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

Hackathon Project Phases Template

Project Title:

VisionTagger AI: Leveraging Google's Gemini Vision Pro for Advanced Image Tagging and Detailed JSON Metadata

Team Name:The steady path

Members:

- M.Niharika
- M.Vaishnavi
- V.Mounika

Phase-1: Brainstorming & Ideation

Objective:

The **VisionTagger AI**, powered by **Google's Gemini Vision Pro**, automates advanced image tagging and generates detailed **JSON metadata**. Its objectives are:

1. **Accurate Image Tagging**: Identifies and labels objects, people, and scenes within images.
2. **Advanced Vision Capabilities**: Utilizes Gemini Vision Pro's cutting-edge AI models for precise object detection and classification.

3. **Detailed JSON Metadata**: Provides rich data like object attributes, locations, and confidence scores for each image.
4. **Efficient Data Labeling**: Speeds up the creation of labeled datasets for machine learning model training.
5. **Improved AI Performance**: Enhances AI model accuracy by supplying high-quality metadata for better training data.

Key Points:

1. **Problem Statement:**
   Manual Image Tagging Challenge: Manually labeling large image datasets is time-consuming, error-prone, and inefficient, hindering the creation of high-quality training data for AI models.

   VisionTagger AI Solution: By leveraging **Google's Gemini Vision Pro**, VisionTagger AI automates accurate image tagging and generates detailed **JSON metadata**, improving tagging speed, model performance, and enabling advanced analysis with minimal manual effort.

2. **Proposed Solution:**
   Automated Image Tagging: Implement VisionTagger AI powered by Google's Gemini Vision Pro to automatically identify and label objects, people, and scenes in images with high accuracy, reducing manual effort and errors.
   Detailed Metadata Generation: The AI will generate comprehensive JSON metadata for each image, including object attributes, relationships, and confidence scores, providing rich data for better analysis, search, and AI model training.
   Scalability and Efficiency: VisionTagger AI will streamline the data labeling process, enabling faster and more consistent creation of high-quality datasets, significantly improving AI model training and real-time application performance.

3. **Target Users:**
   The target users are AI/ML developers and data scientists who need efficient, accurate image tagging for model training and analysis. Additionally, industries like retail, healthcare, and security requiring automated image categorization and metadata generation**.**

4. **Expected Outcome:**

   ○ A functional **AI-powered vehicle information app** that provides insights based on real-time data and user queries.

Phase-2: Requirement Analysis

Objective:

To leverage Google's Gemini Vision Pro for advanced image tagging and generate detailed JSON metadata for enhanced image analysis and insights.

Key Points:

Sure! Here's a concise summary of the technical requirements for the VisionTagger AI project:

1.Technical Requirements

1. Development Environment
- Python : Programming language
- IDE/Editor: Visual Studio Code, PyCharm, Jupyter Notebook
- Version Control : Git and GitHub

2. Libraries and Frameworks
- Machine Learning : TensorFlow, Keras
- Image Processing : OpenCV, Pillow
- Web Framework : Flask
- HTTP Requests : requests
- Data Handling : pandas
- Deployment : pyngrok

3. APIs and Services
- Gemini Vision Pro : API for image processing
- Ngrok : Tool to expose local server to the internet

4. Hardware
- GPU : Recommended for faster model training (optional in local development)
- RAM : 16GB+ recommended

5. Database
- Type : SQLite or PostgreSQL
- Access Library : SQLAlchemy (optional)

6. Hosting and Deployment
- Server : Cloud-based server (AWS, Google Cloud, Heroku)
- Containerization : Docker (optional)

7. Security

- Data Encryption : Secure storage and transmission
- Authentication : User authentication and authorization

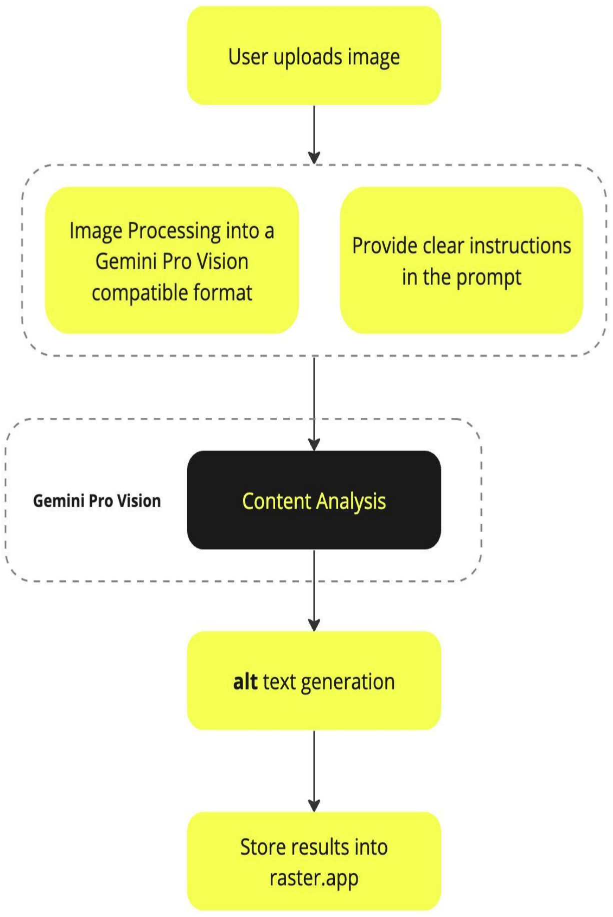
8. Performance and Scalability
- Caching : Redis
- Load Balancing : Implement as needed

This overview covers the key technical requirements for building and deploying the VisionTagger AI project effectively. Let me know if you need more details on any part!

Phase-3: Project Design

Objective:

Architecture


User uploads image
Image Processing into a Gemini Pro Vision compatible format
Provide clear instructions in the prompt
Gemini Pro Vision
Content Analysis
alt text generation
Store results into raster.app

Key Points:

Key Points for VisionTagger AI Architecture

1. Frontend (Client Side)
   - Technologies : HTML, CSS, JavaScript
   - Functionality : Image upload interface, display of metadata
   - Components :
     - Image upload form
     - Display area for metadata and tags

2. Backend (Server Side)
   - Technology : Python with Flask
   - Functionality :
     - Handle image uploads
     - Process images using Gemini Vision Pro API
     - Generate and return JSON metadata
   - Components :
     - API endpoints (`/upload`, `/process`, `/metadata`)
     - Integration with Gemini Vision Pro API
     - Business logic for image processing

3. Gemini Vision Pro API

   - Role : Provide advanced image processing and tagging capabilities
   - Functionality :
     - Analyze images
     - Generate tags and detailed metadata

4. Database

   - Type : SQLite for simplicity or PostgreSQL for scalability
   - Functionality :
     - Store uploaded images
     - Store generated metadata
   - Components :
     - Tables for image data
     - Tables for metadata

5. Deployment

   - Tools :
     - Ngrok for development (exposing local server)
     - Docker (optional, for containerization)
     - Cloud Services (AWS, Google Cloud, Heroku for production)
   - Functionality :
     - Host and manage the Flask application

  - Ensure scalability and reliability

6. Security

 - Measures :

  - Data encryption for secure storage and transmission

  - User authentication and authorization

 - Functionality :

  - Protect user data and ensure secure access

7. Performance and Scalability

 - Strategies :

  - Caching (e.g., Redis)

  - Load balancing for handling high traffic

 - Functionality :

  - Optimize response times

  - Ensure system can scale with demand

User Actions & Feedback:

1. User Actions

Image Upload

- Action : User accesses the web interface and uploads an image from their device.

- Expected Outcome : The image is successfully uploaded to the server.

Image Processing

- Action : User submits the uploaded image for processing.

- Expected Outcome : The backend processes the image using Gemini Vision Pro, and metadata is generated.

View Metadata

- Action : User views the generated metadata and tags in the web interface.

- Expected Outcome : The metadata is displayed in a clear and organized format.

Download JSON

- Action : User downloads the detailed JSON metadata for the uploaded image.

- Expected Outcome : The JSON file is successfully downloaded to the user's device.

Provide Feedback

- Action : User provides feedback on the accuracy of the tags and metadata.

- Expected Outcome : Feedback is recorded and used for improving the system.

Error Handling

- Action : User encounters an error during image upload or processing.

- Expected Outcome : User receives a clear error message and guidance on how to resolve the issue.

2. User Feedback

Accuracy of Tags

- Feedback : Users may provide feedback on the relevance and accuracy of the tags generated by Gemini Vision Pro.

- Use : Feedback is used to fine-tune the tagging algorithm and improve accuracy.

Metadata Clarity

- Feedback : Users may comment on the clarity and comprehensiveness of the generated metadata.

- Use : Feedback helps in refining the format and content of the metadata provided.

User Interface Usability

- Feedback : Users may share their experience with the usability of the web interface, including ease of image upload and navigation.

- Use : Feedback is used to enhance the user experience and interface design.

Performance

- Feedback : Users may report on the speed and responsiveness of the system during image upload and processing.

- Use : Feedback helps identify performance bottlenecks and areas for optimization.

Security and Privacy

- Feedback : Users may express concerns or satisfaction with how their data is handled and stored.

- Use : Feedback informs security enhancements and data privacy measures.

Overall Satisfaction

- Feedback : Users may provide overall satisfaction ratings and suggestions for improvements.

- Use : Overall feedback is used to guide future development and feature updates.

1. **User Flow:**

User Flow for VisionTagger AI

1. Access the Web Interface
   - Action : User opens the VisionTagger AI web application in their browser.
   - Entry Point : Home page of the web application.

2. Upload an Image
   - Action : User clicks the "Upload Image" button.
   - Input : User selects an image file from their local device or cloud storage.
   - System Response : The image is displayed in a preview area, and the upload form is updated to show the file name and an "Upload" button.

3. Submit the Image for Processing
   - Action : User clicks the "Upload" button to submit the image for processing.
   - Input : The selected image file.
   - System Response : The image file is sent to the backend server for processing. A loading indicator is displayed to inform the user that processing is in progress.

4. Process the Image
   - Backend Action : The backend server receives the image and sends it to the Gemini Vision Pro API for processing.
   - API Action : Gemini Vision Pro API processes the image and generates metadata, including tags and descriptions.
   - System Response : The backend receives the metadata from the API and stores it in the database.

5. Display Metadata
   - System Action : The backend retrieves the generated metadata from the database.
   - Output : The metadata, including tags and descriptions, is displayed on the web interface.
   - System Response : The user sees the metadata along with the image preview.

6. Download JSON Metadata
   - Action : User clicks the "Download JSON" button to download the metadata.
   - Output : A JSON file containing the metadata is downloaded to the user's device.

7. Provide Feedback (Optional)
   - Action : User provides feedback on the accuracy and relevance of the tags and metadata.
   - Input : Feedback form or rating system.
   - System Response : Feedback is recorded and used to improve the system.

8. Error Handling (If Applicable)
   - Action : If an error occurs (e.g., failed upload, processing error), the user is informed with a clear error message.
   - System Response : The error message provides guidance on how to resolve the issue or try again.

2. UI/UX Considerations:

1. User Interface (UI) Design
- Simplicity and Clarity:
  - Use a clean and minimalist design to ensure the interface is not cluttered.
  - Ensure that all buttons and interactive elements are clearly labeled.

- Consistency :
  - Maintain a consistent style throughout the application, including fonts, colors, and button styles.
  - Use consistent layouts for different sections of the application.

- Responsiveness :
  - Design the interface to be responsive and adaptable to different screen sizes and devices.
  - Ensure that the application works seamlessly on both desktop and mobile devices.

- Visual Hierarchy :
  - Organize content in a hierarchical manner to guide users' attention to the most important elements first.
  - Use size, color, and spacing to create a clear visual hierarchy.

- Accessibility :
  - Ensure the application is accessible to users with disabilities by adhering to web accessibility standards (e.g., WCAG).
  - Provide alternative text for images and ensure sufficient color contrast.

2. User Experience (UX) Design
- User Flow :
  - Design intuitive user flows to guide users through the process of uploading images and viewing metadata.

  - Simplify complex tasks into manageable steps to prevent user frustration.

- Feedback and Response :
  - Provide immediate feedback for user actions, such as uploading an image or submitting a form.
  - Use loading indicators to show progress during image processing and metadata generation.

- Error Handling :
  - Display clear and concise error messages when something goes wrong.
  - Provide guidance on how users can resolve errors or try again.

- Performance :
  - Optimize the application to ensure fast load times and smooth interactions.
  - Minimize delays during image upload and processing to enhance user satisfaction.

- Security and Privacy :
  - Clearly communicate how user data and images are handled and stored securely.
  - Provide users with control over their data, including options to delete uploaded images and metadata.

3. User Engagement
- User Feedback :
  - Include mechanisms for users to provide feedback on the accuracy and relevance of the tags and metadata.
  - Use feedback to continually improve the application and its features.

- Personalization:
  - Allow users to customize their experience by providing options to select preferred output formats for metadata.
  - Offer personalized recommendations based on user interactions and preferences.

- Onboarding and Tutorials :
  - Provide an onboarding process or tutorial to guide new users through the application's features.
  - Include tooltips and help sections to assist users in understanding how to use the application effectively.

4. Aesthetics
- Visual Design :
  - Use a visually appealing color palette that aligns with the application's branding.
  - Incorporate high-quality images and icons to enhance the overall look and feel.

- Typography :
  - Select readable fonts and appropriate font sizes for different sections of the application.

  - Use font styles to create emphasis and distinguish between headings, subheadings, and body text.

5. Testing and Iteration
- User Testing :
  - Conduct user testing sessions to gather feedback on the interface and user experience.
  - Observe users as they interact with the application and identify areas for improvement.

- Iterative Design :
  - Continuously refine the design based on user feedback and testing results.
  - Implement small changes incrementally to improve the user experience over time.

Phase-4: Project Planning (Agile Methodologies)

Objective:

Break down development tasks for efficient completion.

Sprint	Task	Priority	Duration	Deadline	Assigned To	Dependencies	Expected Outcome
Sprint 1	Environment Setup & API Integration	High	6 hours (Day 1)	End of Day 1	Mounika	Google API Key, Python.	API connection established & working
Sprint 1	Frontend UI Development	Medium	2 hours (Day 1)	End of Day 1	Vaishnavi	API response format finalized	Basic UI with input fields
Sprint 2	Vehicle Search & Comparison	High	3 hours (Day 2)	Mid-Day 2	Niharika	API response, UI elements ready	Search functionality with filters
Sprint 2	Error Handling & Debugging	High	1.5 hours (Day 2)	Mid-Day 2	Niharika, Vaishnavi	API logs, UI inputs	Improved API stability
Sprint 3	Testing & UI Enhancements	Medium	1.5 hours (Day 2)	Mid-Day 2	Mounika	API response, UI layout completed	Responsive UI, better user experience
Sprint 3	Final Presentation & Deployment	Low	1 hour (Day 2)	End of Day 2	Entire Team	Working prototype	Demo-ready project

Sprint Planning with Priorities

Sprint 1 – Setup & Integration (Day 1)

(**High Priority)** Set up the **environment** & install dependencies.
(**High Priority)** Integrate **Google Gemini API**.
(**Medium Priority)** Build a **basic UI with input fields**.

Sprint 2 – Core Features & Debugging (Day 2)

(**High Priority)** Implement **search & comparison functionalities**.
(**High Priority)** Debug API issues & handle **errors in queries**.

Sprint 3 – Testing, Enhancements & Submission (Day 2)

(**Medium Priority)** Test API responses, refine UI, & fix UI bugs.
(**Low Priority)** Final **demo preparation & deployment**.

Phase-5: Project Development

Objective:

To leverage Google's Gemini Vision Pro for advanced image tagging and generate detailed JSON metadata for enhanced image analysis and insights.

Key Points:

1. **Technology Stack Used:**
   - **Frontend:** Html,Css
   - **Backend:** Google Gemini Flash API
   - **Programming Language:** Python
2. **Development Process:**
   - Implement **API key authentication** and **Gemini API integration**.
   - Develop **vision tagger and maintenance tips logic**.
   - Optimize **search queries for performance and relevance**.
3. **Challenges & Fixes:**

- **Challenge:** Delayed API response times.
  Fix: Implement **caching** to store frequently queried results.
- **Challenge:** Limited API calls per minute.
  Fix: Optimize queries to fetch **only necessary data**.

Phase-6: Functional & Performance Testing

Objective:

To leverage Google's Gemini Vision Pro for advanced image tagging and generate detailed JSON metadata for enhanced image analysis and insights.

Test Case ID	Category	Test Scenario	Expected Outcome	Status	Tester
TC-001	Functional Testing	Diverse image upload	High resolution and low resolution images are processed accurately	✅ Passed	Niharika
TC-002	Functional Testing	High Traffic and concurrent users	The System handles multiple concurrent uploads without significant delays or crashes	✅ Passed	Vaishnavi
TC-003	Performance Testing	API response and error handling	Users are able to retry failed requests without issues	⚠️ Needs Optimization	Mounika
TC-004	Bug Fixes & Improvements	Security and Privacy	Users data privacy is maintained, and sensitive information is protected	✅ Fixed	Developer
TC-005	Final Validation	User experience across devices	Users feedback indicates a high level of accuracy of the tags and descriptions provided	❌ Failed - UI broken on mobile	Niharika
TC-006	Deployment Testing	Image processing accuracy	The system maintains high reliability and uptime over an extended period	Deployed	DevOps

---