

INDUSTRY TRAINING PROJECT REPORT

GOVERNMENT POLYTECHNIC

MOHAMMADI KHERI, UTTAR PRADESH

WANDERLUST - A WEB-BASED VACATION RENTAL PLATFORM



STUDENT NAME: PANKAJ KUMAR

STREAM: DIPLOMA IN COMPUTER SCIENCE

SEMESTER: 5TH SEMESTER

SUBMISSION DATE: _____

TABLE OF CONTENTS

1. PROJECT OVERVIEW
2. OBJECTIVES AND SCOPE
3. SYSTEM ARCHITECTURE
4. TECHNOLOGY STACK
5. DATABASE DESIGN
6. KEY FEATURES AND FUNCTIONALITY
7. PROJECT STRUCTURE & SCREENSHOTS
8. INSTALLATION AND SETUP GUIDE
9. WORKING AND IMPLEMENTATION
10. CONCLUSION
11. FUTURE ENHANCEMENTS

1. PROJECT OVERVIEW

PROJECT NAME: WANDERLUST

Wanderlust is a full-stack web application that acts as an Airbnb-like vacation rental platform. It allows users to explore, list, and review unique places to stay around the world. The application provides a complete solution for both travelers seeking accommodations and property owners wanting to list their spaces.

PROJECT DESCRIPTION:

The project is inspired by Airbnb and provides a complete platform for users to:

- Create and manage property listings
- Browse available accommodations
- View detailed property information with interactive maps
- Leave reviews and ratings
- Manage user accounts with secure authentication

The application is built using modern web technologies including Node.js, Express.js, MongoDB, and EJS templating, demonstrating a comprehensive understanding of full-stack web development.

LIVE DEPLOYMENT:

URL: <https://wanderlust-a1m5.onrender.com/listings>

Platform: Render.com

GitHub Repository: <https://github.com/pankj-ctrl/wanderlust>

2. OBJECTIVES AND SCOPE

PRIMARY OBJECTIVES:

1. To develop a complete, production-ready web application using MERN stack technologies
2. To implement user authentication and authorization with secure password handling
3. To create a database schema for managing listings, reviews, and users
4. To integrate third-party APIs (Mapbox, Cloudinary) for enhanced functionality
5. To deploy the application on a cloud platform for real-world usage
6. To demonstrate understanding of MVC architecture and best practices in web development

SCOPE OF PROJECT:

FUNCTIONAL SCOPE:

- User Management:

- * User registration and authentication
- * Secure login/logout functionality
- * Session management with encrypted cookies
- * Password hashing and storage

- Listing Management:

- * Create new property listings with details (title, description, price, location, country, images)
- * View all listings in a grid layout
- * View detailed information for individual listings
- * Edit listings (owner only)
- * Delete listings (owner only)
- * Search functionality for listings by destination

- Review System:

- * Add reviews with ratings (1-5 stars) and comments
- * View all reviews for a listing with reviewer information
- * Delete reviews (author only)
- * Display average ratings

- Interactive Maps:

- * Display property locations on Mapbox GL JS
- * Geocoding addresses to coordinates
- * Interactive map features (zoom, pan)

- Image Management:

- * Upload property images during listing creation
- * Cloud storage using Cloudinary
- * Automatic image optimization

TECHNICAL SCOPE:

- **Frontend:** HTML5, CSS3, EJS templating, Bootstrap 5, Font Awesome icons
- **Backend:** Node.js, Express.js
- **Database:** MongoDB with Mongoose ODM
- **Authentication:** Passport.js with local strategy
- **External Services:** Cloudinary (image storage), Mapbox (geocoding and maps)
- **Deployment:** Render.com

NOT IN SCOPE:

- Payment processing/booking system
- Real-time messaging between users
- Admin dashboard
- Mobile native application
- Advanced analytics

3. SYSTEM ARCHITECTURE

ARCHITECTURAL PATTERN: MVC (Model-View-Controller)

The application follows a clean separation of concerns with:

- Models: MongoDB schemas defining data structure
- Views: EJS templates for rendering HTML
- Controllers: Business logic for handling requests

ARCHITECTURE LAYERS:

1. PRESENTATION LAYER (Frontend)

- EJS Templates for server-side rendering
- Bootstrap 5 for responsive UI
- Mapbox GL JS for interactive maps
- Font Awesome for icons

2. APPLICATION LAYER (Backend)

- Express.js routes for HTTP endpoints
- Middleware for authentication, validation, and error handling
- Controllers containing business logic
- Utilities for error handling and async operations

3. DATA LAYER (Database)

- MongoDB for NoSQL data storage
- Mongoose for schema definition and data validation
- Collections: users, listings, reviews

4. EXTERNAL SERVICES

- Cloudinary API for image storage and manipulation
- Mapbox API for geocoding and mapping

- Passport.js for authentication

DATA FLOW DIAGRAM:

User Interface (EJS Templates)



Express Routes and Middleware



Controllers (Business Logic)



Models (Mongoose Schema)



MongoDB Database



Controllers Process and Return Data



Render View or Send JSON Response



User Sees Updated Interface

REQUEST-RESPONSE CYCLE:

1. User submits form (e.g., create listing)
2. Browser sends HTTP request to Express server
3. Middleware processes request (authentication, validation)
4. Controller receives request and processes business logic
5. Model interacts with MongoDB database
6. Database returns data to controller
7. Controller prepares response
8. View rendered or JSON sent back
9. Browser receives response and displays to user

4. TECHNOLOGY STACK

FRONTEND TECHNOLOGIES:

- └─ HTML5
- └─ CSS3
- └─ EJS (Embedded JavaScript Templating)
- └─ Bootstrap 5.3.8 (Responsive CSS Framework)
- └─ Font Awesome 7.0.0 (Icon Library)
- └─ Mapbox GL JS 3.15.0 (Interactive Maps)
- └─ Google Fonts (Typography)

BACKEND TECHNOLOGIES:

- └─ Node.js (JavaScript Runtime)
- └─ Express.js (Web Framework)
- └─ Passport.js (Authentication)
- └─ Joi (Schema Validation)
- └─ dotenv (Environment Variables)

DATABASE:

- └─ MongoDB (NoSQL Database)
- └─ Mongoose (ODM - Object Document Mapper)
- └─ MongoDB Atlas (Cloud Database Hosting)

EXTERNAL SERVICES:

- └─ Cloudinary (Image Upload and Storage)
- └─ Mapbox (Geocoding and Maps API)
- └─ Render.com (Deployment Platform)

DEVELOPMENT TOOLS:

- └─ npm (Package Manager)
- └─ Git (Version Control)

5. DATABASE DESIGN

DATABASE: MongoDB (Cloud - MongoDB Atlas)

COLLECTIONS:

1. USERS COLLECTION

- └── _id (ObjectId - Primary Key)
- └── username (String - Unique, Required)
- └── password (String - Hashed, Required)
- └── email (String - Optional)
- └── __v (Version number - Auto)

2. LISTINGS COLLECTION

- └── _id (ObjectId - Primary Key)
- └── title (String - Required)
- └── description (String - Required)
- └── price (Number - Required, min: 0)
- └── location (String - Required)
- └── country (String - Required)
- └── owner (ObjectId - Foreign Key to Users)
- └── image (Object)
 - | └── filename (String)
 - | └── url (String - Cloudinary URL)
- └── geometry (GeoJSON Point for mapping)
 - | └── type: "Point"
 - | └── coordinates: [longitude, latitude]
- └── reviews (Array of ObjectIds - References to Reviews)
- └── __v (Version number - Auto)

3. REVIEWS COLLECTION

```
|── _id (ObjectId - Primary Key)  
|── rating (Number - Required, 1-5)  
|── comment (String - Required)  
|── author (ObjectId - Foreign Key to Users)  
└── __v (Version number - Auto)
```

RELATIONSHIPS:

- User → Listings (One-to-Many): One user can have many listings as owner
- Listing → Reviews (One-to-Many): One listing can have many reviews
- Review → User (Many-to-One): Each review has one author



6. KEY FEATURES AND FUNCTIONALITY

FEATURE 1: USER AUTHENTICATION AND AUTHORIZATION

- Passwords hashed using bcrypt algorithm
- Passport.js LocalStrategy for authentication
- Sessions stored in MongoDB with encryption
- HTTPOnly cookies prevent JavaScript access
- Session expiration set to 7 days

FEATURE 2: LISTING MANAGEMENT (CRUD)

- Create: Add new listings with image upload
- Read: Browse all listings or view single listing
- Update: Edit listing (owner only)
- Delete: Remove listing (owner only)

FEATURE 3: REVIEW SYSTEM

- Add reviews with ratings (1-5) and comments
- View all reviews on listing page
- Delete reviews (author only)
- Star ratings visualization

FEATURE 4: SEARCH FUNCTIONALITY

- Search bar in navigation
- Filter by title, location, country
- Case-insensitive matching
- Returns filtered listings

FEATURE 5: INTERACTIVE MAPS

- Display property locations on Mapbox
- Geocoding addresses to coordinates
- Map zoom, pan, and interaction

FEATURE 6: IMAGE MANAGEMENT

- Upload JPEG, JPG, PNG files
- Cloud storage on Cloudinary
- Automatic image optimization
- URL-based image delivery

FEATURE 7: FORM VALIDATION

- Client-side Bootstrap validation
- Server-side Joi schema validation
- Real-time feedback messages
- Error handling and display

FEATURE 8: RESPONSIVE DESIGN

- Mobile: < 576px
- Tablet: 576px - 992px
- Desktop: > 992px
- Bootstrap grid system

FEATURE 9: FLASH MESSAGES

- Success messages (green)
- Error messages (red)
- Auto-clear after display
- Session-based storage

FEATURE 10: NAVIGATION AND UI

- Sticky navigation bar with search
- Footer with links and social media
- Category filters (Trending, Rooms, Mountains, etc.)
- User authentication status display

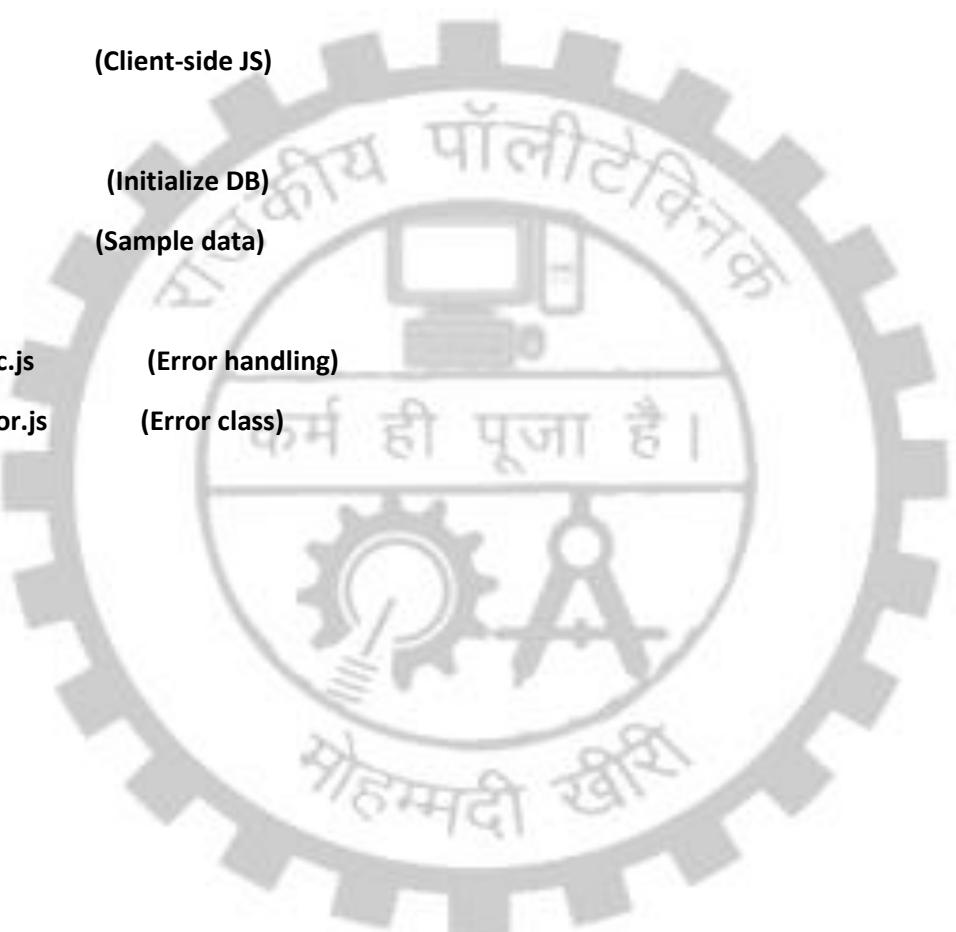
7. PROJECT STRUCTURE & SCREENSHOTS

PROJECT DIRECTORY STRUCTURE:

wanderlust/

└── app.js	(Main application file)
└── middleware.js	(Custom middleware functions)
└── schema.js	(Joi validation schemas)
└── cloudConfig.js	(Cloudinary configuration)
└── controllers/	
└── listing.js	(Listing CRUD operations)
└── review.js	(Review operations)
└── user.js	(User authentication)
└── models/	
└── listing.js	(Listing schema)
└── review.js	(Review schema)
└── user.js	(User schema)
└── routes/	
└── listing.js	(Listing routes)
└── review.js	(Review routes)
└── user.js	(User/Auth routes)
└── views/	
└── layouts/boilerplate.ejs	(Base layout)
└── includes/	
└── navbar.ejs	(Navigation bar)
└── footer.ejs	(Footer)
└── flash.ejs	(Flash messages)
└── listings/	
└── index.ejs	(All listings)
└── show.ejs	(Listing details)
└── new.ejs	(Create listing)

```
| | └ edit.ejs      (Edit listing)
| └ users/
|   └ login.ejs    (Login form)
|   └ signup.ejs   (Registration form)
└ public/
  └ CSS/
    └ style.css     (Custom styles)
    └ rating.css    (Star rating styles)
  └ js/
    └ script.js     (Client-side JS)
└ seeds/
  └ index.js       (Initialize DB)
  └ data.js        (Sample data)
└ utils/
  └ wrapAsync.js   (Error handling)
  └ ExpressError.js (Error class)
└ package.json
└ .env
└ readme.md
```



8. INSTALLATION AND SETUP GUIDE

SYSTEM REQUIREMENTS:

Hardware:

- RAM: Minimum 2GB
- Disk Space: Minimum 500MB
- Processor: Any modern processor

Software:

- Node.js v14 or higher
- npm (Node Package Manager)
- MongoDB (Atlas - cloud) or local MongoDB
- Git (for cloning repository)
- Any modern web browser

STEP-BY-STEP INSTALLATION:

STEP 1: Clone Repository

Command: `git clone https://github.com/pankj-ctrl/wanderlust.git`

Navigate: `cd wanderlust`

STEP 2: Install Dependencies

Command: `npm install --force`

Note: `--force` flag may be needed for dependency conflicts

Time: 2-5 minutes depending on internet speed

STEP 3: Create Environment Variables File

Create file: `.env` (in root directory)

Add following variables:

```
ATLASDB_URL=mongodb+srv://username:password@cluster.mongodb.net/wanderlust  
SECRET=your_super_secret_key_for_sessions_12345  
MAP_TOKEN=your_mapbox_public_api_token  
CLOUD_NAME=your_cloudinary_cloud_name  
CLOUD_API_KEY=your_cloudinary_api_key  
CLOUD_API_SECRET=your_cloudinary_api_secret  
NODE_ENV=development
```

STEP 4: Get Required API Keys

A. MongoDB Atlas:

- Visit: <https://www.mongodb.com/cloud/atlas>
- Create free account
- Create cluster
- Get connection string

B. Mapbox:

- Visit: <https://www.mapbox.com/>
- Sign up for free account
- Go to tokens page
- Copy default public token

C. Cloudinary:

- Visit: <https://cloudinary.com/>
- Sign up for free account
- Copy Cloud Name and API credentials

STEP 5: Initialize Database (Optional)

Command: node seeds/index.js

Purpose: Populate database with 10 sample listings

STEP 6: Start Development Server

Command: node app.js

Output: "Server is listening on Port number 8080"

Access: <http://localhost:8080>

STEP 7: Test Application

- Open <http://localhost:8080/listings> in browser
- You should see listings page
- Try signing up and creating a listing



9. WORKING AND IMPLEMENTATION

APPLICATION WORKFLOW:

USER REGISTRATION WORKFLOW:

1. User clicks "Sign Up" in navbar
2. GET /signup renders signup.ejs form
3. User fills: username, email, password
4. Form submitted: POST /signup
5. Validation and password hashing
6. User document created in MongoDB
7. User automatically logged in
8. Redirect to /listings
9. Flash: "Sign Up successful"

USER LOGIN WORKFLOW:

1. User clicks "Log in" in navbar
2. GET /login renders login.ejs form
3. User enters: username, password
4. Form submitted: POST /login
5. Passport authenticates credentials
6. Session created in MongoDB
7. User redirected to requested page
8. Flash: "Login successful"

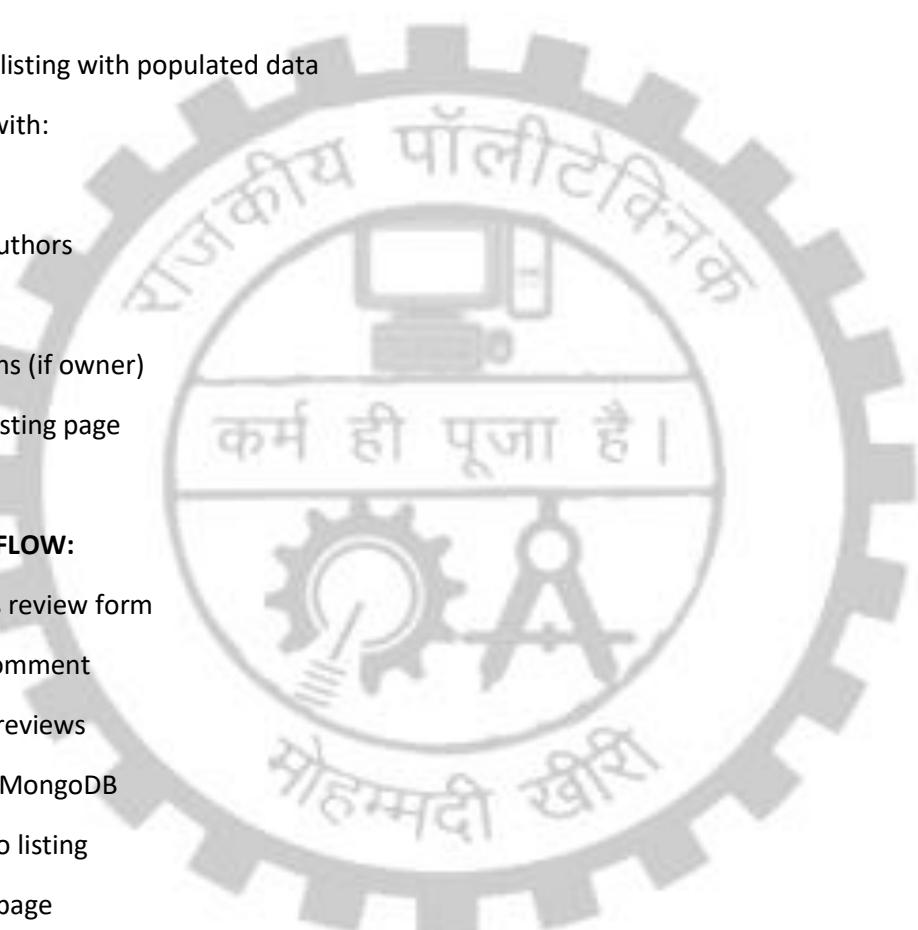
CREATE LISTING WORKFLOW:

1. Logged-in user: GET /listings/new
2. Render new.ejs form
3. User fills details and selects image
4. Form submitted: POST /listings
5. Joi validation

6. Image uploaded to Cloudinary
7. Location geocoded by Mapbox API
8. Listing saved to MongoDB with owner ID
9. Redirect to /listings
10. Flash: "New Listing Created"

VIEW LISTING WORKFLOW:

1. User clicks listing card
2. GET /listings/:id
3. Controller fetches listing with populated data
4. show.ejs renders with:
 - Property details
 - All reviews with authors
 - Interactive map
 - Edit/Delete buttons (if owner)
5. Browser displays listing page



ADD REVIEW WORKFLOW:

1. Logged-in user fills review form
2. Rating (1-5) and comment
3. POST /listings/:id/reviews
4. Review created in MongoDB
5. Review ID added to listing
6. Redirect to listing page
7. Review appears in list
8. Flash: "New Review Created"

DATABASE OPERATIONS:

CREATE:

- User registration: db.users.insertOne({...})
- New listing: db.listings.insertOne({...})
- New review: db.reviews.insertOne({...})

READ:

- All listings: db.listings.find({})
- Specific listing: db.listings.findById(id)
- User: db.users.findOne({username: "..."})

UPDATE:

- Update listing: db.listings.findByIdAndUpdateAndUpdate(id, {...})
- Add review to listing: db.listings.findByIdAndUpdateAndUpdate(id, {\$push: {reviews: ...}})

DELETE:

- Delete listing: db.listings.findByIdAndDelete(id)
- Delete review: db.reviews.findByIdAndDelete(reviewId)



10. CONCLUSION

PROJECT COMPLETION SUMMARY:

The Wanderlust project demonstrates a comprehensive understanding of full-stack web development using modern technologies. The application successfully integrates frontend, backend, database, and external services to create a production-ready vacation rental platform.

KEY ACHIEVEMENTS:

1. Complete CRUD Operations

- Create, read, update, delete operations fully implemented
- Data properly persisted in MongoDB
- Authorization enforced on sensitive operations

2. User Authentication

- Secure registration and login system
- Password hashing with bcrypt
- Session management with encryption
- Authorization middleware protecting routes

3. Database Design

- Normalized MongoDB schema
- Proper relationships between collections
- Data validation at multiple levels

4. API Integration

- Cloudinary for image uploads
- Mapbox for geocoding and maps
- Proper error handling

5. Responsive UI

- Bootstrap framework for rapid development
- Mobile-responsive design
- Intuitive user interface

6. Security

- Password hashing (bcrypt)
- CSRF protection via sessions
- Input validation (Joi schemas)
- Authorization checks
- HTTPOnly cookies

7. Code Organization

- MVC architecture
- Separation of concerns
- Reusable components
- Clean code structure

8. Deployment

- Successfully deployed to Render.com
- Application accessible online
- Database on MongoDB Atlas

LEARNING OUTCOMES:

1. Backend Development: Express.js, Node.js, RESTful APIs
2. Frontend Development: EJS, Bootstrap, Responsive Design
3. Database: MongoDB, Mongoose, Schema Design
4. Authentication: Passport.js, Sessions, Password Security
5. API Integration: Cloudinary, Mapbox
6. Deployment: Cloud hosting, Environment configuration
7. Best Practices: Error handling, validation, code organization

STRENGTHS:

- Production-ready code
- Follows industry best practices
- Secure implementation
- Well-organized structure
- Complete documentation
- Deployed and accessible

AREAS FOR IMPROVEMENT:

- Payment processing (Stripe)
- Advanced search filters
- User messaging system
- Admin dashboard
- Mobile app development
- Unit and integration tests

FINAL ASSESSMENT:

The Wanderlust project successfully demonstrates mastery of full-stack web development. The application is functional, secure, and deployed. The code quality is professional and maintainable. This project provides a strong foundation for future web development work.

11. FUTURE ENHANCEMENTS

SHORT-TERM ENHANCEMENTS (3-6 months):

1. Booking and Reservation System

- Calendar-based date selection
- Booking confirmations
- Cancellation policies

2. Advanced Search Filtering

- Price range filter
- Rating filter
- Amenities filter
- Location radius search

3. User Profile Features

- Profile pages with photo
- Listing history
- Review history

4. Enhanced Review System

- Photo uploads in reviews
- Review responses
- Helpful vote system

5. Wishlist/Favorites

- Save listings
- Create multiple wishlists
- Share wishlists

MID-TERM ENHANCEMENTS (6-12 months):

1. Payment Integration

- Stripe payment gateway
- Multiple payment methods
- Invoice generation

2. Messaging System

- Real-time messaging
- Chat history
- File sharing

3. Notification System

- Email notifications
- SMS notifications
- Push notifications

4. Hosting Features

- Listing analytics
- Booking dashboard
- Income tracking

5. Mobile Application

- Native iOS app
- Native Android app

LONG-TERM ENHANCEMENTS (12+ months):

1. Admin Dashboard

- User management
- Listing moderation
- Revenue analytics

2. Advanced Analytics



- User behavior tracking
- Booking trends
- Revenue forecasting

3. Machine Learning Features

- Personalized recommendations
- Dynamic pricing
- Fraud detection

4. Social Features

- User following
- Social feed
- Community discussions

5. Localization

- Multiple languages
- Currency conversion
- Regional customization

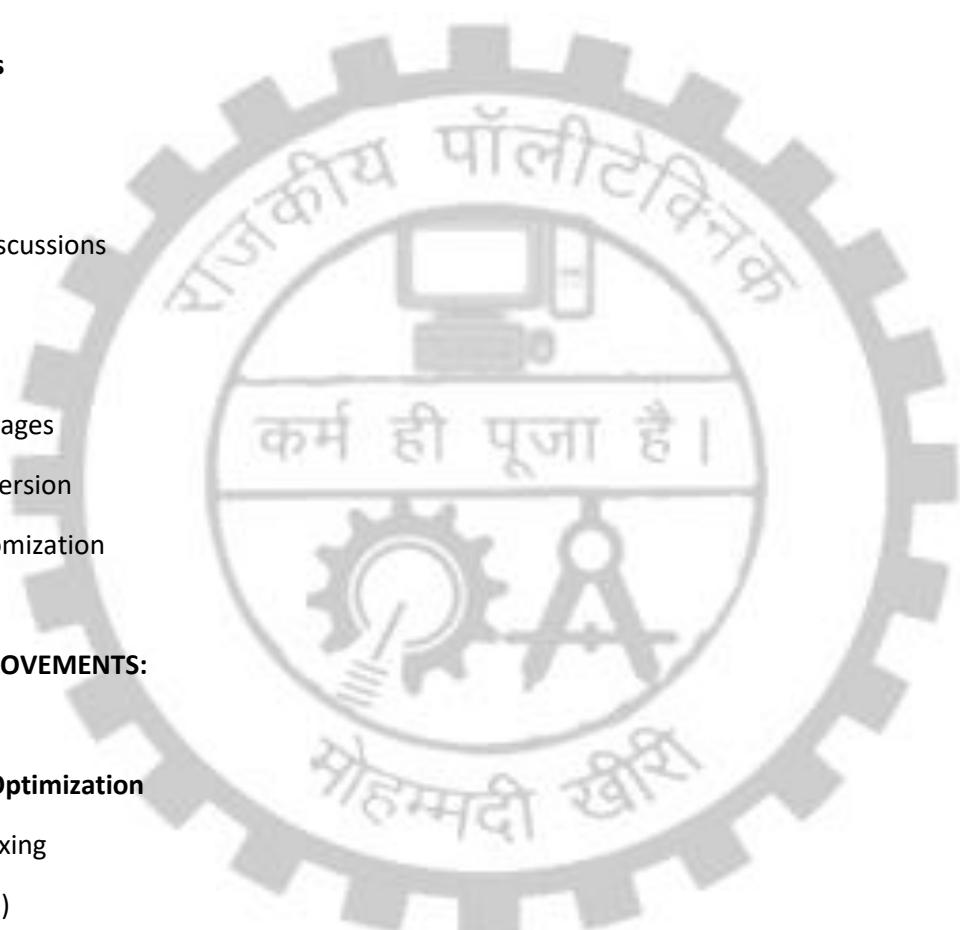
TECHNICAL IMPROVEMENTS:

1. Performance Optimization

- Database indexing
- Caching (Redis)
- CDN implementation
- Image optimization

2. Scalability

- Microservices
- Load balancing
- Database sharding
- Horizontal scaling



3. Testing

- Unit tests (Jest)
- Integration tests
- End-to-end tests
- Load testing

4. Infrastructure

- Docker containerization
- Kubernetes deployment
- CI/CD pipeline
- Automated backups

5. Monitoring

- Application logging
- Error tracking
- Performance monitoring
- Analytics



END OF REPORT

DOCUMENT INFORMATION:

Project: Wanderlust

Student: Pankaj Kumar

Institution: Government Polytechnic Mohammadi Kheri

Course: Diploma in Computer Science

Semester: 5th

Type: Industry Training Project Report

Report Prepared: December 11, 2025

Submission Date: _____

For any queries:

GitHub: <https://github.com/pankj-ctrl/wanderlust>

Live App: <https://wanderlust-a1m5.onrender.com/listings>

