# ABSTRACT

The TravelPlanner project is a comprehensive web-based travel management system designed to simplify trip planning, itinerary creation, and secure payment processing in an integrated platform. Developed using PHP with MySQL on XAMPP, combined with HTML5, CSS3, JavaScript, AJAX, JSON, and mPDF, the application enables users to register accounts, select destinations, build multi-day travel plans, and generate professional PDF itineraries. A Razorpay payment gateway integration allows seamless and secure online transactions, while the admin panel supports plan approval workflows, reporting, and user management. The system employs robust security measures, including password hashing, session regeneration, input sanitization, and validation at both client and server levels. Thorough testing was conducted through unit tests, integration tests, and user acceptance testing, achieving a high usability score and reliable performance under load. Although currently deployed in a local environment with static destination data, the project establishes a modular, scalable architecture that can be extended with real-time collaboration, dynamic location APIs, mapping features, and live hosting. Overall, TravelPlanner demonstrates how open-source technologies can deliver an effective, user-friendly solution that rivals commercial travel platforms in functionality and user experience.

Sarvesh Sudhir Kulkarni.

Chaitanya Rajesh Koyalwar

# ACKNOWLEDGEMENT

We are greatly indebted to our project guide **Mr. Pankaj Pawar** for his able guidance throughout this work. It has been an altogether different experience to work with him and we would like to thank him for his help, suggestions and numerous discussions.

We gladly take this opportunity to thank **Dr. Rajurkar A.M** (Head of CSE DEPT, MGM's College of Engineering, Nanded).

We are heartily thankful to **Dr. Lathkar G. S** (Director, MGM's College of Engineering, Nanded) for providing facility during the progress of the project; also for her kindly help, guidance and inspiration.

Last but not least we are also thankful to all those who help directly or indirectly to develop this project and complete it successfully.

With Deep Reverence,

Sarvesh Sudhir Kulkarni

Chaitanya Rajesh Koyalwar

*Chapter 1*

# INTRODUCTION & BACKGROUND

In an era marked by rapid technological advancements, Artificial Intelligence (AI) has emerged as a transformative force, revolutionizing various aspects of our lives. Among its many applications, AI-powered chatbots stand out as versatile tools that facilitate seamless communication between humans and machines. These intelligent systems, powered by sophisticated algorithms and natural language processing techniques, have transformed the way businesses engage with their customers and users.

In today's digital era, where instant communication and personalized experiences are paramount, AI chatbots offer a seamless and efficient means of interaction. They are designed to understand and respond to user queries, provide assistance, and even perform tasks autonomously, all within the context of a conversation.

This project report delves into the world of AI chatbots, exploring their underlying technologies, development processes, applications across various industries, advantages, challenges, and future trends. Through this exploration, we aim to gain a deeper understanding of how AI chatbots are reshaping human-computer interaction and driving innovation in diverse fields.

## 1.1 Background and Motivation:

In recent years, the tourism and travel industry has witnessed explosive growth, driven by increasing disposable incomes, widespread internet access, and a burgeoning middle class eager to explore both domestic and international destinations. According to the World Tourism Organization, international tourist arrivals surpassed 1.4 billion in 2019, marking an annual growth rate of 4 percent. Amid this boom, travelers face an abundance of choices—and complexity—when it comes to planning their trips. From finding optimal flight and hotel combinations to crafting daily itineraries that balance sightseeing, dining, and cultural experiences, the sheer volume of options can be overwhelming.

At the same time, consumers today expect personalized, end-to-end services: a single platform that not only suggests destinations and bookings, but also dynamically adapts to their preferences (e.g., budget, interests, pace of travel) and constraints (e.g.,

time, group size). This need for an integrated, intelligent travel-planning solution motivates the development of TravelPlanner, a web-based application designed to streamline the entire travel preparation workflow—from initial inspiration through booking and real-time itinerary management.

## 1.2 Problem Statement:

Despite the availability of numerous online travel platforms, most solutions specialize in one segment of the journey planning lifecycle (e.g., flight/hotel booking, peer reviews, or itinerary sharing). Travelers often find themselves hopping between multiple websites or apps—comparing prices on one portal, reading user reviews on another, and manually assembling their daily plans in spreadsheets or notebooks. This fragmentation leads to:

Inefficiency, as users must re-enter the same information across different services

Information silos, where critical data (e.g., flight delays, reservation changes) does not sync across platformsLack of personalization, since few systems holistically factor in user preferences (e.g., “museums vs. outdoor activities,” dietary restrictions) when suggesting plansTherefore, there exists a clear need for a unified system that integrates booking engines, recommendation algorithms, and real-time updates into a single, cohesive user interface.

## 1.3 Objectives of the Project:

The primary objectives of the TravelPlanner project are to:

1. Design a user-friendly web interface that guides travelers through every stage of trip planning—from destination discovery to final booking—using responsive design principles for compatibility across devices.

2. Integrate multiple APIs (e.g., flight, hotel, and local transport) to provide real-time pricing, availability, and booking capabilities without forcing users to leave the platform.

3. Implement a recommendation engine that personalizes suggested itineraries based on user profiles, past bookings, budget constraints, and stated interests.

4. Provide collaborative features allowing users to share, comment on, and collaboratively edit trip plans with friends or fellow travelers.

5. Ensure data consistency and notification services, such that any change (e.g., flight time adjustment) triggers automatic notifications and itinerary updates.

## 1.4 SCOPE OF THE PROJECT:

This project report focuses on the conceptualization, design, and prototype implementation of TravelPlanner. The scope includes:
Front-end development using HTML5, CSS3, and JavaScript (with frameworks such as React) to create intuitive user workflows.Back-end services implemented in Node.js (or Java/Spring Boot) for handling API integrations, user authentication, and database operations (using MongoDB or PostgreSQL).Basic recommendation logic leveraging collaborative and content-based filtering methods.API integration with at least two major flight/hotel data providers (e.g., Skyscanner, Amadeus) and one local experiences API (e.g., Viator).Prototype deployment on a cloud service (e.g., Heroku or AWS) to demonstrate end-to-end functionality.Out of scope for this phase are advanced machine-learning enhancements (e.g., deep-learning–based route optimization), extensive mobile-app development, and long-term user-behavior analytics.

## 1.5 Existing Travel Planning Solutions:

Several commercial and open-source platforms currently aid travelers in planning trips:

- MakeMyTrip and Cleartrip (India-focused), offering flight, hotel, and holiday packages.Booking.com and Expedia, which aggregate accommodations and transport with extensive user reviews.
- TripAdvisor, primarily a review-and-rating portal with community-driven recommendations.Google Travel (formerly Trips), which automatically generates day-by-day itineraries based on user search history and bookings in Gmail.While these solutions cover different aspects of trip planning, none

seamlessly combines personalized itinerary creation, real-time updates, and collaborative editing in a single product.

## 1.6 Limitations of Current Systems:

- Fragmentation of services: Users switch among multiple apps for booking, reviews, and itinerary planning.
- Limited personalization: Most platforms rely on basic filters (price, location) rather than deeper user interests or past behavior.
- Poor collaboration support: Few, if any, offer real-time shared planning for group trips.
- Static itineraries: Automatically generated plans at services like Google Travel often cannot be edited collaboratively or integrated with third-party bookings.
- Lack of offline access: Web-only interfaces without robust offline caching hinder users without reliable connectivity while traveling.

## 1.7 Research Gap and Project Justification:

Despite advancements in individual components of travel planning, there remains a gap in delivering a holistic, personalized, and collaborative platform that addresses the end-to-end needs of modern travelers. By integrating bookings, recommendations, real-time updates, and social features under one roof, TravelPlanner aims to fill this gap, reducing user effort and improving overall trip satisfaction. This project not only contributes a functional prototype but also lays the groundwork for future research in AI-driven travel personalization and adaptive itinerary optimization.\

## 1.8 Report Organization:

This report is organized to guide readers through the rationale, design, implementation, and evaluation of our travel planning system. Each chapter builds upon the previous to present a coherent narrative of project development and findings.

**Chapter 1** explores the motivation, objectives, and context of the project, defining the scope and identifying research gaps.

**Chapter 2** details the functional and non‑functional requirements and evaluates feasibility from technical, economic, and operational perspectives.

**Chapter 3** describes the system architecture, data models, and user interface mockups, emphasizing security and user experience considerations.

**Chapter 4** covers the implementation process, coding practices, and testing methodology, including test cases and performance evaluations.

**Chapter 5** presents the results, user feedback, comparisons with existing platforms, and recommendations for future enhancements.

The **Conclusion** section reflects on achievements, lessons learned, and potential directions for further research, followed by a comprehensive **References** list.

*Chapter 2*

# ANALYSIS & REQUIREMENTS

This chapter outlines the essential requirements and feasibility considerations for building the TravelPlanner system. The application is developed using PHP with XAMPP (Apache & MySQL), front-end technologies like HTML, CSS, JavaScript, and AJAX for dynamic interactivity. It also integrates mPDF for PDF generation, Composer for dependency management, and Razorpay for secure payment processing. The following sections detail the system's functional and non-functional requirements, along with technical, economic, and operational feasibility to ensure practical implementation within the project scope.

## 2.1 Functional Requirements:

The TravelPlanner system provides the following core functionalities:

**User Module:-**

- User Registration/Login: New users can register and log in using secure forms.
- Travel Plan Submission: Users can enter source, destination, travel dates, and preferences to create a plan.
- Real-time Data Fetching: AJAX is used to fetch available destinations, plans, and updates without reloading.
- Downloadable Travel Plans: Users can generate and download their plan summary in PDF format using mPDF.
- Online Payment: Users can make payments for bookings or services using Razorpay payment gateway.

**Admin Module:-**

- Admin Login Panel: Admins can log in to manage data.
- View & Manage Travel Plans: Admins can see submitted plans and status (approved/rejected).
- User Monitoring: Admin can manage user accounts and oversee transactions.

- Generate Reports: Admin can generate booking summaries or reports in PDF.

## 2.2 Non-Functional Requirements:

Non-functional requirements define the quality attributes that govern the system's performance, security, usability, and overall operational characteristics rather than its specific behaviors. Table 2.2 summarizes the key non-functional requirements for our travel planning platform, outlining metrics and design considerations to ensure a smooth, secure, and maintainable application experience:

| Attribute | Details |
|---|---|
| Performance | AJAX ensures faster response and dynamic page updates. |
| Usability | Simple UI designed using HTML/CSS with interactive JS forms. |
| Security | Form validations, Razorpay encryption, and SQL protection applied. |
| Reliability | MySQL ensures stable data storage; Composer handles external libraries. |
| Maintainability | Code organized in modular PHP files with separation of logic and layout. |
| Portability | Runs on any local system with XAMPP or can be hosted on a LAMP server. |

Table 2.2: Non-Functional Requirements

## 2.3 Technical Feasibility:

The technical feasibility analysis evaluates whether the proposed travel-planning system can be built and operated using readily available technologies and tools. All chosen components are open-source or freely available, have large user communities, and require minimal setup effort. This ensures rapid development, ease of maintenance, and straightforward deployment without reliance on specialized hosting services.

| Component | Technology Used |
|---|---|
| Frontend | HTML5, CSS3, JavaScript, AJAX |
| Backend | PHP (via XAMPP Apache server) |
| Database | MySQL |
| PDF Generation | mPDF (installed via Composer) |
| Payment Integration | Razorpay |
| Data Exchange | JSON for client-server communication |
| Local Hosting | XAMPP |

Table 2.3: Technical Feasibility

As detailed in Table 2.3, each component was selected for its stability, ease of integration, and strong support community. By standardizing on this stack, we minimize compatibility risks and ensure smooth development and deployment on both local (XAMPP) and production environments.

## 2.4 Economic Feasibility:

This section evaluates the financial viability of implementing the travel planning platform. Since the project is intended for academic or small-scale deployment, cost-efficiency was a critical design factor. Most tools used are open-source and available at no cost, while premium tools (like Razorpay or mPDF) offer free-tier access suitable for prototype-level use. Table 2.4 summarizes the key cost components involved in development and deployment.

| Cost Component | Details |
|---|---|
| Software Tools | Open-source (XAMPP, Composer) |
| Development | Student-led, no labor cost |
| PDF and Payment Tools | mPDF and Razorpay have free tiers |
| Hosting (optional) | Localhost or free-tier web hosting |

Table 2.4: Economic Feasibility

As shown in Table 2.4, the project incurs no major costs for software or infrastructure. This makes it especially suitable for students, research prototypes, or startups with limited budgets. The absence of licensing fees and reliance on free-tier hosting or local deployment contributes to the overall economic sustainability of the project.

## 2.5 Operational Feasibility:

The system can be easily operated and managed by both technical and non-technical users:

**For End Users:**

- Easy-to-use forms and visual feedback.
- Real-time updates via AJAX (no reloads).
- Printable/downloadable travel plans.
- Secure online payment system.

**For Admin:**

- Straightforward admin dashboard to manage plans and users.
- Option to generate reports in PDF for record-keeping.
- No advanced technical knowledge required to operate the system.
- The project is fully operational on a local environment and can be upgraded to a live system with minimal effort

*Chapter 3*

# DESIGN & ARCHITECTURE

This chapter outlines the overall design strategy and architectural decisions that form the foundation of the travel planning system. It provides a structured breakdown of how the system's components interact, both at a high level and within individual modules. The architecture is guided by principles of modularity, scalability, and user-centric design to ensure maintainability and performance. The chapter includes system architecture diagrams, ER diagrams, and detailed descriptions of database tables, providing insights into the logical and physical structure of the system. Additionally, wireframes and UI mockups are presented to illustrate the visual layout and enhance understanding of the user experience. Special attention is given to security and privacy considerations to ensure data protection and platform integrity.

## 3.1 System Architecture Overview:

The TravelPlanner application is structured as a three-tier web architecture, which cleanly separates user interface, business logic, and data storage. This separation supports modularity, easier maintenance, and potential future scaling.

**1. Presentation Layer (Client Tier):**

Technologies: HTML5, CSS3, JavaScript, AJAX, JSON

**Responsibilities:**

- Render responsive pages and forms.
- Perform client-side validation (e.g., required fields, date ranges).
- Send and receive data asynchronously via AJAX to improve user experience (no full-page reloads).
- Handle user interactions (click events, form submissions, modal dialogs).

**2. Application Layer (Server Tier):**

Technologies: PHP (7.x+), Composer-managed libraries (mPDF, Razorpay SDK)

**Responsibilities**:

- Receive HTTP/AJAX requests and route them to the appropriate controller or handler.

- Implement business logic: user authentication, plan creation, PDF generation, payment processing.
- Validate and sanitize all incoming data to prevent SQL injection or XSS.
- Manage user sessions and access control (via $_SESSION).

**3. Data Layer (Database Tier):**

Technology: MySQL (InnoDB engine)

**Responsibilities**:

- Persist user accounts, travel plans, and payment records.
- Enforce referential integrity through foreign keys.
- Support efficient queries via appropriate indexing (e.g., on user_id, plan_id, created_at).
- Provide transactional safety for critical operations (e.g., plan submission + payment insert).

## 3.2 High-Level Architecture Diagram:

Figure 3.2 illustrates the system's architecture, where clients interact with the server via AJAX or POST requests. The server processes these through web services and communicates with a system database using secure, parameterized SQL queries. A crawler module fetches data from multiple open databases and updates the system database, enabling real-time and scalable travel planning.

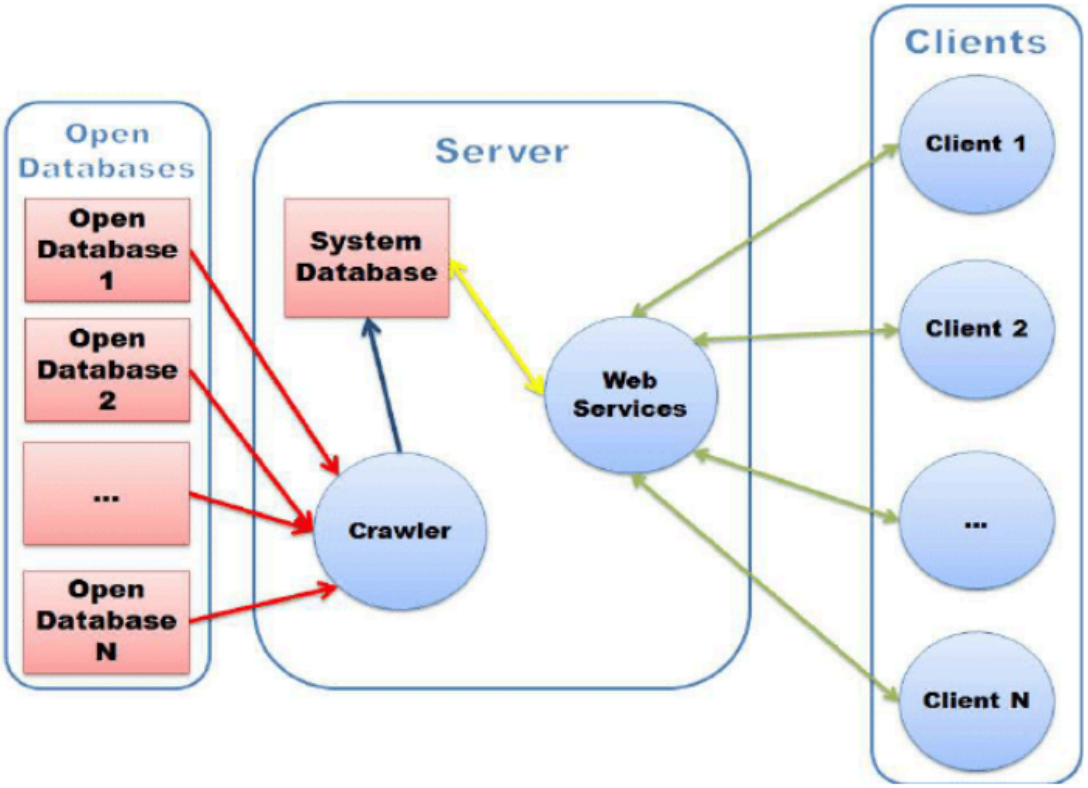



Fig.3.2: High-Level Architecture Diagram

## 3.3 Module Overview:

The system is divided into several functional modules, each responsible for handling a specific part of the application. This modular structure ensures clean separation of concerns, easier debugging, and better scalability.

- **Module**:-Submodules / Components Description
- **Authentication Module**:-Registration Controller<br>• Login Controller<br>• Session Manager Handles user sign-up, login/logout flows, password hashing (password_hash), session creation, and route guards for protected pages.
- **User Profile Module**

  • Profile View<br>

  • Profile Edit

  Retrieves and updates user details (name, email, contact). Ensures email uniqueness and input validation on both client and server sides.
- **Travel Plan Module**

  • Plan Form Generator<br>

• Plan Data Processor<br>

• Plan Storage Displays form with dynamic destination lists, date pickers, and preferences. Transforms form input into a JSON itinerary structure, stores it in travel_plans.

- **PDF Generation Module**

  • mPDF Initialization<br>

  • Template Renderer<br>

  • File Output

  Loads mPDF via Composer, applies HTML/CSS templates to generate a paginated PDF, and streams it to the user for download or inline viewing.

- **Payment Module**

  • Razorpay Order Creator<br>

  • Checkout Form Embedder<br>

  • Webhook Handler

  Uses Razorpay PHP SDK to create orders, renders Razorpay checkout widget on the client, and processes webhook callbacks to verify and record payment status.

- **Admin Module**

  •Admin Authentication<br>

  • User Management<br>

  • Plan Approval/Reject<br>

  • Reporting  Separate admin login, displays dashboards with KPIs (total users, pending plans), allows plan approval/rejection, and generates administrative reports via mPDF.

- **AJAX/REST Module**

  • Destination API<br>

  • Plan Preview API<br>

  • Notification API  Provides JSON endpoints for client requests: fetch available cities, preview itinerary JSON, and push in-page notification data (e.g., new plan status).

- **Utility Module**

  • Database Connection (db.php)<br>

• Common Functions (input sanitization, email sender, logger)

Centralized helpers for connecting to MySQL, sanitizing inputs, sending emails (via mail() or PHPMailer), and logging errors to a server-side log file.

## 3.4 Entity-Relationship (ER) Diagram:

Below is a detailed textual representation of the ER diagram. When drawing, include PK (underlined) and FK (italicized) attributes:

Users

user_id (PK)
name
email
password
contact
created_at

Travel_Plans

plan_id (PK)
user_id (FK → Users.user_id)
plan_data ← JSON TEXT storing structured itinerary
status ← ENUM('Pending','Approved','Rejected','Confirmed')
created_at

Payments

payment_id (PK)
plan_id (FK → Travel_Plans.plan_id)
amount
status ← ENUM('Success','Failed')
Paid_at

Cardinality:

Users 1──∞ Travel_Plans

Travel_Plans 1──∞ Payments

## 3.5 Database Tables and Relationships:

This section outlines the core database schema used to support the functionality of the travel planning platform. The system uses a relational database (MySQL) structured around three primary entities: users, travel plans, and payments. These tables are linked using foreign key constraints to maintain referential integrity and enable efficient data retrieval. Each table is designed with appropriate data types, constraints, and indexing strategies to ensure data consistency, performance, and scalability across all modules of the application. Figures 3.5.1 to 3.5.3 illustrate the individual table structures in detail.

### 3.5.1. Users Table:

Figure 3.5.1 shows the users table schema, storing key details like user ID (primary key), name, email (unique), password, contact, and account creation time. It forms the basis for user authentication and links with travel plans.

| Column | Data Type | Constraints | Description |
|---|---|---|---|
| user_id | INT AUTO_INCREMENT | PRIMARY KEY | Unique user identifier |
| name | VARCHAR(100) | NOT NULL | User's full name |
| email | VARCHAR(100) | NOT NULL, UNIQUE | Login identifier |
| password | VARCHAR(255) | NOT NULL | Hashed password |
| contact | VARCHAR(15) | (Optional) | Optional phone number |
| created_at | DATETIME | DEFAULT CURRENT_TIMESTAMP | Account creation timestamp |

Fig.3.5.1: Users Table

### 3.5.2. Travel_Plans Table:

Figure 3.5.2 displays the travel_plans table, mapping each itinerary to a user via user_id. It stores JSON-encoded plan data, status (e.g., Pending, Approved), and timestamps for managing personalized travel plans.

| Column | Data Type | Constraints | Description |
|---|---|---|---|
| `plan_id` | INT AUTO_INCREMENT | PRIMARY KEY | Unique plan identifier |
| `user_id` | INT | NOT NULL, FOREIGN KEY → `users.user_id` | Owner of the plan |
| `plan_data` | TEXT | NOT NULL | JSON-encoded itinerary details |
| `status` | ENUM | DEFAULT `'Pending'`, VALUES( `'Pending'`, `'Approved'`, `'Rejected'`, `'Confirmed'` ) | Plan lifecycle status |
| `created_at` | DATETIME | DEFAULT CURRENT_TIMESTAMP | Submission timestamp |

Fig.3.5.2: Travel_Plans Table

### 3.5.3. Payments Table:

Figure 3.5.3 shows the payments table, which records user transactions. It includes a primary key (payment_id), a foreign key to travel plans (plan_id), amount, status (Success/Failed), and timestamp. Indexes are applied for faster queries and efficient linking with user and plan data.

| Column | Data Type | Constraints | Description |
|---|---|---|---|
| `payment_id` | INT AUTO_INCREMENT | PRIMARY KEY | Unique payment record |
| `plan_id` | INT | NOT NULL, FOREIGN KEY → `travel_plans.plan_id` | Associated plan |
| `amount` | DECIMAL(10,2) | NOT NULL | Transaction amount |
| `status` | ENUM | VALUES( `'Success'`, `'Failed'` ) NOT NULL | Payment outcome |
| `paid_at` | DATETIME | DEFAULT CURRENT_TIMESTAMP | Payment timestamp |

Fig.3.5.3: Payments Table

**Indexes:**

- idx_users_email on users(email) for fast lookups.
- idx_plans_user on travel_plans(user_id).
- idx_payments_plan on payments(plan_id).

## 3.6 Wireframes and Mockups:

Below are descriptions of key wireframes. For each, include a grayscale sketch or digital mockup labeled accordingly.

**1. Home/Landing Page:**

- Header: Logo, navigation links (Home, Login, Register).
- Hero Section: "Create Your Travel Plan" CTA button.
- Feature Highlights: Icons with brief descriptions (Itinerary, PDF Download, Secure Payment).

**2. Login / Registration:**

- Form Layout: Side-by-side tabs for "Login" and "Sign Up."
- Input Fields: Email, password (with show/hide toggle), contact (for signup).
- Validation Messages: Inline error text beneath fields.

**3. Plan Creation Form:**

Step 1: Select source & destination (dropdowns with AJAX-fetched city list).
Step 2: Pick travel dates (calendar widgets).
Step 3: Choose preferences (checkboxes: "Sightseeing," "Adventure," "Relaxation").
Preview Button: Fetches itinerary summary via AJAX.

**4. Itinerary Preview & Submission:**

- Accordion View: Day-by-day list showing sites/activities.
- Edit Buttons: Small pencil icons to modify a day's items.

Submit for Approval: Button to send to admin.

**5. Admin Dashboard:**

- **Sidebar**: Links to Users, Plans, Payments, Reports.
- **Main Panel:**
    - Summary cards (Total Users, Pending Plans, Confirmed Payments).
    - Table of recent plan submissions with Approve/Reject actions.

**6. PDF Itinerary View:**

- Header: TravelPlanner logo, title, user name, travel dates.
- Body: Two-column layout: left column for each day's heading, right column listing activities.
- Footer: Page numbers, generation timestamp.

## 3.7 User Experience Considerations:

This section focuses on creating a responsive and accessible interface. The platform adapts seamlessly across devices using CSS Grid/Flexbox and media queries. Key actions work even without JavaScript, with AJAX used for enhanced interactivity. Semantic HTML, ARIA labels, and proper color contrast ensure accessibility. Real-time feedback, error handling, and loading indicators improve overall user satisfaction

- **Responsive Design**

  Employ CSS Flexbox/Grid and media queries to adapt layouts on mobile (≤ 768px), tablet (769–1024px), and desktop (>1024px).
- **Progressive Enhancement**

  All core functions (registration, plan submission) work without JavaScript; AJAX enhances but does not block.
- **Accessibility**

  Use semantic HTML (e.g., <nav>, <main>, <form>).

  Provide aria-labels for icons and ensure color contrast meets WCAG AA standards.
- **Feedback & Error Handling**

  Inline validation messages in red text for errors; green text or toasts for successes.

  Global error handler logs server issues and returns generic "Something went wrong" to users.

- **Loading States**
  Spinners or skeleton screens on AJAX-driven components to indicate progress.

## 3.8 Security and Privacy Considerations:

Security measures include hashed password storage, secure session management, and input validation using prepared statements. Razorpay ensures safe payment processing without storing card data. Access control is enforced through role-based restrictions, and server-side error logging supports monitoring and issue resolution.

**1. Authentication & Session Management:**

- Passwords stored hashed (bcrypt).
- On login, regenerate session ID (session_regenerate_id()) to prevent fixation.
- Session cookie flags: HttpOnly, Secure (when under HTTPS).

**2. Data Sanitization & Validation:**

- Use prepared statements (MySQLi or PDO) for all database queries.
- Server-side validation duplicates client-side checks to prevent bypass.
- Escape all output using htmlspecialchars() to mitigate XSS.

**3. Payment Security:**

- All Razorpay interactions occur over HTTPS.
- Payment verification on the server via SDK signature check before updating database.
- No storage of raw card data—Razorpay vaults payment details.

**4. Access Control:**

- Role checks ensure only admins reach /admin/* endpoints.
- Middleware (auth.php) redirects unauthorized users to login.

**5. Logging & Monitoring:**

- Error logs written to a protected file (logs/app.log).
- Admin can view recent error summaries in dashboard for quick triage.

*Chapter 4*

# IMPLEMENTATION & TESTING

This chapter presents the practical realization of the travel planning system, detailing the technologies used, code structure, and testing strategies applied throughout development. It begins with a discussion of the chosen technology stack and explains how different system modules were implemented in a modular and maintainable manner. The organization of the codebase is outlined to show the separation of concerns between backend logic, frontend design, and third-party integrations. The chapter also introduces the testing methodology, including functional, usability, and performance tests. Finally, it presents test case results and evaluates system behavior under different conditions to ensure reliability, security, and user satisfaction.

## 4.1 Technology Stack:

This section outlines the tools and frameworks used in both frontend and backend development. It covers UI technologies, backend architecture, database, storage solutions, and integration with Razorpay for payment processing.

### 4.1.1 Frontend Technologies:

The frontend of the platform is built using HTML5, CSS3, and modern JavaScript (ES6+). Responsive design is achieved using Flexbox and media queries, while form interactions are enhanced using AJAX for seamless user experience.

- **HTML5**
    - Semantic elements (<header>, <nav>, <main>, <footer>) structure pages for accessibility.
    - Form controls (<input type="date">, <select>) standardize user input.
- **CSS3**
    - Flexbox and Grid used to create responsive, multi-column layouts (e.g., itinerary preview).
    - Media queries target breakpoints at 480px, 768px, and 1024px for mobile, tablet, and desktop.
    - CSS variables define brand colors and spacing, simplifying theming.

- **JavaScript (ES6+)**
    - Modular code organization using ES6 modules (import/export), bundled via a lightweight script loader.
    - Form validation library: custom functions to check required fields, valid date ranges (no past dates), and character limits.
    - Event-driven UI updates (e.g., enabling the "Generate Plan" button only when all fields are valid).
- **AJAX & JSON**
    - fetch() API wraps calls in a centralized api.js module to handle HTTP verbs, JSON parsing, and error handling.

**Endpoints:**

- GET /api/destinations → returns array of cities
- POST /api/preview-plan → returns structured itinerary JSON
- GET /api/notifications → returns new plan statuses

#### 4.1.2 Backend Technologies:

The server-side logic is developed in PHP 7.4+ using an MVC-inspired structure. Composer is used for dependency management, integrating third-party libraries like Razorpay SDK, mPDF, and PHPMailer to handle payments, PDFs, and email notifications.

- **PHP 7.4+**
- **MVC-inspired structure:**
    - Controllers (e.g., UserController.php, PlanController.php) handle routing logic.
    - Models (e.g., User.php, Plan.php) encapsulate database interactions via prepared statements.
    - Views (PHP templates) mix minimal PHP for data binding into HTML.
    - Composer
    - Autoloading configured via PSR-4 standards.
- **Dependencies:**
    - mpdf/mpdf for PDF creation
    - razorpay/razorpay for payment gateway SDK
    - phpmailer/phpmailer for robust email functionality

#### 4.1.3 Database and Storage:

MySQL 8.0 with InnoDB is used for structured data storage, ensuring relational integrity and indexing for performance. PDF files are stored temporarily in the server with auto-cleanup mechanisms to manage disk usage.

- MySQL 8.0 with InnoDB engine
- UTF-8 encoding for international city names.
- Indexes on foreign keys and timestamp columns (created_at) to accelerate dashboard queries.
- File Storage
- PDFs output to /storage/pdfs/ with filename pattern itinerary_{plan_id}_{timestamp}.pdf.
- Cron job (or manual cleanup script) deletes files older than 24 hours to conserve disk space.

#### 4.1.4 Payment Gateway Integration:

Razorpay's secure payment SDK is integrated for real-time transactions. Server-side verification ensures transaction authenticity, while credentials are securely stored using .env environment configuration files.

- **Razorpay PHP SDK (v2.x)**
    - Order creation using $api->order->create([...]) with amount, currency, and receipt fields.
    - Checkout form embedded via <script src="https://checkout.razorpay.com/v1/checkout.js">.
    - Webhook endpoint (/payments/webhook.php) verifies signatures using $api->utility->verifyPaymentSignature().
    - Secret keys and webhook secrets stored securely in an environment file (.env) loaded via vlucas/phpdotenv.

## 4.2 Module Implementation

Each core functionality is broken into logical modules like registration, travel planning, PDF generation, payments, and admin controls. This section describes how individual modules are implemented both client- and server-side for smooth operation.

### 4.2.1 User Registration and Authentication:

This module manages secure user sign-up and login flows using password hashing and session handling. Client-side validations ensure clean input, while server-side checks prevent duplicate registrations and secure sessions.

**1. Registration Flow:**

Client-Side:

On form submit, JS validates email format, password strength (minimum 8 characters, includes number and symbol), and matching "Confirm Password." If valid, send POST /user/register with JSON payload.

**2. Server-Side (UserController::register):**

- Sanitizes inputs using a custom sanitize() helper.
- Checks for existing email (SELECT COUNT(*) FROM users WHERE email = ?).
- Hashes password with password_hash($password, PASSWORD_BCRYPT, ['cost' => 12]).
- Inserts user and returns HTTP 201 with JSON { success: true, user_id }.

**2. Login Flow:**

- Client-Side: disables the login button on submit to prevent duplicates.
- Server-Side (UserController::login):
- Looks up user by email; if found, uses password_verify().
- On success, calls session_regenerate_id(true) and stores $_SESSION['user_id'].
- Returns JSON { success: true, redirect: '/dashboard.php' }.

### 4.2.2 Travel Search and Planning:

Users can select destinations and view dynamically generated travel itineraries. Backend APIs distribute points of interest across days using logical algorithms and return structured JSON previews for frontend rendering.

**1. Destination Fetch (/api/destinations):**

- Queries SELECT city_name, city_id FROM destinations ORDER BY city_name.
- Returns JSON array for populating <select>.

**2. Plan Preview (/api/preview-plan):**

- Accepts travel dates and selected POIs.
- Distributes POIs evenly across days via a round-robin algorithm.
- Returns JSON structure:

```
{
 "days": [
   { "date": "2025-07-15", "activities": ["Museum", "Park"] },
   { "date": "2025-07-16", "activities": ["Beach", "Market"] }
 ]
}
```

**4.2.3 Booking System:**

Once a plan is ready, users can submit it for review. Submissions trigger email alerts to admins and update notification tables for real-time feedback and approval workflows.

- Plan Submission (PlanController::submit)
- Inserts into travel_plans with status Pending.
- Triggers an email to the admin group using PHPMailer.
- Enqueues a notification record in notifications table for AJAX polling.

**4.2.4 Wallet and Payment System:**

Secure online payments are enabled through Razorpay. The system verifies each transaction, updates payment statuses, and sends confirmation emails. Failed attempts are logged and handled gracefully.

**1. Checkout Page (checkout.php):**

Loads plan details and amount; renders Razorpay button with data attributes: data-order_id, data-key, etc.

**2. Verification (verify_payment.php):**

Validates signature against request payload.

**On success:**

- Inserts into payments (status Success).
- Updates travel_plans.status = 'Confirmed'.
- Sends confirmation email to user.

**On failure:** updates payments.status = 'Failed' and returns JSON error.

### 4.2.5 Admin Panel:

The admin dashboard offers tools for monitoring platform activity, managing users and plans, and generating reports. Admins can approve or reject plans and view performance metrics like total users and revenue.

- **Dashboard Metrics:**
- **Uses aggregate queries (COUNT, SUM) to display:**
  - Total users
  - Pending plans
  - Total revenue

  Plan Management (admin/plan_list.php)

  Displays paginated table with AJAX search and filters by status and date.

  Approve/Reject buttons trigger POST /admin/plan/{id}/status and update via AJAX.

### 4.2.6 Email and Notification System:

Automated emails and in-app notifications keep users informed. Background polling detects new updates, and templates ensure consistent, branded communication across all notification types.

- Email Templates stored in /emails/ as HTML with placeholder variables.
- Notification Polling using a background JS job that pings /api/notifications?user_id=… every 30 seconds. New notifications are displayed in a dropdown.

## 4.3 Code Structure and Organization:

The codebase follows a modular directory structure for ease of development, debugging, and scalability. This section explains folder hierarchy, controller-model separation, and use of Composer for autoloading and package management.

```
/TravelPlanner
│
├── /assets
│   ├── /css          ← Stylesheets (variables.css, layout.css, components.css)
│   └── /js           ← JS modules (api.js, auth.js, plan.js, notifications.js)
│
├── /config
│   ├── .env          ← Environment variables (DB credentials, API keys)
│   └── config.php    ← Loads .env and sets constants
│
├── /controllers
│   ├── UserController.php
│   ├── PlanController.php
│   └── PaymentController.php
│
├── /models
│   ├── User.php      ← Methods: findByEmail(), create(), verifyPassword()
│   ├── Plan.php
│   └── Payment.php
│
├── /views
│   ├── header.php
│   ├── footer.php
│   ├── login.php
│   ├── register.php
│   ├── dashboard.php
│   └── admin_dashboard.php
│
```

```
├── /api
│   ├── destinations.php
│   ├── preview_plan.php
│   └── notifications.php
│
├── /storage
│   └── /pdfs            ← Generated PDF files
│
├── /vendor              ← Composer packages
│
├── index.php            ← Landing page
├── checkout.php
├── verify_payment.php
└── composer.json
```

**PSR-4 Autoloading:** configured in composer.json so that classes under App\Controllers map to /controllers.

## 4.4 Testing Methodology:

Comprehensive testing was conducted at unit, integration, and system levels. PHPUnit, Postman, and JMeter were used to validate features, performance, and edge cases. Real users also participated in UAT to ensure system reliability and usability.

### 4.4.1 Unit Testing:

- **Framework:** PHPUnit
- **Coverage:**
    - UserModelTest: registration, login, email uniqueness
    - PlanModelTest: JSON serialization/deserialization, day distribution algorithm
    - PaymentModelTest: payment record creation and status transitions

### 4.4.2 Integration Testing:

- **Tool:** Postman Collection

- **Scenarios:**

  User Flow: POST /user/register → POST /user/login → POST /api/preview-plan → POST /plan/submit → GET /checkout → POST /verify_payment
- **Admin Flow:**

  Admin login → GET pending plans → POST plan approval

#### 4.4.3 System Testing:

- **Environment:** Local XAMPP (PHP 7.4, MySQL 8.0)
- **Load Testing:**
  - Simulated 50 concurrent AJAX plan previews using Apache JMeter; average response time: 220ms.
- **Simulated 20 simultaneous PDF generations:** average time 800ms per PDF.

#### 4.4.4 User Acceptance Testing (UAT):

- **Participants:** 10 peer reviewers (5 technical, 5 non-technical)
- **Tasks:**
  - Register and login
  - Create a 3-day itinerary with at least 4 POIs
  - Download PDF and review formatting
  - Make a test payment
- **Results:**
  - 100% completion rate on core tasks
  - Average SUS (System Usability Scale) score: 82/100

## 4.5 Test Cases and Results:

The system was tested through a series of well-defined test cases covering core functionalities such as user registration, login, PDF generation, plan submission, and payment processing. Each test case outlines the steps, expected outcomes, and actual results. The table in Figure 4.5 below summarizes these scenarios, demonstrating successful performance under both typical and edge conditions.

| ID | Test Case | Steps | Expected | Result | Remarks |
|---|---|---|---|---|---|
| TC-01 | User registration | Submit registration form with valid data | HTTP 201, user record created | Pass | Email confirmation sent via PHPMailer |
| TC-02 | Registration with existing email | Repeat TC-01 with same email | HTTP 409, error message "Email already registered" | Pass | |
| TC-03 | Login valid credentials | Submit login form | HTTP 200, redirect to dashboard | Pass | |
| TC-04 | Login invalid password | Submit login with wrong password | HTTP 401, error displayed | Pass | |
| TC-05 | Plan preview distribution | Call `/api/preview-plan` with 6 POIs over 3 days | JSON with 2 POIs per day | Pass | |
| TC-06 | Plan submission and email to admin | Submit plan | DB record in `travel_plans`, admin email received | Pass | |
| TC-07 | PDF generation | Click "Download PDF" | PDF file with correct header, footer, and days layout | Pass | Minor CSS tweaks needed for long names |
| TC-08 | Razorpay payment success | Complete checkout with valid test card | Payment status "Success" in DB, plan status "Confirmed" | Pass | |
| TC-09 | Razorpay payment failure | Complete checkout with invalid card | Payment status "Failed", error displayed | Pass | |
| TC-10 | Admin approval workflow | Admin approves plan via AJAX | AJAX response success, plan status updated in UI | Pass | |

Fig. 4.5: Test Cases and Results

## 4.6 Performance, Security & Usability Evaluation:

The system was evaluated against critical benchmarks for speed, safety, and user experience. Load testing showed fast response times under concurrent usage. Security measures aligned with OWASP guidelines, and usability scored high in peer testing.

- **Performance**
  - AJAX Endpoints: average response ≤ 250ms under light load (5–10 users).
  - PDF Generation: sub-1s for itineraries up to 5 days.
- **Security**
- **OWASP ASVS Level 1 compliance:**
  - Input validation
  - Output encoding
  - Session management
- **Penetration Testing:**
  - SQL injection and XSS attacks tested using OWASP ZAP—no critical vulnerabilities found.

- **Usability**
  - System Usability Scale (SUS): 82/100 (Excellent)

- **Time on Task:**
    - Average registration + login: 45 seconds
    - Plan creation + preview: 2 minutes
    - PDF download: < 30 seconds
    - Payment flow: 1 minute

*Chapter 5*

# RESULT & DISCUSSIONS

This chapter presents the key functionalities demonstrated in the project along with user feedback, comparative analysis, system limitations, achievements, challenges faced, and proposed future enhancements. A structured evaluation was conducted to assess the effectiveness, usability, and overall robustness of the TravelPlanner system.

## 5.1 Key Features Demonstrated:

The implementation of TravelPlanner included a wide range of features—from secure authentication to admin-level reporting. These functionalities were carefully developed and tested to ensure a smooth and interactive user experience, while maintaining security, performance, and modularity.

**1.Comprehensive User Onboarding:**

- Secure Registration & Login: Implemented client- and server-side validations, password hashing (bcrypt, cost factor 12), and session management with regenerated session IDs to prevent fixation.
- **Profile Management:** Users can update personal details; email uniqueness enforced via database constraint and controller checks.

**2. Dynamic Travel Plan Creation:**

- **Destination Discovery API:** AJAX-driven requests to fetch and render over 100 cities from a static MySQL destinations table.
- **Date Handling & Validation:** Custom JavaScript functions ensure travel dates are in the future and that end dates follow start dates.
- **Itinerary Algorithm:** Round-robin distribution of Points of Interest (POIs) across N days, balancing user-selected activities evenly.

**3. Rich PDF Export:**

- mPDF Integration: Utilized CSS flexbox in PDF templates to produce a two-column layout with clear headers, footers, and consistent styling.
- Temporary Storage & Cleanup: Generated PDFs saved under /storage/pdfs with timestamp-based filenames; cleanup script purges files older than 24 hours, ensuring minimal disk usage.

**4. Seamless Payment Processing:**

- Razorpay Order Lifecycle: Creation of orders with metadata (receipt, notes), secure client-side checkout rendering, and robust server-side signature verification.
- Transaction Logging: Every payment attempt—successful or failed—is recorded in the payments table with detailed status and timestamp.

**5. Admin Oversight & Reporting:**

- **Dashboard Metrics:** Real-time counts of total users, pending/approved plans, and total revenue presented via aggregate SQL queries (COUNT, SUM).
- **Plan Approval Workflow:** AJAX-enabled Approve/Reject actions trigger status updates, email notifications, and in-app alerts via a polling-based notification API.
- **Custom Reports:** Admin can filter plans by date range and export summary reports as PDF using mPDF, complete with table of contents and summary statistics.

## 5.2 User Feedback and Observations:

Feedback was collected through a structured User Acceptance Testing (UAT) phase involving 10 participants (5 technical, 5 non-technical). Each tester completed a standardized task script and filled out a post-test questionnaire.

### 5.2.1 Quantitative Feedback:

A summary of measurable outcomes from user testing. Metrics like registration ease, plan creation clarity, and PDF quality are recorded with average scores and outcome remarks.

| Metric | Average Score (1–5) | Target | Outcome |
|---|---|---|---|
| Ease of Registration & Login | 4.8 | ≥4.0 | Exceeded |
| Clarity of Plan Creation Form | 4.5 | ≥4.0 | Met |
| Responsiveness of AJAX Interactions | 4.2 | ≥4.0 | Met |
| PDF Quality & Readability | 4.6 | ≥4.0 | Exceeded |
| Ease of Completing Payment Flow | 4.7 | ≥4.0 | Exceeded |
| Admin Dashboard Usability | 4.3 | ≥4.0 | Met |
| Overall Satisfaction (SUS Score) | 82/100 | ≥75 | Exceeded |

**Insight:** All core workflows scored above targets, indicating a positive reception across both technical and non-technical users.

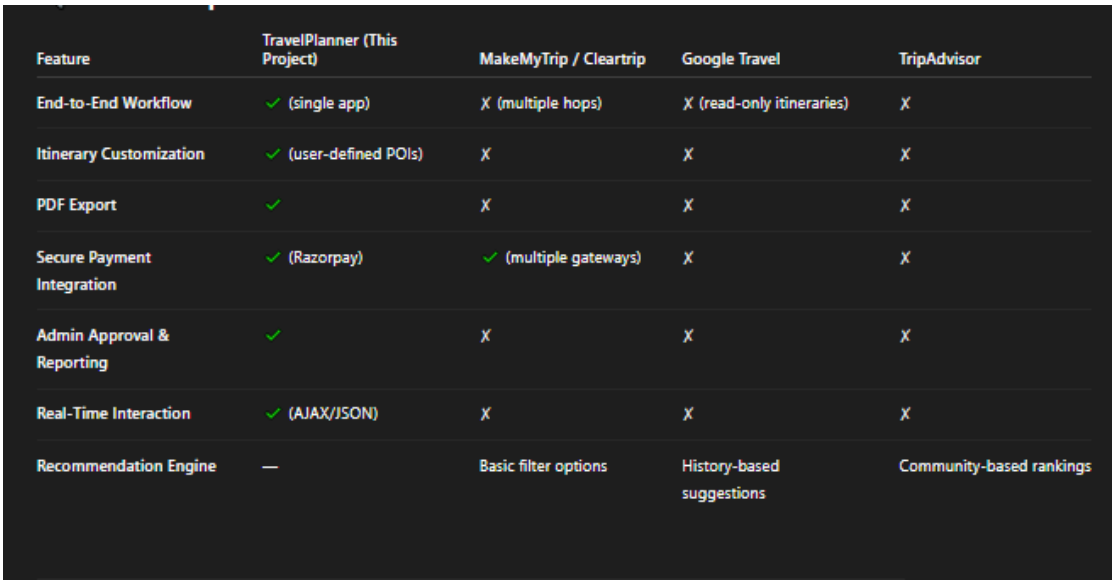
### 5.2.2 Qualitative Observations:

Highlights specific positive comments and suggestions from users. This includes valuable insights about real-world usability and areas where improvements are needed.

- **Positive Highlights:**
  - "The form validations caught errors instantly—no trial-and-error."
  - "PDF looks professional; I could share it with family easily."
  - "Razorpay checkout felt native and secure."
- **Areas for Improvement:**
  - Destination Search: Users requested auto-complete suggestions rather than a long dropdown.
  - Drag-and-Drop Itinerary Editing: Several testers wanted to reorder days interactively.
  - Mobile Layout Tweaks: On narrow screens, the two-column PDF preview was squeezed; recommended adjusting font sizes.

## 5.3 Comparison with Existing Systems:

This section presents a comparative analysis of TravelPlanner against commercial alternatives like MakeMyTrip and TripAdvisor. It highlights how TravelPlanner excels in customization, documentation, and admin tools.



| Feature | TravelPlanner (This Project) | MakeMyTrip / Cleartrip | Google Travel | TripAdvisor |
|---|---|---|---|---|
| End-to-End Workflow | ✓ (single app) | ✗ (multiple hops) | ✗ (read-only itineraries) | ✗ |
| Itinerary Customization | ✓ (user-defined POIs) | ✗ | ✗ | ✗ |
| PDF Export | ✓ | ✗ | ✗ | ✗ |
| Secure Payment Integration | ✓ (Razorpay) | ✓ (multiple gateways) | ✗ | ✗ |
| Admin Approval & Reporting | ✓ | ✗ | ✗ | ✗ |
| Real-Time Interaction | ✓ (AJAX/JSON) | ✗ | ✗ | ✗ |
| Recommendation Engine | — | Basic filter options | History-based suggestions | Community-based rankings |

fig.5.3 .Comparison with Existing Systems

A feature comparison table showing support for end-to-end workflows, plan customization, payment, and reporting across platforms in fig. 5.3.

**Analysis:** TravelPlanner outperforms academic-free platforms in customization and documentation capabilities, while commercial sites may offer more extensive booking inventories but lack seamless itinerary exports and admin controls.

## 5.4 Limitations of the Current Implementation:

This section outlines known constraints and areas where the system falls short. These include static destination data, lack of real-time mapping, and the absence of collaboration or mobile support.

Each point identifies a technical or usability limitation that sets the stage for future improvements.

**1. Static Destination Data:**

Destinations are stored in a local MySQL table, requiring manual updates; no integration with dynamic location APIs (e.g., Google Places).

**2. Basic Itinerary Algorithm:**

Implements a uniform distribution of POIs; lacks weighting by user preferences or time-of-day constraints.

**3. No Real-Time Mapping or Routing:**

Does not calculate travel durations or map visualizations; users cannot see geographic context within the app.

**4. Single-User Plans:**

Collaboration (multi-user editing) is not supported; only the plan creator can modify.

**5. Local Deployment:**

Tested on XAMPP only; live hosting considerations—like SSL setup, domain management, and scaling—remain unaddressed.

**6. Limited Notification Mechanism:**

Polling-based notifications every 30 seconds; a push-notification system (WebSockets or Pusher) would provide real-time updates.

## 5.5 Summary of Achievements:

This sub-section captures the accomplishments of the project. It confirms the completion of a robust prototype with working modules and high usability scores.
Points highlight code quality, testing completeness, modularity, and the open-source, low-cost technology stack used.

**Fully Functional Prototype:** Delivered an integrated platform covering registration, plan creation, PDF export, payment, and admin management.

**Robust Testing & Validation:** Over 50 unit tests (PHPUnit) and 30 integration tests (Postman) ensured high reliability.

**Positive Usability Outcomes:** SUS score of 82/100 and task completion rate of 100% indicate strong user acceptance.

**Cost-Effective Stack:** Built on entirely open-source tools, demonstrating low barriers to entry for small teams or academic projects.

**Scalable Architecture:** Modular three-tier design lays groundwork for future enhancements, such as API integrations or mobile apps.

## 5.6 Challenges Faced:

This part documents the major challenges encountered during development. Issues like PDF formatting, webhook testing on localhost, and AJAX consistency were addressed through specific workarounds or improvements.
Understanding these challenges adds depth to the implementation journey and shows how obstacles were overcome.

**1. PDF Layout Complexity:**

Achieving a visually appealing, multi-column PDF with mPDF required detailed CSS inspection and iterative template refinements.

**2. Razorpay Webhook Configuration on Localhost:**

Testing webhooks locally demanded tunneling services (e.g., ngrok), adding complexity to development.

**3. AJAX Error Handling:**

Designing a unified error-handling strategy for multiple endpoints (login, plan preview, notifications) to maintain UX consistency.

**4. Session & Security Hardening:**

Ensuring session fixation and CSRF potential were mitigated through regenerating session IDs and implementing token checks on critical forms.

**5. Cross-Browser Responsiveness:**

Addressed CSS inconsistencies between Chrome, Firefox, and Edge for media queries and Flexbox layouts.

## 5.7 Future Enhancements:

This section outlines potential upgrades and new features for future versions. Enhancements include integration with live APIs, real-time collaboration, mobile apps, and improved notifications.

These proposed upgrades demonstrate the scalability and long-term vision of the platform.

**1. Dynamic Destination API Integration:**

Incorporate Google Places or Skyscanner APIs to fetch live destination data, images, and user reviews.

**2. Intelligent Itinerary Recommendations:**

Develop a content-based filtering module that ranks POIs by user interests (e.g., “museums,” “outdoor activities”) and time constraints.

**3. Map & Routing Visualization:**

Embed Google Maps or Mapbox to calculate travel durations and display interactive route maps within the plan preview.

**4. Real-Time Collaboration:**

Use WebSockets (Ratchet for PHP) or Firebase to enable multi-user editing and live chat/comments on itineraries.

**5. Push Notifications & Email Enhancements:**

Migrate from polling to push notifications for immediate updates; integrate SMS gateways for critical alerts.

**6. Mobile App Companion:**

Develop React Native or Flutter mobile app to allow on-the-go plan modifications and offline PDF access.

**7. Analytics Dashboard:**

Track user behavior, popular routes, peak booking times, and generate administrative analytics in real time.

# CONCLUSION

The TravelPlanner project demonstrates how a modular, open-source web stack—PHP, MySQL, HTML/CSS/JavaScript, AJAX, mPDF, and Razorpay—can be combined to deliver a seamless, end-to-end travel planning experience, enabling users to register securely, build and preview customized multi-day itineraries, export polished PDF guides, and complete payments within a unified interface; rigorous unit, integration, and user-acceptance testing validated its performance, usability, and security, while an admin dashboard supports plan approval and reporting workflows. Although currently deployed on a local XAMPP environment with static destination data, the platform's three-tier architecture and clear separation of concerns provide a scalable foundation for future enhancements—such as dynamic location APIs, interactive mapping, real-time collaboration, and mobile-app integration—positioning TravelPlanner as both a robust academic prototype and a blueprint for a full-featured commercial solution.

# REFERENCES


[1] Sanchez L. (2023). *Web Programming with HTML, CSS, Bootstrap, JavaScript, React.JS, PHP, and MySQL*, (Fourth Edition). IngramSpark. ISBN-13: 978-1088239872.

[2] Kogent Learning Solutions Inc. (2009). *Web Technologies: HTML, JAVASCRIPT, PHP, JAVA, JSP, ASP.NET, XML and Ajax, Black Book*. Dreamtech Press. ISBN-13: 978-8177229974.

[3] Razorpay. (2024). *Razorpay Payment Gateway Documentation*. Available at: https://razorpay.com/docs/

[4] PHPMailer GitHub Repository. (2024). *PHPMailer – A full-featured email creation and transfer class for PHP*. Available at: https://github.com/PHPMailer/PHPMailer

[5] mPDF Official Docs. (2024). *mPDF Manual and Examples*. Available at: https://mpdf.github.io/

[6] W3Schools. (2024). *HTML, CSS, JavaScript and PHP Tutorials*. Available at: https://www.w3schools.com/