**Hackathon Project Phases Template** that ensures students can complete it efficiently while covering all six phases. The template is structured to capture essential information without being time-consuming.

Hackathon Project Phases Template

# Project Title:

GEMINI VISION ANNOTATION

# Team Name:

TECHNOS

# Team Members:

* RAMPEY SREEVARDHINI
* THIRUVAIPATI SATYA PRIYA
* YEMPATI SOWMYA

# Phase-1: Brainstorming & Ideation

## Objective:

* Traditional image annotation is time-consuming and labor-intensive. Annotators often spend time on redundant or easily predictable annotations.
* Define the purpose and impact of the project.

## Key Points:

1. **Problem Statement:**  Traditional image annotation is time-consuming and labor-intensive. Annotators often spend time on redundant or easily predictable annotations.
2. **Proposed Solution: GVA utilizes Gemini's vision API for initial image understanding and combines it with an active learning approach. This means the system intelligently selects the most informative images or image regions for human annotators to focus on, maximizing the value of their time.**
3. **Target Users: Security and Surveillance, Autonomous Vehicles, Healthcare and Medical Imaging:**
4. **Expected Outcome: Efficient and Accurate Image Labeling, Efficient and Accurate Image Labeling,The system will help human annotators focus their efforts on the most critical parts of the image data, rather than having to manually annotate large volumes of less informative data. This maximizes the impact of human involvement and minimizes burnout or fatigue.**

# Phase-2: Requirement Analysis

## Objective:

The system must integrate with the Gemini vision API for initial image analysis, which can process raw images to identify relevant features and provide outputs such as bounding boxes, classifications, and uncertainty scores for active learning.

The API should have high accuracy and low latency for real-time or near-real-time image analysis and feedback.

## Key Points:

1. **Technical Requirements: Tool: Gemini's vision API for initial image understanding (image classification, object detection, segmentation, etc.).**

**Languages: Python, JavaScript (for web integration), or any language supported by the Gemini API SDK.**

1. **Functional Requirements: The system should use Gemini's vision API (or similar) to process input images and classify objects, detect regions of interest, or segment the image into meaningful components.**

**The system must automatically identify and extract key features, such as objects, boundaries, or specific regions in the images that are crucial for further processing.**

**Constraints & Challenges:**  **Challenge**: The success of the system depends heavily on the **quality** and **quantity** of annotated data available for training the model. Insufficient or poor-quality labeled data may lead to **biased models** or **inaccurate predictions**.

 **Constraint**: In some use cases, high-quality labeled datasets may be scarce, or they may require expensive manual annotation efforts.

# Phase-3: Project Design

## Objective:

* + Create the architecture and user flow.

## Key Points:

1. **System Architecture Diagram: +---------------------------------------------------------------+**
2. **| Input Data Layer |**
3. **| |**
4. **| +------------------+ +-----------------------------+ |**
5. **| | Raw Image Input | | External Data Sources | |**
6. **| +------------------+ +-----------------------------+ |**
7. **+---------------------------------------------------------------+**
8. **|**
9. **v**
10. **+---------------------------------------------------------------+**
11. **| Image Processing Layer |**
12. **| |**
13. **| +-------------------------+ +-----------------------+ |**
14. **| | Gemini Vision API | | Pre-processing Module | |**
15. **| +-------------------------+ +-----------------------+ |**
16. **+---------------------------------------------------------------+**
17. **|**
18. **v**
19. **+---------------------------------------------------------------+**
20. **| Active Learning Loop |**
21. **| |**
22. **| +----------------------------+ +---------------------+ |**
23. **| | Model Confidence Scoring | | Data Selection | |**
24. **| +----------------------------+ | Algorithm (Uncertainty)| |**
25. **| +---------------------+ |**
26. **| +---------------------+ |**
27. **| | Human Annotators | |**
28. **| | (Annotation UI) | |**
29. **| +---------------------+ |**
30. **+---------------------------------------------------------------+**
31. **|**
32. **v**
33. **+---------------------------------------------------------------+**
34. **| Model Training and Improvement |**
35. **| |**
36. **| +----------------------+ +------------------------+ |**
37. **| | Training Dataset | | Model Retraining | |**
38. **| +----------------------+ +------------------------+ |**
39. **| | Model Evaluation | | Model Deployment | |**
40. **| +----------------------+ +------------------------+ |**
41. **+---------------------------------------------------------------+**
42. **|**
43. **v**
44. **+---------------------------------------------------------------+**
45. **| Data Management and Storage |**
46. **| |**
47. **| +------------------------+ +---------------------+ |**
48. **| | Image Database | | Version Control | |**
49. **| +------------------------+ +---------------------+ |**
50. **| | Metadata Storage | |**
51. **| +------------------------+ |**
52. **+---------------------------------------------------------------+**
53. **|**
54. **v**
55. **+---------------------------------------------------------------+**
56. **| User Management and Security |**
57. **| |**
58. **| +---------------------+ +------------------------+ |**
59. **| | RBAC | | Audit Logs | |**
60. **| +---------------------+ +------------------------+ |**
61. **+---------------------------------------------------------------+**
62. **|**
63. **v**
64. **+---------------------------------------------------------------+**
65. **| Feedback and Reporting |**
66. **| |**
67. **| +------------------+ +----------------------------+ |**
68. **| | Performance | | Feedback Loop | |**
69. **| | Dashboard | +----------------------------+ |**
70. **| +------------------+ |**

**+---------------------------------------------------------------+**

**2.User Flow: The user interaction with this project, which combines Gemini's Vision API for image understanding and active learning, can be described through a series of user tasks and interfaces. Users in this context could include human annotators, administrators, and data scientists. Each user role has a different way of interacting with the system, contributing to its overall function. Below is a breakdown of how each user type will interact with the project.**

**1. Human Annotator Interaction**

**Goal: Annotators label images or image regions that have been selected by the system for annotation.**

* **Login/Authentication:**
  + **The annotator logs into the system through a secure authentication interface.**
  + **RBAC (Role-Based Access Control) ensures that only users with proper credentials can access the annotation interface.**
* **Receiving Annotation Tasks:**
  + **The annotator sees a list of images or image regions that need annotation, selected by the active learning algorithm based on the confidence score (i.e., the system chooses the most uncertain or ambiguous data).**
  + **The images may come with pre-determined bounding boxes, object segments, or classifications generated by the Gemini Vision API. Annotators can verify or correct these labels.**
* **Annotation Interface:**
  + **The annotator interacts with an intuitive UI that allows them to:**
    - **Correct existing labels: Modify, add, or delete labels (e.g., "cat" → "dog").**
    - **Draw Bounding Boxes or Segments: If the system has suggested an object detection or segmentation task, annotators can modify or adjust bounding boxes or segmentation masks.**
    - **Classify Unlabeled Objects: Annotators can identify and label previously unseen objects.**

**3.UI/UX Considerations: +--------------------------+--------------------------+**

**| Left Sidebar | Image Display Area |**

**|--------------------------+--------------------------|**

**| [ ] To Do | +--------------------+ |**

**| [ ] In Progress | | | |**

**| [ ] Completed | | Image Here | |**

**|--------------------------+ | | |**

**| [ ] Filter by Class | +--------------------+ |**

**|--------------------------+--------------------------+**

**| [ ] Filter by Confidence | [ ] Label: \_\_\_\_\_ |**

**|--------------------------+--------------------------+**

**| [ ] Mark as Reviewed | [ ] Add Bounding Box |**

**+--------------------------+--------------------------+**

# Phase-4: Project Planning (Agile Methodologies)

## Objective:

* + Break down the tasks using Agile methodologies.

## Key Points:

1. **Sprint Planning: Team Member 1: UI/UX Design & Implementation (2 tasks)**
2. **Task 1: Annotator Dashboard Design**
3. **Duration: 2 Days**
4. **Description: Design the UI for the Annotator Dashboard. Implement basic layout for:**
5. **Task management (view, filter, and select images for annotation).**
6. **Image display with current bounding boxes and labels.**
7. **Annotation tools (draw bounding boxes, label objects).**
8. **Progress tracker for annotations.**
9. **Deliverables: Annotator dashboard wireframe implemented and ready for integration with backend.**
10. **Task 2: Admin Dashboard Design**
11. **Duration: 2 Days**
12. **Description: Design the UI for the Admin Dashboard, which includes:**
13. **User management (creating, editing, and deleting users).**
14. **System health (status indicators for Gemini API, model, and active learning system).**
15. **Audit logs and performance reporting.**
16. **Deliverables: Admin dashboard implemented with user management and health monitoring.**
17. **Team Member 2: Image Display & Prediction Results (2 tasks)**
18. **Task 3: End User Interface for Image Upload & Predictions**
19. **Duration: 3 Days**
20. **Description: Create the interface for end users to upload images and display model predictions, including:**
21. **Drag-and-drop area for image upload.**
22. **Display prediction results with confidence scores.**
23. **Show images with bounding boxes/segments for detected objects.**
24. **Deliverables: End user interface that allows image uploads and displays prediction results.**
25. **Task 4: Integration with Backend API (Frontend-Backend Integration)**
26. **Duration: 2 Days**
27. **Description: Ensure the frontend for image upload is connected to the backend to send and receive image prediction data.**
28. **Deliverables: Integrated frontend for image upload and predictions, ensuring proper communication with the backend API.**

**2.Task Allocation:** SOWMYA- Building the solution

**Satya Priya –** building the solution

**Sreevardhini -**submisson of the solution

**3.Timeline & Milestones:**

# Phase-5: Project Development

## Objective:

* + Code the project and integrate components.

## Key Points:

1. **Technology Stack Used: Python: For backend logic, machine learning tasks, and interacting with the Gemini Vision API. Python is widely used in AI/ML-related applications and allows easy integration with libraries and tools like TensorFlow, PyTorch, etc.**
2. **Development Process: Collaborate with stakeholders to finalize the technical and functional requirements.**

**Define the scope of the system, which includes:**

**User types: Annotators, data scientists, administrators.**

**Core features: Image upload, annotations (bounding boxes, segmentation), active learning pipeline, model predictions, role-based access, real-time updates, and system monitoring.**

**Plan the development phases using Sprint planning, breaking down the work into tasks (with milestones) for each team member.**

**Output:**

**Project requirements and feature list.**

**Development plan and task breakdown.**

**Wireframes and system architecture.**

**3.Challenges & Fixes: Challenge: Integration with Gemini Vision API**

**Problem:**

* **Gemini Vision API (or any external image recognition API) could have issues such as high latency, unpredictable API response times, or rate limits, which can delay image processing or even cause failures.**

**Solution:**

* **Asynchronous Processing: Implement asynchronous API calls to ensure the user interface is not blocked while waiting for API responses. This allows users to continue working while the backend processes requests.**
* **Retry Logic & Caching: Add retry mechanisms and caching of previously processed images or results to reduce the number of API calls and speed up subsequent requests.**
* **Rate Limiting Handling: Implement error handling for rate limits (e.g., exponential backoff or queuing requests), ensuring smooth operation during high traffic periods.**

# Phase-6: Functional & Performance Testing

## Objective:

* + Ensure the project works as expected.

## Key Points:

1. **Test Cases Executed: Scenario: Test login functionality for valid users (Annotators, Admins).**

**Expected Result: The user is successfully logged in, and the respective dashboard (Annotator/ Admin) is displayed.**

1. **Bug Fixes & Improvements: Image Upload Failure for Large Files**

**Problem: Users were unable to upload images larger than a certain size (e.g., > 10 MB). The upload process failed without any error message.**

**Fix: Increased the file size limit for image uploads and implemented file size validation checks. An appropriate error message is now displayed when the user exceeds the upload size limit (e.g., "File size too large").**

1. **Final Validation: The final validation process ensures that the Image Annotation System with Active Learning and Gemini Vision API Integration fulfills all the initial requirements outlined during the planning phase. Below is a detailed breakdown of whether the project meets the functional, technical, and non-functional requirements:**

**1. Functional Requirements Validation**

**1.1 Image Upload & Storage**

**Requirement: The system should support the upload of images in various formats and store them securely.**

**Validation: The system successfully handles image uploads in multiple formats (JPG,PNG) and stores them in a cloud storage solution (e.g., AWS S3). Image metadata is saved correctly in the database, ensuring retrieval is possible for future annotation tasks.**

**Conclusion: Met.**

1. **Deployment (if applicable): Hosting Details**

**The Image Annotation System has been deployed using cloud-based services to ensure scalability, reliability, and performance. Below are the details of the hosting infrastructure:**

**Cloud Provider: Amazon Web Services (AWS)**

**EC2 Instances: Used for hosting the backend services and the API layer. Auto-scaling is enabled to manage traffic spikes efficiently.**

**S3 Storage: Utilized for secure image storage. All uploaded images and related metadata are stored and accessed via AWS S3 buckets.**

**RDS (Relational Database Service): AWS RDS is used to store all annotation data, user information, and task-related metadata. The database is backed up regularly for data safety.**

**Elastic Load Balancer (ELB): Distributes incoming traffic evenly across multiple backend servers to ensure high availability and optimal performance.**

**CloudFront CDN: For fast delivery of images and assets to users, a Content Delivery Network (CDN) like AWS CloudFront is used to ensure that data is served from the nearest location to users worldwide.**

# Final Submission

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