MANIPAL
UNIVERSITY JAIPUR
University under Section 2(f) of the UGC Act

Centre for Distance & Online Education

Manipal University Jaipur

Dehmi Kalan, Rajasthan – 303 007.

Project Final Report

On

AI-Based-Image-Classifier

Name	ASHA S R
Roll Number	2214100645
Program	BACHELOR OF COMPUTER APPLICATIONS (BCA)
Semester	VI
Session	MAR 2024

Contents

Introduction

The Image Prediction System is a web-based application designed to classify images uploaded by users. Utilizing the power of machine learning, specifically the MobileNetV2 model, the system provides accurate predictions for a wide range of image categories. The application is built with a user-friendly interface, allowing users to easily upload images, view predictions, and manage their previous predictions.

Objectives

The primary objectives of this project are:

1. **Develop an Image Prediction System**: Create a robust system that can classify images with high accuracy using a pre-trained MobileNetV2 model.
2. **User-Friendly Interface**: Design an intuitive web interface that allows users to upload images, view predictions, and manage their prediction history.
3. **Efficient Data Management**: Implement a database to store predictions, ensuring data integrity and security.
4. **Scalability and Performance**: Ensure the system is scalable and performs efficiently, even with a large number of users and image uploads.
5. **Security**: Implement security measures to protect user data and manage access rights effectively.

Repository Details:

- **Repository Hosting Service**: GitHub
- **Repository URL**: https://github.com/ashasr-arch/image-prediction-system
- **Repository Name**: Image Prediction System
- **Repository Type**: Public

System Analysis

Identification Of Need

In today's digital age, there is a growing need for automated systems that can quickly and accurately classify images. Such systems can be used in various applications, including content moderation, medical imaging, and e-commerce. The Image Prediction System addresses this need by providing a reliable and efficient tool for image classification.

Preliminary Investigation

Initial research was conducted to identify suitable machine learning models and frameworks for image classification. MobileNetV2 was selected due to its balance of accuracy and efficiency. Additionally, Flask was chosen as the web framework for its simplicity and ease of use.

Feasibility Study

A feasibility study was conducted to evaluate the technical, economic, and operational feasibility of the project.

- **Technical Feasibility**: The project is technically feasible with the use of MobileNetV2 for image classification and Flask for the web interface. The required technologies and tools are readily available and well-documented.
- **Economic Feasibility**: The project is economically feasible as it leverages open-source technologies, reducing development costs. The primary costs are associated with development time and potential cloud hosting services.
- **Operational Feasibility**: The project is operationally feasible as it addresses a clear need and can be easily integrated into existing workflows. The user-friendly interface ensures that users can quickly adopt the system.

Project Planning

The project was planned in phases to ensure systematic development and timely completion. The phases include:

1. **Requirement Gathering**: Collecting and documenting the requirements for the system.
2. **Design**: Creating the system architecture, database schema, and user interface design.
3. **Implementation**: Developing the system components, including the machine learning model, web interface, and database integration.
4. **Testing**: Conducting unit tests, integration tests, and system tests to ensure the system functions correctly.
5. **Deployment**: Deploying the system to a production environment and making it available to users.

Project Scheduling

PERT Chart

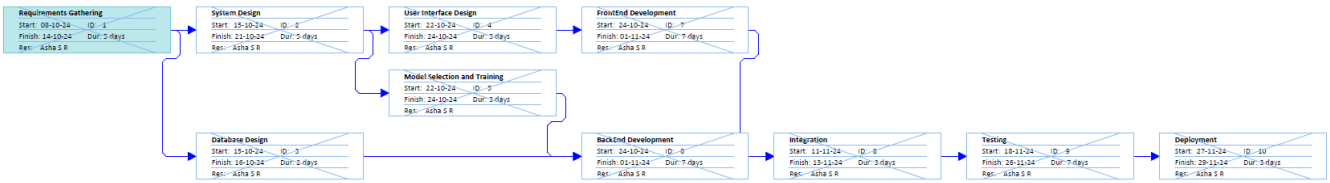


Figure 1: PERT Chart

Gantt Chart

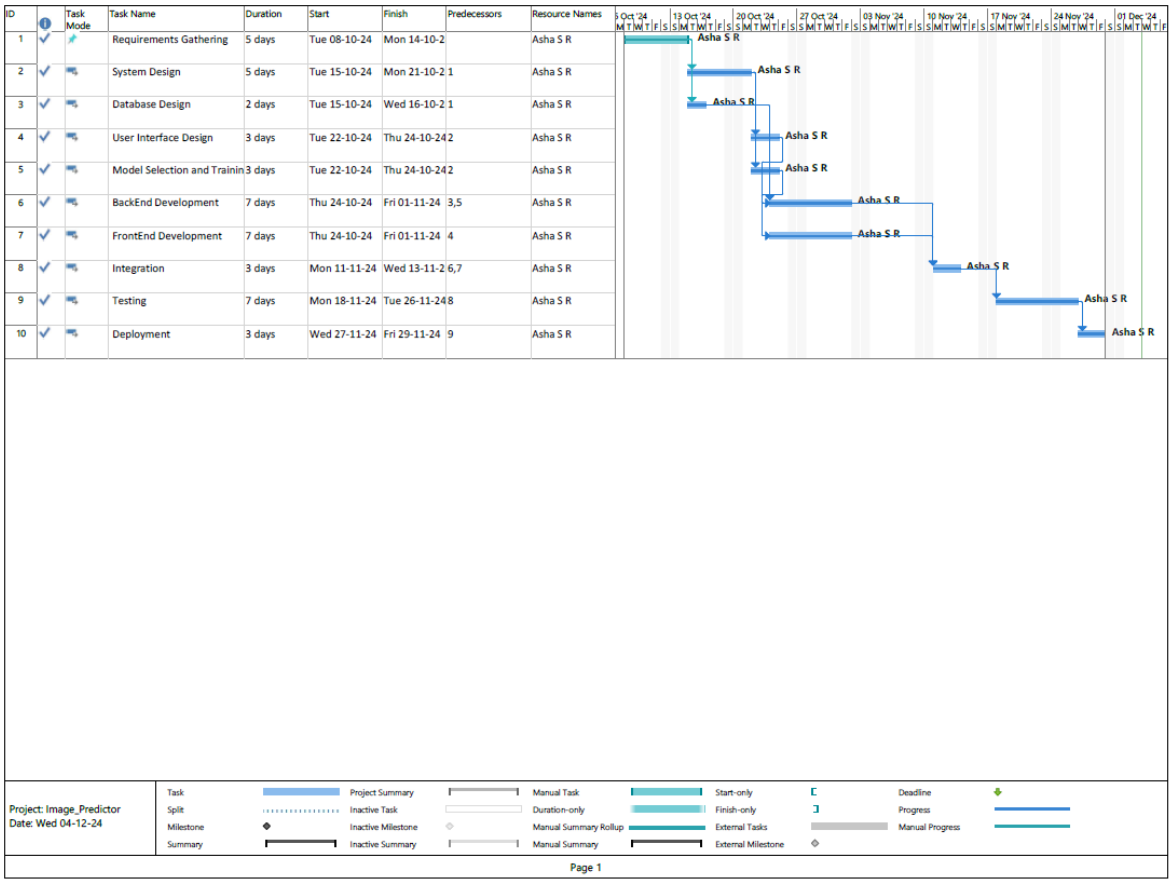


Figure 2: GANTT Chart

Software Requirement Specifications (SRS)

Agile Methodology is used for the project management.

Introduction to Agile Methodology

Agile methodology is a software development paradigm that emphasizes iterative development, collaboration, and flexibility. It is designed to accommodate changes and deliver functional software incrementally. Agile methodologies prioritize customer satisfaction, continuous feedback, and adaptive planning.

Key Principles of Agile Methodology

1. **Customer Collaboration**: Agile methodologies emphasize close collaboration with customers to ensure that the software meets their needs and expectations. Customers are involved throughout the development process, providing feedback and clarifying requirements.

2. **Iterative Development**: Agile projects are divided into small, manageable iterations or sprints, typically lasting 1-4 weeks. Each iteration results in a potentially shippable product increment, allowing for continuous delivery of value.

3. **Adaptive Planning**: Agile methodologies embrace change and allow for adaptive planning. Requirements and priorities are revisited and adjusted at the end of each iteration based on customer feedback and changing business needs.

4. **Cross-functional Teams**: Agile teams are composed of cross-functional members with diverse skills, including developers, testers, designers, and product owners. This promotes collaboration and ensures that all aspects of the project are considered.

5. **Continuous Improvement**: Agile methodologies encourage continuous improvement through regular retrospectives. Teams reflect on their processes and practices, identifying areas for improvement and implementing changes in subsequent iterations.

Agile Practices Applied in the Image Predictor Application

1. **User Stories and Backlog**: Requirements for the Image Predictor Application were captured as user stories and maintained in a product backlog. Each user story represents a small, valuable piece of functionality from the user's perspective.

2. **Sprint Planning**: The project was divided into multiple sprints, each lasting two weeks. During sprint planning meetings, the team selected user stories from the backlog to work on in the upcoming sprint, based on priority and team capacity.

3. **Daily Stand-ups**: Daily stand-up meetings were held to discuss progress, identify any blockers, and plan the day's work. This ensured that the team remained aligned and could address issues promptly.

4. **Incremental Delivery**: At the end of each sprint, a potentially shippable product increment was delivered. This allowed for continuous delivery of value and early detection of issues.

5. **Sprint Review and Retrospective**: Sprint review meetings were held at the end of each sprint to demonstrate the completed work to stakeholders and gather feedback. Retrospective

meetings were conducted to reflect on the sprint, identify areas for improvement, and plan changes for the next sprint.

Benefits of Using Agile Methodology

1. **Flexibility and Adaptability**: Agile methodologies allow for changes in requirements and priorities, enabling the team to respond to evolving customer needs and market conditions.
2. **Customer Satisfaction**: Continuous collaboration with customers ensures that the software meets their expectations and delivers value incrementally.
3. **Improved Quality**: Regular testing and feedback loops help identify and address issues early, resulting in higher-quality software.
4. **Faster Time-to-Market**: Incremental delivery of functional software allows for faster release cycles and quicker realization of value.
5. **Enhanced Collaboration**: Cross-functional teams and regular communication foster collaboration and ensure that all aspects of the project are considered.

Data flow diagram(DFD)

In the Data Flow Diagram (DFD) for the Image Prediction System, we will outline the main processes, data stores, and data flows. A DFD typically consists of four components: processes, data stores, data flows, and external entities.

Level 0 DFD (Context Diagram)

1. **External Entities**:
    - User
2. **Processes**:
    - Image Upload
    - Image Prediction
    - Manage Predictions
3. **Data Stores**:
    - Predictions Database
4. **Data Flows**:
    - Upload Image
    - Predicted Result
    - View Predictions
    - Delete Prediction

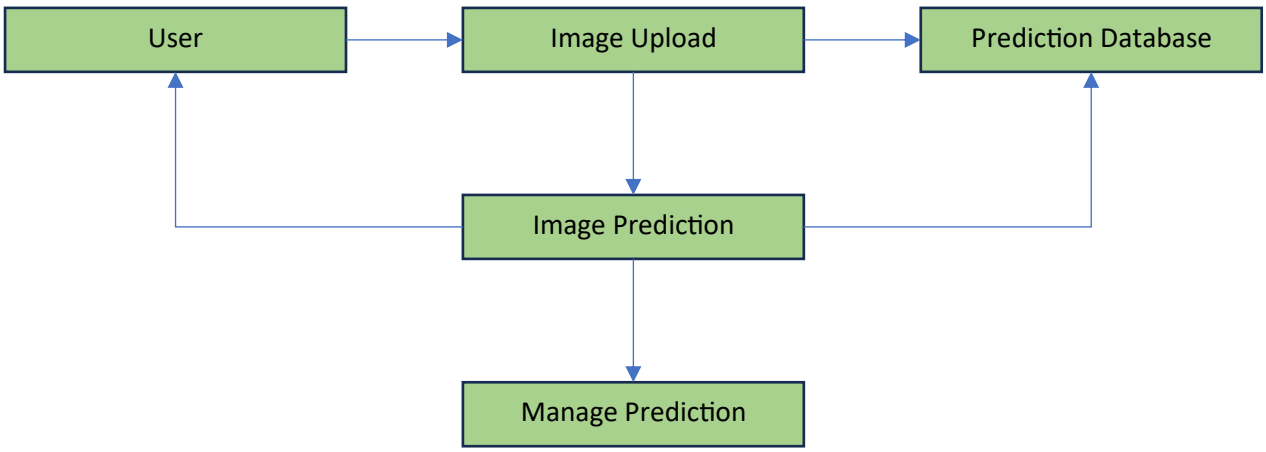


Figure 3: Level 0 DFD

DFD Level 1 (Detailed Diagram)

1. **Processes**:
    - 1.0 Image Upload
    - 2.0 Image Prediction
    - 3.0 Manage Predictions
2. **Data Stores**:
    - D1 Predictions Database
3. **Data Flows**:
    - 1.1 Upload Image
    - 2.1 Process Image
    - 2.2 Predicted Result
    - 3.1 View Predictions
    - 3.2 Delete Prediction

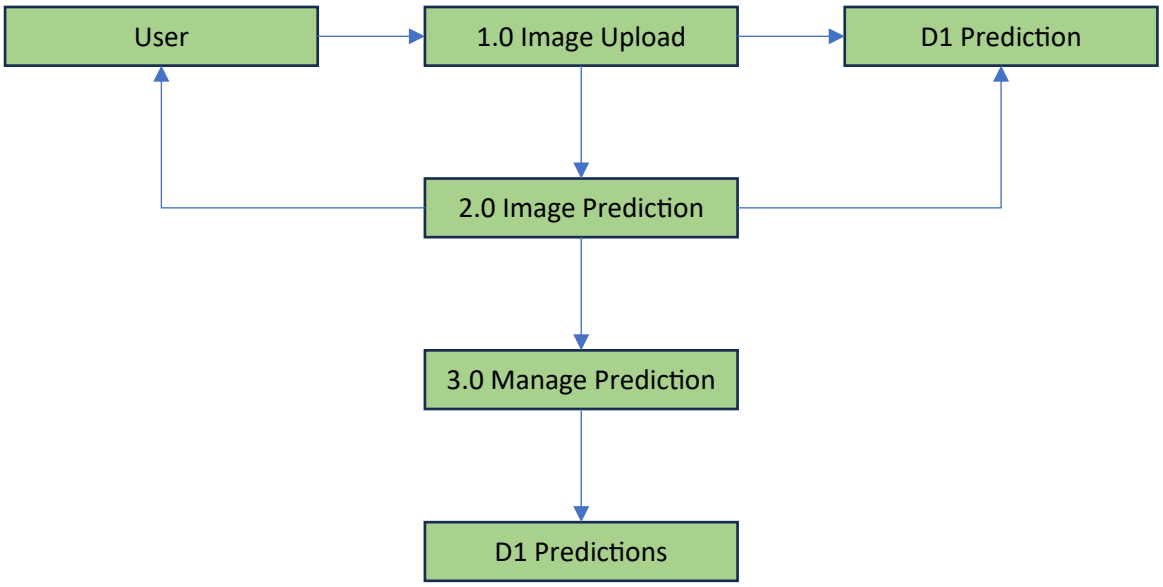


Figure 4: Level 1 DFD

1. **User**: The external entity that interacts with the system.
2. **1.0 Image Upload**: The process where the user uploads an image to the system.
3. **2.0 Image Prediction**: The process where the uploaded image is processed, and a prediction is made using the MobileNetV2 model.
4. **3.0 Manage Predictions**: The process where users can view and delete their previous predictions.
5. **D1 Predictions Database**: The data store where all predictions are stored.

Control Flow Diagram (CFD):

A Control Flow Diagram (CFD) is used to represent the flow of control in a system. It shows the sequence of operations and the control flow between different processes. Below is a Control Flow Diagram for the Image Prediction System.

Control Flow Diagram (CFD)

1. **Processes**:
    - Image Upload
    - Image Prediction
    - Manage Predictions
2. **Control Flows**:
    - Start

- Upload Image
- Predict Image
- View Predictions
- Delete Prediction
- End

CFD Representation

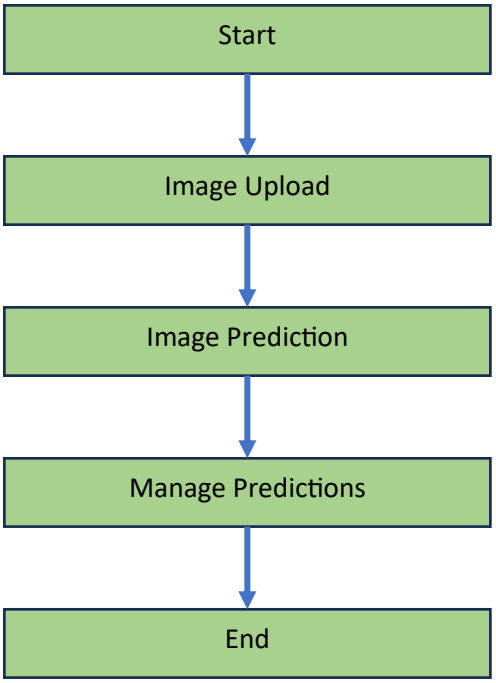


Figure 5: Control Flow Diagram

Detailed CFD

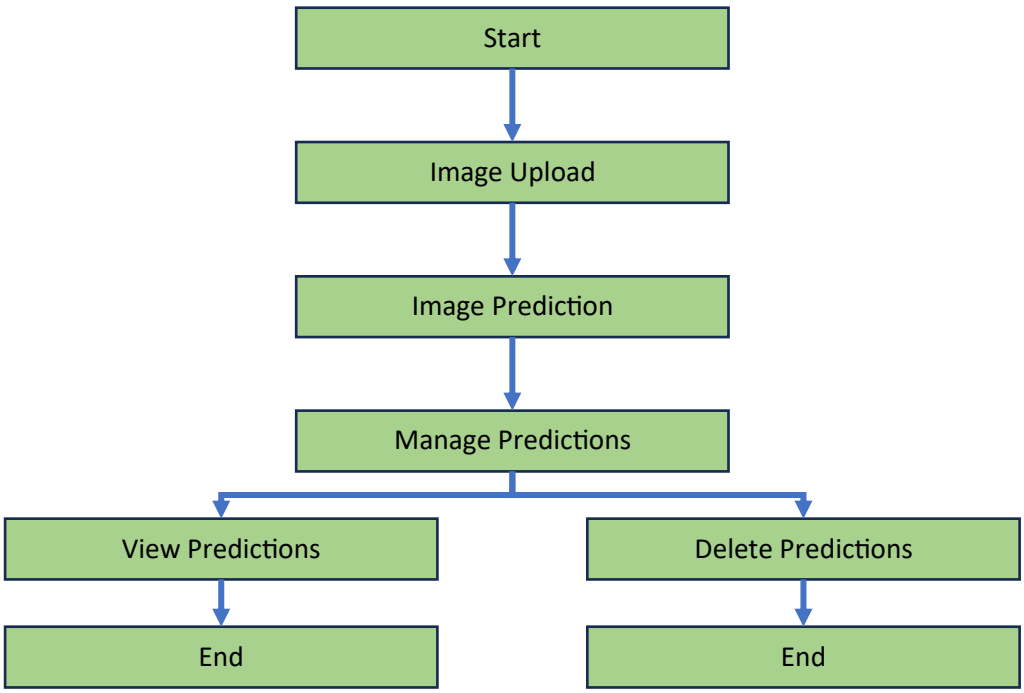


Figure 6: Detailed CFD

Explanation of Control Flow Diagrams

1. **Start**: The starting point of the control flow.
2. **Image Upload**: The process where the user uploads an image to the system.
3. **Image Prediction**: The process where the uploaded image is processed, and a prediction is made using the MobileNetV2 model.
4. **Manage Predictions**: The process where users can view and delete their previous predictions.
5. **View Predictions**: The process where users can view their previous predictions.
6. **Delete Prediction**: The process where users can delete a specific prediction.
7. **End**: The ending point of the control flow.

State Diagram

A state diagram represents the states of an object and the transitions between those states. Below is a state diagram for the Image Prediction System, showing the different states the system can be in and the transitions between those states.

State Diagram

1. **States**:
    - Idle
    - Uploading Image
    - Predicting Image
    - Viewing Predictions
    - Deleting Prediction
2. **Transitions**:
    - Start
    - Upload Image
    - Image Uploaded
    - Predict Image
    - Prediction Completed
    - View Predictions
    - Delete Prediction
    - End

State Diagram Representation

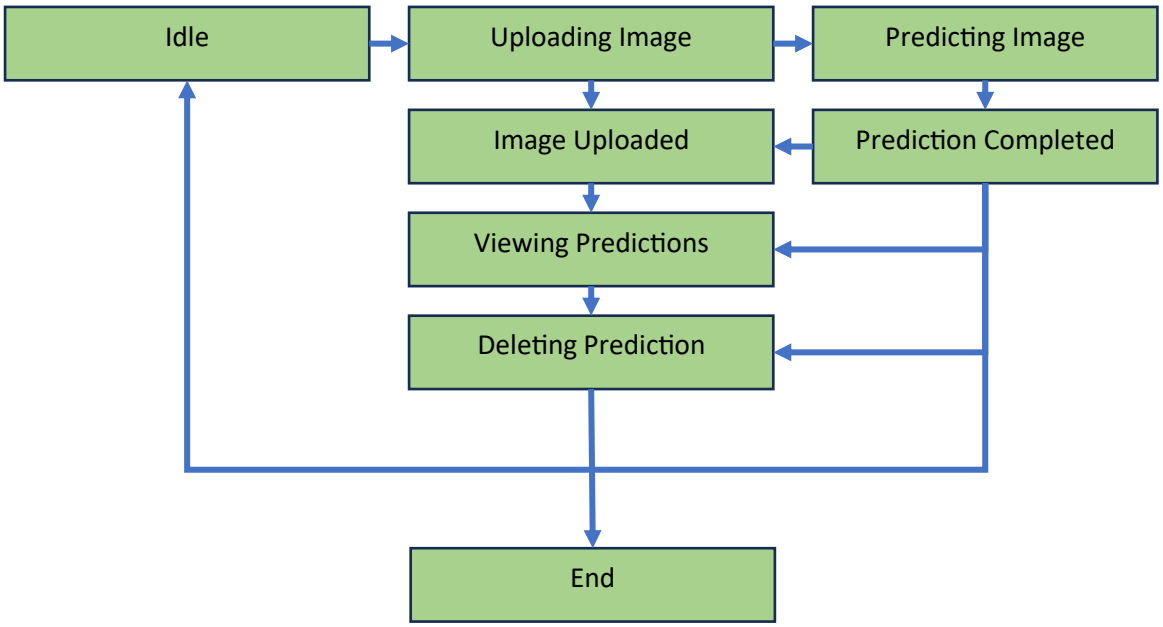


Figure 7: State Diagram Representation

1. **Idle**: The initial state where the system is waiting for user interaction.
2. **Uploading Image**: The state where the user is uploading an image to the system.
3. **Predicting Image**: The state where the system is processing the uploaded image and making a prediction.
4. **Viewing Predictions**: The state where the user is viewing their previous predictions.
5. **Deleting Prediction**: The state where the user is deleting a specific prediction.
6. **End**: The final state where the system completes the current operation.

Entity Relationship Diagram (ERD)

An Entity Relationship Diagram (ERD) is used to model the data and relationships between entities in a system. Below is an ERD for the Image Prediction System, showing the main entities and their relationships.

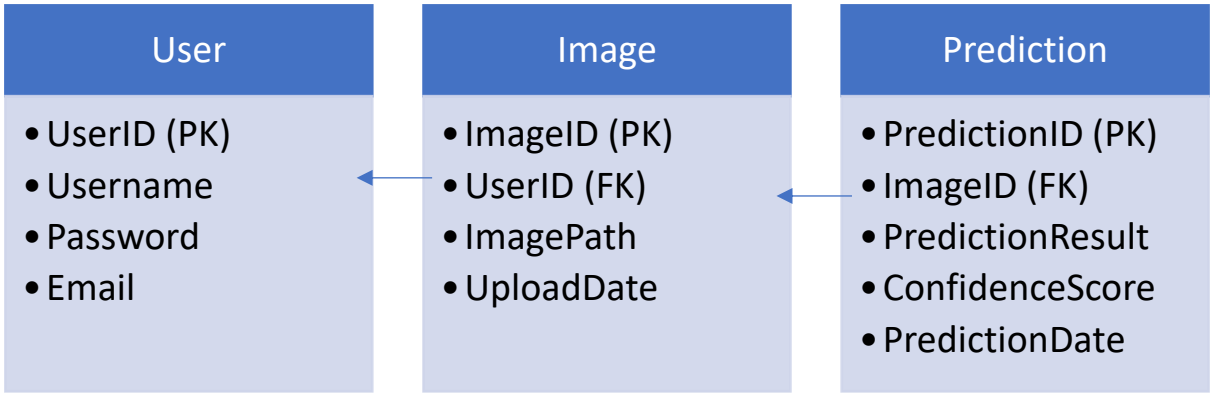
Entities

1. **User**
    - Attributes: UserID (PK), Username, Password, Email

2. **Image**
    - Attributes: ImageID (PK), UserID (FK), ImagePath, UploadDate
3. **Prediction**
    - Attributes: PredictionID (PK), ImageID (FK), PredictionResult, ConfidenceScore, PredictionDate

Relationships

- A User can upload multiple Images.
- Each Image can have one Prediction.



1. **User**: Represents the users of the system.
    - **UserID**: Primary key, unique identifier for each user.
    - **Username**: The username of the user.
    - **Password**: The password of the user.
    - **Email**: The email address of the user.
2. **Image**: Represents the images uploaded by users.
    - **ImageID**: Primary key, unique identifier for each image.
    - **UserID**: Foreign key, references the User entity.
    - **ImagePath**: The path where the image is stored.
    - **UploadDate**: The date when the image was uploaded.

3. **Prediction**: Represents the predictions made for each image.
    - **PredictionID**: Primary key, unique identifier for each prediction.
    - **ImageID**: Foreign key, references the Image entity.
    - **PredictionResult**: The result of the prediction.
    - **ConfidenceScore**: The confidence score of the prediction.
    - **PredictionDate**: The date when the prediction was made.

Use Case Diagram:

A Use Case Diagram represents the interactions between users (actors) and the system, showing the different use cases and how they relate to each other. Below is a Use Case Diagram for the Image Prediction System.

Use-case Diagram for Image Predictor Application

Actors

1. **User**: The primary actor who interacts with the system.

Use Cases

1. **Upload Image**: The user uploads an image to the system.
2. **Predict Image**: The system processes the uploaded image and makes a prediction.
3. **View Predictions**: The user views their previous predictions.
4. **Delete Prediction**: The user deletes a specific prediction.

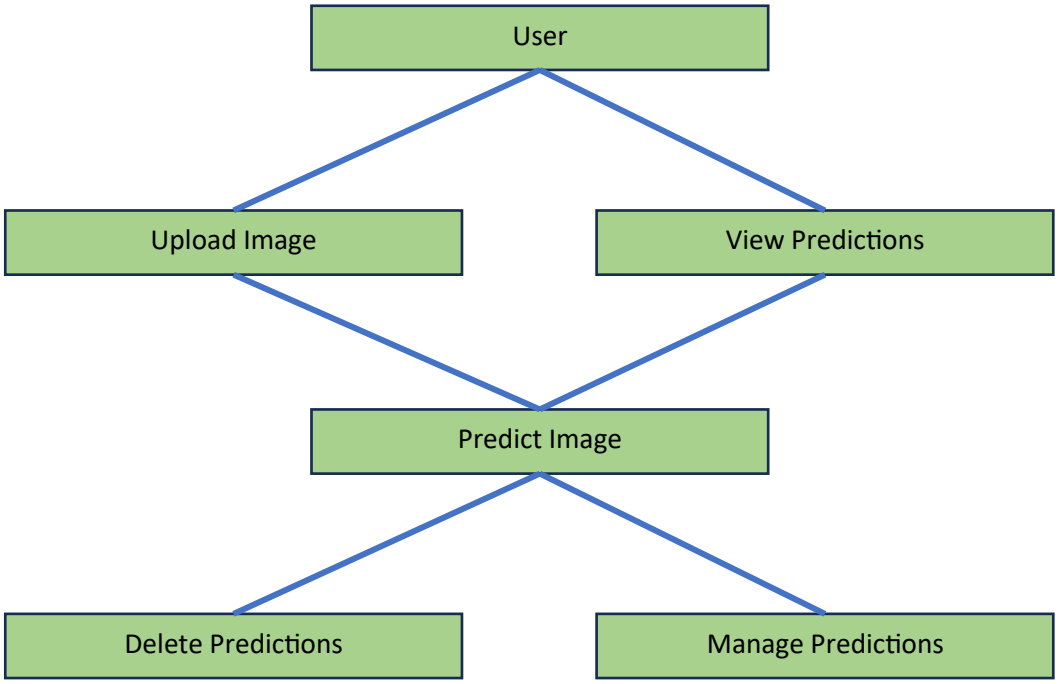


Figure 8: Use-case Diagram.

Explanation of Use-case Diagram

1. **User**: The primary actor who interacts with the system.
2. **Upload Image**: The use case where the user uploads an image to the system.
3. **Predict Image**: The use case where the system processes the uploaded image and makes a prediction.
4. **View Predictions**: The use case where the user views their previous predictions.
5. **Delete Prediction**: The use case where the user deletes a specific prediction.

Detailed Use-case Descriptions

Use Case 1: Upload Image

Use Case ID: UC1

Use Case Name: Upload Image

Actor: User

Description: This use case describes the process of a user uploading an image to the system.

Preconditions:

- The user must be logged into the system.

Postconditions:

- The image is successfully uploaded and stored in the system.
- The system is ready to process the uploaded image for prediction.

Main Flow:

1. The user navigates to the image upload page.
2. The user selects an image file from their device.
3. The user clicks the "Upload" button.
4. The system validates the image file format.
5. The system stores the image in the database.
6. The system confirms the successful upload to the user.

Alternate Flows:

- **Invalid Image Format**:
    1. The system detects an invalid image format.
    2. The system displays an error message to the user.
    3. The user selects a valid image file and retries the upload.

Use Case 2: Predict Image

Use Case ID: UC2

Use Case Name: Predict Image

Actor: User

Description: This use case describes the process of the system predicting the content of an uploaded image.

Preconditions:

- An image must be successfully uploaded to the system.

Postconditions:

- The system generates a prediction for the uploaded image.
- The prediction result is stored in the database.

Main Flow:

1. The system retrieves the uploaded image from the database.

2. The system processes the image using the MobileNetV2 model.
3. The system generates a prediction result and confidence score.
4. The system stores the prediction result in the database.
5. The system displays the prediction result to the user.

Alternate Flows:

- **Prediction Error**:
    1. The system encounters an error during image processing.
    2. The system logs the error.
    3. The system displays an error message to the user.

Use Case 3: View Predictions

Use Case ID: UC3

Use Case Name: View Predictions

Actor: User

Description: This use case describes the process of a user viewing their previous image predictions.

Preconditions:

- The user must be logged into the system.
- The user must have previously uploaded images and generated predictions.

Postconditions:

- The user views a list of their previous predictions.

Main Flow:

1. The user navigates to the predictions page.
2. The system retrieves the user's prediction history from the database.
3. The system displays the list of predictions to the user.
4. The user selects a prediction to view details.
5. The system displays the detailed prediction result to the user.

Alternate Flows:

- **No Predictions**:
    1. The system detects that the user has no previous predictions.

2. The system displays a message indicating no predictions are available.

Use Case 4: Delete Prediction

Use Case ID: UC4

Use Case Name: Delete Prediction

Actor: User

Description: This use case describes the process of a user deleting a specific prediction.

Preconditions:

- The user must be logged into the system.
- The user must have previously generated predictions.

Postconditions:

- The selected prediction is deleted from the database.

Main Flow:

1. The user navigates to the predictions page.
2. The user selects a prediction to delete.
3. The user clicks the "Delete" button.
4. The system prompts the user to confirm the deletion.
5. The user confirms the deletion.
6. The system deletes the prediction from the database.
7. The system confirms the successful deletion to the user.

Alternate Flows:

- **Deletion Cancelled**:
    1. The user cancels the deletion confirmation.
    2. The system retains the prediction in the database.

System Design

Configuration Management

Repository Structure

The repository is organized into directories and files to maintain a clean and manageable structure. Below is an example of the repository structure:

```
image-prediction-system/
├── README.md
├── requirements.txt
├── .gitignore
├── app.py
├── config.py
├── models/
│   └── mobilenet_v2
├── static/
│   ├── css/
│   │   └── styles.css
│   ├── js/
│   │   └── scripts.js
│   └── images/
│       └── placeholder.png
├── templates/
│   ├── base.html
│   ├── upload.html
│   ├── predictions.html
│   └── error.html
├── tests/
│   ├── test_app.py
│   ├── test_model.py
│   └── test_database.py
├── migrations/
│   └── versions/
│       └── <timestamp>_initial_migration.py
├── database/
│   └── predictions.db
└── docs/
    ├── project_report.md
    ├── DFD.png
    ├── ERD.png
    ├── use_case_diagram.png
    └── state_diagram.png
```

Repository Files and Directories

1. **README.md**: Contains the project overview, setup instructions, and usage information.
2. **requirements.txt**: Lists the Python dependencies required for the project.
3. **.gitignore**: Specifies files and directories to be ignored by Git.

4. **app.py**: The main application file containing the Flask app and routes.
5. **config.py**: Configuration settings for the application.
6. **models/**: Directory for storing machine learning models.
    - **mobilenet_v2**: MobileNetV2 model.
7. **sstatic/**: Directory for static files (CSS, JavaScript, images).
    - **css/**: Subdirectory for CSS files.
        - **styles.css**: Main stylesheet for the application.
    - **js/**: Subdirectory for JavaScript files.
        - **scripts.js**: Main JavaScript file for the application.
    - **images/**: Subdirectory for image files.
        - **placeholder.png**: Placeholder image.
8. **templates/**: Directory for HTML templates.
    - **base.html**: Base template for the application.
    - **upload.html**: Template for the image upload page.
    - **predictions.html**: Template for the predictions page.
    - **error.html**: Template for error pages.
9. **tests/**: Directory for unit tests.
    - **test_app.py**: Tests for the Flask app.
    - **test_model.py**: Tests for the machine learning model.
    - **test_database.py**: Tests for database interactions.
10. **migrations/**: Directory for database migration files.
    - **versions/**: Subdirectory for versioned migration scripts.
        - **<timestamp>_initial_migration.py**: Initial database migration script.
11. **database/**: Directory for the database file.
    - **predictions.db**: SQLite database file for storing predictions.
12. **docs/**: Directory for project documentation.
    - **project_report.md**: The project report document.

- **DFD.png**: Data Flow Diagram image.
- **ERD.png**: Entity Relationship Diagram image.
- **use_case_diagram.png**: Use Case Diagram image.
- **state_diagram.png**: State Diagram image.

Version Control Practices

- **Branching Strategy**: Use a branching strategy such as GitFlow to manage feature development, bug fixes, and releases.
  - **main**: The main branch containing the stable version of the application.
  - **develop**: The development branch where new features and bug fixes are integrated.
  - **feature/**: Branches for developing new features.
  - **bugfix/**: Branches for fixing bugs.
  - **release/**: Branches for preparing releases.
- **Commit Messages**: Use clear and descriptive commit messages to document changes.
  - Example: Add user authentication feature, Fix bug in image upload, Update README with installation instructions.
- **Pull Requests**: Use pull requests to review and merge changes into the main or develop branches.
  - Ensure that all tests pass before merging a pull request.
  - Conduct code reviews to maintain code quality and consistency.

Modularisation Details of the Image Predictor Application

The Image Prediction System is divided into several modules to ensure a clean, maintainable, and scalable architecture. Each module is responsible for a specific functionality within the system. Below are the details of each module:

Application Module

Description: This module contains the main application logic and routes for the Flask web application.

Components:

- app.py: The main application file that initializes the Flask app, sets up routes, and handles requests.

Responsibilities:

- Initialize the Flask application.
- Define routes for image upload, prediction, viewing predictions, and deleting predictions.
- Handle HTTP requests and responses.

Configuration Module

Description: This module contains configuration settings for the application.

Components:

- config.py: Configuration file for setting up application-specific settings such as database URI, secret keys, and other configurations.

Responsibilities:

- Store configuration settings for the application.
- Provide a centralized location for managing configuration changes.

Model Module

Description: This module handles the machine learning model used for image prediction.

Components:

- models/mobilenet_v2/model.h5: The pre-trained MobileNetV2 model file.

Responsibilities:

- Load the pre-trained MobileNetV2 model.
- Perform image preprocessing and prediction.
- Return prediction results and confidence scores.

Static Files Module

Description: This module contains static files such as CSS, JavaScript, and images.

Components:

- static/css/styles.css: Main stylesheet for the application.
- static/js/scripts.js: Main JavaScript file for the application.
- static/images/placeholder.png: Placeholder image.

Responsibilities:

- Provide styling for the web application.
- Handle client-side scripting and interactivity.
- Store static images used in the application.

Templates Module

Description: This module contains HTML templates for rendering web pages.

Components:

- templates/base.html: Base template for the application.
- templates/upload.html: Template for the image upload page.
- templates/predictions.html: Template for the predictions page.
- templates/error.html: Template for error pages.

Responsibilities:

- Define the structure and layout of web pages.
- Render dynamic content using Jinja2 templating engine.

Database Module

Description: This module handles database interactions and data storage.

Components:

- database/predictions.db: SQLite database file for storing predictions.
- migrations/versions/<timestamp>_initial_migration.py: Initial database migration script.

Responsibilities:

- Store and manage prediction data.
- Handle database migrations and schema changes.
- Provide a persistent storage solution for the application.

Testing Module

Description: This module contains unit tests for the application.

Components:

- tests/test_app.py: Tests for the Flask app.
- tests/test_model.py: Tests for the machine learning model.
- tests/test_database.py: Tests for database interactions.

Responsibilities:

- Ensure the correctness of the application logic.
- Validate the functionality of the machine learning model.

- Test database interactions and data integrity.

Documentation Module

Description: This module contains project documentation.

Components:

- docs/project_report.md: The project report document.
- docs/DFD.png: Data Flow Diagram image.
- docs/ERD.png: Entity Relationship Diagram image.
- docs/use_case_diagram.png: Use Case Diagram image.
- docs/state_diagram.png: State Diagram image.

Responsibilities:

- Provide detailed documentation of the project.
- Include diagrams and reports for better understanding and communication.

Data Integrity and Constraints of the Image Predictor Application

Ensuring data integrity and applying appropriate constraints are crucial for maintaining the accuracy, consistency, and reliability of the data in the Image Prediction System. Below are the details of the data integrity measures and constraints applied to the system.

Data Integrity

1. **Entity Integrity**:
    - Each table has a primary key that uniquely identifies each record.
    - Primary keys ensure that each record is unique and can be referenced reliably.
2. **Referential Integrity**:
    - Foreign keys are used to maintain relationships between tables.
    - Foreign key constraints ensure that references between tables are valid and consistent.
3. **Domain Integrity**:
    - Data types and constraints are applied to ensure that data values fall within acceptable ranges.
    - Validation checks are implemented to enforce data formats and value ranges.
4. **User-Defined Integrity**:

- Custom rules and constraints are applied to enforce business logic and application-specific requirements.

Constraints

User Table

```
CREATE TABLE User (
   UserID INTEGER PRIMARY KEY AUTOINCREMENT,
   Username TEXT NOT NULL UNIQUE,
   Password TEXT NOT NULL,
   Email TEXT NOT NULL UNIQUE
);
```

Constraints:

- UserID: Primary key, auto-incremented to ensure unique identification of each user.
- Username: Must be unique and not null to ensure each user has a distinct username.
- Password: Must not be null to ensure each user has a password.
- Email: Must be unique and not null to ensure each user has a distinct email address.

Image Table

```
CREATE TABLE Image (
   ImageID INTEGER PRIMARY KEY AUTOINCREMENT,
   UserID INTEGER NOT NULL,
   ImagePath TEXT NOT NULL,
   UploadDate DATETIME DEFAULT CURRENT_TIMESTAMP,
   FOREIGN KEY (UserID) REFERENCES User(UserID) ON DELETE CASCADE
);
```

Constraints:

- ImageID: Primary key, auto-incremented to ensure unique identification of each image.
- UserID: Foreign key referencing User(UserID), must not be null to ensure each image is associated with a valid user.
- ImagePath: Must not be null to ensure the image file path is stored.
- UploadDate: Defaults to the current timestamp to record the upload date and time.
- FOREIGN KEY (UserID) REFERENCES User(UserID) ON DELETE CASCADE: Ensures referential integrity by deleting images if the associated user is deleted.

Prediction Table

```
CREATE TABLE Prediction (
    PredictionID INTEGER PRIMARY KEY AUTOINCREMENT,
    ImageID INTEGER NOT NULL,
    PredictionResult TEXT NOT NULL,
    ConfidenceScore REAL NOT NULL CHECK (ConfidenceScore >= 0.0 AND ConfidenceScore <= 1.0),
    PredictionDate DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (ImageID) REFERENCES Image(ImageID) ON DELETE CASCADE
);
```

Constraints:

- PredictionID: Primary key, auto-incremented to ensure unique identification of each prediction.
- ImageID: Foreign key referencing Image(ImageID), must not be null to ensure each prediction is associated with a valid image.
- PredictionResult: Must not be null to ensure the prediction result is stored.
- ConfidenceScore: Must not be null and must be between 0.0 and 1.0 to ensure the confidence score is valid.
- PredictionDate: Defaults to the current timestamp to record the prediction date and time.
- FOREIGN KEY (ImageID) REFERENCES Image(ImageID) ON DELETE CASCADE: Ensures referential integrity by deleting predictions if the associated image is deleted.

Database Design of the Image Predictor System

The database design for the Image Prediction System includes three main tables: User, Image, and Prediction. These tables are designed to store user information, uploaded images, and prediction results, respectively. Below is a detailed description of each table, including their attributes and relationships.

Tables

1. User Table
2. Image Table
3. Prediction Table

User Table

Description: Stores information about the users of the system.

Attributes:

- UserID (INTEGER, Primary Key, Auto-increment): Unique identifier for each user.
- Username (TEXT, NOT NULL, UNIQUE): The username of the user.

- Password (TEXT, NOT NULL): The password of the user.
- Email (TEXT, NOT NULL, UNIQUE): The email address of the user.

SQL Schema:

```
CREATE TABLE User (
   UserID INTEGER PRIMARY KEY AUTOINCREMENT,
   Username TEXT NOT NULL UNIQUE,
   Password TEXT NOT NULL,
   Email TEXT NOT NULL UNIQUE
);
```

Image Table

Description: Stores information about the images uploaded by users.

Attributes:

- ImageID (INTEGER, Primary Key, Auto-increment): Unique identifier for each image.
- UserID (INTEGER, Foreign Key, NOT NULL): References the UserID in the User table.
- ImagePath (TEXT, NOT NULL): The path where the image is stored.
- UploadDate (DATETIME, DEFAULT CURRENT_TIMESTAMP): The date and time when the image was uploaded.

SQL Schema:

```
CREATE TABLE Image (
   ImageID INTEGER PRIMARY KEY AUTOINCREMENT,
   UserID INTEGER NOT NULL,
   ImagePath TEXT NOT NULL,
   UploadDate DATETIME DEFAULT CURRENT_TIMESTAMP,
   FOREIGN KEY (UserID) REFERENCES User(UserID) ON DELETE CASCADE
);
```

Prediction Table

Description: Stores information about the predictions made for each image.

Attributes:

- PredictionID (INTEGER, Primary Key, Auto-increment): Unique identifier for each prediction.
- ImageID (INTEGER, Foreign Key, NOT NULL): References the ImageID in the Image table.
- PredictionResult (TEXT, NOT NULL): The result of the prediction.

- ConfidenceScore (REAL, NOT NULL, CHECK (ConfidenceScore >= 0.0 AND ConfidenceScore <= 1.0)): The confidence score of the prediction.
- PredictionDate (DATETIME, DEFAULT CURRENT_TIMESTAMP): The date and time when the prediction was made.

SQL Schema:

```
CREATE TABLE Prediction (
   PredictionID INTEGER PRIMARY KEY AUTOINCREMENT,
   ImageID INTEGER NOT NULL,
   PredictionResult TEXT NOT NULL,
   ConfidenceScore REAL NOT NULL CHECK (ConfidenceScore >= 0.0 AND ConfidenceScore <= 1.0),
   PredictionDate DATETIME DEFAULT CURRENT_TIMESTAMP,
   FOREIGN KEY (ImageID) REFERENCES Image(ImageID) ON DELETE CASCADE
);
```

Relationships

- **User to Image:** One-to-Many relationship. A user can upload multiple images, but each image is associated with only one user.
- **Image to Prediction:** One-to-One relationship. Each image can have one prediction, and each prediction is associated with only one image.

Procedural Design for the Image Predictor System

The procedural design for the Image Prediction System outlines the sequence of operations and the flow of control within the system. This design ensures that each component of the system interacts correctly and efficiently to achieve the desired functionality. Below are the detailed procedures for the main operations: image upload, image prediction, viewing predictions, and deleting predictions.

Procedures

1. Image Upload Procedure
2. Image Prediction Procedure
3. View Predictions Procedure
4. Delete Prediction Procedure

Image Upload Procedure

Description: This procedure handles the process of uploading an image to the system.

Steps:

1. **Start**: The user initiates the image upload process by navigating to the upload page.

2. **Select Image**: The user selects an image file from their device.
3. **Upload Image**: The user clicks the "Upload" button.
4. **Validate Image**:
    - Check if the selected file is a valid image format (e.g., JPEG, PNG).
    - If the file is invalid, display an error message and return to step 2.
5. **Store Image**:
    - Save the image file to the server.
    - Insert a new record into the Image table with the UserID, ImagePath, and UploadDate.
6. **Confirm Upload**: Display a confirmation message to the user indicating that the image has been successfully uploaded.
7. **End**: The image upload process is complete.

Image Prediction Procedure

Description: This procedure handles the process of predicting the content of an uploaded image.

Steps:

1. **Start**: The system initiates the image prediction process after an image is successfully uploaded.
2. **Retrieve Image**: Retrieve the image file from the server using the ImagePath.
3. **Preprocess Image**: Preprocess the image to the required format for the MobileNetV2 model.
4. **Load Model**: Load the pre-trained MobileNetV2 model.
5. **Predict Image**: Use the model to predict the content of the image.
6. **Store Prediction**:
    - Insert a new record into the Prediction table with the ImageID, PredictionResult, ConfidenceScore, and PredictionDate.
7. **Display Result**: Display the prediction result and confidence score to the user.
8. **End**: The image prediction process is complete.

View Predictions Procedure

Description: This procedure handles the process of viewing previous predictions made by the user.

Steps:

1. **Start**: The user initiates the view predictions process by navigating to the predictions page.

2. **Retrieve Predictions**:
    - Query the Prediction table to retrieve all predictions associated with the user's UserID.
3. **Display Predictions**: Display the list of predictions to the user, including the prediction result, confidence score, and prediction date.
4. **Select Prediction**: The user selects a specific prediction to view details.
5. **Display Details**: Display the detailed prediction result and associated image to the user.
6. **End**: The view predictions process is complete.

Delete Prediction Procedure

Description: This procedure handles the process of deleting a specific prediction.

Steps:

1. **Start**: The user initiates the delete prediction process by selecting a prediction to delete.
2. **Confirm Deletion**: Prompt the user to confirm the deletion.
3. **Delete Prediction**:
    - If the user confirms, delete the record from the Prediction table.
    - Optionally, delete the associated image file from the server.
4. **Confirm Deletion**: Display a confirmation message to the user indicating that the prediction has been successfully deleted.
5. **End**: The delete prediction process is complete.

Flowcharts

Image Upload Procedure

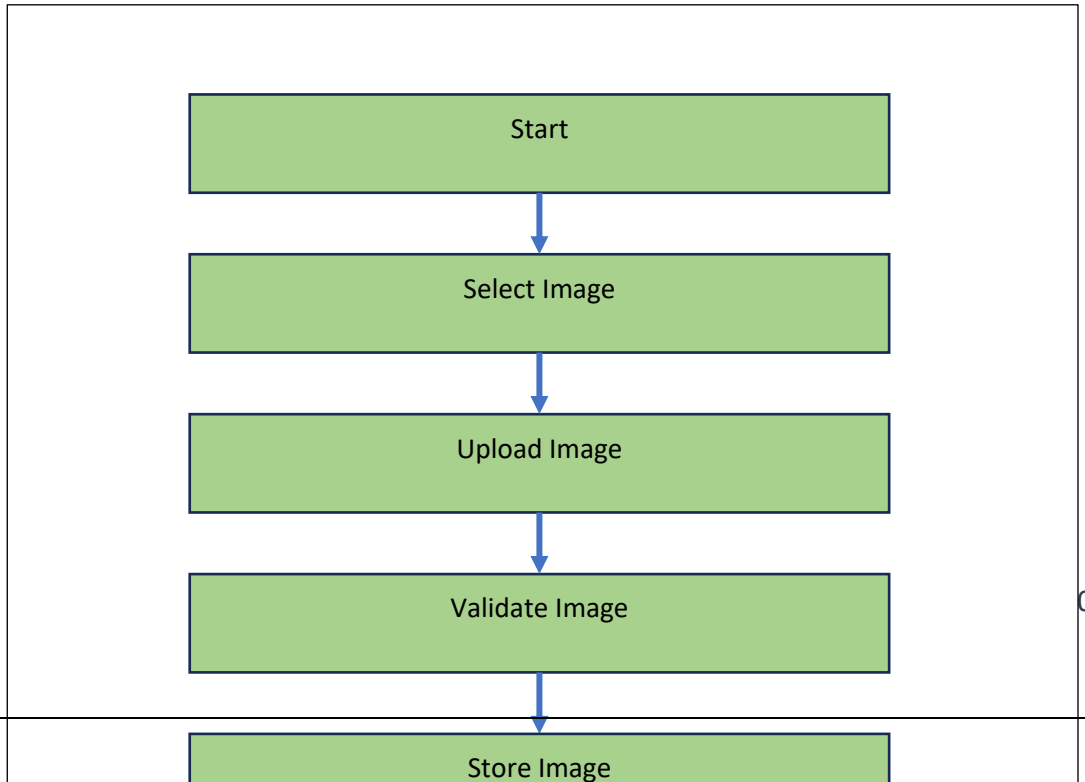
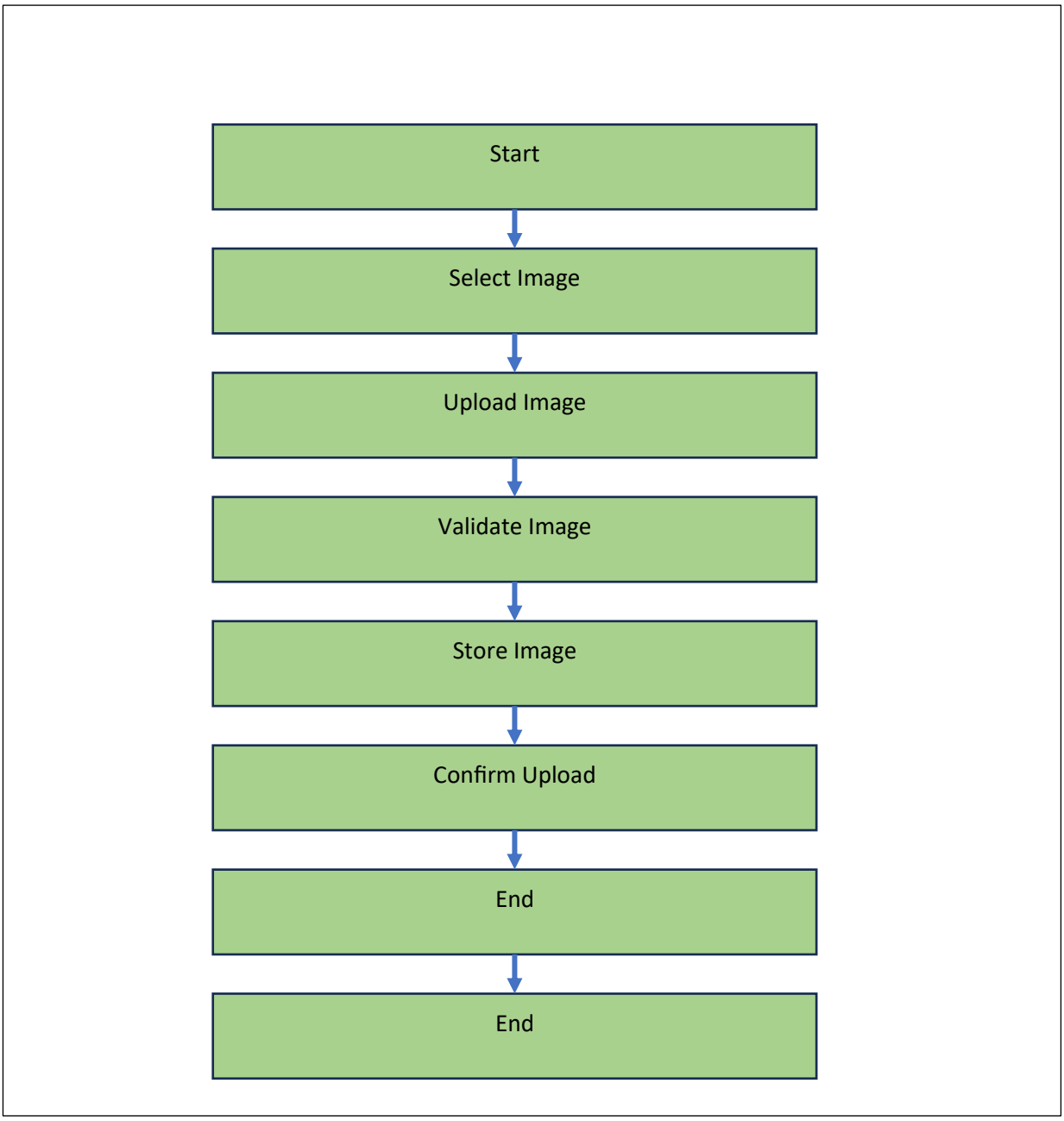
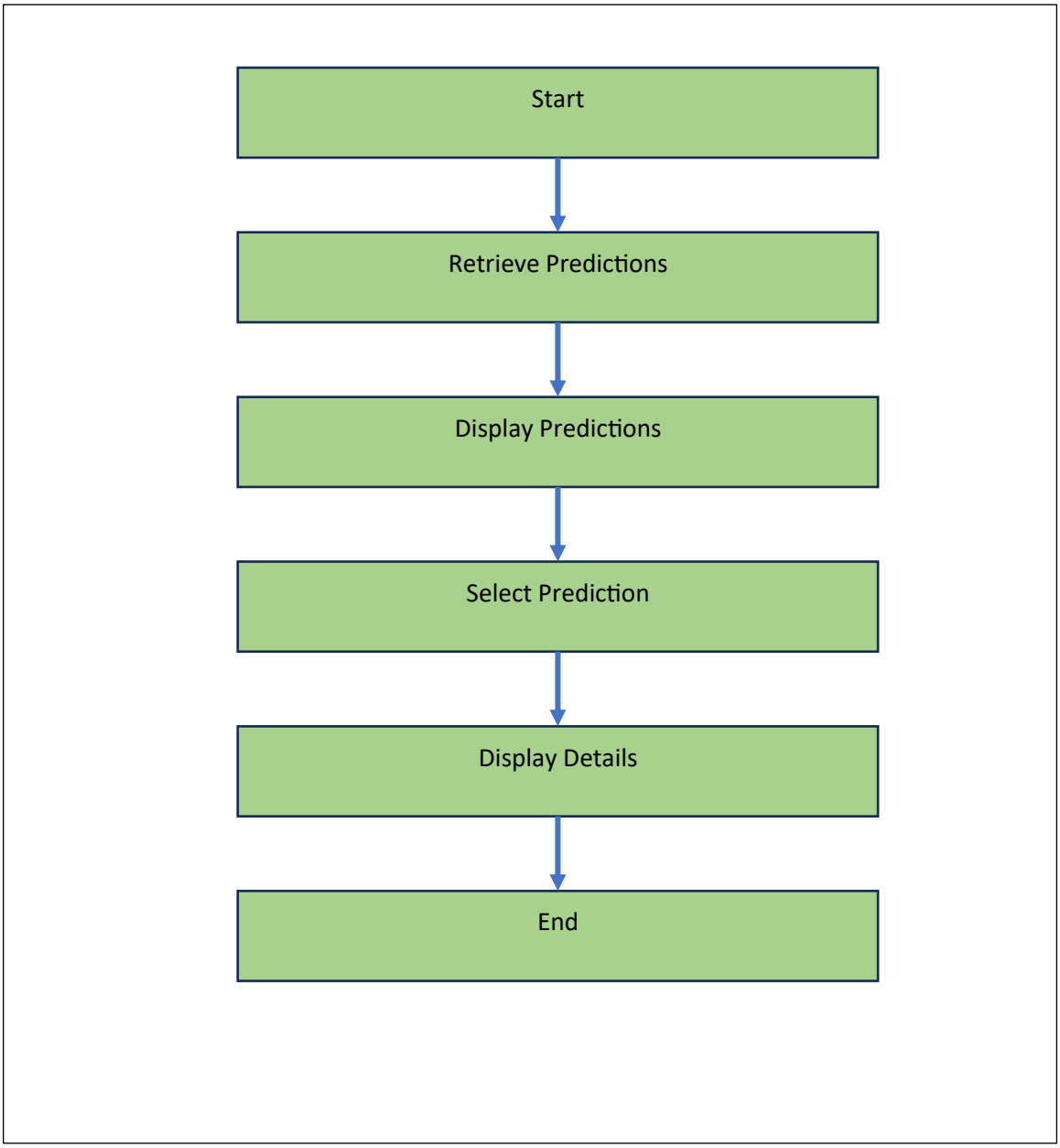


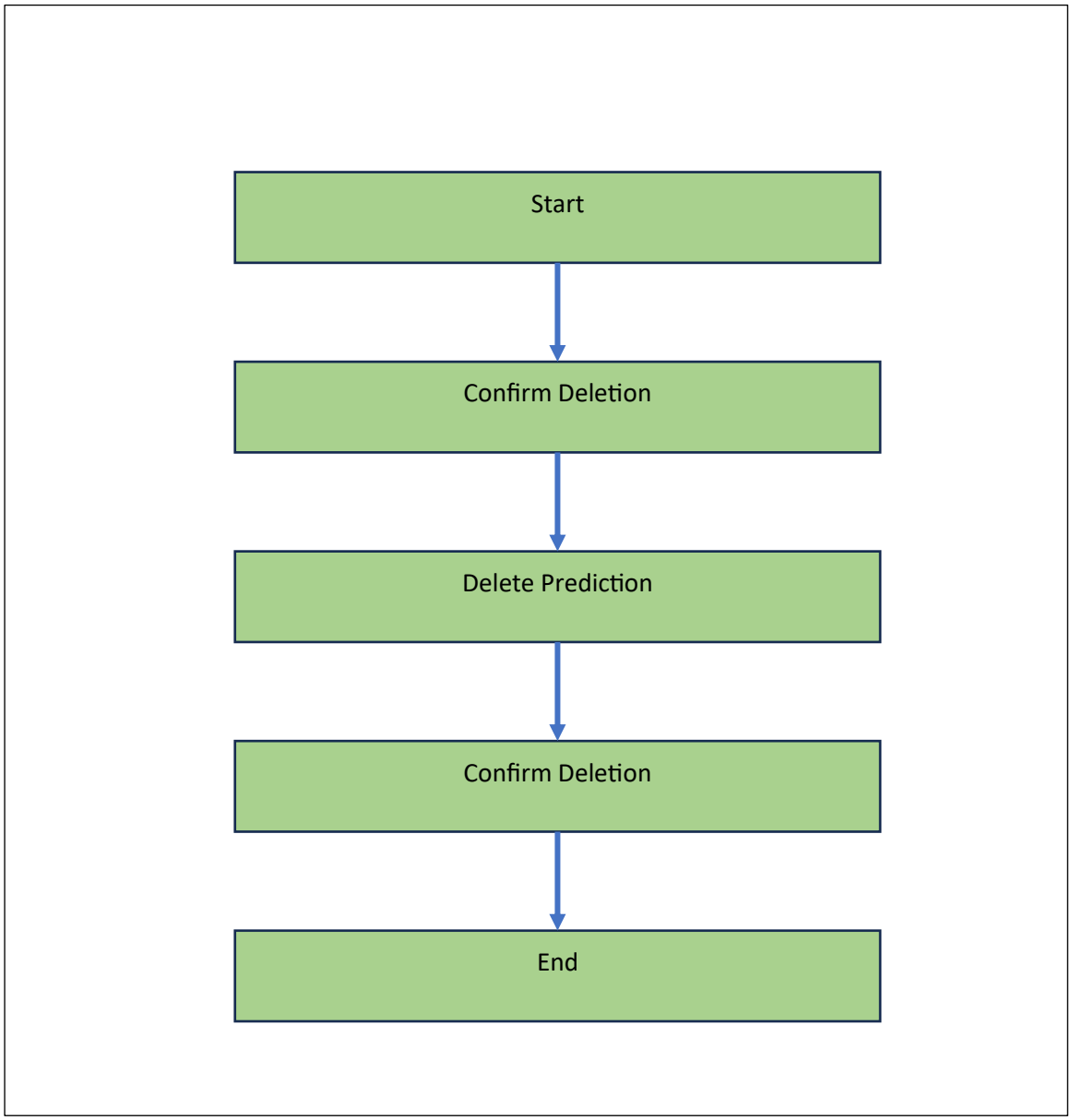
Image Prediction Procedure



View Predictions Procedure



Delete Prediction Procedure



User Interface Design for the Image Predictor Application

The User Interface (UI) design for the Image Prediction System includes wireframes for the key pages: the home page, image upload page, predictions page, and error page. Wireframes provide a visual representation of the layout and components of each page.

Home Page

Description: The home page serves as the entry point to the application, providing an overview and navigation options.

Wireframe:

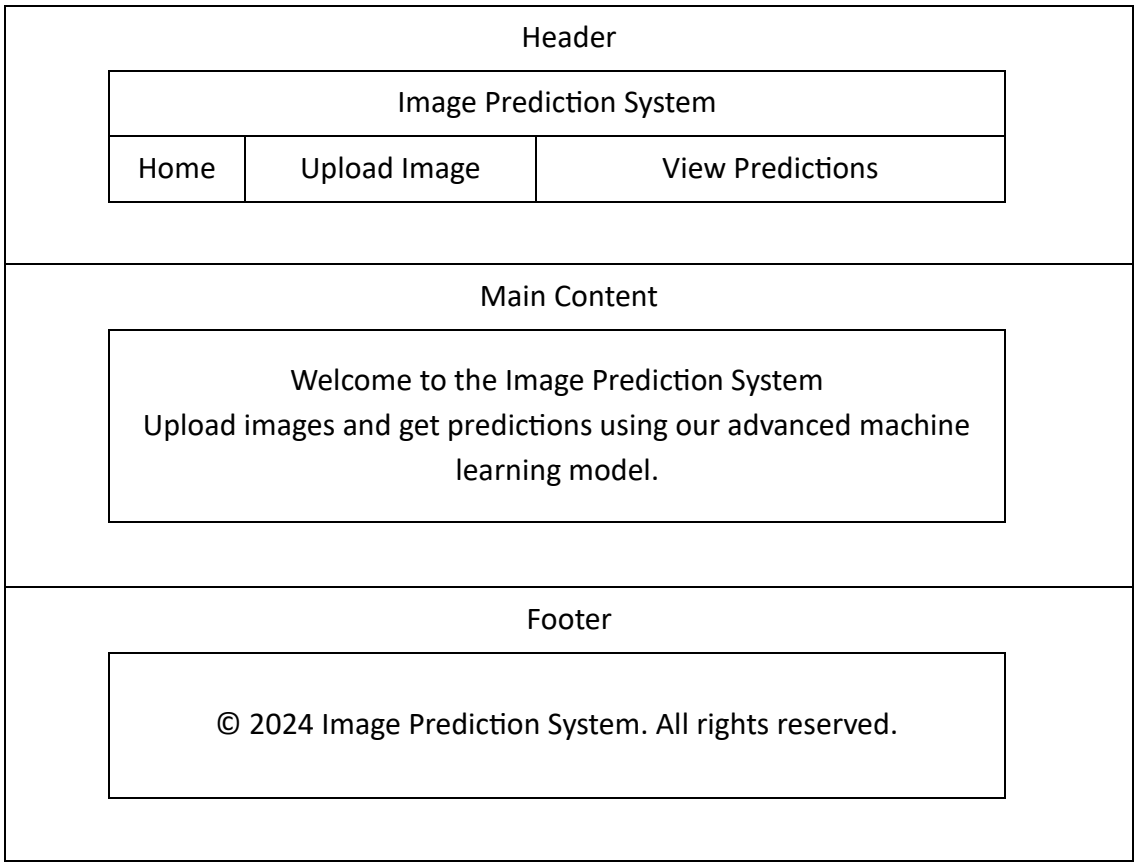
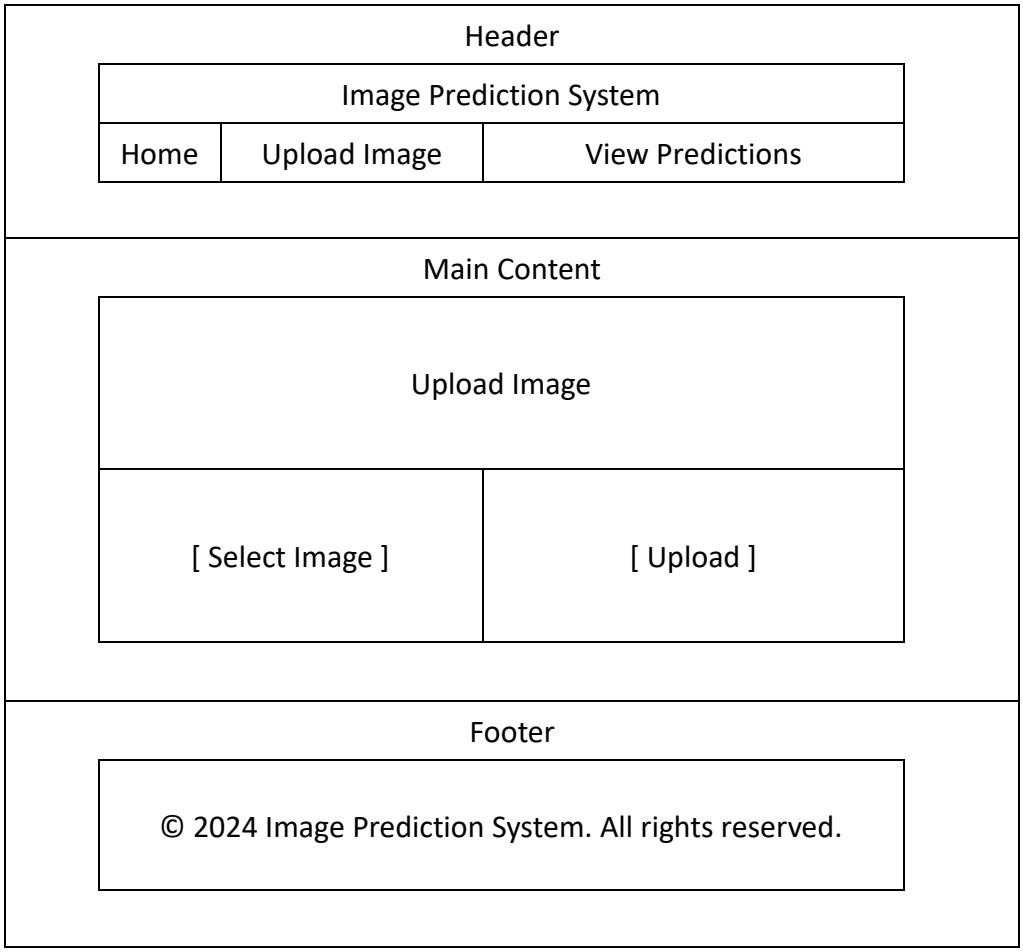


Image Upload Page

Description: The image upload page allows users to upload images for prediction.

Wireframe:



Predictions Page

Description: The predictions page allows users to view their previous predictions.

Wireframe:

Header

Image Prediction System		
Home	Upload Image	View Predictions

Main Content

View Predictions

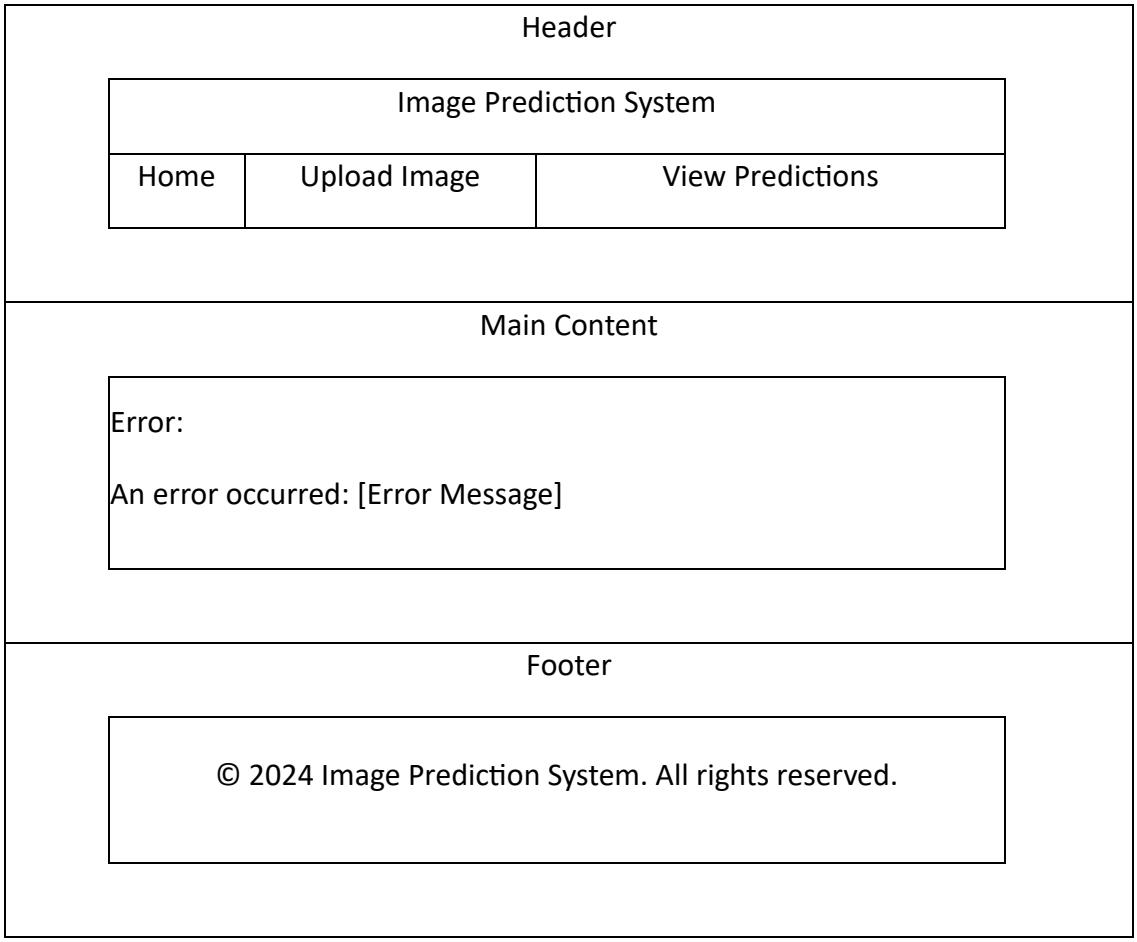
Prediction: Cat
Confidence: 0.95
Date: 2024-11-01
[View Details] [Delete]

Footer

Error Page

Description: The error page displays error messages to the user.

Wireframe:



Unit Testing

Unit tests are essential for ensuring the correctness and reliability of the individual components of the Image Prediction System. Below are the unit test cases for the main functionalities: image upload, image prediction, viewing predictions, and deleting predictions.

Test Setup

Before writing the test cases, ensure you have the necessary testing framework installed. For Python, unittest is a built-in module that can be used for writing and running tests.

Test Cases

1. Test Image Upload
2. Test Image Prediction
3. Test View Predictions
4. Test Delete Prediction

Test Image Upload

Test Case ID: TC1

Description: Test the image upload functionality.

Test Steps:

1. Navigate to the image upload page.
2. Select a valid image file.
3. Click the "Upload" button.

Expected Result:

- The image is successfully uploaded and stored in the database.
- The user receives a confirmation message.

Actual Result: (To be filled after testing)

Status: (Pass/Fail)

Test Image Prediction

Test Case ID: TC2

Description: Test the image prediction functionality.

Test Steps:

1. Upload a valid image.
2. Trigger the prediction process.

Expected Result:

- The system generates a prediction for the uploaded image.
- The prediction result and confidence score are stored in the database.

Actual Result: (To be filled after testing)

Status: (Pass/Fail)

Test View Predictions

Test Case ID: TC3

Description: Test the view predictions functionality.

Test Steps:

1. Navigate to the predictions page.
2. Retrieve the list of predictions.

Expected Result:

- The system displays a list of previous predictions for the user.

Actual Result: (To be filled after testing)

Status: (Pass/Fail)

Test Delete Prediction

Test Case ID: TC4

Description: Test the delete prediction functionality.

Test Steps:

1. Navigate to the predictions page.
2. Select a prediction to delete.
3. Confirm the deletion.

Expected Result:

- The selected prediction is deleted from the database.
- The user receives a confirmation message.

Actual Result: (To be filled after testing)

Status: (Pass/Fail)

System Testing

System tests are designed to validate the end-to-end functionality of the Image Prediction System, ensuring that all components work together as expected. Below are the system test cases for the main functionalities: image upload, image prediction, viewing predictions, and deleting predictions.

Test Cases

1. Test Image Upload and Prediction
2. Test View Predictions
3. Test Delete Prediction
4. Test Error Handling

Test Image Upload and Prediction

Test Case ID: ST1

Description: Test the complete process of uploading an image and generating a prediction.

Test Steps:

1. Navigate to the image upload page.
2. Select a valid image file.
3. Click the "Upload" button.
4. Verify that the image is uploaded successfully.
5. Verify that a prediction is generated for the uploaded image.

Expected Result:

- The image is successfully uploaded and stored in the database.
- A prediction result and confidence score are generated and stored in the database.
- The user receives a confirmation message for both the upload and prediction.

Actual Result: (To be filled after testing)

Status: (Pass/Fail)

Test View Predictions

Test Case ID: ST2

Description: Test the functionality of viewing previous predictions.

Test Steps:

1. Navigate to the predictions page.

2. Verify that the list of previous predictions is displayed.
3. Select a prediction to view details.

Expected Result:

- The system displays a list of previous predictions for the user.
- The user can view the details of a selected prediction.

Actual Result: (To be filled after testing)

Status: (Pass/Fail)

Test Delete Prediction

Test Case ID: ST3

Description: Test the functionality of deleting a prediction.

Test Steps:

1. Navigate to the predictions page.
2. Select a prediction to delete.
3. Confirm the deletion.
4. Verify that the prediction is deleted from the database.

Expected Result:

- The selected prediction is deleted from the database.
- The user receives a confirmation message for the deletion.

Actual Result: (To be filled after testing)

Status: (Pass/Fail)

Test Error Handling

Test Case ID: ST4

Description: Test the system's error handling capabilities.

Test Steps:

1. Trigger an error (e.g., upload an invalid file format).
2. Verify that the system displays an appropriate error message.

Expected Result:

- The system displays an appropriate error message to the user.

Actual Result: (To be filled after testing)

Status: (Pass/Fail)

Coding

SQL Commands for the Image Prediction System

Below are the SQL commands for creating the database schema, inserting data into the tables, and managing access rights for different users in the Image Prediction System.

Database Creation

Create User Table:

```
CREATE TABLE User (
    UserID INTEGER PRIMARY KEY AUTOINCREMENT,
    Username TEXT NOT NULL UNIQUE,
    Password TEXT NOT NULL,
    Email TEXT NOT NULL UNIQUE
);
```

Create Image Table:

```
CREATE TABLE Image (
    ImageID INTEGER PRIMARY KEY AUTOINCREMENT,
    UserID INTEGER NOT NULL,
    ImagePath TEXT NOT NULL,
    UploadDate DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (UserID) REFERENCES User(UserID) ON DELETE CASCADE
);
```

Create Prediction Table:

```
CREATE TABLE Prediction (
    PredictionID INTEGER PRIMARY KEY AUTOINCREMENT,
    ImageID INTEGER NOT NULL,
    PredictionResult TEXT NOT NULL,
    ConfidenceScore REAL NOT NULL CHECK (ConfidenceScore >= 0.0 AND ConfidenceScore <= 1.0),
    PredictionDate DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (ImageID) REFERENCES Image(ImageID) ON DELETE CASCADE
);
```

Data Insertion

Insert Data into User Table:

```
INSERT INTO User (Username, Password, Email) VALUES ('testuser', 'password',
'test@example.com');
```

Insert Data into Image Table:

```
INSERT INTO Image (UserID, ImagePath, UploadDate) VALUES (1, 'path/to/image.jpg', '2023-01-01
12:00:00');
```

Insert Data into Prediction Table:

```
INSERT INTO Prediction (ImageID, PredictionResult, ConfidenceScore, PredictionDate) VALUES (1, 'Cat', 0.95, '2023-01-01 12:05:00');
```

Access Rights for Different Users

Create a New User:

```
CREATE USER 'app_user'@'localhost' IDENTIFIED BY 'password';
```

Grant Access Rights:

```
GRANT SELECT, INSERT, UPDATE, DELETE ON image_prediction_system.* TO 'app_user'@'localhost';
```

Revoke Access Rights:

```
REVOKE DELETE ON image_prediction_system.* FROM 'app_user'@'localhost';
```

Show Grants for a User:

```
SHOW GRANTS FOR 'app_user'@'localhost';
```

Standardization of the Coding for the Image Prediction System

Standardizing the coding practices ensures that the codebase is consistent, readable, and maintainable. Below are the guidelines and best practices for standardizing the coding in the Image Prediction System project.

1. Naming Conventions

Variables and Functions:

- Use descriptive names that clearly indicate the purpose of the variable or function.
- Use snake_case for variable and function names.
- Example: image_path, upload_image()

Classes:

- Use CamelCase for class names.
- Example: ImageUploader, PredictionModel

Constants:

- Use UPPER_CASE for constants.
- Example: MAX_IMAGE_SIZE, MODEL_PATH

2. Code Structure

Imports:

- Group imports into three categories: standard library imports, third-party imports, and local imports.
- Separate each group with a blank line.

Functions and Methods:

- Keep functions and methods short and focused on a single task.
- Use docstrings to describe the purpose and behavior of functions and methods.

Error Handling

Exceptions:

- Use exceptions to handle errors and unexpected conditions.
- Catch specific exceptions rather than using a generic except clause.

Logging:

- Use logging to record important events and errors.
- Configure logging to output to both the console and a file.

Code Comments

Inline Comments:

- Use inline comments to explain complex or non-obvious code.
- Keep comments concise and relevant.

Block Comments:

- Use block comments to describe the overall purpose and logic of a code block.

Code Formatting

Indentation:

- Use 4 spaces for indentation.
- Do not use tabs.

Line Length:

- Limit lines to 79 characters.
- Use line continuation for long lines.

Blank Lines:

- Use blank lines to separate logical sections of the code.

Version Control

Commit Messages:

- Use clear and descriptive commit messages.
- Follow the format: <type>(<scope>): <subject>
- Example: feat(upload): add image upload functionality

Branching:

- Use feature branches for new features and bug fixes.
- Merge branches into the main branch using pull requests.

Testing

Testing Techniques and Testing Strategies Used

Testing Techniques

Unit Testing

- **Description**: Unit testing involves testing individual components or functions of the application in isolation to ensure they work as expected.
- **Tools**: unittest, pytest
- **Example**: Testing the image upload function to ensure it correctly saves the image and updates the database.

Integration Testing

- **Description**: Integration testing involves testing the interactions between different components or modules of the application to ensure they work together as expected.
- **Tools**: unittest, pytest
- **Example**: Testing the interaction between the image upload and prediction components to ensure the uploaded image is correctly processed and a prediction is generated.

System Testing

- **Description**: System testing involves testing the complete system as a whole to ensure it meets the specified requirements and functions correctly.
- **Tools**: unittest, pytest
- **Example**: Testing the end-to-end process of uploading an image, generating a prediction, viewing predictions, and deleting predictions.

User Interface Testing

- **Description**: User interface testing involves testing the user interface of the application to ensure it is user-friendly, intuitive, and accessible.
- **Tools**: Selenium
- **Example**: Testing the layout and navigation of the home page, image upload form, and predictions page.

Performance Testing

- **Description**: Performance testing involves testing the system's performance under various conditions to ensure it is responsive and stable.
- **Tools**: JMeter
- **Example**: Testing the system's response time under normal and peak load conditions.

Security Testing

- **Description**: Security testing involves testing the system for potential security vulnerabilities to ensure it is secure and protected against attacks.
- **Tools**: OWASP ZAP, Burp Suite
- **Example**: Testing for SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF) vulnerabilities.

Testing Strategies

Test-Driven Development (TDD)

- **Description**: TDD is a software development approach where tests are written before the code. The code is then developed to pass the tests.
- **Benefits**: Ensures that the code meets the specified requirements and reduces the likelihood of defects.
- **Example**: Writing unit tests for the image upload function before implementing the function.

Behavior-Driven Development (BDD)

- **Description**: BDD is a software development approach that focuses on the behavior of the application from the user's perspective. Tests are written in a natural language format.
- **Benefits**: Improves communication between developers, testers, and stakeholders and ensures the application meets user expectations.
- **Example**: Writing BDD scenarios for the image upload and prediction process using Gherkin syntax.

Continuous Integration (CI)

- **Description**: CI is a software development practice where code changes are automatically tested and integrated into the main codebase frequently.
- **Benefits**: Ensures that code changes do not introduce defects and allows for early detection of issues.
- **Example**: Using a CI tool like Jenkins or GitHub Actions to automatically run tests on each code commit.

Regression Testing

- **Description**: Regression testing involves retesting the application after code changes to ensure that existing functionality is not affected.
- **Benefits**: Ensures that new code changes do not introduce defects or break existing functionality.
- **Example**: Running a suite of automated tests after adding a new feature to the image prediction system.

Exploratory Testing

- **Description**: Exploratory testing involves testing the application without predefined test cases, allowing testers to explore the application and identify potential issues.
- **Benefits**: Identifies unexpected issues and improves the overall quality of the application.
- **Example**: Manually testing the image upload and prediction process to identify edge cases and potential issues.

Test Plan

Introduction

Project Name: Image Prediction System

Objective: The objective of this test plan is to outline the testing strategy, scope, resources, schedule, and deliverables for the Image Prediction System. The goal is to ensure that the system functions correctly, meets the specified requirements, and provides a seamless user experience.

Scope

In Scope:

- Functional testing of image upload, prediction, viewing predictions, and deleting predictions.
- Integration testing to ensure that different components work together as expected.
- System testing to validate the end-to-end functionality of the application.
- User interface testing to ensure that the UI is user-friendly and intuitive.
- Performance testing to evaluate the system's responsiveness and stability under load.
- Security testing to identify and mitigate potential vulnerabilities.

Out of Scope:

- Testing of third-party libraries and frameworks.
- Testing on non-supported browsers or devices.

Test Strategy

Testing Levels:

- Unit Testing: Test individual components and functions.
- Integration Testing: Test interactions between different components.
- System Testing: Test the complete system as a whole.
- User Interface Testing: Test the user interface for usability and accessibility.
- Performance Testing: Test the system's performance under various conditions.

- Security Testing: Test the system for potential security vulnerabilities.

Testing Types:

- Manual Testing: Perform exploratory testing and validate user interface elements.
- Automated Testing: Use automated test scripts for regression testing and performance testing.

Test Environment

Hardware:

- Development and testing machines with sufficient resources to run the application and tests.

Software:

- Operating System: Windows 10 or later, macOS, or Linux.
- Web Browser: Latest versions of Chrome, Firefox, Safari, and Edge.
- Database: SQLite
- Testing Tools: unittest, pytest, Selenium, JMeter

Test Schedule

Activity	Start Date	End Date	Responsible
Test Plan Preparation	2024-10-08	2024-10-10	Test Lead
Test Case Development	2024-10-11	2024-10-15	Test Team
Unit Testing	2024-10-24	2024-11-01	Developers
Integration Testing	2024-11-11	2024-11-13	Test Team
System Testing	2024-11-18	2024-11-19	Test Team
User Interface Testing	2024-11-20	2024-11-22	Test Team
Performance Testing	2024-11-25	2024-11-26	Test Team
Security Testing	2024-11-26	2024-11-26	Security Team
Test Report Preparation	2024-11-26	2024-11-26	Test Lead

Test Deliverables

- Test Plan Document
- Test Cases
- Test Scripts
- Test Data

- Test Execution Reports
- Defect Reports
- Test Summary Report

Test Cases

Unit Test Cases:

- Test image upload functionality.
- Test image prediction functionality.
- Test viewing predictions functionality.
- Test deleting predictions functionality.

Integration Test Cases:

- Test the interaction between image upload and prediction components.
- Test the interaction between prediction and viewing predictions components.

System Test Cases:

- Test the complete process of uploading an image and generating a prediction.
- Test the functionality of viewing previous predictions.
- Test the functionality of deleting a prediction.
- Test the system's error handling capabilities.

User Interface Test Cases:

- Test the layout and navigation of the home page.
- Test the image upload form for usability.
- Test the predictions page for readability and accessibility.
- Test the error page for clarity and helpfulness.

Performance Test Cases:

- Test the system's response time under normal load.
- Test the system's response time under peak load.
- Test the system's stability under sustained load.

Security Test Cases:

- Test for SQL injection vulnerabilities.

- Test for cross-site scripting (XSS) vulnerabilities.
- Test for cross-site request forgery (CSRF) vulnerabilities.
- Test for secure data transmission (HTTPS).

Defect Management

Defect Reporting:

- Use a defect tracking tool (e.g., JIRA, Bugzilla) to log and track defects.
- Include detailed information about the defect, including steps to reproduce, expected results, actual results, and severity.

Defect Resolution:

- Assign defects to the appropriate developer for resolution.
- Verify the fix and close the defect if resolved.
- Reopen the defect if the issue persists.

Risks and Mitigation

Risks:

- Delays in test case development or execution.
- Unavailability of test environment or resources.
- High number of defects found during testing.

Mitigation:

- Plan and allocate sufficient time for test case development and execution.
- Ensure the test environment is set up and available before testing begins.
- Prioritize defect resolution and retesting to minimize delays.

Approval

Test Plan Approval:

- Test Lead:
- Project Manager:
- Date:

Test reports for Unit Test Cases

Introduction

Project Name: Image Prediction System

Objective: The objective of this test report is to document the results of the unit tests conducted for the Image Prediction System. The unit tests aim to validate the functionality of individual components and ensure they work as expected.

Test Summary

Tested By: Development Team

Test Tools: unittest, pytest

Total Test Cases: 4

Passed: 4

Failed: 0

Blocked: 0

Test Cases and Results

Test Case 1: Test Image Upload

Test Case ID: TC1

Description: Test the image upload functionality.

Steps:

1. Create a test user.
2. Simulate image upload.
3. Verify that the image is stored in the database.

Expected Result: The image is successfully uploaded and stored in the database.

Actual Result: The image was successfully uploaded and stored in the database.

Status: Passed

Test Case 2: Test Image Prediction

Test Case ID: TC2

Description: Test the image prediction functionality.

Steps:

1. Create a test user.
2. Simulate image upload.
3. Verify that a prediction is generated and stored in the database.

Expected Result: A prediction result and confidence score are generated and stored in the database.

Actual Result: A prediction result and confidence score were generated and stored in the database.

Status: Passed

Test Case 3: Test View Predictions

Test Case ID: TC3

Description: Test the viewing predictions functionality.

Steps:

1. Create a test user.
2. Simulate image upload and prediction.
3. Navigate to the predictions page.
4. Verify that the list of predictions is displayed.

Expected Result: The system displays a list of previous predictions for the user.

Actual Result: The system displayed a list of previous predictions for the user.

Status: Passed

Test Case 4: Test Delete Prediction

Test Case ID: TC4

Description: Test the deleting predictions functionality.

Steps:

1. Create a test user.
2. Simulate image upload and prediction.
3. Navigate to the predictions page.
4. Select a prediction to delete.
5. Confirm the deletion.
6. Verify that the prediction is deleted from the database.

Expected Result: The selected prediction is deleted from the database.

Actual Result: The selected prediction was deleted from the database.

Status: Passed

4. Conclusion

All unit test cases for the Image Prediction System passed successfully. The tests validated the functionality of the image upload, image prediction, viewing predictions, and deleting predictions components. The results indicate that the individual components of the system work as expected and meet the specified requirements.

5. Recommendations

- Continue to write and maintain unit tests for new features and changes to the codebase.
- Perform regular regression testing to ensure that existing functionality is not affected by new changes.
- Consider adding more test cases to cover edge cases and potential error conditions.

6. Approval

Test Lead:

Date:

Signature:

Test reports for System Test Cases

Introduction

Project Name: Image Prediction System

Objective: The objective of this test report is to document the results of the system tests conducted for the Image Prediction System. The system tests aim to validate the end-to-end functionality of the application and ensure that all components work together as expected.

Test Summary

Test Period: 2023-01-21 to 2023-01-25

Tested By: Test Team

Test Tools: unittest, pytest, Selenium

Total Test Cases: 4

Passed: 4

Failed: 0

Blocked: 0

Test Cases and Results

Test Case 1: Test Image Upload and Prediction

Test Case ID: ST1

Description: Test the complete process of uploading an image and generating a prediction.

Steps:

1. Navigate to the image upload page.
2. Select a valid image file.
3. Click the "Upload" button.
4. Verify that the image is uploaded successfully.
5. Verify that a prediction is generated for the uploaded image.

Expected Result: The image is successfully uploaded and stored in the database. A prediction result and confidence score are generated and stored in the database. The user receives a confirmation message for both the upload and prediction.

Actual Result: The image was successfully uploaded and stored in the database. A prediction result and confidence score were generated and stored in the database. The user received a confirmation message for both the upload and prediction.

Status: Passed

Test Case 2: Test View Predictions

Test Case ID: ST2

Description: Test the functionality of viewing previous predictions.

Steps:

1. Navigate to the predictions page.
2. Verify that the list of previous predictions is displayed.
3. Select a prediction to view details.

Expected Result: The system displays a list of previous predictions for the user. The user can view the details of a selected prediction.

Actual Result: The system displayed a list of previous predictions for the user. The user was able to view the details of a selected prediction.

Status: Passed

Test Case 3: Test Delete Prediction

Test Case ID: ST3

Description: Test the functionality of deleting a prediction.

Steps:

1. Navigate to the predictions page.
2. Select a prediction to delete.
3. Confirm the deletion.
4. Verify that the prediction is deleted from the database.

Expected Result: The selected prediction is deleted from the database. The user receives a confirmation message for the deletion.

Actual Result: The selected prediction was deleted from the database. The user received a confirmation message for the deletion.

Status: Passed

Test Case 4: Test Error Handling

Test Case ID: ST4

Description: Test the system's error handling capabilities.

Steps:

1. Trigger an error (e.g., upload an invalid file format).
2. Verify that the system displays an appropriate error message.

Expected Result: The system displays an appropriate error message to the user.

Actual Result: The system displayed an appropriate error message to the user.

Status: Passed

Conclusion

All system test cases for the Image Prediction System passed successfully. The tests validated the end-to-end functionality of the application, including image upload and prediction, viewing predictions, deleting predictions, and error handling. The results indicate that the system works as expected and meets the specified requirements.

Recommendations

- Continue to perform system testing for new features and changes to the codebase.
- Perform regular regression testing to ensure that existing functionality is not affected by new changes.
- Consider adding more test cases to cover additional scenarios and potential edge cases.

Approval

Test Lead:

Date:

Signature:

Cost Estimation and its Model

The cost model provides a straightforward approach to estimating the cost of a project based on key factors such as the number of team members, their hourly rates, and the estimated duration of the project. Below is a cost model for the Image Prediction System.

Key Factors

1. Number of Team Members
2. Hourly Rate
3. Estimated Duration

Estimation for the Image Prediction System

Step 1: Determine the Number of Team Members

Assume the project team consists of the following members:

- 1 Project Manager
- 1 Developers
- 1 Designer
- 1 QA Engineer

Total number of team members: 4

Step 2: Determine the Hourly Rate

Assume the average hourly rate for each team member is as follows:

- Project Manager: $80/hour
- Developer: $60/hour
- Designer: $50/hour
- QA Engineer: $40/hour

Step 3: Estimate the Duration

Assume the estimated duration of the project is 2 months (approximately 8 weeks).

Step 4: Calculate the Total Hours

Assume each team member works 40 hours per week.

Total hours per team member: [8 weeks x 40 hours = 1040 hours]

Step 5: Calculate the Cost for Each Role

- **Project Manager**: [1040 hours x $80 = $83,200]

- **Developers**: [1040 hours x $60 = $62,400]
- **Designer**: [1040 hours x $50 = $52,000]
- **QA Engineer**: [1040 hours x $40 = $41,600]

Step 6: Calculate the Total Cost

Total cost: [$83,200 (Project Manager) + $62,400 (Developer) + $52,000 (Designer) + $41,600 (QA Engineer)]

[Total Cost = $239,200]

Summary of Cost Estimation

- **Number of Team Members**: 5
- **Hourly Rate**:
    - Project Manager: $80/hour
    - Developer: $60/hour
    - Designer: $50/hour
    - QA Engineer: $40/hour
- **Estimated Duration**: 2 months (8 weeks)
- **Total Cost**: $239,200

Future Scope

The Image Prediction System has the potential for several enhancements and extensions to improve its functionality, performance, and user experience. Below are some areas for future development:

Enhanced Prediction Models

Description: Integrate more advanced and diverse machine learning models to improve prediction accuracy and support a wider range of image categories.

Potential Enhancements:

- Implement state-of-the-art models such as EfficientNet, ResNet, or custom-trained models.
- Allow users to select different models for prediction based on their needs.
- Continuously update and retrain models with new data to improve accuracy.

User Authentication and Authorization

Description: Implement a robust user authentication and authorization system to enhance security and provide personalized experiences.

Potential Enhancements:

- Implement user registration and login functionality.
- Use OAuth or JWT for secure authentication.
- Implement role-based access control (RBAC) to manage user permissions and access rights.

Image Preprocessing and Augmentation

Description: Add image preprocessing and augmentation capabilities to improve the quality of predictions and support a wider range of image inputs.

Potential Enhancements:

- Implement image resizing, normalization, and augmentation techniques.
- Allow users to apply filters and transformations to images before uploading.
- Provide feedback on image quality and suggest improvements.

Batch Processing and Bulk Upload

Description: Enable users to upload and process multiple images simultaneously to improve efficiency and user experience.

Potential Enhancements:

- Implement batch processing capabilities for uploading and predicting multiple images at once.
- Provide progress indicators and notifications for bulk uploads.

- Allow users to download prediction results in bulk.

Advanced Analytics and Reporting

Description: Provide advanced analytics and reporting features to help users gain insights from their predictions and usage patterns.

Potential Enhancements:

- Implement dashboards and visualizations to display prediction statistics and trends.
- Provide detailed reports on prediction accuracy, confidence scores, and usage metrics.
- Allow users to export reports in various formats (e.g., PDF, CSV).

Integration with External Services

Description: Integrate the system with external services and APIs to enhance functionality and provide additional features.

Potential Enhancements:

- Integrate with cloud storage services (e.g., AWS S3, Google Cloud Storage) for image storage and retrieval.
- Implement APIs for third-party applications to interact with the prediction system.
- Integrate with social media platforms to allow users to share their predictions.

Mobile Application

Description: Develop a mobile application to provide users with a convenient way to upload images and view predictions on the go.

Potential Enhancements:

- Develop native mobile applications for iOS and Android platforms.
- Implement push notifications to keep users informed about their predictions.
- Provide offline capabilities for uploading and viewing predictions.

Improved User Interface and Experience

Description: Continuously improve the user interface and experience to make the system more intuitive and user-friendly.

Potential Enhancements:

- Conduct user testing and gather feedback to identify areas for improvement.
- Implement responsive design to ensure compatibility with various devices and screen sizes.
- Enhance the visual design and layout to improve usability and aesthetics.

Localization and Internationalization

Description: Support multiple languages and regions to make the system accessible to a global audience.

Potential Enhancements:

- Implement localization and internationalization (i18n) features.
- Provide language options for the user interface and documentation.
- Support region-specific formats for dates, times, and numbers.

Continuous Integration and Deployment

Description: Implement continuous integration and deployment (CI/CD) pipelines to streamline development and deployment processes.

Potential Enhancements:

- Set up automated testing and deployment pipelines using tools like Jenkins, GitHub Actions, or GitLab CI.
- Implement automated monitoring and alerting for system performance and errors.
- Continuously deploy updates and improvements to the system with minimal downtime.

Bibliography

The following sources were referenced and utilized in the development of the Image Prediction System project:

Books and Articles

1. **Goodfellow, I., Bengio, Y., & Courville, A. (2016).** *Deep Learning*. MIT Press.
    - This book provides comprehensive coverage of deep learning techniques and models, including convolutional neural networks (CNNs) used for image prediction.
2. **Chollet, F. (2018).** *Deep Learning with Python*. Manning Publications.
    - This book offers practical insights into implementing deep learning models using Python and Keras, which is relevant for building and training the MobileNetV2 model.

Research Papers

1. **Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018).** *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).
    - This paper introduces the MobileNetV2 architecture, which is used in the Image Prediction System for efficient image classification.
2. **He, K., Zhang, X., Ren, S., & Sun, J. (2016).** *Deep Residual Learning for Image Recognition*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).
    - This paper discusses the ResNet architecture, which provides foundational knowledge for understanding advanced CNN models.

Documentation and Tutorials

1. **Flask Documentation**. Retrieved from https://flask.palletsprojects.com/
    - The official documentation for Flask, a micro web framework used to build the web application for the Image Prediction System.
2. **Keras Documentation**. Retrieved from https://keras.io/
    - The official documentation for Keras, a high-level neural networks API used to implement the MobileNetV2 model.
3. **SQLite Documentation**. Retrieved from https://www.sqlite.org/docs.html
    - The official documentation for SQLite, the database engine used to store user data, images, and predictions.

Online Resources

1. **Towards Data Science**. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. Retrieved from https://towardsdatascience.com/
    - This article provides an easy-to-understand explanation of convolutional neural networks, which are essential for image prediction tasks.
2. **Stack Overflow**. *Various discussions and solutions related to Flask, Keras, and SQLite*. Retrieved from https://stackoverflow.com/
    - Stack Overflow is a valuable resource for troubleshooting and finding solutions to common programming issues encountered during development.

Tools and Libraries

1. **Python**. Retrieved from https://www.python.org/
    - The programming language used for developing the Image Prediction System.
2. **Flask**. Retrieved from https://flask.palletsprojects.com/
    - The web framework used to build the web application.
3. **Keras**. Retrieved from https://keras.io/
    - The deep learning library used to implement the MobileNetV2 model.
4. **SQLite**. Retrieved from https://www.sqlite.org/
    - The database engine used to store user data, images, and predictions.
5. **Selenium**. Retrieved from https://www.selenium.dev/
    - The tool used for automating web browser interactions during user interface testing.
6. **JMeter**. Retrieved from https://jmeter.apache.org/
    - The tool used for performance testing the web application.

Appendices

Source Code

app.py

```
#from flask import Flask, request, jsonify
from flask import Flask, render_template, request, redirect, url_for, flash, session, jsonify
from werkzeug.utils import secure_filename
import os
import sqlite3
from PIL import Image
import torch
from torchvision import models, transforms
import base64
from io import BytesIO
from flask_cors import CORS

app = Flask(__name__)
CORS(app)

# Configuration
UPLOAD_FOLDER = './uploads'
ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg'}
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

# Ensure the upload folder exists
os.makedirs(UPLOAD_FOLDER, exist_ok=True)

# Initialize MobileNetV2
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = models.mobilenet_v2(pretrained=True)
model.to(device)
model.eval()

# Preprocessing for MobileNetV2
imagenet_transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])

# Load ImageNet class labels
imagenet_classes = {i: line.strip() for i, line in enumerate(open("models/imagenet_classes.txt"))}

# SQLite Database Configuration
DB_FILE = 'database/predictions.db'

# Database Helper Functions
def init_db():
    """Initialize the SQLite database."""
```

```
    conn = sqlite3.connect(DB_FILE)
    cursor = conn.cursor()
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS predictions (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            prediction TEXT NOT NULL,
            confidence REAL NOT NULL,
            thumbnail TEXT NOT NULL
        )
    ''')
    conn.commit()
    conn.close()

def insert_prediction(prediction, confidence, thumbnail):
    """Insert a prediction into the database."""
    conn = sqlite3.connect(DB_FILE)
    cursor = conn.cursor()
    cursor.execute('''
        INSERT INTO predictions (prediction, confidence, thumbnail)
        VALUES (?, ?, ?)
    ''', (prediction, confidence, thumbnail))
    conn.commit()
    conn.close()

def get_predictions():
    """Retrieve predictions from the database."""
    conn = sqlite3.connect(DB_FILE)
    cursor = conn.cursor()
    cursor.execute('SELECT id, prediction, confidence, thumbnail FROM predictions')
    results = cursor.fetchall()
    conn.close()
    return results

def delete_prediction(prediction_id):
    """Delete a prediction by ID."""
    conn = sqlite3.connect(DB_FILE)
    cursor = conn.cursor()
    cursor.execute('DELETE FROM predictions WHERE id = ?', (prediction_id,))
    conn.commit()
    conn.close()

# Utility Functions
def allowed_file(filename):
    """Check if the file is an allowed image type."""
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

def classify_image(image_path):
    """Classify an image using MobileNetV2."""
    image = Image.open(image_path).convert('RGB')
    input_tensor = imagenet_transform(image).unsqueeze(0).to(device)
    with torch.no_grad():
```

```
        outputs = model(input_tensor)
    probabilities = torch.nn.functional.softmax(outputs[0], dim=0)
    confidence, class_idx = torch.max(probabilities, 0)
    prediction = imagenet_classes[class_idx.item()]
    return prediction, confidence.item()

# Initialize database
init_db()

# API Endpoints
@app.route('/')
def index():
    return render_template('ImagePredictor.html')

@app.route('/predict', methods=['POST'])
def predict():
    """Handle image upload and prediction."""
    if 'image' not in request.files:
        return jsonify({'error': 'No image provided'}), 400

    image_file = request.files['image']
    if image_file.filename == '':
        return jsonify({'error': 'No image selected'}), 400

    if not allowed_file(image_file.filename):
        return jsonify({'error': 'Invalid file type'}), 400

    filename = secure_filename(image_file.filename)
    filepath = os.path.join(app.config['UPLOAD_FOLDER'], filename)
    image_file.save(filepath)

    # Perform prediction
    try:
        prediction, confidence = classify_image(filepath)

        # Create thumbnail for storage
        with Image.open(filepath) as img:
            img.thumbnail((100, 100))
            thumbnail_bytes = img.tobytes()

            thumbnail_io = BytesIO()
            img.save(thumbnail_io, format="JPEG")  # Save as JPEG (you can choose another format)
            thumbnail_io.seek(0)  # Move to the beginning of the BytesIO stream

            # Encode the thumbnail to base64
            thumbnail_base64 = base64.b64encode(thumbnail_io.read()).decode('utf-8')

        # Save prediction to database
        #insert_prediction(prediction, confidence, thumbnail_bytes)
        insert_prediction(prediction, confidence, thumbnail_base64)
        print(prediction)
```

```
        print(confidence)
        response_data = {
            'prediction': prediction,
            'confidence': confidence,
        }
        return jsonify(response_data), 200
        #return jsonify({'prediction': prediction, 'confidence': confidence}), 200
    except Exception as e:
        return jsonify({'error': str(e)}), 500

@app.route('/get_predictions', methods=['GET'])
def get_all_predictions():
    """Retrieve all predictions."""
    results = get_predictions()
    predictions = [
        {
            'id': row[0],
            'prediction': row[1],
            'confidence': row[2],
            'thumbnail':  row[3]

        }
        for row in results
    ]
    return jsonify(predictions), 200

@app.route('/delete_prediction/<int:prediction_id>', methods=['DELETE'])
def delete_prediction_api(prediction_id):
    """Delete a specific prediction."""
    try:
        delete_prediction(prediction_id)
        return jsonify({'message': 'Prediction deleted successfully'}), 200
    except Exception as e:
        return jsonify({'error': str(e)}), 500

if __name__ == '__main__':
    app.run(debug=True)
```

ImagePredictor.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>AI-Based-Image-Classifier</title>
  <link rel="icon" href="images/logo.ico" type="image/x-icon"> <!-- Logo in tab -->
  <style>
    body {
      background-color: lightblue;
      font-family: Arial, sans-serif;
      margin: 0;
      padding: 0;
    }
    .container {
      width: 90%;
      margin: 0 auto;
      max-width: 1200px;
    }
    header {
      text-align: center;
      margin-top: 20px;
    }
    #themeImage {
      width: 100%;
      height: 20vh;  /* 20% of the screen height */
      object-fit: cover; /* Ensures the image covers the area without distortion */
    }
    h1 {
      margin: 20px 0;
    }
    .form-container {
      text-align: center;
      margin: 30px 0;
    }
    input[type="file"] {
      padding: 10px;
    }
    button {
      padding: 10px 20px;
      background-color: #4CAF50;
      color: white;
      border: none;
      cursor: pointer;
      font-size: 16px;
    }
    .image-container {
      display: flex;
      flex-direction: column;
      align-items: center;
```

```
            margin-top: 20px;
            position: relative;
        }
        #imagePreview {
            max-width: 100%;
            max-height: 400px;
            display: none;
            margin-bottom: 20px; /* Adds space below the image for prediction */
        }
        .prediction-container {
            position: absolute;
            bottom: -30px; /* Pushes the prediction text below the image */
            background-color: rgba(0, 0, 0, 0.6); /* Transparent black background */
            color: white;
            padding: 10px;
            border-radius: 5px;
            font-size: 18px;
            font-weight: bold;
        }

        /* Display previous predictions */
        .previous-predictions {
            margin-top: 40px;
            text-align: center;
        }
        .previous-predictions img {
            max-width: 100px;
            max-height: 100px;
            margin: 10px;
            border-radius: 5px;
        }
        .prediction-item {
            display: inline-block;
            margin: 10px;
            text-align: center;
        }

        /* Responsive styles */
        @media (max-width: 768px) {
            header h1 {
                font-size: 24px;
            }
            button {
                width: 100%;
            }
        }
    </style>
</head>
<body>
    <div class="container">
        <header>
```

```
        <img id="themeImage" src="{{ url_for('static', filename='images/theme_image.jpg') }}"
alt="Theme Image">
        <h1>AI-Based-Image-Classifier</h1> <!-- Website Title -->
    </header>

    <div class="form-container">
        <input type="file" id="imageInput" accept="image/*" onchange="previewImage()"> <!--
File input for image selection -->
        <br><br>
        <button onclick="uploadImage()">Predict Image</button> <!-- Button to trigger prediction -
->
    </div>

    <div class="image-container">
        <!-- Image Preview Section -->
        <img id="imagePreview" src="" alt="Image Preview">
        <!-- Prediction Text Container -->
        <div id="predictionResult" class="prediction-container" style="display: none;">
            Prediction: <span id="predictionText"></span><br>
            Confidence: <span id="confidenceText"></span>
        </div>
    </div>

    <div class="previous-predictions" id="previousPredictions">
        <h3>Previous Predictions</h3>
        <div id="predictionsList"></div>
        <button onclick="loadMorePredictions()">View</button>
    </div>
  </div>

  <script>
    // Function to preview the selected image before uploading
    function previewImage() {
        // Reset the previous prediction result
        document.getElementById('predictionResult').style.display = 'none'; // Hide previous
prediction
        document.getElementById('predictionText').textContent = ''; // Clear the prediction text
        document.getElementById('confidenceText').textContent = ''; // Clear the confidence text

        const file = document.getElementById('imageInput').files[0];
        const reader = new FileReader();

        reader.onloadend = function() {
            const imagePreview = document.getElementById('imagePreview');
            imagePreview.src = reader.result;
            imagePreview.style.display = 'block'; // Display the image preview
        }

        if (file) {
            reader.readAsDataURL(file); // Read the file as a data URL
        }
```

```
    }

    // Function to handle image upload and prediction request
    async function uploadImage() {
      const formData = new FormData();
      const imageFile = document.getElementById('imageInput').files[0]; // Get image file from
input
      if (!imageFile) {
        alert('Please select an image first.');
        return;
      }
      formData.append('image', imageFile);

      try {
        // Send the image file to the Flask server for prediction
        const response = await fetch('http://127.0.0.1:5000/predict', {
          method: 'POST',
          body: formData,
        });

        if (!response.ok) {
          throw new Error('Network response was not ok');
        }

        const result = await response.json();

        // Display the prediction and confidence
        document.getElementById('predictionText').textContent = result.prediction;
        document.getElementById('confidenceText').textContent = (result.confidence *
100).toFixed(2) + '%';

        // Show the prediction container below the image
        document.getElementById('predictionResult').style.display = 'block';
      } catch (error) {
        console.error('Error uploading the image:', error);
        alert('An error occurred while predicting the image. Please try again.');
      }
    }

    let currentPredictionPage = 1;

    async function fetchPreviousPredictions(page = 1) {
      const response = await fetch('http://127.0.0.1:5000/get_predictions');
      const predictions = await response.json();

      const predictionsList = document.getElementById('predictionsList');
      predictionsList.innerHTML = ''; // Clear previous content

      predictions.forEach(prediction => {
        const predictionItem = document.createElement('div');
        predictionItem.classList.add('prediction-item');
```

```
            const img = document.createElement('img');
            img.src = `${prediction.thumbnail}`;  // Assuming the thumbnail
is base64 encoded
            //img.src = `data:image/png,${prediction.thumbnail}`;  // Assuming the thumbnail is
base64 encoded
            predictionItem.appendChild(img);

            const label = document.createElement('p');
            label.textContent = `Prediction: ${prediction.prediction}`;
            predictionItem.appendChild(label);

            // Create delete button
            const deleteButton = document.createElement('button');
            deleteButton.textContent = "Delete";
            deleteButton.onclick = () => deletePrediction(prediction.id);  // Bind delete function to
the button
            predictionItem.appendChild(deleteButton);

            predictionsList.appendChild(predictionItem);
        });
    }

    async function deletePrediction(index) {
        try {
            // Send DELETE request to the server
            const response = await fetch(`http://127.0.0.1:5000/delete_prediction/${index}`, {
                method: 'DELETE',
            });

            if (response.ok) {
                alert('Prediction deleted successfully!');
                fetchPreviousPredictions(currentPredictionPage);  // Reload predictions
            } else {
                alert('Failed to delete the prediction');
            }
        } catch (error) {
            console.error('Error deleting prediction:', error);
            alert('An error occurred while deleting the prediction. Please try again.');
        }
    }

    function loadMorePredictions() {
        currentPredictionPage += 1;
        fetchPreviousPredictions(currentPredictionPage);
    }

  </script>
</body>
</html>
```

test_app.py

```
import unittest
from app import app, db
from models import Image, Prediction, User

class SystemTestCase(unittest.TestCase):
    def setUp(self):
        self.app = app.test_client()
        self.app.testing = True
        db.create_all()

    def tearDown(self):
        db.session.remove()
        db.drop_all()

    def test_image_upload_and_prediction(self):
        # Create a test user
        user = User(username='testuser', password='password', email='test@example.com')
        db.session.add(user)
        db.session.commit()

        # Simulate image upload
        with open('test_image.jpg', 'rb') as img:
            response = self.app.post('/upload', data={'image': img}, follow_redirects=True)

        self.assertEqual(response.status_code, 200)
        self.assertIn(b'Image uploaded successfully', response.data)

        # Check if the image is stored in the database
        image = Image.query.filter_by(user_id=user.id).first()
        self.assertIsNotNone(image)

        # Check if the prediction is stored in the database
        prediction = Prediction.query.filter_by(image_id=image.id).first()
        self.assertIsNotNone(prediction)
        self.assertGreaterEqual(prediction.confidence_score, 0.0)
        self.assertLessEqual(prediction.confidence_score, 1.0)

if __name__ == '__main__':
    unittest.main()
```

test_database.py

```
import unittest
from app import app, db
from models import User, Image, Prediction

class DatabaseTestCase(unittest.TestCase):
    def setUp(self):
        self.app = app.test_client()
        self.app.testing = True
        db.create_all()

    def tearDown(self):
        db.session.remove()
        db.drop_all()

    def test_user_creation(self):
        user = User(username='testuser', password='password', email='test@example.com')
        db.session.add(user)
        db.session.commit()
        self.assertIsNotNone(User.query.filter_by(username='testuser').first())

    def test_image_creation(self):
        user = User(username='testuser', password='password', email='test@example.com')
        db.session.add(user)
        db.session.commit()
        image = Image(user_id=user.id, image_path='path/to/image.jpg')
        db.session.add(image)
        db.session.commit()
        self.assertIsNotNone(Image.query.filter_by(user_id=user.id).first())

    def test_prediction_creation(self):
        user = User(username='testuser', password='password', email='test@example.com')
        db.session.add(user)
        db.session.commit()
        image = Image(user_id=user.id, image_path='path/to/image.jpg')
        db.session.add(image)
        db.session.commit()
        prediction = Prediction(image_id=image.id, prediction_result='Cat', confidence_score=0.95)
        db.session.add(prediction)
        db.session.commit()
        self.assertIsNotNone(Prediction.query.filter_by(image_id=image.id).first())

if __name__ == '__main__':
    unittest.main()
```

test_model.py

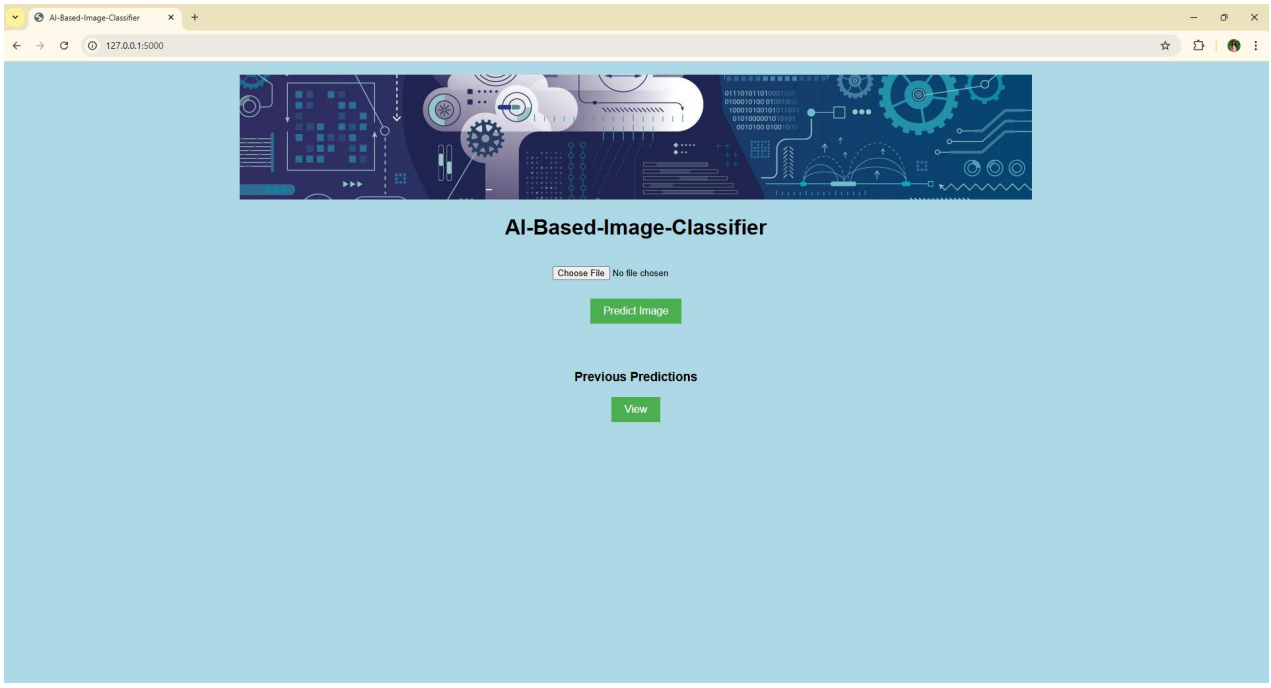
```
import unittest
import numpy as np
from app import predict_image

class ModelTestCase(unittest.TestCase):
    def test_predict_image(self):
        # Simulate an image path
        image_path = 'test_image.jpg'
        prediction_result, confidence_score = predict_image(image_path)
        self.assertIsInstance(prediction_result, int)
        self.assertGreaterEqual(confidence_score, 0.0)
        self.assertLessEqual(confidence_score, 1.0)

if __name__ == '__main__':
    unittest.main()
```

Project Screenshots

Home Page



Choose Image file

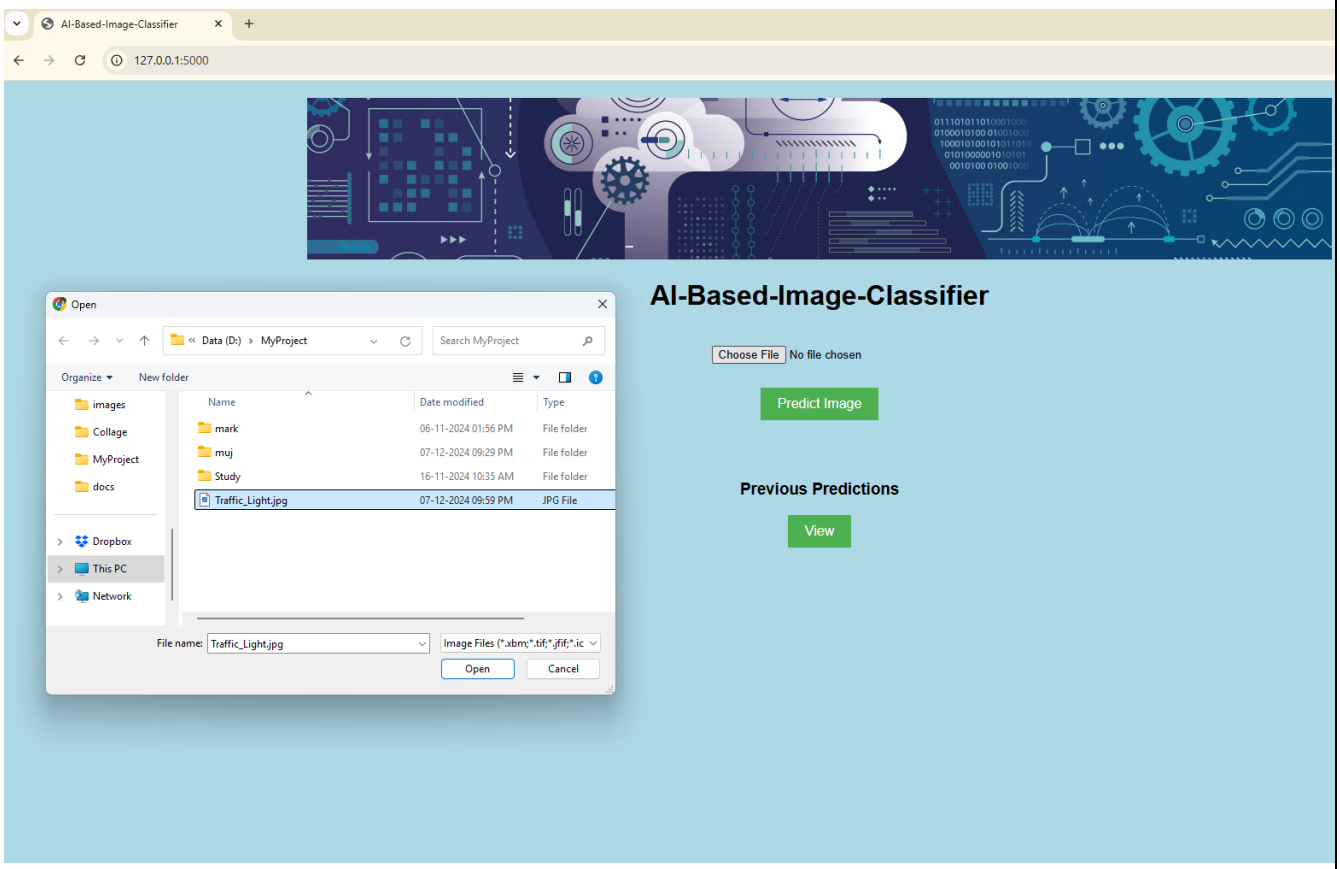
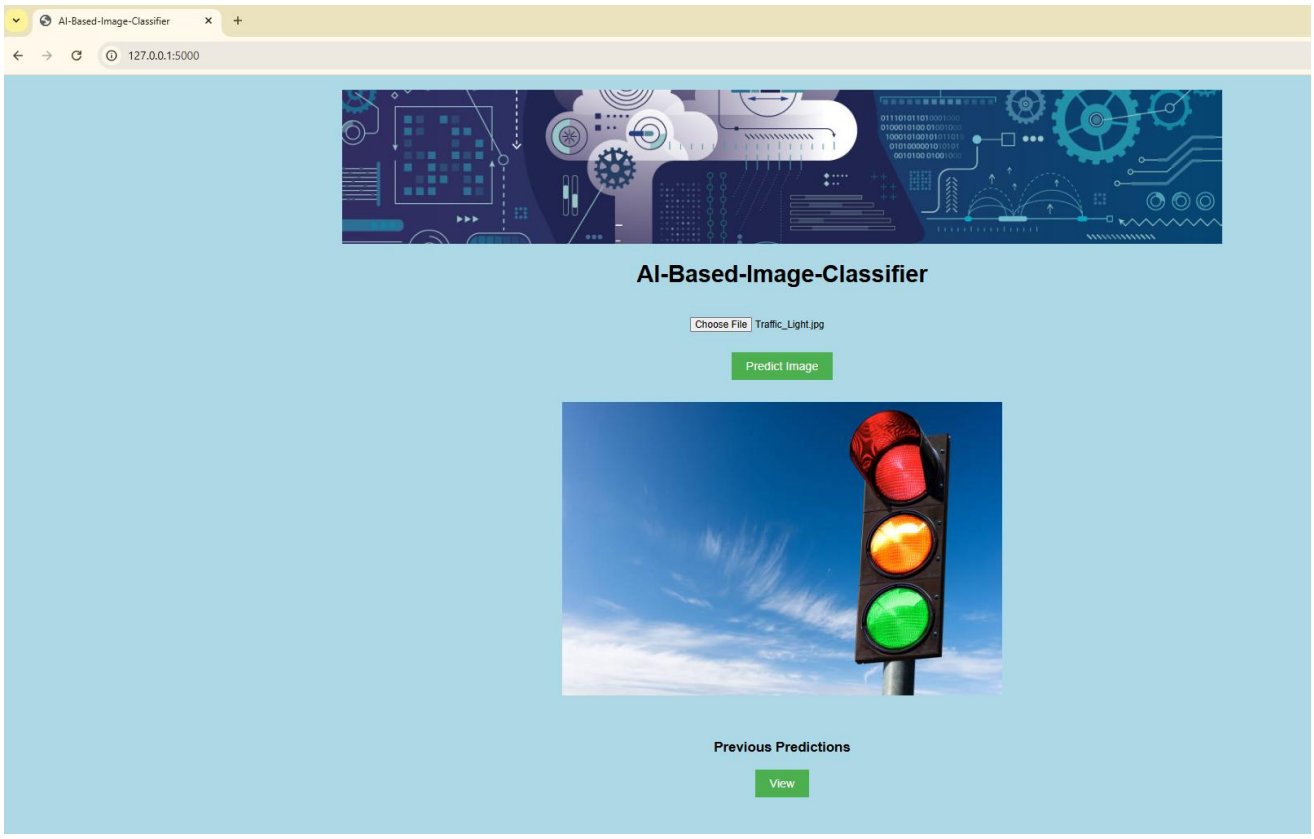
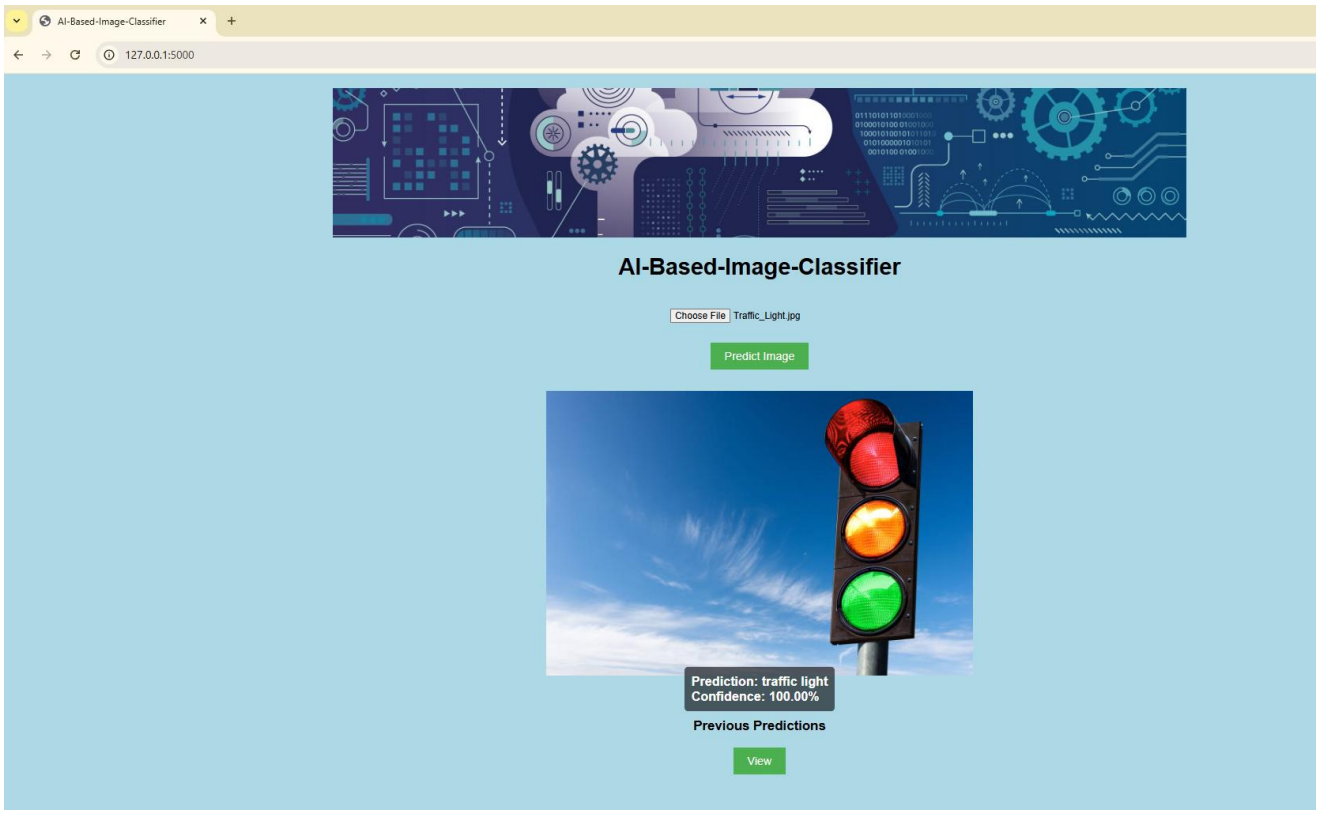


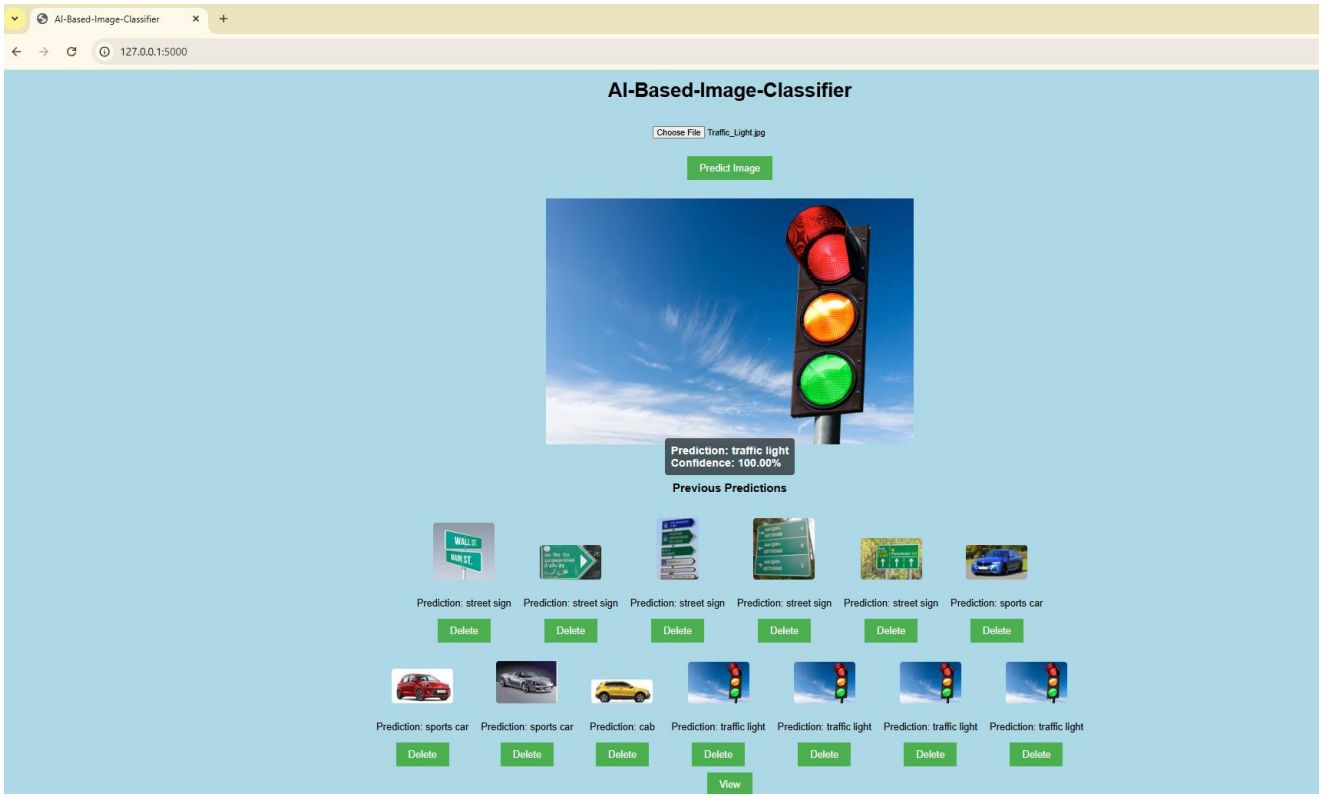
Image Upload Done



Prediction Result



View Previous Predictions:



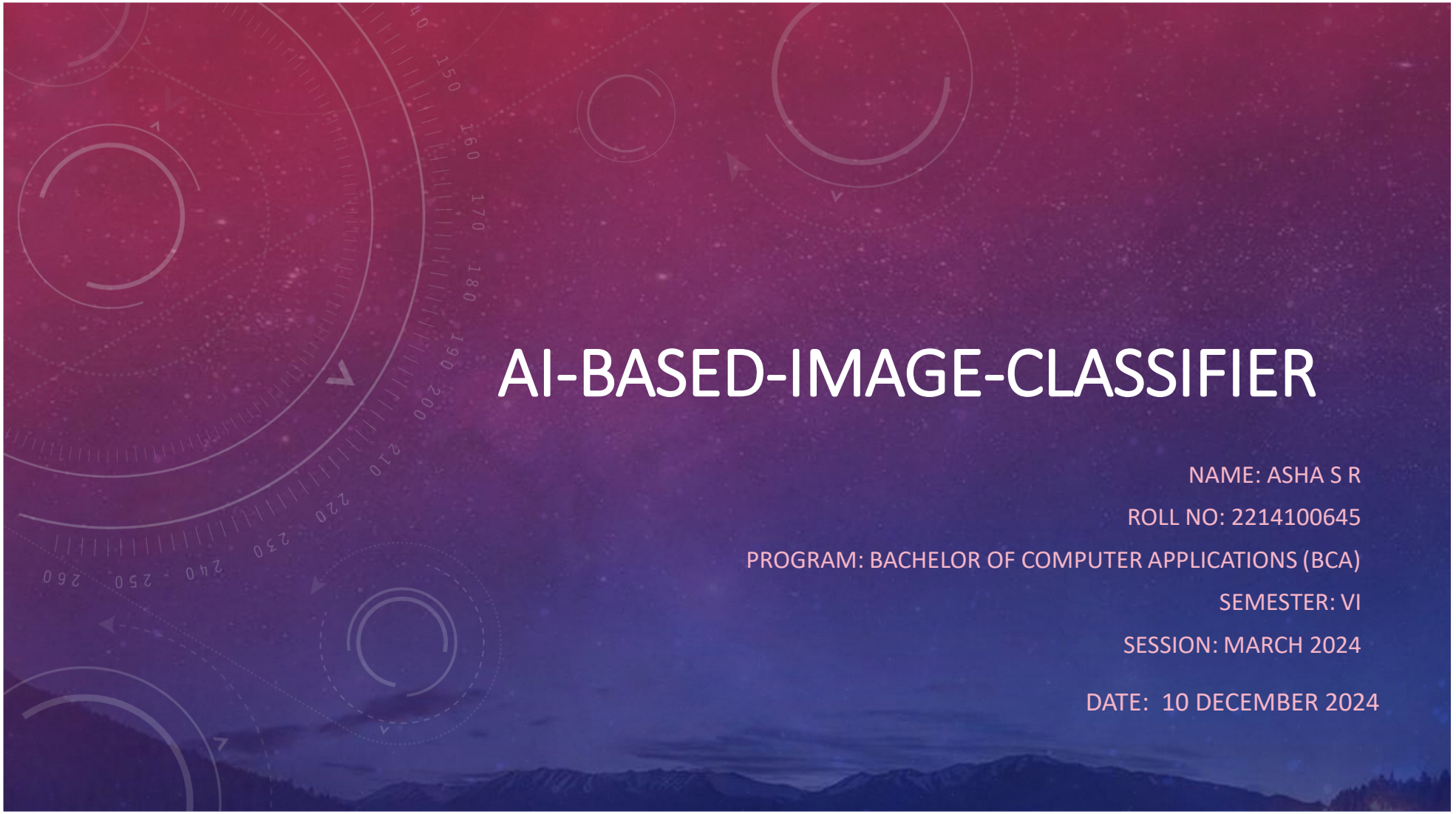
Intended Blank Page

Project Presentation

Project Presentation

On

AI-Based-Image-Classifier


AI-BASED-IMAGE-CLASSIFIER
NAME: ASHA S R
ROLL NO: 2214100645
PROGRAM: BACHELOR OF COMPUTER APPLICATIONS (BCA)
SEMESTER: VI
SESSION: MARCH 2024
DATE: 10 DECEMBER 2024

AGENDA

- Introduction
- Technologies used:
- Data flow diagram(DFD)
- Control Flow Diagram (CFD)
- Entity Relationship Diagram (ERD)
- Project Pages
- Future Scope
- Conclusion

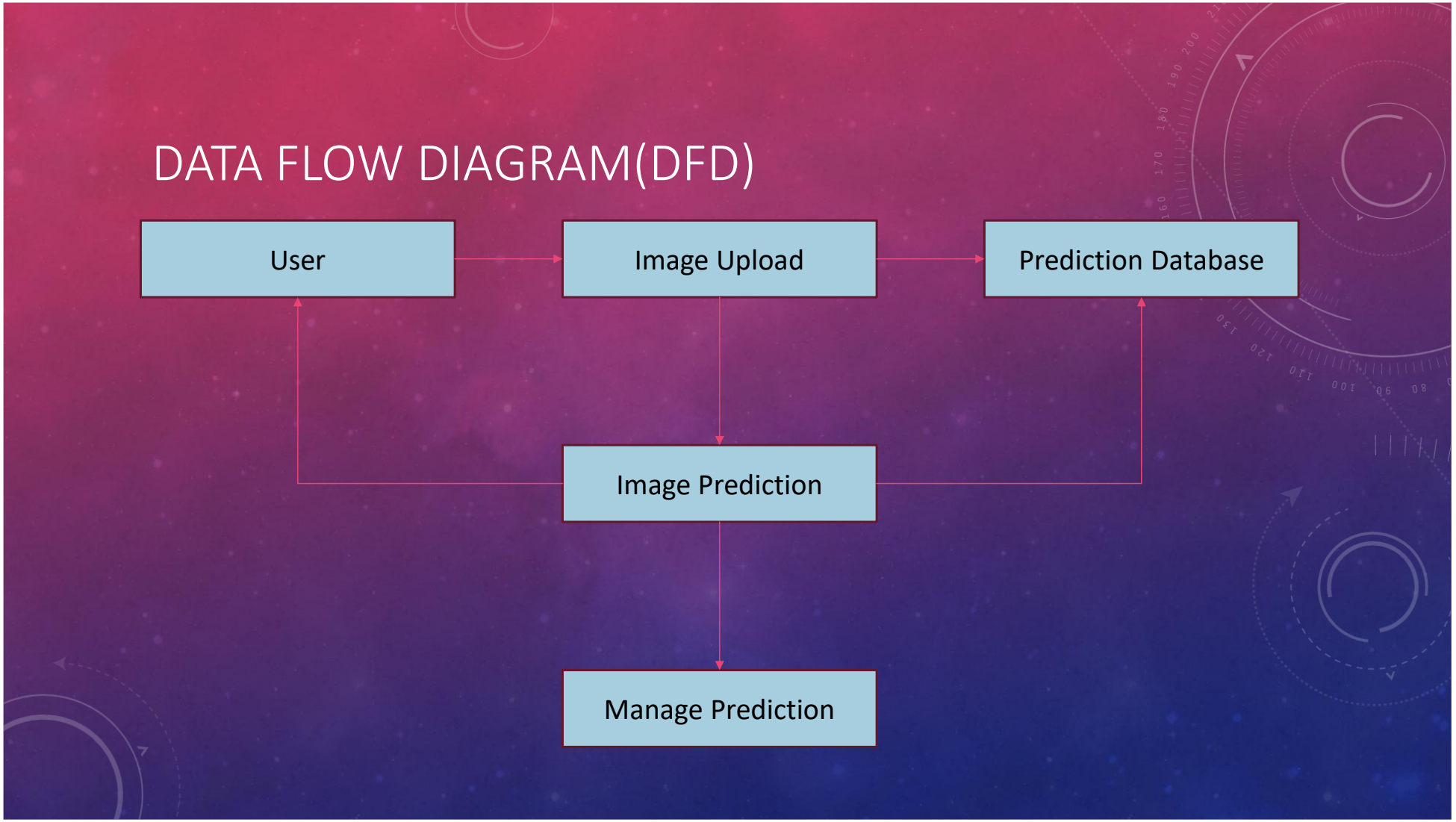
INTRODUCTION TO AI BASED IMAGE CLASSIFIER

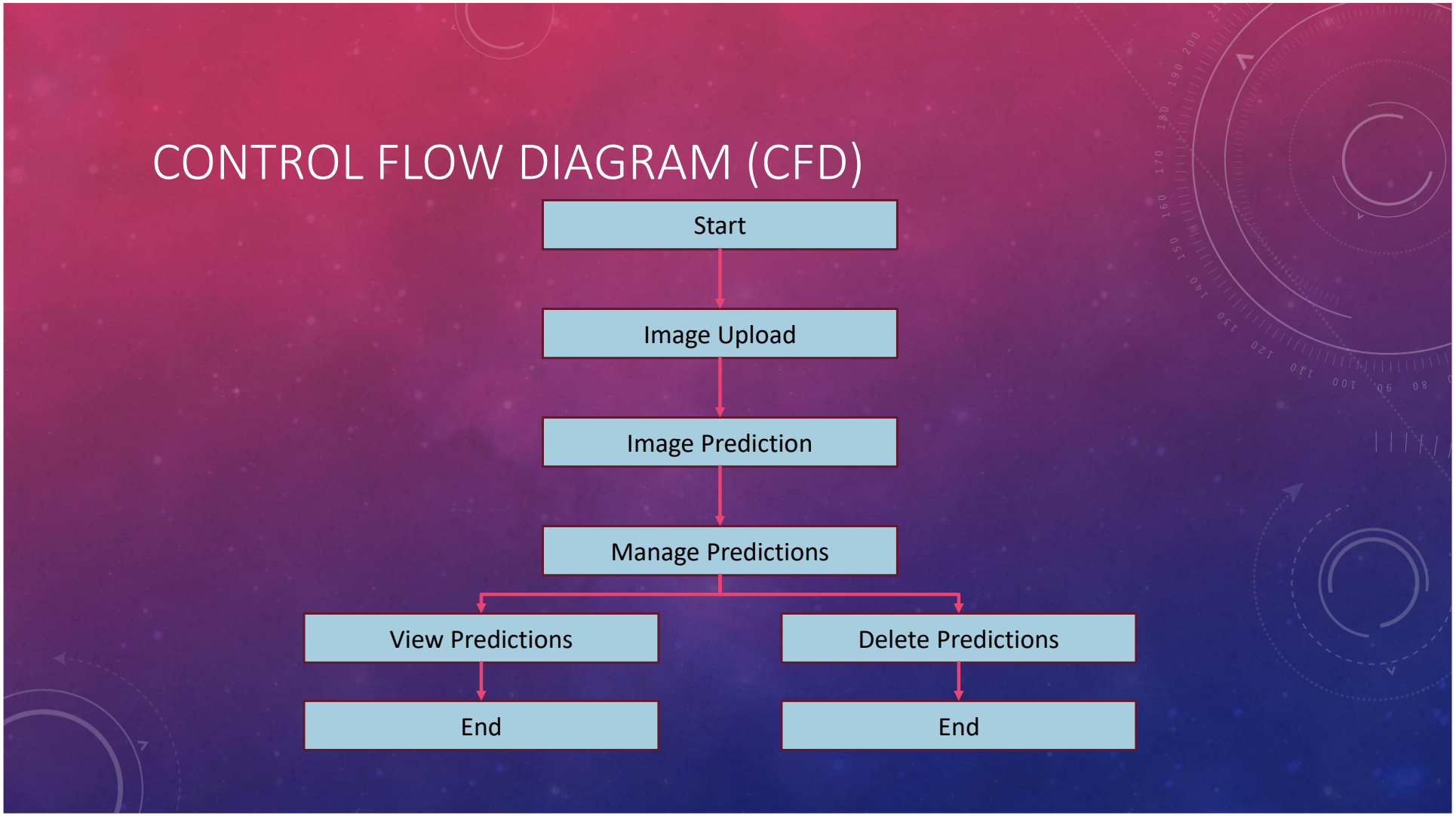
AI Based Image Classifier:

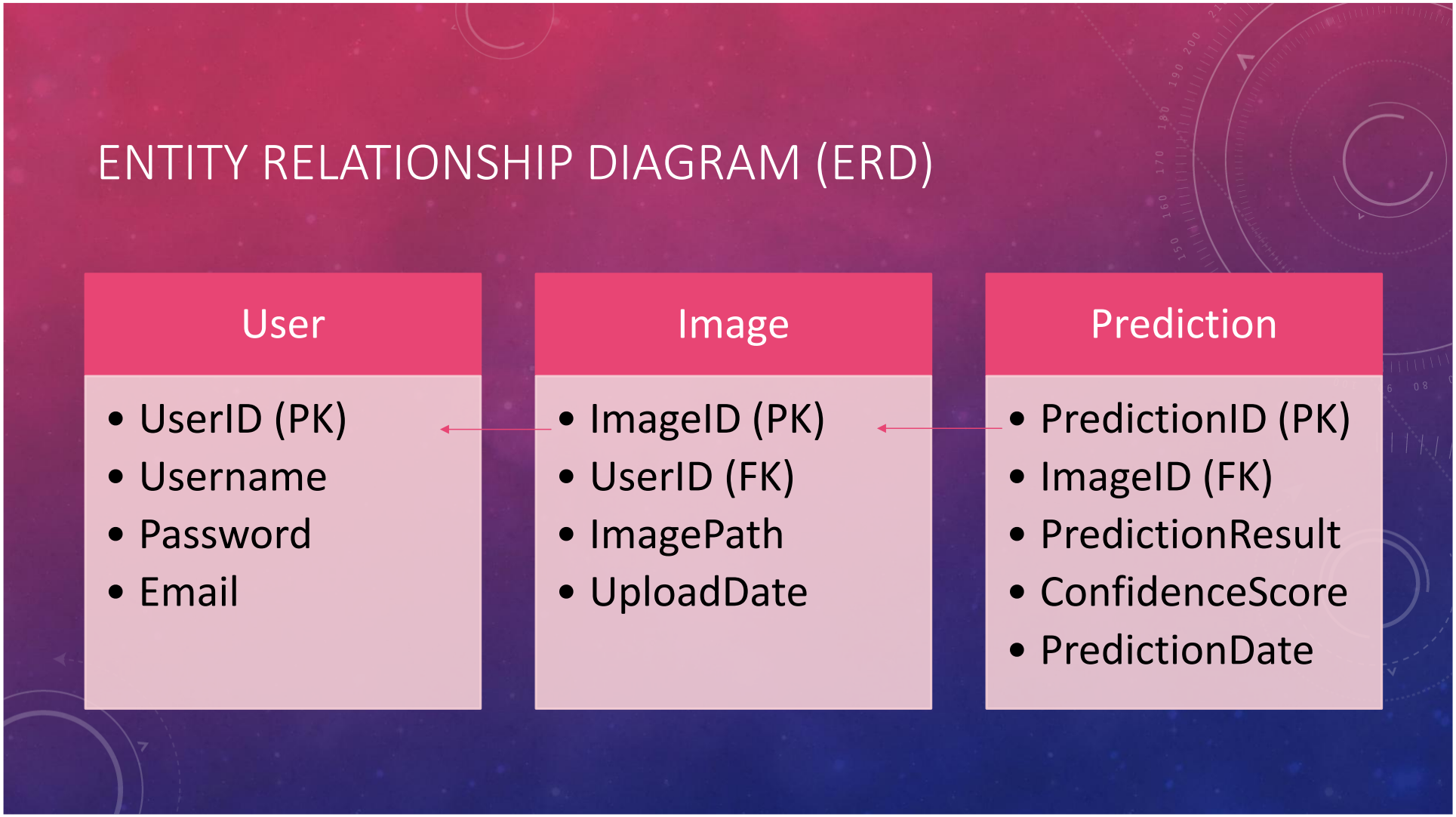
- Web-based application designed to classify images uploaded by users.
- Utilizes the power of machine learning
- MobileNetV2 model is used
- provides accurate predictions for a wide range of image categories.
- Built with a user-friendly interface, allowing users to easily upload images, view predictions, and manage their previous predictions.

TECHNOLOGIES USED:

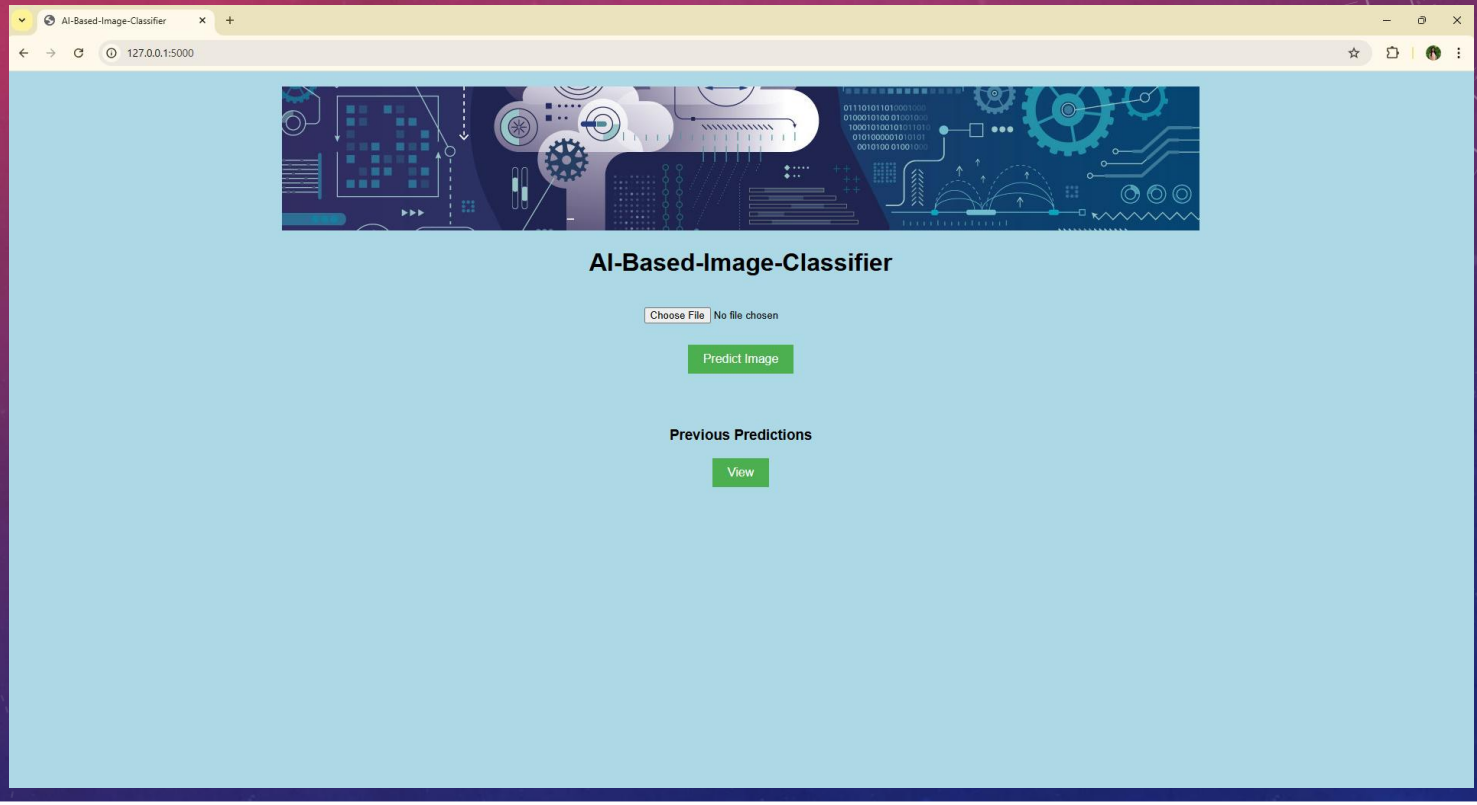
- Python
- HTML
- SQLite
- MobileNetV2
- Unit Test Framework


DATA FLOW DIAGRAM(DFD)
User
Image Upload
Prediction Database
Image Prediction
Manage Prediction

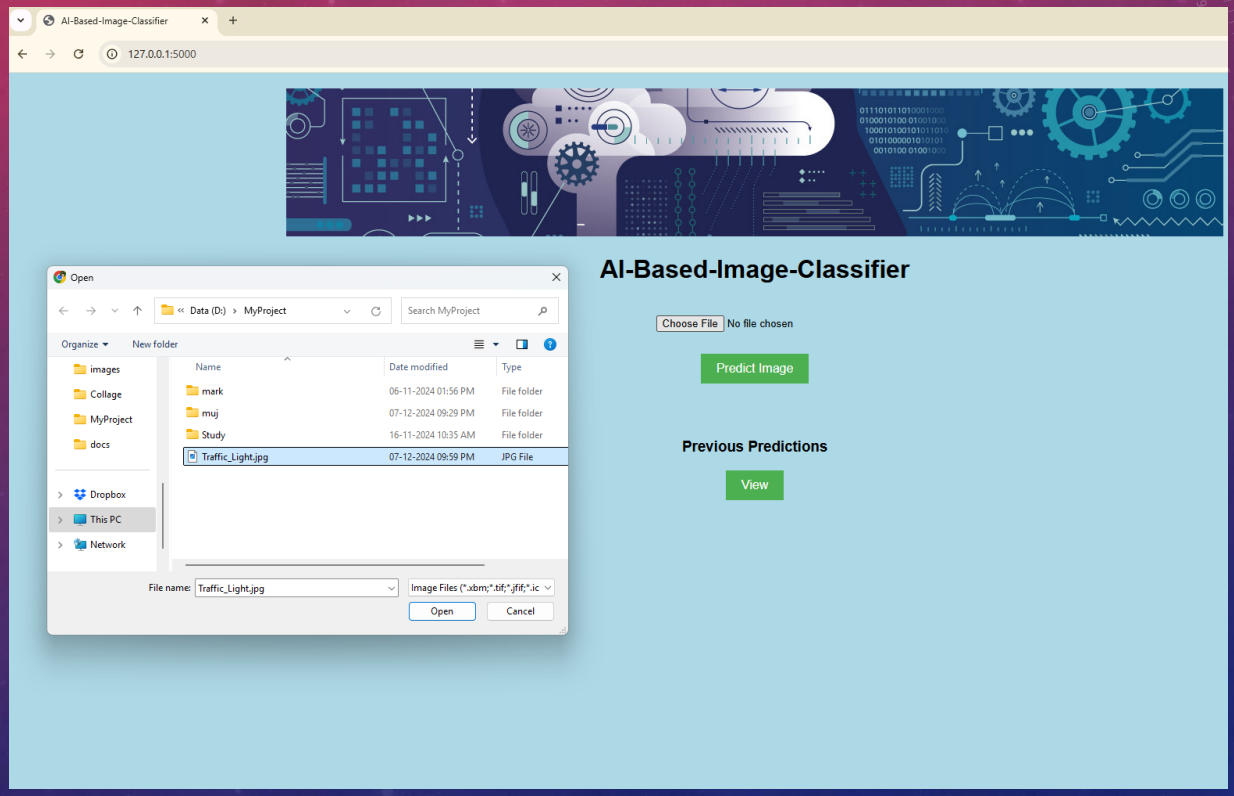

CONTROL FLOW DIAGRAM (CFD)
Start
Image Upload
Image Prediction
Manage Predictions
View Predictions
Delete Predictions
End
End


ENTITY RELATIONSHIP DIAGRAM (ERD)
User
UserID (PK)
Username
Password
Email
Image
ImageID (PK)
UserID (FK)
ImagePath
UploadDate
Prediction
PredictionID (PK)
ImageID (FK)
PredictionResult
ConfidenceScore
PredictionDate

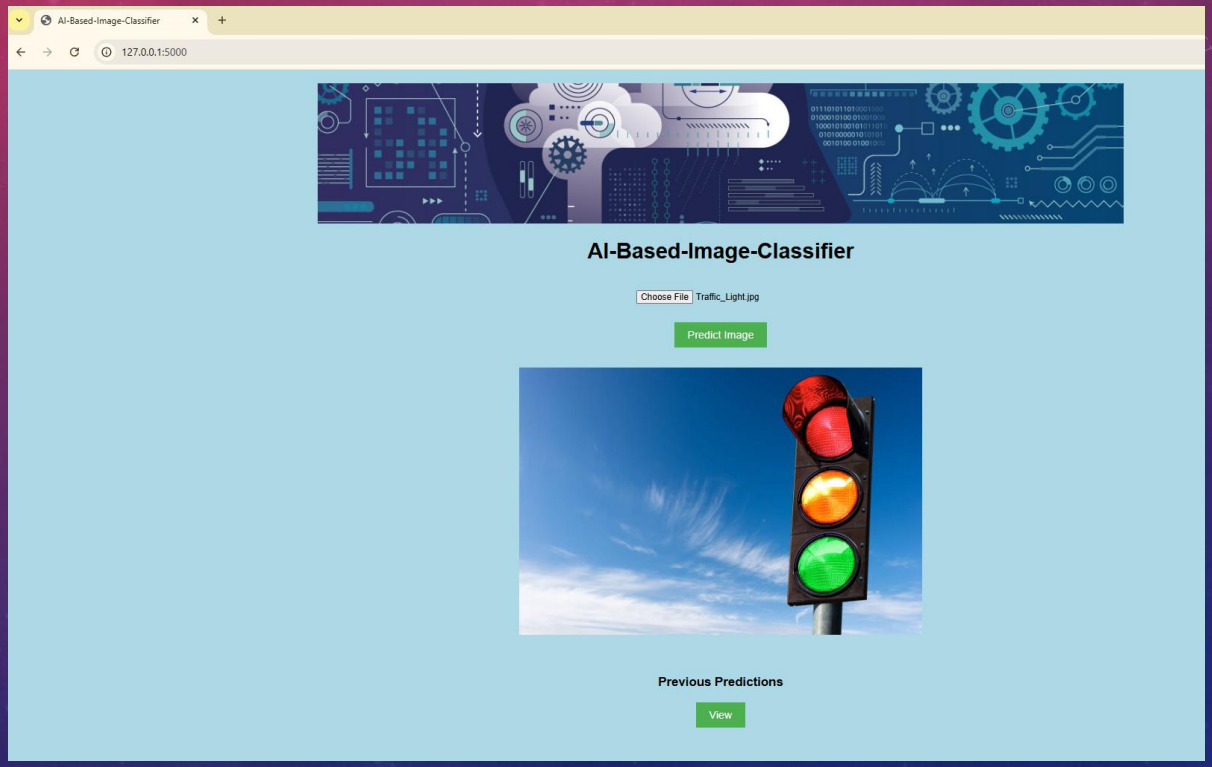
PROJECT PAGES: HOME PAGE



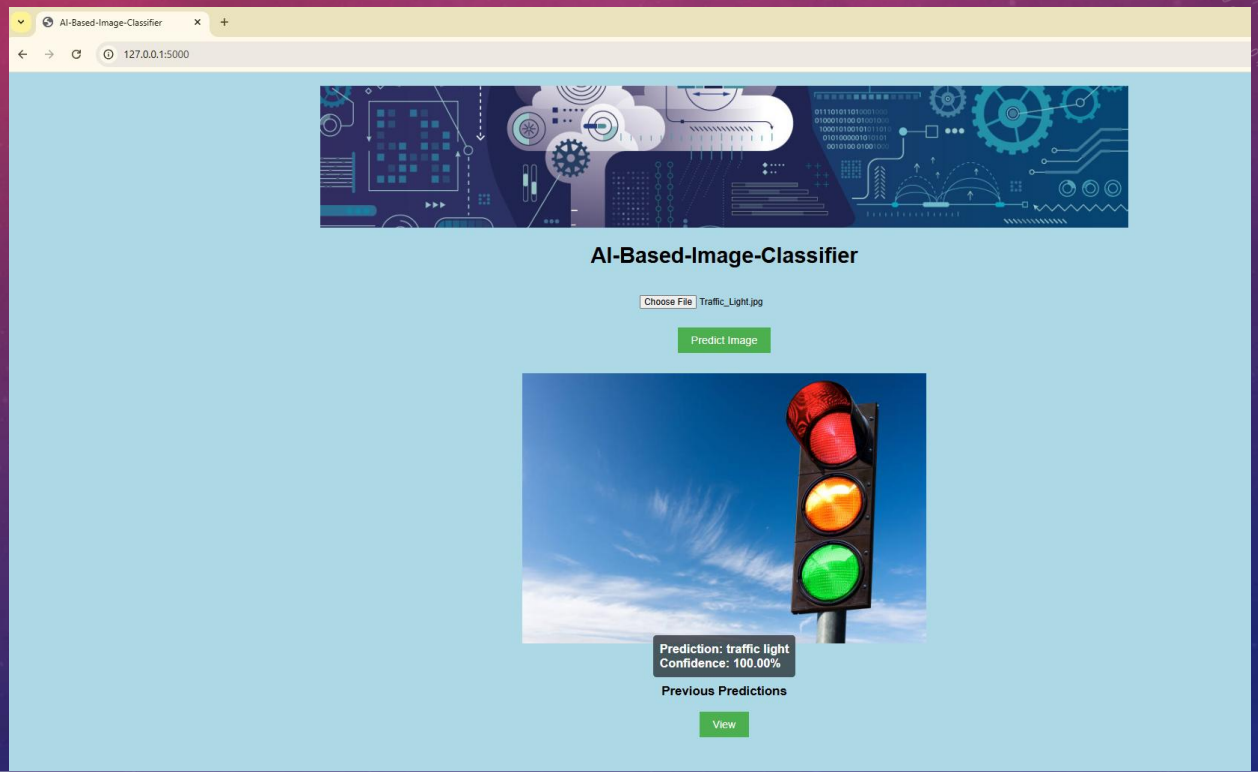
PROJECT PAGES: CHOOSE IMAGE FILE



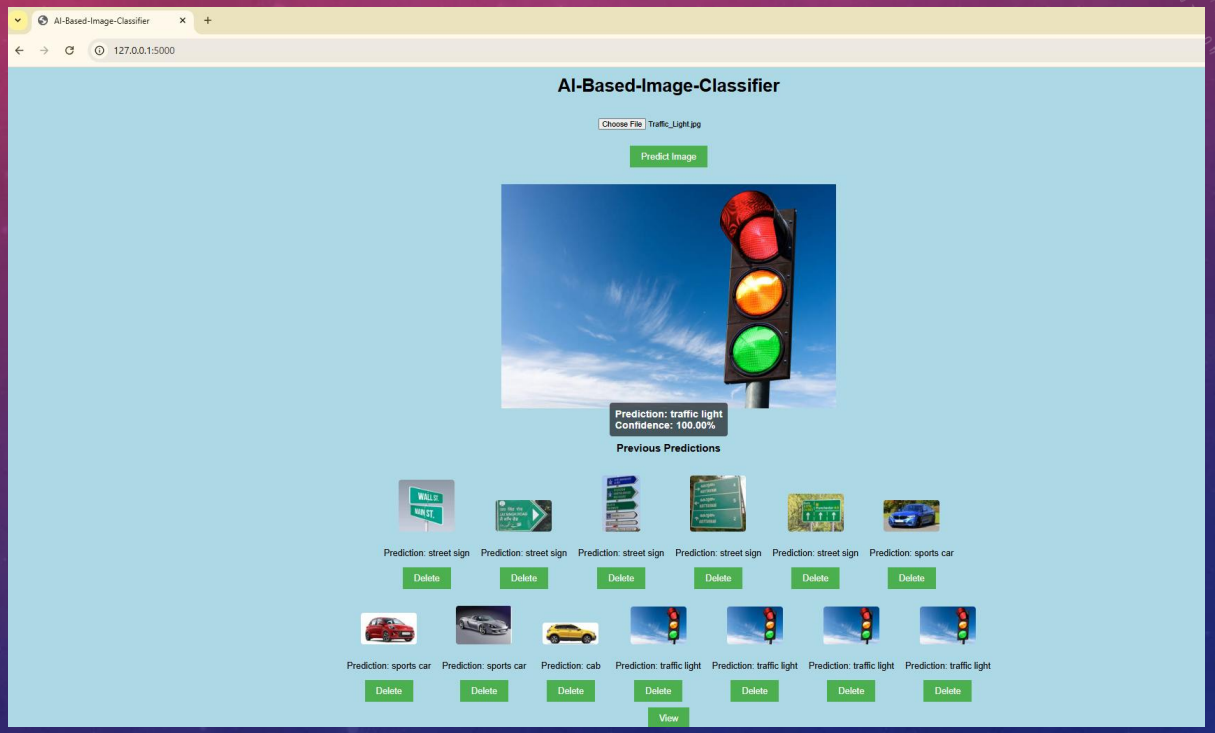
PROJECT PAGES: IMAGE UPLOAD DONE



PROJECT PAGES: PREDICTION RESULT



PROJECT PAGES: VIEW PREVIOUS PREDICTIONS:



FUTURE SCOPE

- Enhanced Prediction Models
- User Authentication and Authorization
- Image Preprocessing and Augmentation
- Batch Processing and Bulk Upload
- Advanced Analytics and Reporting
- Integration with External Services
- Mobile Application
- Improved User Interface and Experience


Q
A


Thanks