**Hackathon Project Phases Template** for the **AutoSage App** project.

Hackathon Project Phases Template

# Project Title:

# Gemini Land Mark Description App Enhancing Tourist Experiences With AI

# Team Name:

Karthikeya

# Team Members:

* Edla Mahesh
* Abhiram
* Raj kumar
* Laxman
* sathvik

# Phase-1: Brainstorming & Ideation

## Objective:

## 1️⃣ Features & Enhancements

## 🖼️ Image Recognition

## Upload an image of a landmark and get an AI-generated description.

## AI detects historical significance, cultural importance, and fun facts.

## Real-time object detection to highlight key parts of the landmark.

## 🌎 AR & VR Integration

## Augmented Reality (AR): Use a camera to scan landmarks and overlay historical facts in real-time.

## Virtual Reality (VR): Create 360° landmark tours for remote users.

## 🗣️ Multilingual Support

## Translate landmark descriptions into multiple languages.

## Text-to-Speech (TTS) for tourists who prefer audio guides.

## 🎟️ Nearby Recommendations

## AI suggests nearby attractions, restaurants, and hotels based on user preferences.

## Show opening hours, ticket prices, and crowd levels.

## 📍 Interactive Maps

## GPS-powered maps guide tourists to the landmark.

## "Hidden Gems" Mode: Suggest less-known attractions around the area.

## 2️⃣ AI Technologies to Use

## 🧠 AI Models

## Google Gemini / OpenAI GPT-4: Generate landmark descriptions.

## Hugging Face Vision Transformers: Recognize landmarks from images.

## YOLOv8 / EfficientNet: For real-time landmark detection.

## 🗺️ APIs & Tools

## Google Maps API: Show locations & directions.

## Wikipedia API: Fetch landmark history & details.

## OpenWeather API: Show real-time weather at the landmark.

## Google Translate API: Provide multilingual descriptions.

## 3️⃣ Monetization Ideas 💰

## Freemium Model

## Free tier: Basic landmark descriptions.

## Premium tier: In-depth analysis, AR features, and audio guides.

## Affiliate Partnerships

## Partner with hotels, local tours, and restaurants for discounts & commissions.

## Advertising

## Show ads for tourist services near landmarks.

## 4️⃣ Future Expansion Ideas 🚀

## 🗿 Museum Mode: AI recognizes artifacts & paintings inside museums.

## 🎭 Culture Explorer: AI-generated stories based on historical events.

## 🌍 Global Landmark Challenges: Users compete to visit & scan multiple landmarks.

# Phase-2: Requirement Analysis

## Objective: Requirement Analysis for Landmark AI App

## This analysis covers functional requirements, non-functional requirements, and technical specifications to ensure a well-defined project scope.

## 1️⃣ Project Overview

## Name: Landmark AI Analyzer Purpose: Enhance tourist experiences by providing AI-powered landmark recognition, descriptions, and interactive features. Users: Tourists, travelers, students, and history enthusiasts.

## 2️⃣ Functional Requirements (FRs)

## Image Recognition

## Users can upload an image of a landmark. AI identifies the name, location, and history of the landmark. Provide cultural and historical insights about the landmark.

## Location-Based Features

## GPS-based landmark detection (using Google Maps API). ✅Show nearby attractions, hotels, and restaurants. Display real-time weather and crowd density at the landmark.

## 🗣️ Multilingual Support

## ✅ AI provides descriptions in multiple languages. ✅ Text-to-Speech (TTS) for audio-based guides.

## 📊 User Interaction

## ✅ Users can rate and review landmark descriptions. ✅ Users can save visited landmarks and create a personal travel diary. ✅ Option to share experiences on social media.

## 🎟️ Premium Features

## ✅ Access to in-depth historical analysis (Freemium model). ✅ Augmented Reality (AR) mode for real-time overlays. ✅ AI-generated personalized travel recommendations.

## 3️⃣ Non-Functional Requirements (NFRs)

## 📡 Performance & Scalability

## ✅ App should process image recognition within 3-5 seconds. ✅ Should handle high traffic loads (e.g., 1M+ users/month). ✅ Use cloud storage (AWS, Firebase) for image uploads.

## 🔒 Security & Privacy

## ✅ Use OAuth 2.0 / API key authentication for secure API access. ✅ Encrypt user data and images to protect privacy. ✅ GDPR compliance for handling user data.

## 💡 Usability

## ✅ Mobile-first design (Responsive UI for web & mobile). ✅ Simple, intuitive UX with easy navigation. ✅ Support for low-bandwidth internet connections.

## 4️⃣ Technical Specifications

## 🛠️ Technology Stack

## Frontend: React (Next.js) + Tailwind CSS

## Backend: Python (FastAPI / Flask)

## AI Models: Google Gemini, OpenAI GPT-4, Hugging Face Vision Transformers

## Database: Firebase Firestore / PostgreSQL

## Hosting: AWS / Vercel / Google Cloud

* + .

# Phase-3: Project Design

## Objective:

**📌 Project Design for Landmark AI App (Powered by Gemini AI)**

This document outlines the **architectural design, system components, data flow, and UI wireframe ideas** for the **Landmark AI app** powered by **Google Gemini AI**.

**1️⃣ System Architecture 🏗️**

The app follows a **client-server model** with AI processing on the backend.

**🌍 High-Level Architecture**

css

CopyEdit

[User] → [Frontend (React/Streamlit)] → [Backend (FastAPI)] → [Google Gemini AI API] → [Response]

**🛠️ Components & Technologies**

| **Component** | **Technology** |
| --- | --- |
| **Frontend** | React (Next.js) / Streamlit |
| **Backend** | Python (FastAPI / Flask) |
| **AI Model** | Google Gemini AI (via API) |
| **Database** | Firebase / PostgreSQL / MongoDB |
| **APIs Used** | Google Maps, Wikipedia, OpenWeather |
| **Authentication** | OAuth 2.0 / Firebase Auth |

**2️⃣ Data Flow Diagram (DFD) 🖥️**

**Step 1:** User uploads an image → **Frontend processes it**  
**Step 2:** Image is sent to **backend for AI analysis**  
**Step 3:** Backend **calls Gemini API** for landmark recognition  
**Step 4:** AI **returns landmark details** → stored in the database  
**Step 5:** The result is **displayed to the user**

css

CopyEdit

[User] → [Upload Image] → [Backend API] → [Google Gemini AI] → [Get Landmark Info] → [Display in App]

**3️⃣ Database Design 📂**

**🔹 Users Table**

| **UserID** | **Name** | **Email** | **SavedLandmarks** |
| --- | --- | --- | --- |
| 1 | John | j@xyz | [Eiffel Tower, Colosseum] |

**🔹 Landmarks Table**

| **LandmarkID** | **Name** | **Location** | **Description** |
| --- | --- | --- | --- |
| 101 | Eiffel Tower | France | A famous tower... |

**4️⃣ API Design 🔗**

**🔹 Endpoint: Upload Image & Get Landmark Details**

http

CopyEdit

POST /analyze\_landmark

**Request Body:**

json

CopyEdit

{

"image": "base64\_encoded\_string"

}

**Response:**

json

CopyEdit

{

"landmark": "Eiffel Tower",

"location": "Paris, France",

"history": "Built in 1889...",

"interesting\_facts": ["It was the tallest structure until 1930."]

}

**5️⃣ UI Wireframe 🎨**

**📌 Key Screens**

1. **Home Screen** → Upload an image
2. **Landmark Recognition Screen** → Displays landmark info
3. **Nearby Recommendations** → Restaurants, hotels, attractions
4. **Saved Landmarks** → User’s history

**6️⃣ Future Enhancements 🚀**

* 🏛️ **AR Integration**: Real-time landmark overlay
* 🌍 **Multilingual AI**: Auto-translate descriptions
* 🔊 **Voice Recognition**: Ask AI about landmarks

**🔥 Next Steps**

1. **Finalize API Integration with Gemini**
2. **Develop Frontend (React/Streamlit)**
3. **Test AI Model Accuracy**

# Phase-4: Project Planning (Agile Methodologies)

## Objective:

## 📌 Project Planning Using Agile Methodology (Landmark AI App)

## This document outlines the Agile-based project planning for the Landmark AI App powered by Google Gemini AI.

## 1️⃣ Agile Approach Overview 🏗️

## We will follow Scrum methodology, focusing on iterative development and continuous feedback.

## 📌 Key Agile Principles for This Project:

## ✅ Short sprints (2 weeks each) for quick releases ✅ Daily stand-ups to track progress ✅ User stories to define functionalities ✅ Frequent testing & feedback loops

## 2️⃣ Project Timeline (4 Sprint Plan) 📅

## The project is estimated to be completed in 8 weeks (4 sprints, each 2 weeks).

| Sprint | Goal | Key Tasks | Deliverables |
| --- | --- | --- | --- |
| Sprint 1 | Set up project structure & backend | - Set up React & FastAPI  - Integrate Gemini AI API  - Implement image upload | Backend API + Basic UI |
| Sprint 2 | AI-powered landmark recognition | - Process images via Gemini AI  - Fetch & display landmark details  - Implement database (Firebase/PostgreSQL) | Working AI model |
| Sprint 3 | UI & User Interaction Features | - Build frontend UI (React/Streamlit)  - Implement Google Maps API for location  - Add Nearby places & weather info | Complete front-end |
| Sprint 4 | Testing & Deployment | - User testing & bug fixes  - Deploy on Vercel/AWS  - Optimize AI responses | Fully functional app |

## 3️⃣ User Stories (Epics & Features) 📝

## Epic 1: Landmark Recognition ✅ As a user, I want to upload an image, so AI can recognize the landmark. ✅ As a user, I want AI-generated landmark descriptions, so I can learn history.

## Epic 2: Enhanced User Experience ✅ As a traveler, I want real-time GPS-based suggestions, so I can explore nearby places. ✅ As a user, I want to save landmarks in my profile, so I can revisit details later.

## Epic 3: AI-Powered Insights ✅ As a tourist, I want AI to suggest travel tips, so I can plan better. ✅ As a user, I want to translate descriptions, so I can read in my language.

## 4️⃣ Agile Tools & Collaboration 🛠️

## We will use: ✅ Jira / Trello → Task tracking ✅ GitHub → Version control ✅ Slack / Standups → Team collaboration ✅ Postman → API testing

## 5️⃣ Risk Assessment & Mitigation ⚠️

| Risk | Mitigation Strategy |
| --- | --- |
| API downtime (Gemini AI) | Implement fallback AI models |
| Slow image processing | Use efficient image compression |
| Limited AI accuracy | Fine-tune Gemini AI model with user feedback |
| Privacy concerns (user images) | Encrypt & anonymize images before processing |

## 6️⃣ Next Steps 🚀

## ✅ Finalize Sprint 1 (Backend & AI Setup) ✅ Create Figma UI Mockups ✅ Start landmark AI training & testing

# Phase-5: Project Development

## Objective:

## 📌 Project Development Plan for Landmark AI App

## This document outlines the step-by-step development process for the Landmark AI App powered by Google Gemini AI.

## 1️⃣ Development Workflow 🔄

## We will follow the Agile development process, with iterative releases and continuous integration.

## 📌 Key Development Steps:

## 1️⃣ Backend Development (FastAPI + Gemini AI API) 2️⃣ Frontend Development (React / Streamlit) 3️⃣ Database & API Integration 4️⃣ Testing & Bug Fixing 5️⃣ Deployment & Optimization

## 2️⃣ Tech Stack & Tools 🛠️

| Component | Technology |
| --- | --- |
| Frontend | React (Next.js) / Streamlit |
| Backend | FastAPI (Python) |
| AI Model | Google Gemini AI API |
| Database | Firebase / PostgreSQL / MongoDB |
| Cloud Storage | Google Cloud / AWS S3 |
| APIs Used | Google Maps, Wikipedia, OpenWeather |
| Authentication | OAuth 2.0 / Firebase Auth |

## 3️⃣ Development Phases & Tasks 🏗️

## 📌 Phase 1: Backend Development (FastAPI + Gemini AI)

## ✅ Set up FastAPI server ✅ Create API endpoints for image upload & analysis ✅ Integrate Gemini AI API for landmark recognition ✅ Return AI-generated landmark details

## 🚀 Sample API Endpoint:

## python

## CopyEdit

## from fastapi import FastAPI, File, UploadFile

## import requests

## import base64

## app = FastAPI()

## GEMINI\_API\_KEY = "YOUR\_API\_KEY"

## @app.post("/analyze\_landmark/")

## async def analyze\_landmark(image: UploadFile = File(...)):

## img\_data = base64.b64encode(await image.read()).decode("utf-8")

## response = requests.post(

## f"https://api.gemini.com/v1/landmark-detection?key={GEMINI\_API\_KEY}",

## json={"image": img\_data}

## )

## return response.json()

## 📌 Phase 2: Frontend Development (React / Streamlit)

## ✅ Build UI for image upload ✅ Create API request to backend ✅ Display AI-generated landmark details ✅ Enhance UI with Google Maps API

## 🚀 Sample React Code:

## javascript

## CopyEdit

## const analyzeLandmark = async () => {

## if (!image) return;

## const formData = new FormData();

## formData.append("image", image);

## const response = await fetch("http://localhost:8000/analyze\_landmark/", {

## method: "POST",

## body: formData,

## });

## const data = await response.json();

## setDescription(data.landmark);

## };

## 📌 Phase 3: Database & API Integration

## ✅ Set up Firebase / PostgreSQL for storing user history ✅ Integrate Google Maps API for displaying landmark locations ✅ Implement authentication (OAuth 2.0 / Firebase Auth)

## 🚀 Sample Database Schema (PostgreSQL):

## sql

## CopyEdit

## CREATE TABLE landmarks (

## id SERIAL PRIMARY KEY,

## name TEXT NOT NULL,

## location TEXT,

## description TEXT,

## image\_url TEXT

## );

## 📌 Phase 4: Testing & Bug Fixing 🧪

## ✅ Unit Testing (Test API responses) ✅ Frontend Testing (Ensure UI works smoothly) ✅ Performance Testing (Optimize AI processing speed) ✅ Security Testing (Check API security & user data protection)

## 🚀 Testing Tools: 🔹 Postman (API testing) 🔹 Jest (Frontend testing) 🔹 Pytest (Backend testing)

## 📌 Phase 5: Deployment & Optimization 🚀

## ✅ Deploy backend on Google Cloud / AWS Lambda ✅ Deploy frontend on Vercel / Netlify ✅ Optimize AI API calls for efficiency ✅ Monitor app performance & user feedback

## 4️⃣ Next Steps 📅

## ✅ Complete AI API integration ✅ Enhance UI with more interactivity ✅ Start Beta Testing with real users

# Phase-6: Functional & Performance Testing

## Objective: 📌 Functional & Performance Testing Plan for Landmark AI App

## This document outlines the testing strategy for the Landmark AI App, covering functional and performance testing to ensure a reliable and efficient system.

## 1️⃣ Functional Testing 🧪

## Functional testing ensures that all features work correctly as per the requirements.

## 📌 Key Functional Testing Areas:

| Feature | Test Case | Expected Result |
| --- | --- | --- |
| Image Upload | Upload different image formats (JPEG, PNG, etc.) | Image is uploaded successfully |
| AI Landmark Recognition | Upload an image of a well-known landmark | Correct landmark is identified |
| Incorrect Image Handling | Upload a non-landmark image (random objects) | AI should return "No landmark found" |
| API Error Handling | Simulate API failure | App should display an error message |
| User Authentication | Login with valid/invalid credentials | Correct user validation is performed |
| Database Integration | Store user searches in Firebase/PostgreSQL | Search history is stored correctly |
| Google Maps Integration | Fetch and display landmark location on the map | Correct location is shown on the map |

## 🚀 Example Functional Test:

## Scenario: Upload an image of the Eiffel Tower.

## Expected Output: AI correctly identifies it as the Eiffel Tower and provides a description.

## 2️⃣ Performance Testing ⚡

## Performance testing ensures the app runs smoothly under different conditions.

## 📌 Key Performance Testing Metrics:

| Metric | Test Scenario | Expected Result |
| --- | --- | --- |
| Response Time | AI recognition should return a result in <3s | Response time remains optimal |
| Concurrent Users | Simulate 100+ users uploading images | App handles concurrent requests smoothly |
| API Latency | Measure API call time to Gemini AI | Response is within 1-2s |
| Server Load | Test backend under high request volume | No server crashes or timeouts |
| Image Size Handling | Upload large images (5MB+) | Image is processed within limits |
| Database Performance | Retrieve search history for 1000+ users | Database queries execute efficiently |

## 🚀 Example Performance Test:

## Scenario: Simulate 100 concurrent users uploading images.

## Expected Output: The AI system should process requests within acceptable response times without crashing.

## 3️⃣ Tools for Testing 🛠️

## To ensure thorough testing, the following tools will be used:

| Testing Type | Tool |
| --- | --- |
| Unit Testing | Pytest, Jest |
| API Testing | Postman |
| Load Testing | JMeter, Locust |
| UI Testing | Cypress, Selenium |
| Monitoring | Google Cloud Monitoring |

## 4️⃣ Test Execution Plan 📅

## 1️⃣ Phase 1: Unit Testing (Individual components) 2️⃣ Phase 2: API Testing (Backend endpoints) 3️⃣ Phase 3: UI Testing (User Interface) 4️⃣ Phase 4: Load Testing (Stress & performance) 5️⃣ Phase 5: Security & Optimization (Final improvements)

## 5️⃣ Final Deliverables ✅

## ✅ Test Report: Summary of passed/failed test cases. ✅ Performance Report: Metrics for response time & scalability. ✅ Bug Fixes: Address identified issues.

# Final Submission

1. **Project Report Based on the templates**
2. **Demo Video (3-5 Minutes)**
3. **GitHub/Code Repository Link**
4. **Presentation**