_classes([IsAuthenticated])**

**def update\_note\_by\_slug(request, slug):**

**note = get\_object\_or\_404(Note, slug=slug, user=request.user)**

**serializer = NoteSerializer(note, data=request.data, partial=True)**

**if serializer.is\_valid():**

**serializer.save() # This triggers Note.save() with slug logic**

**return Response(serializer.data, status=status.HTTP\_200\_OK)**

**return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST**

**6. TESTING**

Testing ensures that the Notes App works reliably, fulfills requirements, and provides a user-friendly experience. Both black box and white box testing strategies are employed.

#### ****Black Box Testing****

### Black box testing validates the app from a user perspective—without needing to understand internal code.

### ****Input Validation****

* **Objective**: Ensure that all form inputs (e.g., note creation, profile update, registration) are properly validated both on frontend (React Hook Form) and backend (Django).
* **Test Scenarios**:
  + Enter valid note title and content → Form submits successfully, and note appears in dashboard.
  + Enter invalid data (e.g., title exceeding length limits, invalid email format in profile update) → Appropriate validation errors are shown.
  + Leave required fields (title/content) empty → Submission blocked with relevant error messages.
  + Attempt to register with an already used email → Error: “Email already registered.

### ****Login****

* **Objective**: Confirm that user authentication works as expected.
* **Test Scenarios**:
* Login with correct email & password → Redirect to dashboard, session stored.
* Login with incorrect credentials → Error message: “Invalid email or password.”
* Try accessing protected routes (e.g., /dashboard) without logging in → Redirect to login page.
* Logout and then try accessing restricted pages → Access blocked, redirect to login.

### ****Search****

* **Objective**: Ensure the contact search functionality returns accurate and relevant results.
* **Test Scenarios**:
  + Search by existing title, content, or category → Matching notes are displayed.
  + Use partial keywords (e.g., first few characters of a title) → Results are shown via fuzzy or partial matching.
  + Search with random/unrelated terms → Message like “No notes found” is shown.
  + If filtering by category is implemented → Confirm filters work correctly (e.g., only “Important” notes appear when selected).

### Note Management (Create/Edit/Delete)****Objective****:

* **Objective:  
  Ensure full CRUD functionality of a note works and is user-specific.**
* **Test Scenarios**:
  + Add a new note → Note appears in dashboard immediately.
  + Edit a note → Changes are saved and visible after update.
  + Delete a note → Note is removed from dashboard and backend.
  + Try to edit/delete notes created by other users (via API manipulation) → Backend denies with 403 or 401.

#### ****White Box Testing****

White box testing focuses on testing the internal logic, structure, and behavior of the system’s codebase. This involves validating individual functions, methods, and modules to ensure they behave as expected under various scenarios. Testing is typically performed using unit testing tools like **unittest** or **pytest** in Django.

Test Cases

**Empty Field Errors**

* Objective: Ensure that required fields (e.g., note title and content) are properly enforced.
* Scenario: Submit a note creation or registration form with blank fields.
* Expected Result: Backend validation returns errors (e.g., “Title is required”), and no data is saved to the database.

#### ****Wrong Login****

* Objective: Verify the authentication logic handles incorrect credentials correctly.
* Scenario: User attempts login with an invalid password or unregistered email.
* Expected Result: Authentication fails, and a clear error (e.g., “Invalid email or password”) is returned via the API.

#### ****Valid Contact Entry****

#### Objective: Confirm that creating a new note with valid data works correctly.

#### Scenario: Submit a note with a valid title, content, and optional category.

#### Expected Result: Note is saved successfully and returned in the note list API.

#### ****Invalid Contact Data****

* **Objective: Ensure backend validations catch improper inputs.**
* **Scenarios:**
* Submit a note with a title exceeding max length.
* Include unsupported HTML/JS in the content field (if applicable).
* Submit note with missing required fields (e.g., title or content).
* Expected Result: The API responds with clear validation messages; data is not saved

## ****7. LIMITATIONS****

### While the Notes App developed using Django (backend) and Next.js + React Query (frontend) provides essential functionality for managing and organizing personal or categorized notes, the current implementation has some limitations that may impact scalability, offline usability, or integration capabilities:

### **No Mobile App Version**

### The application is web-based and does not include a native Android or iOS app. Although the frontend is responsive and accessible via mobile browsers, the absence of a dedicated mobile application may limit usability, speed, and offline accessibility for users who primarily work from smartphones or tablets.

### **Requires Internet Connectivity**

### All features — including note creation, updating, categorization, and user authentication — require a stable internet connection. Without internet access, users are unable to interact with their notes, which limits usability in offline or low-connectivity environments.

### No Integration with External Platforms

### The Notes App functions independently and does not support integration with external services like Google Keep, Notion, Evernote, or cloud storage platforms (e.g., Google Drive, Dropbox). This restricts multi-platform synchronization and may lead to fragmented workflows for users accustomed to external productivity tools.

### No Smart Features or Analytics

Currently, the app does not include AI-powered features such as:

* + **Automatic note categorization**
  + **Smart suggestions based on usage patterns**
  + **Reminders or tagging automation**

### ****Limited Role-Based Features****

The application is designed for **single-user usage** and does not implement role-based features such as admin/editor/viewer access. This makes it less suitable for collaborative environments where multiple users might want to share, manage, or review notes under different permissions.

**8. CONCLUSION**

The **Notes App** is a web-based application developed using **Django (backend)** and **Next.js with React Query and Tailwind CSS (frontend)**, designed to simplify the way users create, organize, and manage their personal or categorized notes. By offering core functionalities like **adding**, **editing**, **deleting**, and **categorizing notes**, the system provides a clean, responsive, and efficient solution for digital note management.

Built with modern and scalable technologies, the platform ensures:

* **Seamless user experience** across devices through a responsive UI
* **Fast client-server communication** using React Query
* **Secure authentication** and data handling via Django and token-based sessions
* **Basic validations** and **search functionality** to streamline note retrieval and entry accuracy

Although the current implementation has a few limitations—such as the absence of a native mobile app, offline capabilities, and third-party integrations—it establishes a solid foundation for future enhancements. The modular and scalable architecture supports potential features like:

* Integration with productivity tools (e.g., Google Keep, Notion)
* AI-powered suggestions and categorization
* Role-based access for shared note collaboration
* Mobile app development for offline note access

In conclusion, this project delivers a practical and user-friendly system for note-taking, ideal for both personal and academic use. It serves as a modern replacement for traditional note-keeping methods, with strong potential to grow into a more intelligent and collaborative platform in the future.

**9. BIBLIOGRAPHY**

* Django Documentation – Official documentation for the Django web framework.  
  Retrieved from: <https://docs.djangoproject.com>
* Next.js Documentation – Comprehensive documentation for the Next.js React framework.  
  Retrieved from: https://nextjs.org/docs
* React Query (TanStack Query) Docs – Guide for handling data fetching and caching in React apps.  
  Retrieved from: https://tanstack.com/query/latest
* Tailwind CSS Documentation – Utility-first CSS framework used for styling.  
  Retrieved from: <https://tailwindcss.com/docs>
* Stack Overflow – Community-driven platform for resolving technical and programming issues.  
  Retrieved from: <https://stackoverflow.com>
* W3Schools – HTML, CSS, and JavaScript tutorials used for frontend design references.  
  Retrieved from: <https://www.w3schools.com>

# 10.REFERENCES

[Django Official Website](https://www.djangoproject.com/) – Backend framework used.

[Bootstrap Official Website](https://getbootstrap.com/) – Front-end styling and layout.

[MySQL Official Website](https://www.mysql.com/) – Relational database used for data storage.

Nextjs Official Website – Front-end design resources and references.