**Project TItle**

## DARSHAN EASE: YOUR EFFORTLESS PATH TO DIVINE

**Team Members :**

1.**Revathi .M(Full Stack Developer)** : Combines both frontend and backend responsibilities, ensuring smooth communication between the two. This role also handles bug fixing, feature integration, and overall system performance.

**2. Y.Phani Sikhar(Backend Developer):** Responsible for designing the user interface using **React.js**. This role focuses on ensuring a responsive, user-friendly design, as well as integrating the frontend with backend APIs.

**3. CH.Bhanu Prakash(Frontend Developer):** Develops the backend server using

**Node.js** and **Express.js**, ensuring the creation of secure, scalable RESTful APIs, as well as handling authentication, data processing, and business logic.

**4.D.Nithin Reddy(Database Administration):** Manages the **MongoDB** database,

focusing on schema design, data integrity, and database optimization to ensure efficient data storage and retrieval

**1. INTRODUCTION:**

The The "Temple Management System" leverages modern web technologies to digitize and streamline temple operations, using the MERN stack—MongoDB, Express.js, React.js, and Node.js. Temples, as vital centers of religious and cultural life, manage a range of spiritual and community activities that often require substantial administration. Traditionally, such tasks have relied on manual records and in-person interactions, which can become cumbersome and error-prone as temple activities grow. The Temple Management System addresses these challenges by digitizing essential processes like ritual scheduling, donation management, and event coordination, offering a more accessible and efficient experience for both devotees and temple administrators.

**2. PROJECT IDEA:**

This web-based system allows devotees to easily book rituals, donate, and receive updates online. By providing an intuitive, centralized interface, devotees can schedule pujas, make contributions, and stay informed about temple events from anywhere, streamlining their connection with the temple community. Meanwhile, administrators gain significant efficiencies as they handle multiple bookings, donations, and event schedules digitally, allowing them to focus more on enhancing the spiritual experience rather than managing logistics. This system not only reduces administrative overhead but also ensures that temples can meet the needs of their devotees more effectively.

One major innovation is the system’s donation module, which facilitates online transactions and generates instant receipts, making the process transparent and error-free. By digitizing donations, the system ensures that funds are accurately tracked and allows for secure, convenient contributions. This data-driven approach to finance management also assists administrators with budgeting and planning, enabling them to allocate resources more strategically based on insights from donation patterns and trends.

**MOTIVATION OF THE IDEA:**

The motivation behind developing a Temple Management System using the MERN stack stems from the desire to modernize temple operations, improve accessibility for devotees, and promote seamless digital engagement within religious communities. With the increasing adoption of digital solutions in various sectors, temples and other religious institutions can benefit significantly from an organized, efficient, and user-friendly management platform. Here are some specific motivations for this project Streamlining Administrative Tasks, Enhanced Devotee Engagement and Accessibility, Efficient Donation Management and Transparency, Fostering a Digital Community for Spiritual Growth, Modernization and Appeal to Younger Generations.

**Goals:**

**1.Streamline and Modernize Temple Operations:** Enable temple administrators and staff to efficiently manage events, donations, volunteer coordination, and puja bookings through a centralized digital platform.

**2.Enhance Devotee Engagement and Accessibility:** Provide devotees with online access to temple services, event information, and donation options, creating a more accessible and inclusive experience for local and distant devotees alike.

**3.Ensure Transparency and Accountability:** Offer transparent and secure donation tracking, along with clear communication of temple activities and fund utilization, to build trust with devotees and foster a sense of community.

**Objectives:**

**User Management and Role-Based Access Control:**

* + - Implement role-based access control (e.g., Admin, Priest, Volunteer, Devotee) to ensure that users have appropriate access to system functions.
    - Enable secure login and authentication processes using JWT, along with user account management features for administrators.

**Donation Tracking and Management:**

* + - Provide an easy-to-use, secure platform for one-time and recurring donations, integrated with popular payment gateways (e.g., Stripe, PayPal).
    - Ensure that devotees receive digital receipts and records of their donations, adding a layer of transparency to all contributions.
    - **Event and Puja Management:**

* + - Develop a system for creating, managing, and scheduling temple events and puja services.

* + - Implement a booking feature that allows devotees to reserve puja services online, with secure payment options for booking fees.

* + - Enable event RSVPs and provide a calendar view for devotees to easily access upcoming events and schedules.

**Mobile Responsiveness and User-Friendly Interface:**

* + - * Design a responsive web interface that works seamlessly on both desktop and mobile devices.

* + - * Ensure that the user interface is intuitive and user-friendly, enabling users of all backgrounds to navigate the platform with ease.

**3.ARCHITECTURE:**

The Temple Management System (TMS) is designed as a scalable, secure, and user-friendly web application using the MERN stack (MongoDB, Express.js, React, and Node.js). Below is a high-level architecture overview of TMS, including the core components, services, and flow of data between the frontend, backend, and database.

**Overall Architecture Overview:**

**TMS follows a three-tier architecture consisting of:**

* + **Frontend (Client Layer)**: Built using React.js for user interaction.
  + **Backend (Application Layer):** Managed by Express.js and Node.js to handle application logic, API requests, and data processing.
  + **Database Layer:** MongoDB is used as the NoSQL database for efficient data storage and retrieval. The system also integrates several external services (e.g., payment gateway, notification service) and ensures security through authentication and access control.

**Core Components of the System Architecture:**

**a. Frontend (Client Layer):**

* + - React.js Application: The user interface is built with React, offering a responsive and interactive experience for various user roles (Admin, Devotee, Priest, Volunteer). Each user role has different access levels and views based on permissions.
    - Component-Based UI: The application is structured with reusable components for features like event display, donation forms, booking modules, volunteer schedules, and dashboards.
    - State Management: Redux (or Context API) is used to manage application state across components, ensuring smooth data handling and user experience.
    - API Communication: Axios or Fetch API is used to communicate with the backend, handling requests for data (e.g., user details, event information, donation records).

1. **Backend (Application Layer):** 
   * + Node.js and Express.js Server: The backend server manages application logic, routes, and business processes. It serves as the intermediary between the frontend and the database, processing requests, executing business logic, and delivering responses to the frontend.
     + RESTful API: Exposes RESTful endpoints for CRUD operations related to events, donations, bookings, users, and notifications.
     + JWT Authentication: JSON Web Tokens (JWT) are used to securely authenticate users and manage sessions, reducing the risk of unauthorized access.
     + Middleware: Custom middleware functions handle request validation, logging, and error handling to ensure data integrity and maintain server performance.

Folder Structure:

**backend/**

│── server.js # Main entry point

│── config/ # Configuration files (DB connection, environment variables)

│── models/ # Mongoose schemas for User, Doctor, Appointment

│── routes/ # Express route handlers

│── controllers/ # Business logic for handling API requests

│── middleware/ # Authentication, error handling, logging

│── package.json # Project dependencies

│── .env # Environment variables

1. **Database Layer:** 
   * + MongoDB Database: Stores information about users, events, donations, bookings, and volunteer records in a document-oriented format. MongoDB is chosen for its flexibility and scalability, allowing for the efficient storage of hierarchical data.
     + Collections:
       - Users Collection: Stores user information, including roles, personal details, and access permissions.
       - Events Collection: Stores details about temple events, pujas, and ceremonies, including date, time, and booking availability.
       - Donations Collection: Tracks donation transactions, donor details, and receipt information. o Bookings Collection: Contains records of all puja and service bookings made by devotees.
       - Volunteer Collection: Manages volunteer schedules, roles, and assigned tasks.

**4. Setup Instructions**

To develop a full-stack web application for a Doctor Appointment Booking System using React.js, Node.js, and MongoDB, several essential prerequisites must be installed. These include setting up Node.js and npm for backend development, installing MongoDB for database management, and configuring React.js for the frontend. Additionally, necessary dependencies such as Express.js for API handling, Mongoose for database interaction, and React Router for navigation should be installed. Proper setup of authentication, state management, and UI libraries will ensure a smooth and efficient development process.Node.js and npm.

Node.js is essential for running JavaScript on the server side, and npm (Node Package Manager) is used for managing project dependencies.

**Download Node.js**: Download Node.js

* 1. **Installation Instructions**: Install Node.js via Package Manager
  2. **a.MongoDB :**

MongoDB is the NoSQL database used to store data such as users, pooja schedules,Donations. You can either install MongoDB locally or use a cloud-based MongoDB service like MongoDB Atlas.

**Download MongoDB**: Download MongoDB Community Edition

**Installation Instructions**: MongoDB Installation Guide

**b.MongoDB**

MongoDB is the NoSQL database used to store data such as users, pooja schedules, Donations. You can either install MongoDB locally or use a cloud-based MongoDB service like MongoDB Atlas.

**Download MongoDB**: Download MongoDB Community Edition

**Installation Instructions**: MongoDB Installation Guide

**C.Express.js**

Express.js is a web framework for Node.js that simplifies server-side development by providing tools for routing, middleware, and API development.

Express.js **Install Express.js:** Open your terminal or command prompt and run the following command:

npm install express

**D.React.js**

React.js is the JavaScript library used to build the frontend user interface. React enables the development of dynamic, component-based applications that allow for fast and responsive user experiences.

**Steps to Set Up React:**

1. **Create a New React Project:**

Install the Create React App tool, which sets up a new project with all required configurations: npx create-react-app client

**2.Navigate to the Project Directory:** cd client

**3.Start the React Development Server**:

Launch the development server by running: npm run dev

Open your browser and go to http://localhost:3000 to view your running React app.

4. **HTML, CSS, and JavaScript**

You need a solid understanding of HTML and CSS for structuring and styling the user interface, and JavaScript to handle client-side logic and interactivity in React.

**5.Database Connectivity with Mongoose** **:**

* 1. Use Mongoose, an Object-Document Mapping (ODM) library for MongoDB, to connect your Node.js server with the database and perform CRUD (Create, Read, Update, Delete) operations.
  2. **Install Mongoose:** npm install mongoose
  3. **Mongoose Documentation:** https://mongoosejs.com/docs/api/document.html
  4. Use Mongoose, an Object-Document Mapping (ODM) library for MongoDB, to connect your Node.js server with the database and perform CRUD (Create, Read, Update, Delete) operations.

**6.Version Control with Git**

Use Git for version control, enabling collaboration, and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

Download Git: Download GIT

**Visual Studio Code**: Download VS Code

**Sublime Text**: Download Sublime Text

**WebStorm:** Download WebStorm

**5. Folder Structure**

**The core structure of the React frontend project typically looks like this:**

**/frontend**

├── /node\_modules

├── /public

├── /src

│ ├── /assets

│ ├── /components

│ ├── /context

│ ├── /pages

├── App.jsx

├── index.css

├── main.jsx

├── index.js

├── index.css

├── package-lock,js

├── package.json

└── .env

**The core structure of the React Backend project typically looks like this:**

**backend/**

│── config/

│── models/

│── routes/

│── controllers/

│── middleware/

├── package-lock,js

│── package.json

│── .env

│── server.js

**8.Authentication:**

To implement user authentication and authorization, we'll need to create a login and registration system. Users should be able to sign up for an account, enter their personal information, and create a username and password. Once they've registered, they should be able to log in and access the website's features. To secure the website, we can use JWT to authenticate users and control access to sensitive data. When a user logs in, the server will generate a JWT token containing their user ID and any relevant permissions. This token can then be sent to the client-side and stored in a cookie or local storage. On subsequent requests, the client-side will send the token to the server, which will use it to verify the user's identity and grant access to protected routes.By implementing user authentication and authorization, we can ensure that only authorized users can access sensitive data on our e-book website. This helps to protect user privacy and prevent unauthorized access to personal information or payment details.

**9.User Interface**



**10.TESTING :**

Testing is a critical phase in the development of Darshan Ease, ensuring the platform’s functionality, performance, security, and usability. By thoroughly testing each component, we verify that users can submit, track, and resolve complaints seamlessly. Testing helps identify and address any potential issues before deployment, maintaining a high standard of reliability and user satisfaction**.**

**PURPOSSE OF TESTING:**

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub- assemblies, assemblies and/or a finished product It is the process of exercising of ware with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

**UNIT TESTING:**

Unit testing is crucial for ensuring that each component of the Temple Management System (TMS) works as expected. Here’s a breakdown of how unit tests can be structured for key modules in the TMS. Testing each module individually can catch bugs early and improve the system's overall reliability. Below is a sample outline for unit tests for various modules of the system.

1. **User Authentication and Authorization:**

**Objective**: Verify secure and accurate login, registration, and role-based access.

**Features to Test**:

* + Test with valid and invalid inputs (e.g., missing email, short passwords).
  + Verify valid and invalid credentials, and token generation on success.
  + Ensure users with different roles (e.g., admin, devotee) can access only permitted features.

1. **Event Management:**

**Objective**: Enable creation, update, and deletion of events with appropriate admin access.

**Features to Tests:**

* + - * Test for valid data inputs and admin permissions.
      * Check if admins can update event details, while other roles are restricted.
      * Ensure only admins can delete events and verify correct removal from the database.
      * Test that users can fetch events and see correct details.

1. **Puja Booking System:**

**Objective**: Allow devotees to book puja slots with payment verification and booking confirmation.

**Features to Tests**:

* + Test for successful booking with valid data and payment details.
  + Test booking for the same slot by multiple users to verify conflict handling.
  + Ensure users can cancel bookings and confirm the system updates the status.
  + Test various payment scenarios, including valid, invalid, and duplicate transactions.

1. **Donation Management**

**Objective**: Enable one-time and recurring donations, ensuring transaction accuracy and receipt generation.

**Features to Tests**:

* + Validate successful and failed transactions, including retries for failed payments.
  + Test setting up recurring donations and ensure the system schedules them correctly.
  + Ensure that receipts are created, emailed, and accessible after each donation.
  + Verify that donations are accurately recorded in the database.

**7.1.2 INTEGRATION TESTING:**

Integration testing for a Temple Management System (TMS) is essential for verifying that individual modules and services work together cohesively. This testing ensures the end-to-end workflow operates correctly, as data moves across various modules like user management, event handling, donation processing, and notifications.

1. **User Authentication and Authorization with Role-Based Access:**

**Objective:** Test seamless integration between authentication, authorization, and access control for user roles.

**Integration Tests:**

* + **Login and Access Control:** 
    - * **Scenario:** Log in as a user with different roles (e.g., admin, devotee, volunteer) and attempt to access different modules.
      * **Expected Outcome:** Access should be granted or denied based on roles, and a valid token should be passed to protect secure routes.
  + **Session Persistence:** 
    - * **Scenario:** Verify that authenticated sessions remain valid across modules such as donation, event booking, and profile management.
      * Ex**pected Outcome:** Token-based session remains active across requests.

1. **Event Management and Notifications:**

**Objective:** Ensure event creation, updating, and notification systems work together to inform users of new events.

**Integration Tests:**

o **Event Creation and Notification:**

* + - * + **Scenario:** Admin creates a new event, triggering a notification to all users.
        + **Expected Outcome:** The event is saved in the database, and notifications are sent via email or SMS to users who have opted in for event updates.

o **Event Reminders:**

* + - * + **Scenario:** A day before the event, the system should send reminders to all users who RSVP’d.
        + **Expected Outcome:** Only users who registered for the event receive reminders, ensuring proper data flow from booking to notification.

1. **Puja Booking System with Payment Gateway:**

**Objective:** Verify the entire booking workflow, from booking to payment confirmation, works smoothly.

**Integration Tests:**

o Puja Booking and Payment Processing:

* + - * + **Scenario:** A user books a puja and completes payment through the integrated payment

gateway.

* + - * + **Expected Outcome:** Booking details are recorded, payment status is verified, and a confirmation message is sent to the user.

o **Booking Conflict Handling:**

* + - * + **Scenario:** Two users attempt to book the same time slot.
        + **Expected Outcome:** Only one user’s booking is processed, and the other user receives an error message about unavailability.

o **Payment Failure Recovery:**

* + - * + **Scenario:** User attempts a booking, and the payment fails midway.
        + **Expected Outcome:** Booking remains pending, and user receives an error with an option to retry payment.

1. **Donation Management with Receipt Generation:**

**Objective:** Ensure donations are processed correctly, receipts are generated, and donor data is recorded in the database.

**Integration Tests:**

o **Donation Processing and Receipt Generation:**

**Scenario:** User makes a donation, and the payment is processed.

**Expected Outcome:** A donation receipt is generated, recorded in the user’s profile, and emailed to the user. o **Recurring Donation Setup:**

**Scenario:** User sets up a recurring donation.

**Expected Outcome:** Donation is processed successfully on the recurring schedule, and receipts are generated after each donation. o **Donation Reports:**

**Scenario:** Admin generates a report based on donation history.

**Expected Outcome:** Report accurately displays all donations within the specified range, ensuring data integrity across modules.

**7.1.3 USER INTERFACE TESTING:**

User Interface (UI) testing for a Temple Management System (TMS) is essential to ensure that the interface is user-friendly, intuitive, responsive, and consistent across different devices and screen sizes. This type of testing involves evaluating the visual aspects, functionality, accessibility, and performance of the UI elements that users interact with.

1. **Login and Registration Screens:**

**Objective:** Ensure the login and registration screens are intuitive, responsive, and guide users through successful account creation or login.

**UI Tests:**

o **Responsive Design:**

* + - * + **Scenario:** Open login and registration screens on various screen sizes (mobile, tablet, desktop).
        + **Expected Outcome:** All fields, buttons, and text are readable and properly aligned, and the design adjusts responsively.

o **Error Handling:**

* + - * + **Scenario:** Attempt to register/login with invalid data (e.g., empty fields, invalid email format).
        + **Expected Outcome:** Appropriate error messages appear next to the invalid fields, guiding the user to correct the input.

1. **Dashboard:**

**Objective:** Verify the dashboard displays all necessary information in an organized manner and that interactions are seamless.

**UI Tests:**

o **Data Display and Layout:**

* + - * + **Scenario:** Log in as an admin, devotee, or volunteer and view the dashboard.
        + **Expected Outcome:** Each user role sees relevant information (e.g., event summary for devotees, volunteer tasks, and management options for admins) arranged in a user-friendly layout.

o **Interactive Elements:**

* + - * + **Scenario:** Click on dashboard elements such as event cards, recent donation summaries, and upcoming bookings.
        + **Expected Outcome:** Interactive elements open detailed views, update appropriately, and provide clear feedback (like loading indicators). o **Responsiveness and Scaling:**
        + **Scenario:** Test the dashboard on different devices.
        + **Expected Outcome:** Dashboard adjusts to fit screen sizes, with content rearranging effectively on smaller screens.

1. **Event Management:**

**Objective:** Ensure that creating, updating, and viewing events is straightforward and error-free.

**UI Tests:**

o **Event Creation Form:**

* + - * + **Scenario:** Admin adds a new event with valid and invalid inputs.
        + **Expected Outcome:** Form accepts valid inputs, displays error messages for invalid entries, and provides feedback upon successful submission. o **Event Calendar and List View:**
        + **Scenario:** View events in both calendar and list views.
        + **Expected Outcome:** Events are displayed accurately in both formats, and users can switch between views easily.

o **Event Details Modal:**

* + - * + **Scenario:** Click on an event in the list or calendar.
        + **Expected Outcome:** A modal or new page opens, displaying detailed information about the event with options to register, donate, or volunteer if available.

1. **Puja Booking Interface:**

**Objective:** Ensure the puja booking interface is easy to navigate and prevents booking conflicts.

**UI Tests:**

o **Booking Form Completion:**

* + - * + **Scenario:** User fills out the booking form with valid and invalid inputs.
        + **Expected Outcome:** Form validates required fields, displays error messages for missing or invalid data, and shows a confirmation message upon successful booking..

o **Booking Confirmation:**

* + - * + **Scenario:** User completes the booking process.
        + **Expected Outcome:** A confirmation message appears, including a summary of the booking details and an option to view booking history.

1. **Donation Page:**

**Objective:** Test donation functionality, ensuring it is simple and secure.

**UI Tests:**

o **Donation Form:**

* + - * + **Scenario:** Fill out the donation form with varying donation amounts and payment methods.
        + **Expected Outcome:** Form accepts valid payment information, shows error messages for invalid inputs, and directs to a confirmation page on success.

o **Success Page:**

* + - * + **Scenario:** Complete a donation.
        + **Expected Outcome:** A confirmation page appears with a message, receipt link, and option to view donation history.

**7.2 Known Issues:**

* + 1. **ACCEPTANCE TESTING:**

User Acceptance Testing is acritical phase of any project and requires significant participation by the end user .It also ensures that the system meets the functional requirements.

* + 1. **RESULTS:**

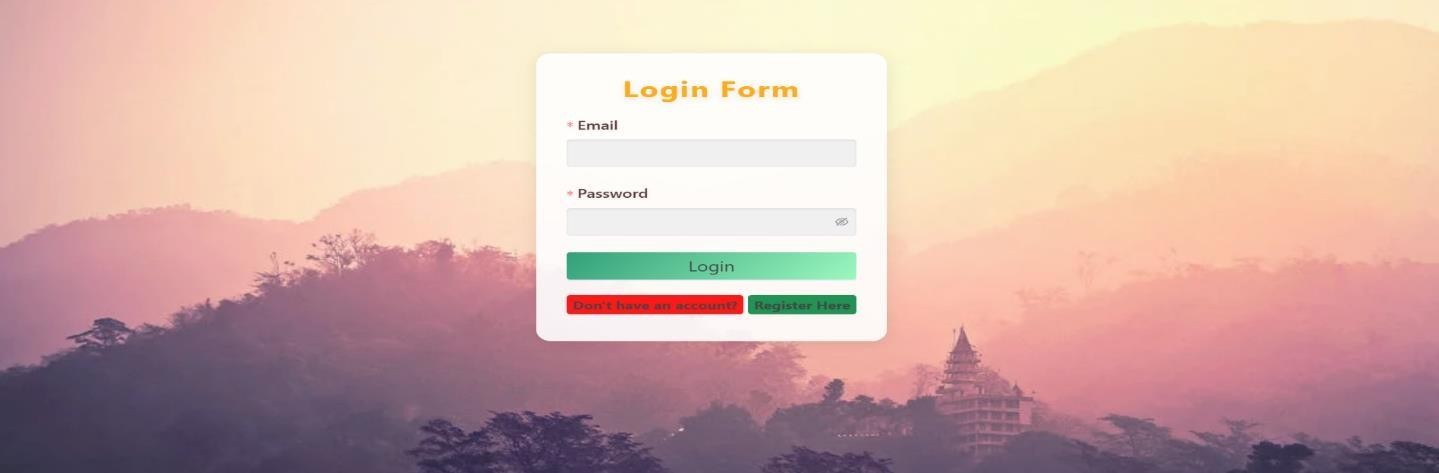
The expected outcome from testing the Book Nest project includes:

Successful execution of all user workflows with no critical bugs.

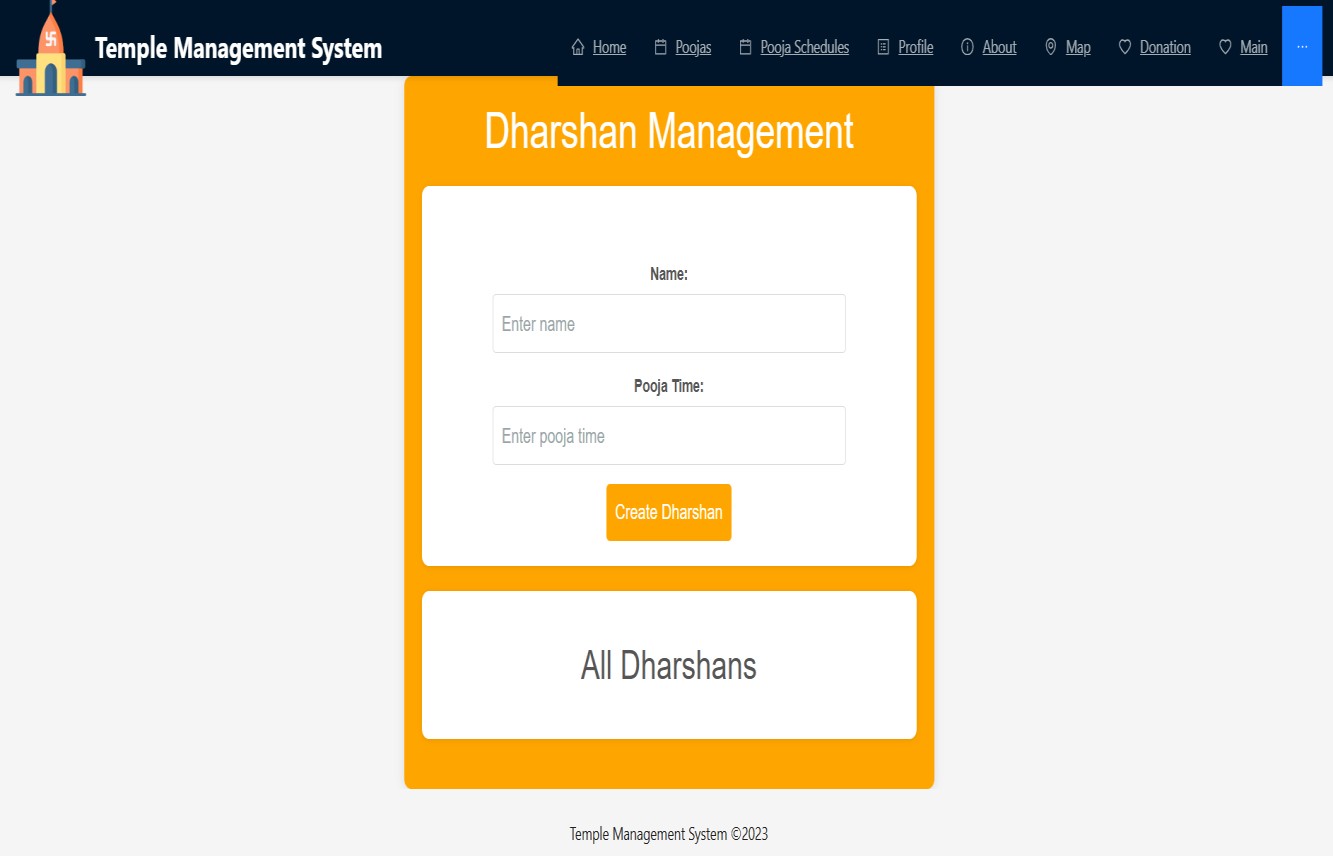
Smooth interactions between the frontend, backend, and database.

Role-based access functioning correctly.

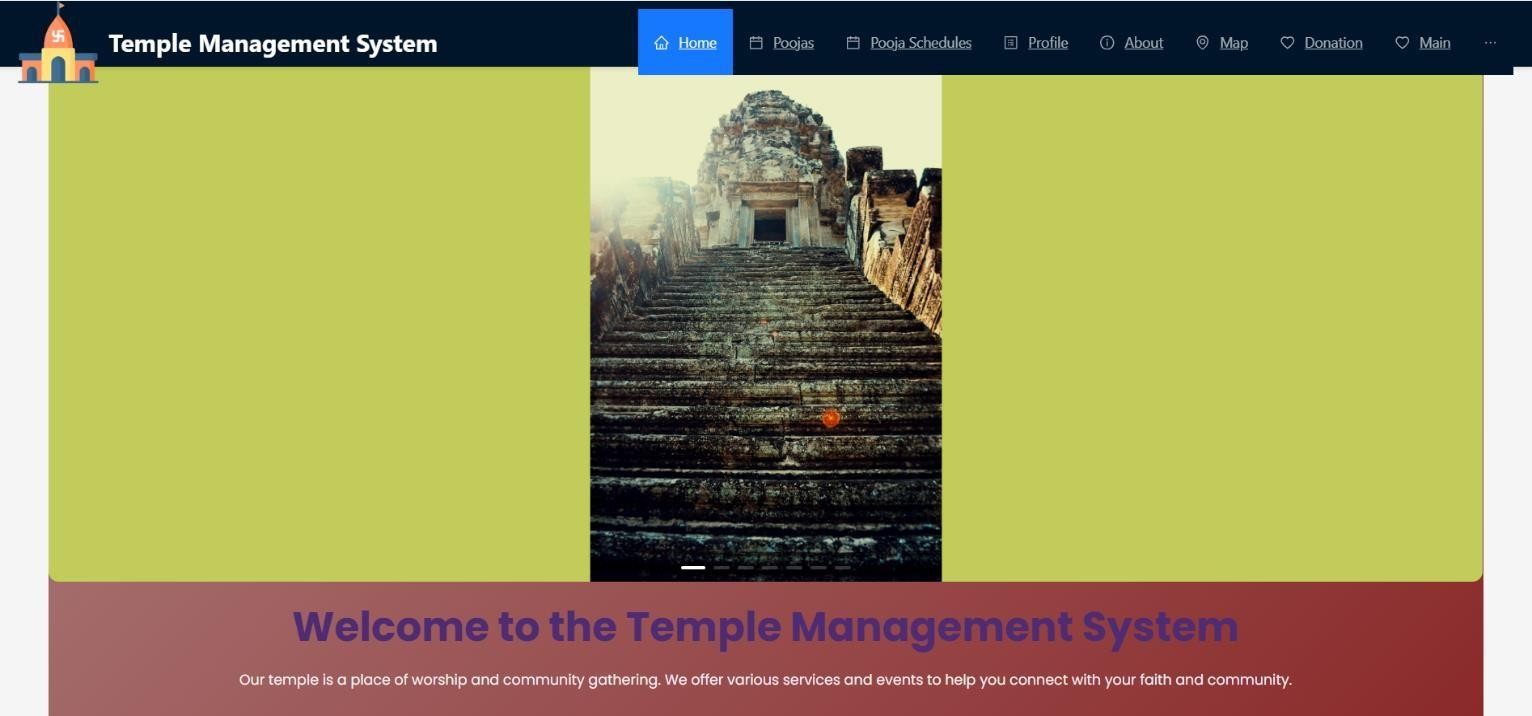
**11. OUTPUT SCREENS**



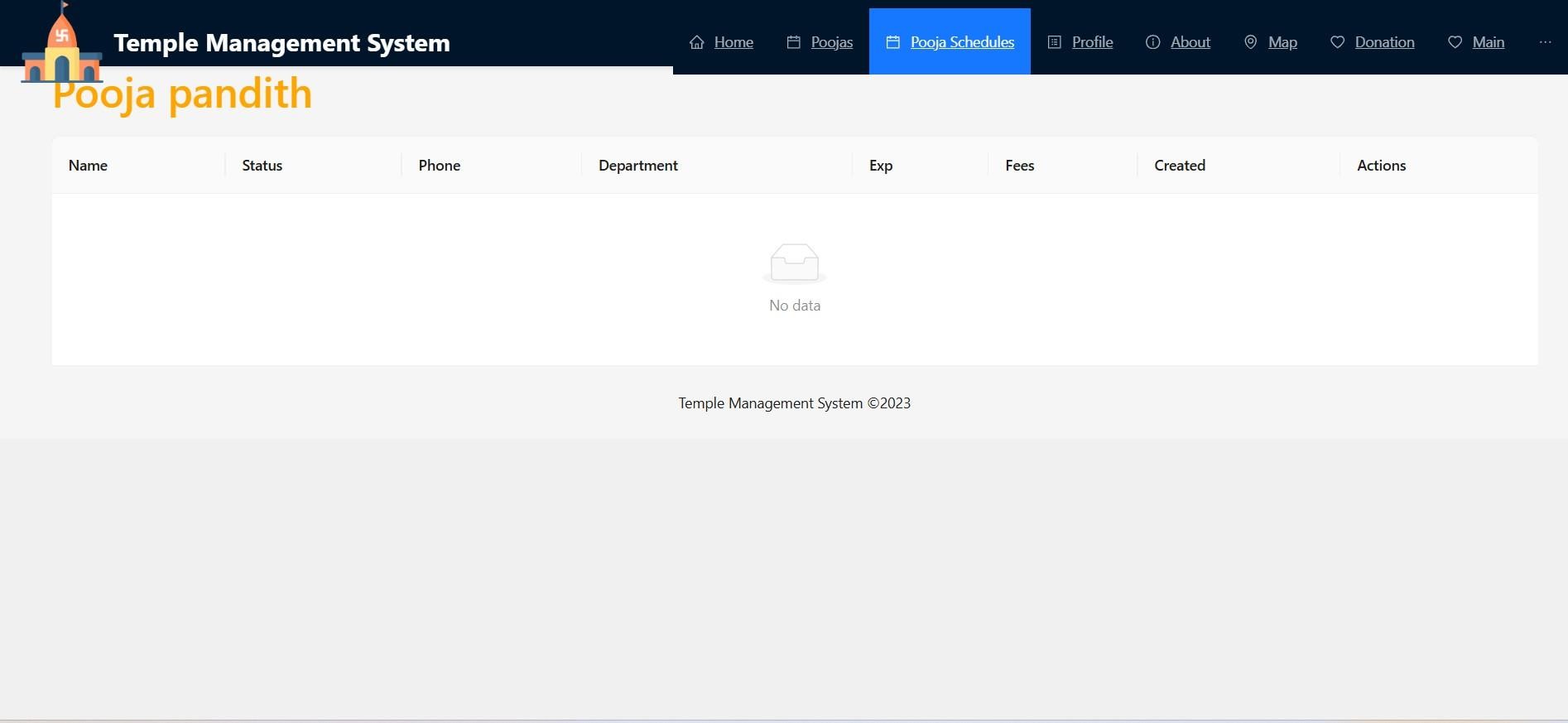
**11.1LOGIN PAGE**



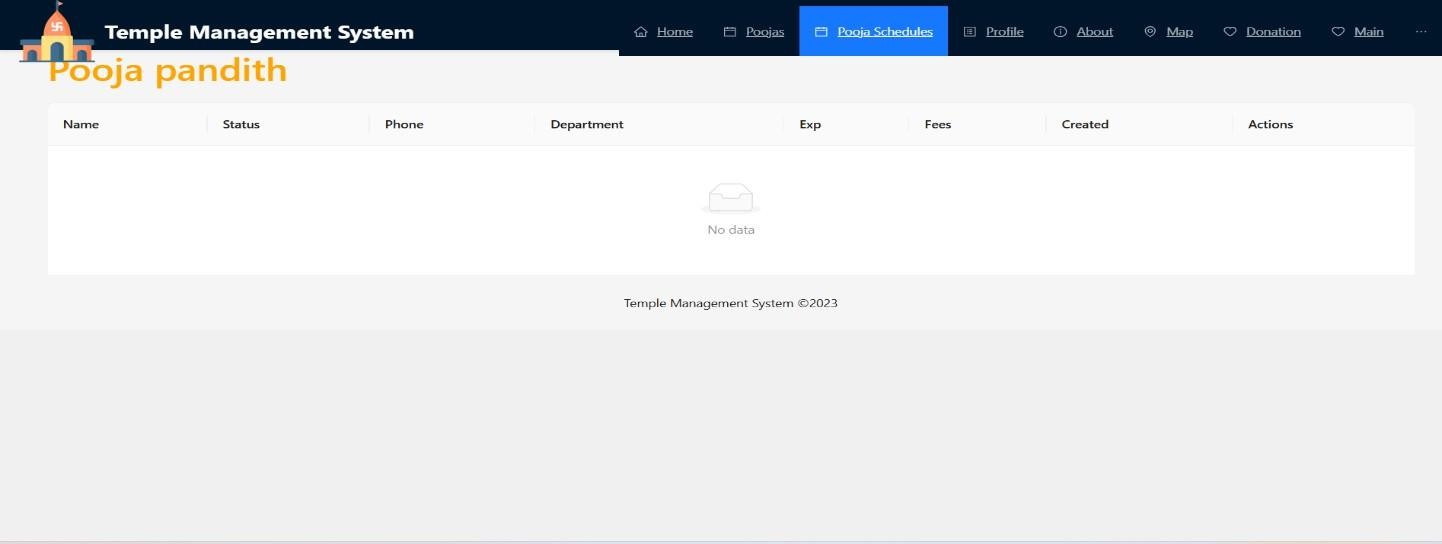
* 1. **DARSHAN PAGE**



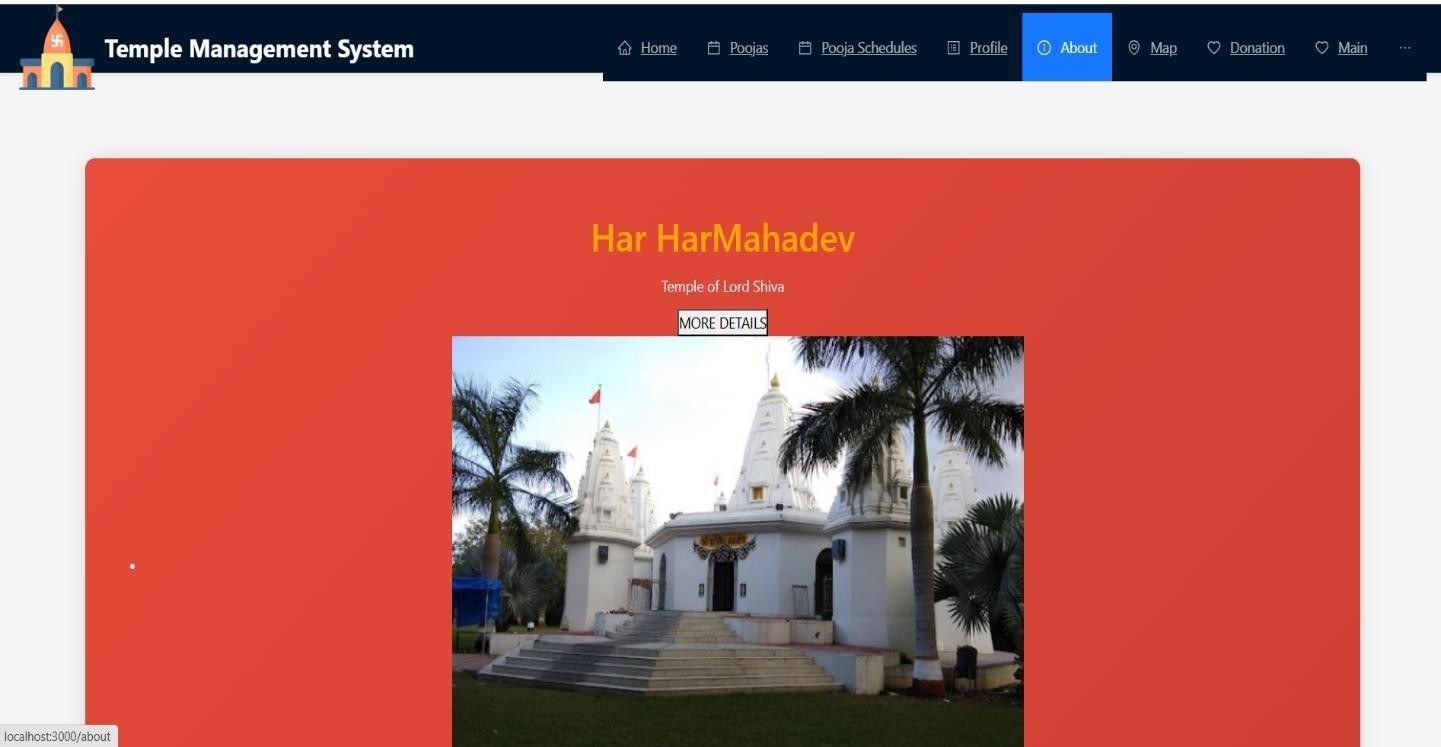
### 8.3 HOME PAGE



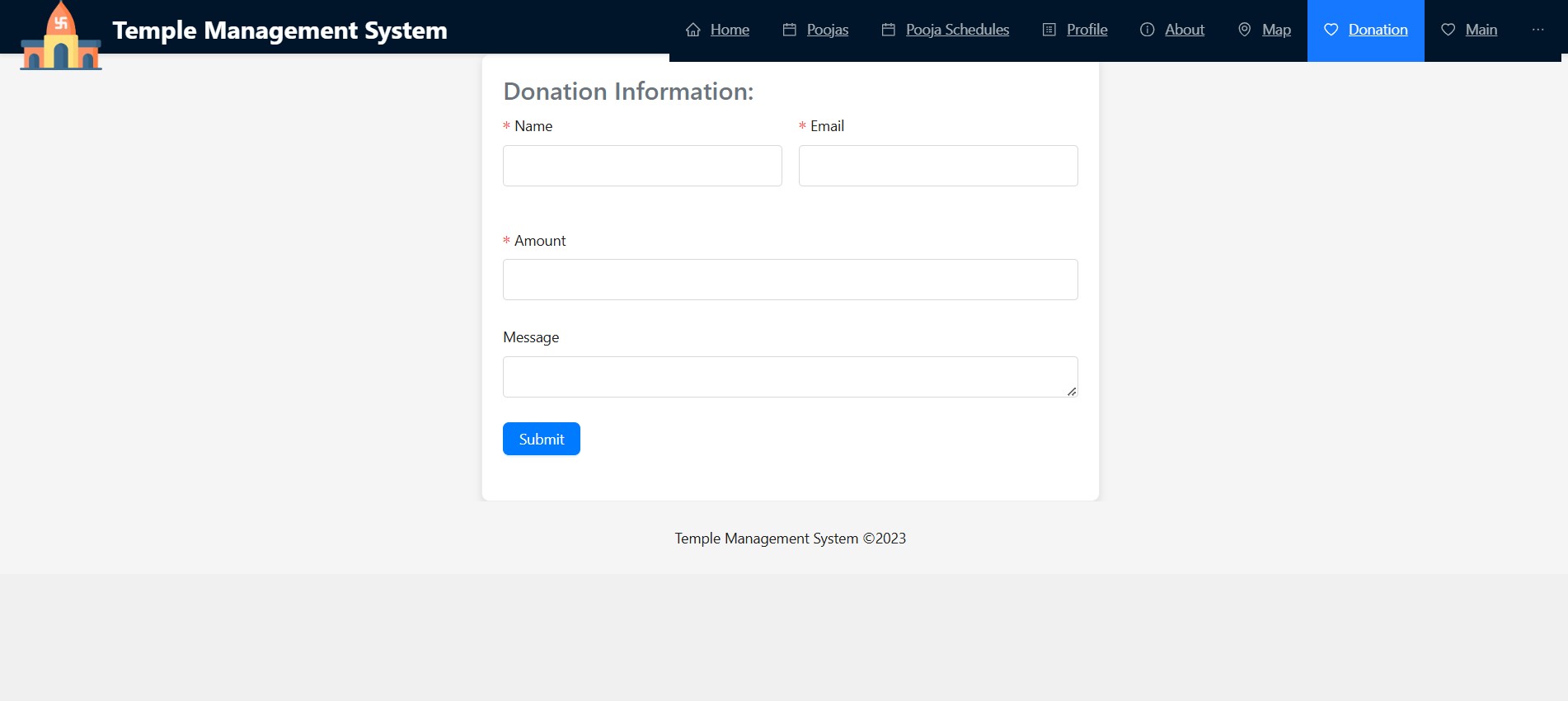
### 8.4 POOJAS PAGE



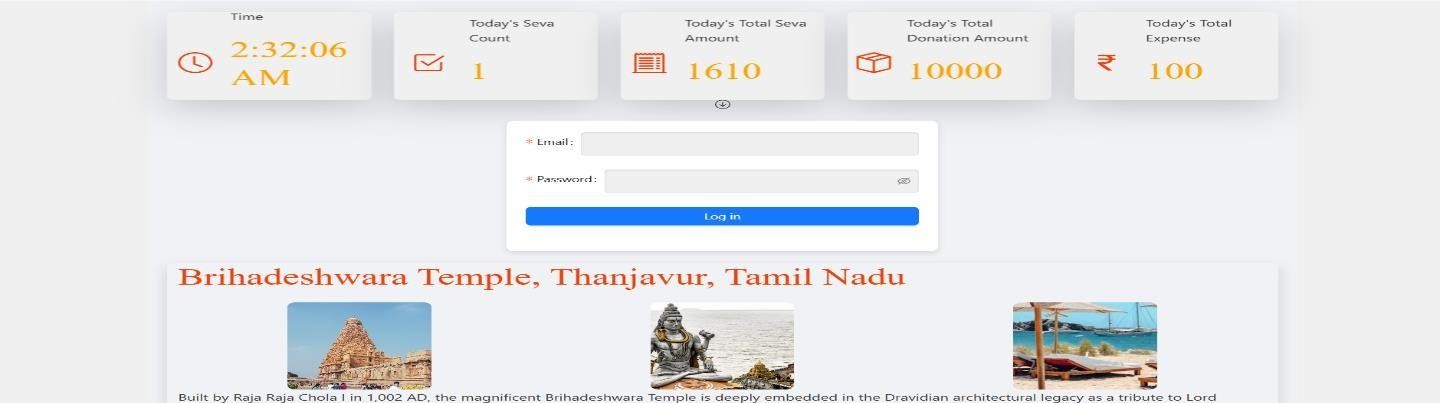
### 8.5 POOJA SCHEDULE PAGE



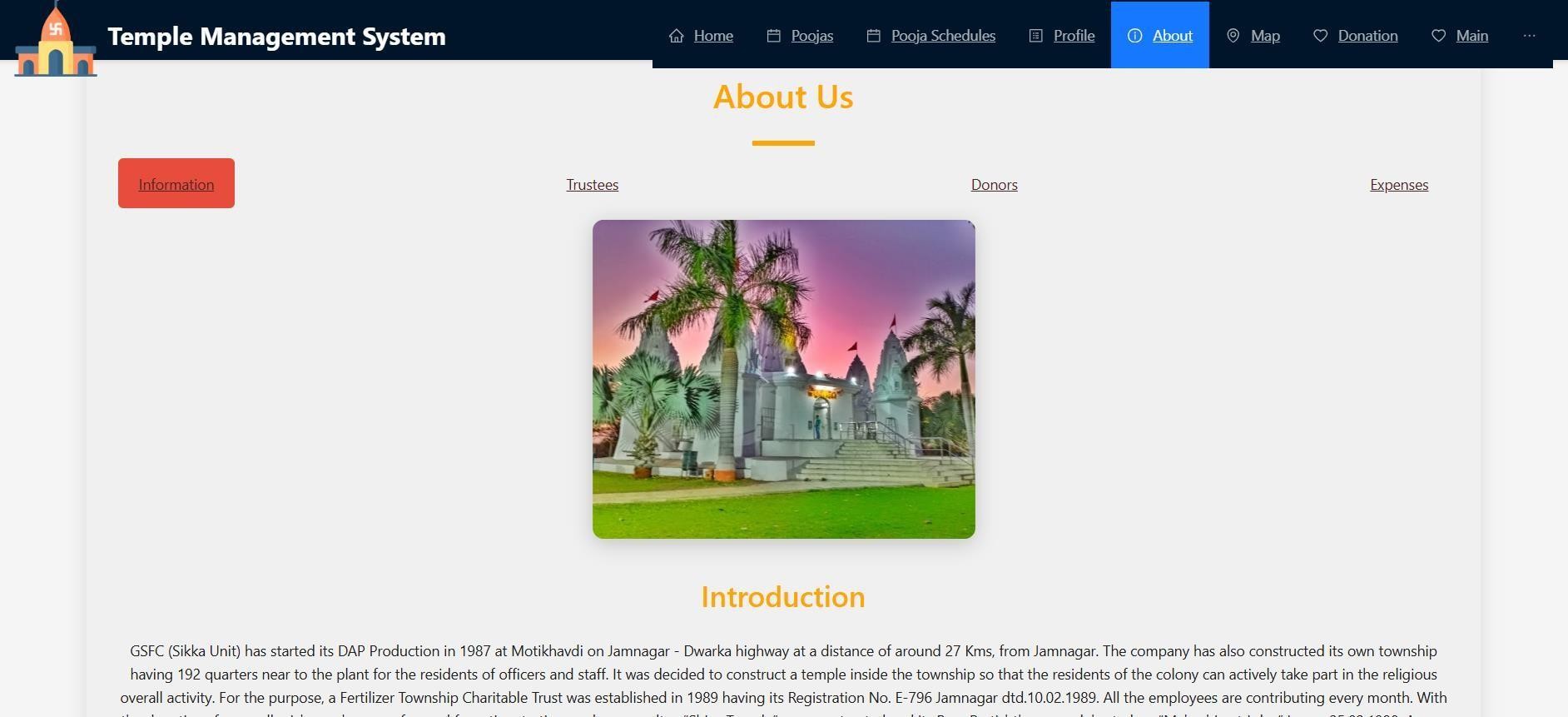
### 8.6 ABOUT PAGE



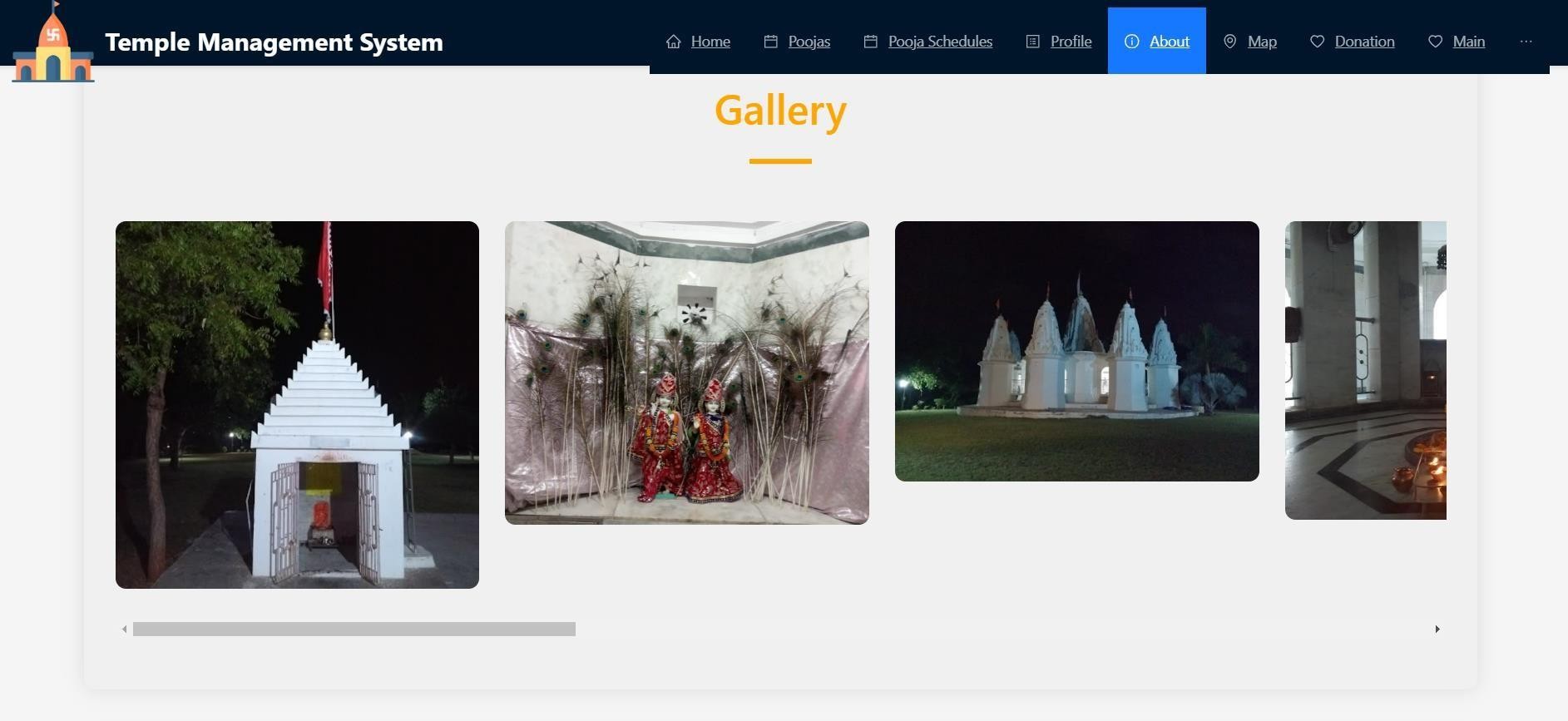
### 8.7 DONATION PAGE



### 8.8 MAIN PAGE



### ABOUT PAGE



.

**Security and Privacy**: With secure user authentication, role-based access controls, and robust data handling protocols, the system prioritizes user data privacy and ensures that sensitive information remains protected

**12. FUTURE ENHANCEMENTS**

* 1. **FUTURE SCOPE:**

The Temple Management System can be expanded with additional features such as multi-language support, integration with social media, detailed analytics, and predictive insights to further enrich user experience and operational efficiency. Additionally, incorporating mobile apps and further optimizing for accessibility can extend reach and usability.

* 1. **FUTURE ENHANCEMENTS:**

The Temple Management System has significant potential for future enhancements and expansions, including:

**Mobile Application Development:** Creating a dedicated mobile app for iOS and Android platforms to increase accessibility and user engagement, allowing devotees to manage activities and donations seamlessly on their mobile devices.

**Enhanced Analytics**: Implementing advanced data analytics and reporting tools to provide insights into temple activities, donation patterns, and user engagement, helping administrators make informed decisions.

**Integration with IoT Devices**: Utilizing IoT technology to monitor temple facilities, such as energy usage and security systems, facilitating better resource management and operational efficiency.

**Multi-language Support**: Expanding the system's reach by incorporating multiple language options to cater to a diverse user base, enhancing inclusivity.

**Event Management Features**: Adding features for managing events such as festivals and ceremonies, including registration, scheduling, and communication tools for attendees.

**AI Chatbot Integration**: Implementing an AI-powered chatbot for 24/7 user support, providing instant assistance with queries related to temple services and events.

47

**Blockchain for Transparency**: Exploring the use of blockchain technology to enhancetransparency and security in financial transactions, ensuring trust in donation processes.

**Community Engagement Tool**s: Developing forums and social features to encourage community interaction, sharing of experiences, and participation in temple activities.

These enhancements will ensure that the Temple Management System remains relevant and continues to serve the community effectively while adapting to technological advancements and user needs.