BIL 481 – Design Document

TOBB ETÜ

University of Economics & Technology

16 Mar 2025

TASK MATRIX

Section	Description	Contributor
1. System Overview	Brief project description, system architecture, and technology stack.	Kaan Arslan
2. Implementation Details	Codebase structure, key implementations, component interfaces, and visual interfaces.	Kaan Arslan
3. Use Case Support in Design	Use case selection, requirement mapping, use case design, and demo requirement.	Ali Şahin
4. Design Decisions	Technology comparisons, AI model choice, backend/frontend framework choice, external API choice, cloud hosting choice.	Ali Şahin
5. GitHub Commit Requirement	Code implementations, interfaces, and technology comparisons.	Joint Contribution

Table of Contents

1. System Overview

Brief Project Description

AI-Powered Food Ingredient Detection and Recipe Suggestion System (PicDish) is an AI-driven web platform that enables users to upload food ingredient images, detects the ingredients using image processing and deep learning techniques, and provides recipe suggestions. The system utilizes **YOLO**, fine-tuned with a custom dataset for improved accuracy, and generates recipes using **OpenAI API**.

System Architecture

The system follows a **layered architecture**, separating the AI model, backend services, and frontend components for modularity and scalability.

Technology Stack

- **Frontend:** React (for a dynamic and user-friendly interface)
- **Backend:** Django/Flask (handling API requests and AI processing)
- **Database:** No Database Used
- **AI Model:** YOLO (for food ingredient detection)
- **External APIs:** OpenAI API (for generating recipe suggestions)
- **Cloud Services:** AWS/GCP (for model hosting and storage)

2. Implementation Details

Codebase Structure

- **frontend/**: Contains all frontend components, including UI elements and styles.
- **backend/**: Handles business logic, API routes, and server-side processing.
- **model/**: Includes the YOLO model for food ingredient detection.
- **docs/**: Documentation and design files.

Key Implementations

- **Ingredient Detection:** Uses a YOLO-based model to detect food items in images.
- **Recipe Generation:** Uses OpenAI API to generate recipe suggestions based on detected ingredients.
- **User Interface:** Intuitive web interface for seamless user experience.

Component Interfaces

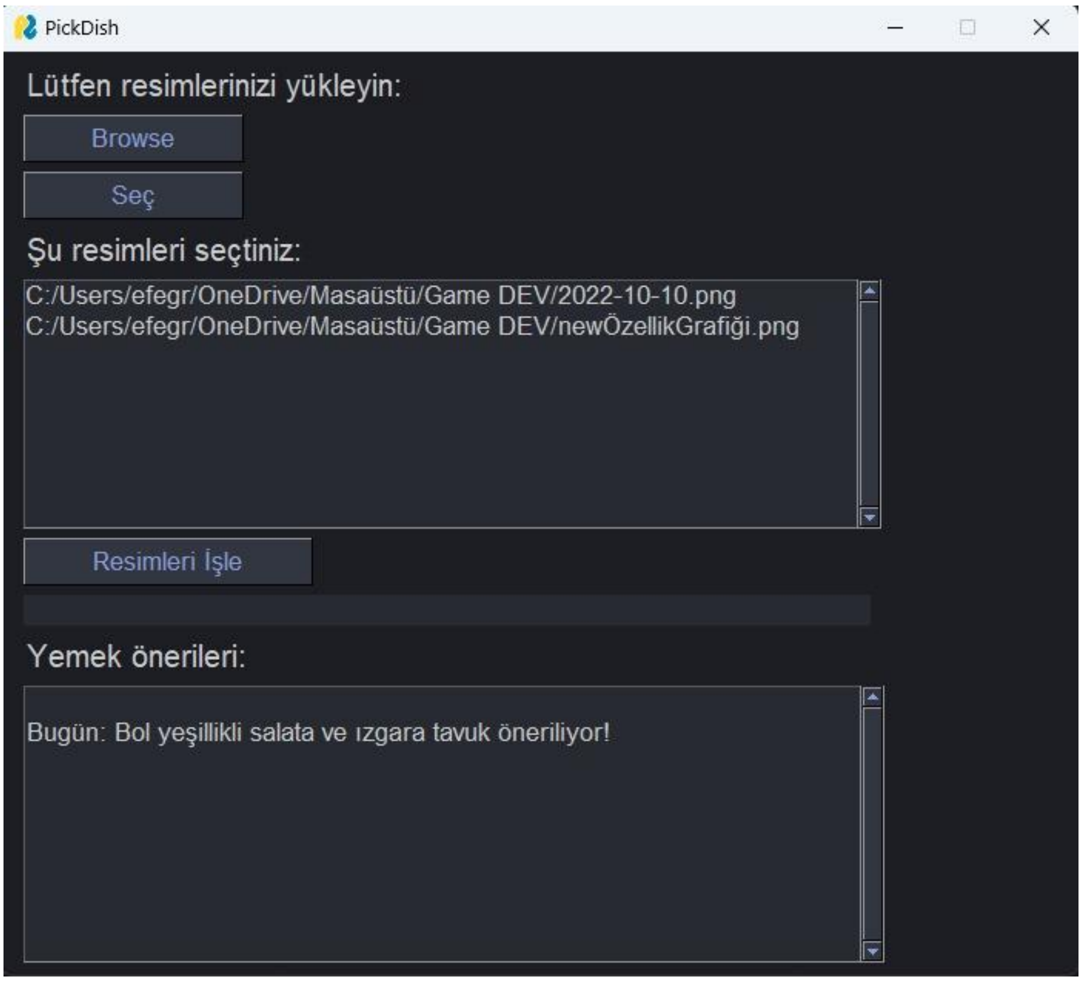
- **API Endpoints:**

```
POST /api/detect - Uploads an image and returns detected ingredients

POST /api/recipe - Generates recipes based on detected ingredients
```

Visual Interfaces

Below is a screenshot of the PickDish application's user interface. The interface allows users to upload food ingredient images, process them, and receive AI-generated meal recommendations.



C:/Users/efegr/OneDrive/Masaüstü/Game DEV/2022-10-10.png
C:/Users/efegr/OneDrive/Masaüstü/Game DEV/newÖzellikGrafiği.png

Bugün: Bol yeşillikli salata ve ızgara tavuk öneriliyor!

- **Image Upload Section:** Users can select multiple images using the "Browse" or "Seç" (Select) buttons.
- **Selected Images Display:** A text area displays the paths of selected images.
- **Process Button:** Clicking the **"Resimleri İşle" (Process Images)** button triggers the AI model to detect ingredients.
- **Meal Recommendation Display:** The detected ingredients are processed, and a meal suggestion is displayed in the lower text box.

3. Use Case Support in Design

Use Case Selection:

The project is structured around the following four key use cases:

1. **Image Upload and Ingredient Detection:** The user uploads an image of food ingredients through the web interface, and the system detects the ingredients using the fine-tuned YOLOv8 model.
2. **Recipe Generation:** Based on the detected ingredients, the system generates recipe suggestions using the OpenAI API.
3. **User Interface Interaction:** Users can view the detected ingredients and suggested recipes on a responsive web interface.
4. **Alternative Recipe Suggestions:** The system suggests alternative recipes based on the same detected ingredients to increase user options.

Use Case	**Functional Requirement**	**Description**
Image Upload and Ingredient Detection	FR1	The user should be able to upload an image.
	FR2	The system should accurately detect ingredients from the uploaded image.
Recipe Generation	FR3	The system should generate meaningful recipes based on the detected ingredients.
User Interface Interaction	FR4	The user should be able to view detected ingredients and recipes through the web interface.
Alternative Recipe Suggestions	FR5	The system should suggest alternative recipes based on the detected ingredients.

Table1: Requirement Mapping

Use Case Design:

Use Case	Data Flow	State Changes
1. Image Upload and Ingredient Detection	User → Image Upload → YOLOv8 Model → Detected Ingredients	After the image is uploaded, the YOLOv8 model processes the image and returns the detected ingredients as a JSON response.
2. Recipe Generation	Detected Ingredients → OpenAI API → Recipe Suggestions	The system sends the detected ingredients to the OpenAI API, which returns recipe suggestions as a structured text output.
3. User Interface Interaction	Recipe Suggestions → Web Interface → Display	The generated recipes are shown in a user-friendly format on the web interface.
4. Alternative Recipe Suggestions	Detected Ingredients → OpenAI API → Alternative Recipe Suggestions	The system generates multiple recipe options based on the same ingredients and displays them for user selection.

Table2: Use Case Design Table

Demo Requirement:

These four use cases must be fully implemented and demonstrated during the final project presentation:

- Image upload and ingredient detection using the fine-tuned YOLOv8 model
- Recipe generation via OpenAI API
- Display of detected ingredients and recipe suggestions on the interface
- Alternative recipe suggestions

4. Design Decisions

Technology Comparisons

Aspect	YOLOv8	TensorFlow Object Detection API	EfficientDet
Architecture	Single neural network for real-time detection.	Multiple models (SSD, Faster R-CNN) with flexible architecture.	Compound scaling for balanced accuracy and efficiency.
Performance	Fast inference with high accuracy.	Varies by model; SSD for speed, Faster R-CNN for accuracy.	High accuracy with efficient computation.
Ease of Use	Simple to implement and fine-tune.	Complex setup depending on architecture.	Easy to fine-tune with TensorFlow/Keras.
Integration	Works with TensorFlow and PyTorch.	Best for TensorFlow ecosystem.	TensorFlow-based, smooth deployment.
Community Support	Strong, active development and resources.	Large community and strong support.	Supported by TensorFlow community.

Table3: Model comparison

Technology Comparisons

1. **AI Model Choice**
   We selected **YOLOv8** for food ingredient detection due to its combination of speed and accuracy. YOLOv8's ability to process images in real time makes it well-suited for user-facing applications where fast response times are critical. The model was fine-tuned using a custom dataset to increase accuracy for specific food ingredients, ensuring more reliable detection.
   - Fast inference time (~15-20ms) ensures a responsive user experience.
   - High detection accuracy (>85%) improves recipe relevance.
   - Efficient handling of multiple objects in a single image.

2. **Backend Framework Choice**
   Flask was chosen as the backend framework because of its lightweight nature and ease of integration with AI models and APIs. Flask enables quick prototyping and minimal overhead, which is suitable for real-time AI-based applications.
   - Lightweight and fast, ideal for handling AI-related requests.
   - Simple to set up and integrate with the YOLO model and OpenAI API.
   - Facilitates easy scaling if required in the future.

3. **Frontend Framework Choice**
   React.js was selected for the frontend due to its component-based architecture and high performance. React's ability to create dynamic and responsive interfaces ensures a smooth user experience when displaying detected ingredients and recipe suggestions.
   - Fast rendering and state updates.
   - Strong community support and large ecosystem.
   - Component-based structure allows modular design and easy maintenance.

4. **No Database Used**
   Since the project does not require persistent data storage, no database was implemented. Detected ingredients and generated recipes are handled in memory during the session.
   - Reduces complexity and improves system performance.
   - Simplifies deployment and maintenance.
   - Potential to add a lightweight storage option (e.g., Redis) if needed in the future.

5. **External API Choice**
   OpenAI API was selected for recipe generation due to its ability to generate creative and contextually relevant text-based suggestions.
   - High-quality and diverse recipe generation.
   - Simple integration with Flask backend.
   - Efficient handling of multiple ingredients and edge cases.

6. **Cloud Hosting Choice**
   The system is hosted on **AWS/GCP** to ensure high availability and scalability.
   - Reliable infrastructure with automatic scaling.
   - Ensures low latency and high uptime.
   - Easy integration with Flask and YOLO models.