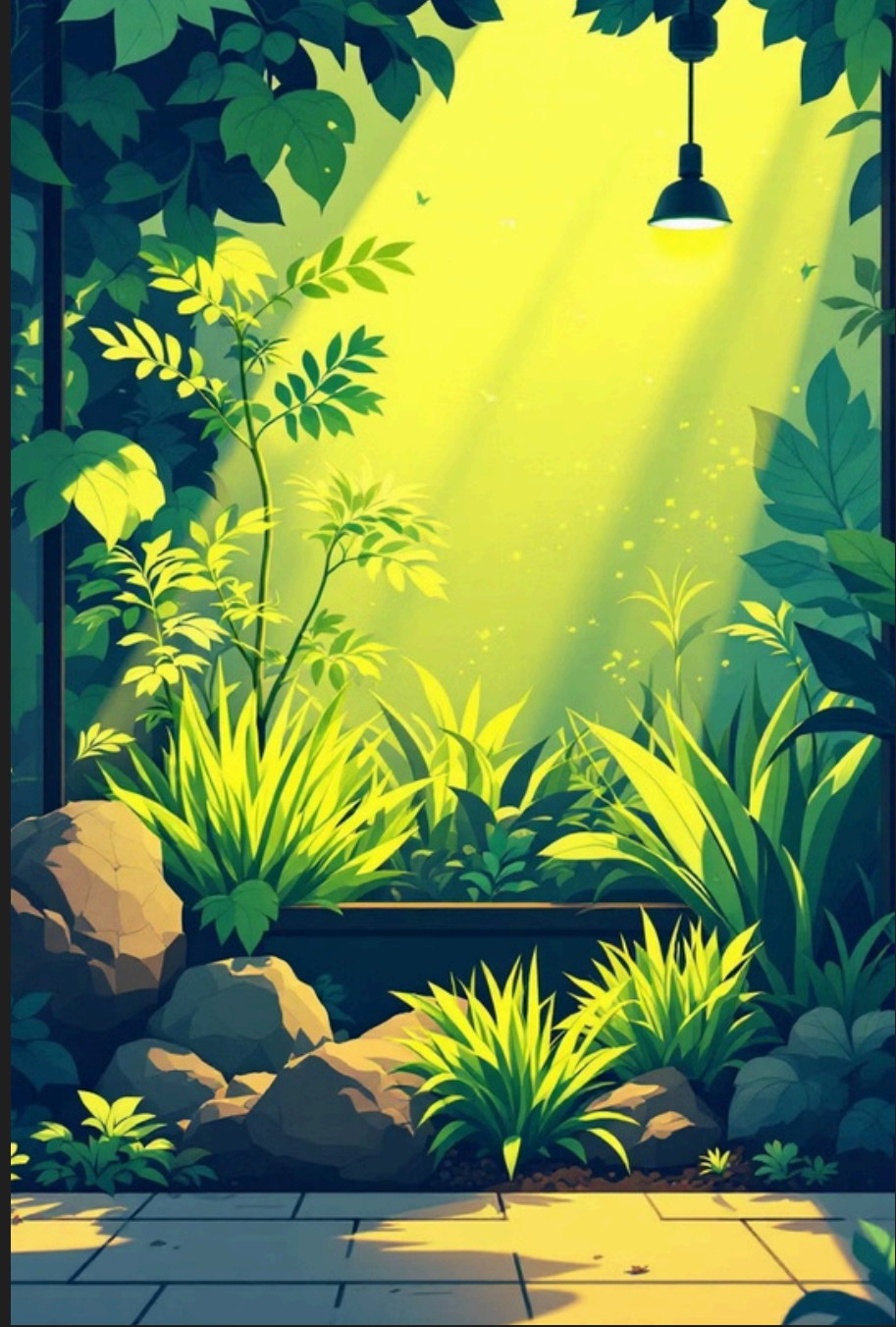
MyGarden Smart Assistant

Final Project Architecture Presentation

project no. 088

Designed by Sahar Friedman & Adar Kliger



Project Overview

MyGarden Platform

A comprehensive smart garden assistant that revolutionizes plant care through intelligent monitoring and personalized guidance.

Interactive Web Application

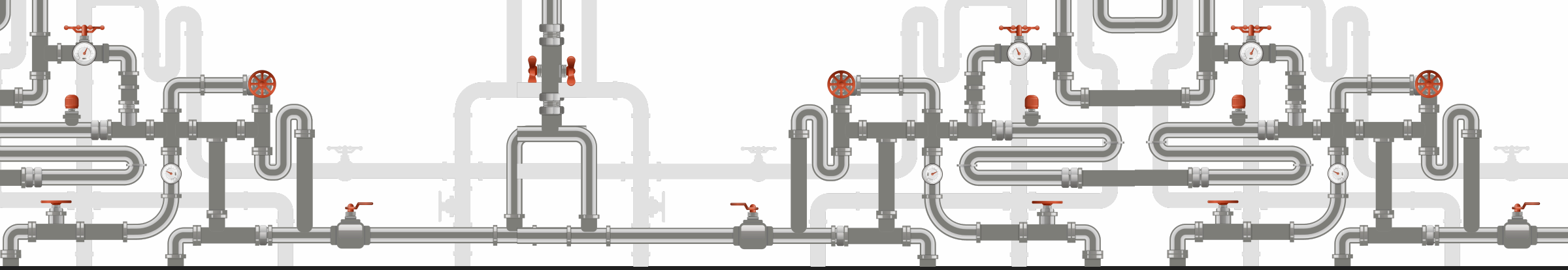
Intuitive UI for seamless plant management, photo organization, and garden area tracking with real-time updates.

AI-Powered Detection

Advanced YOLO-based plant recognition through dedicated Python micro service for accurate species identification.

Personalized Recommendations

OpenAI integration delivers tailored care advice based on weather conditions, plant history, and growth patterns.



High-Level System Architecture

Frontend Layer

React-basedresponsive web interface providing intuitive user experience and real-time garden management capabilities.

Backend API

Node.js/Express server handling business logic, user authentication, and MongoDB data persistence with RESTful endpoints.

AI Microservice

Python-powered detection service integrating YOLO models with weather APIs for intelligent plant analysis and recommendations.

All system components are containerized using Docker for consistent deployment and scalable infrastructure management.

Frontend

React-powered user interface delivering seamless garden management experience with responsive design and intuitive navigation.

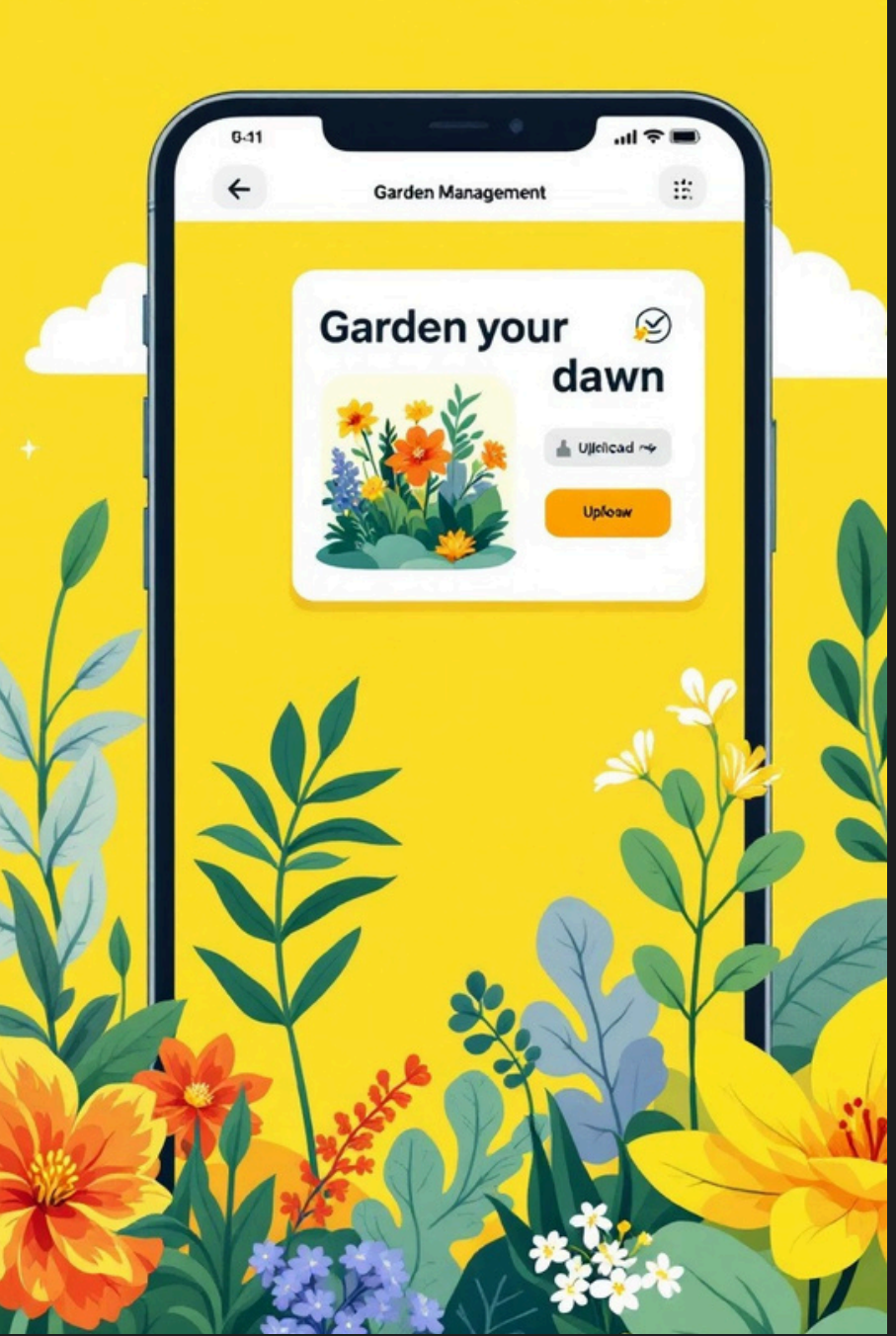
Home Dashboard

Central Command Center

The main application hub designed to engage users daily with actionable insights and garden maintenance reminders.

- Daily gardening tips to encourage regular app usage.
- Educational content about plant care best practices.
- Prioritized task list for optimal garden maintenance.
- Quick access to all major application features.
- using AI to generate new tasks and information with history saving and prompt engineering.

Strategic Role: Serves as the primary user orientation point, establishing daily engagement patterns while providing immediate value through personalized recommendations.



MyGarden Workspace

Photo Management

Upload, replace,and organize garden photos with intelligent processing for plant detection and area mapping.

Plant Identification

Interactive bounding box selection allows precise plant positioning and species recognition.

Data Organization

Comprehensive plant database with care history, and customizable garden zone management.

MyBot AI Assistant

Conversational Plant Expert

Advanced LLM-powered chatbot providing intelligent, context-aware gardening advice tailoredt oyour specific plants and conditions.

Smart Integration Tools

- Plant lookup by name or database ID.
- Real-time weather data integration.
- Persistent note-taking for plant observations.
- Historical data analysis for trend insights.

Token-efficiency

synthesized tolow butrich data, at JSON format.

Each active user will cost approximately about **1.5$ per month** in API usage.

Key Innovation: Seamlessly combines stored plant history, live weather APIs, and user interactions to deliver highly personalized, actionable gardening guidance.

Backend

Robust Node.js foundation powering secure data management and seamless frontend-microservice communication.

Backend Architecture

01

MVC Design Pattern

Clean separation of concerns ensuring maintainable, scalable code architecture with clear data flow and business logic organization.

02

Node.js Runtime

High-performanceJavaScript server environment enabling rapid development and efficient handling of concurrent user requests.

03

MongoDB Integration

NoSQL database with Mongoose ODM providing flexible schema design and robust data persistence for plant and user information.

04

Express Framework

Lightweight web server framework delivering RESTful API endpoints with middleware support for authentication and request processing.

Python Microservice Components

YOLO Plant Detection

State-of-the-art Ultralytics YOLO implementation for accurate, real-time plant species identification and bounding box generation.

Image Processing

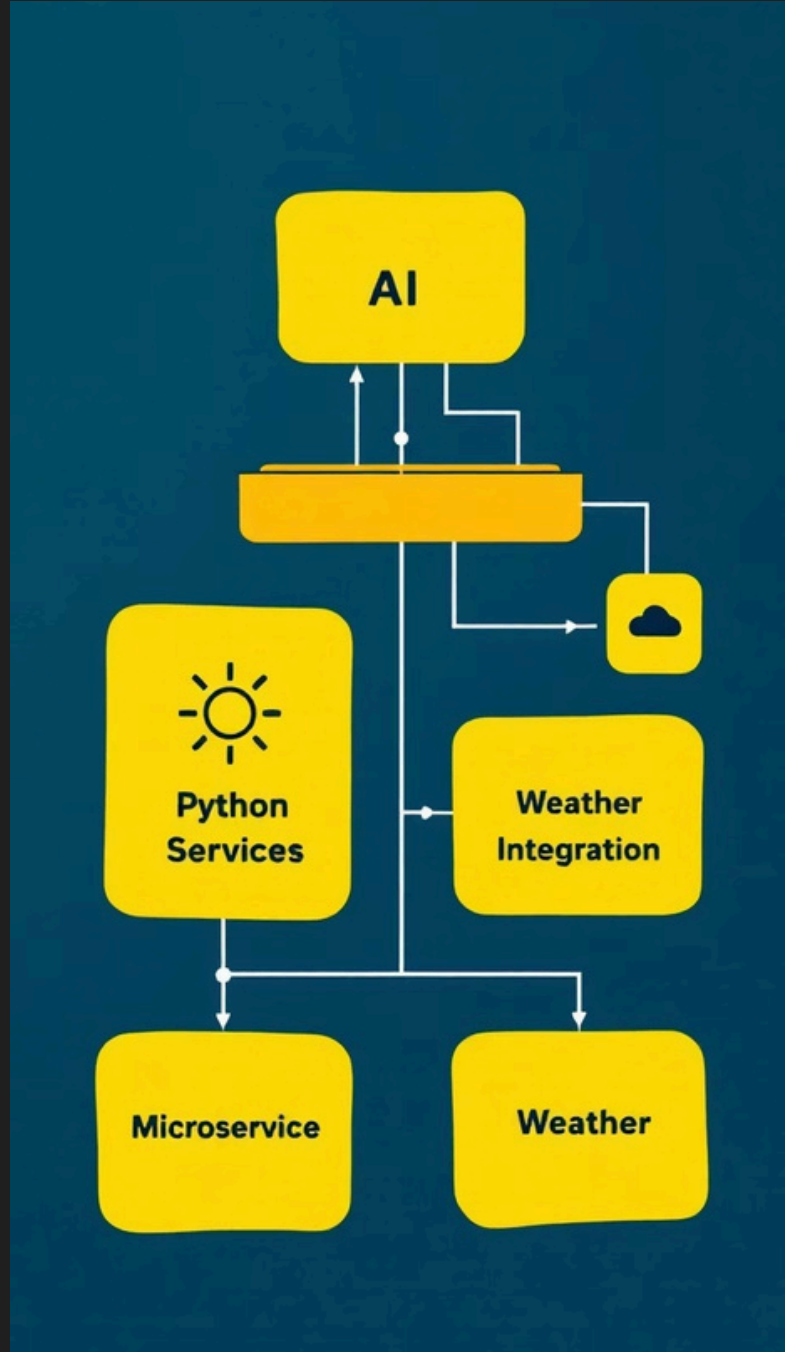
Advanced computervision pipeline using OpenCV and NumPy for image handling, preprocessing, and feature extraction.

Weather Intelligence

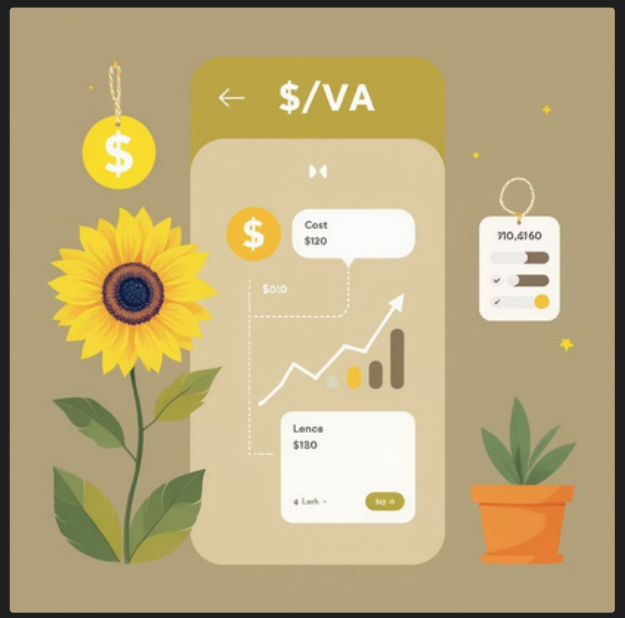
Comprehensive environmental data integration through Open-Meteo, Astral, and Geopy APIs for location-aware plant care recommendations.

Flask REST API

Lightweight, high-performance endpoints enabling seamless communication between backend services and AI processing capabilities.



monthly app cost per user



The model

using **gpt-4o-mini the price is** $0.15 **per 1M** input tokens, $0.60 per 1M output tokens.

Daily tips and weekly assignments

- **User's plant and weather data** (~3,500 input tokens)
- The model then responds with the **tip JSON** (~1,000 output tokens).
- At gpt-4o-mini prices, this works out to about **$0.0011 per tip**.
- For 30days: **~3 to 4 cents per user per month**.

Chatbot Conversations (interactive help)

- Suppose a user chats with the assistant **every day**, asking **10 messages**.
- Each chatbot turn resends the system prompt + garden data (~3,500 input tokens) and gets back a short answer (~150 output tokens).
- At gpt-4o-mini prices, that costs about **$0.007 per day per user**.
- For 30 days: **~21 cents per user per month**.

Total: ~25 cents per user per month

With GPT-4o, the app would cost about $7 per active user per month

considering all of the above, best cost-efficient is to generate daily and weekly tips with GPT-4o at ~ under one dollar, and chatBot with GPT-4o mini. at total cost of under 1.5$ per user per month

Key Innovation: Seamlessly combines stored plant history, live weather APIs, and user interactions to deliver highly personalized, actionable gardening guidance.

Libraries & Tools

Libraries

01

Python and pyServer

flask, flask-cors, ultralytics, opencv-python-headless, numpy, requests, requests-cache, retry-requests, geopy, openmeteo-requests, astral, pytz

02

Node.js Backend

express,cors,body-parser,mongoos,image-size,jsonwebtoken,multer, openai

03

Frontend (React + CSS)

react,axios,react-router,react-markdown, remark-gfm, dompurify

04

Other

Docker,MongoDB, label-studio, jupiter notebook, iNaturalist API,open Images v7 API

Picture Detection

press here for the model training code

First Model

object detection for {Tree, Flower,Plant,Cactus,Pot,Raised bed,Garden bed,Grass}. with ~900 classified picture out of ~10,000 images

Second Model

plant detection for {Basil, Geranium, Jasmin, Lavender, Lemon, Olive, Orange, Parsley, Peppermint} by processing the relevant objects {Tree, Flower, Plant}. 1,000 classified out of 1,800

Yolov8

we used YOLOv8 for our models because it's fast, simple, and efficient on low resources, while still giving accurate bounding boxes. Other detectors like Faster R-CNN are heavier and slower, and SSD is lighter but less accurate. YOLO gives the best trade-off for our dataset and hardware limits.

IoU

IoU or Intersection over Union was used for estimating the plants container {Ground, raised bed, pots}. IoU is calculated as follows: area of intersection / Area of union (union = area_box_a + area_box_b - area of intersaction)

Key Innovation: Seamlessly combines stored plant history, live weather APIs, and user interactions to deliver highly personalized, actionable gardening guidance.

model precision

Second Model

Precision: **86.3%**

Recall: **92.7%**

mAP50(Mean average with threshold<0.5): 92.9%

mAP50-95(50<t<95) : 84.2%

in conclusion: The model is reliable, and detect correctly the species, but in practice needs more plant data

First Model

Precision: **75.1%**

Recall: **52.7%**

mAP50(Mean average with threshold<0.5): 59.2%

mAP50-95(50<t<95) : 36.5%

in conclusion: The model is reliable, but it does not catch everything. which works well for our needs especially with smaller relevant database.

Key Innovation: Seamlessly combines stored plant history, live weather APIs, and user interactions to deliver highly personalized, actionable gardening guidance.

Application Setup & conclusions

Docker setup

01

Docker built with three images: frontend, backend, pyserver.

02

Run by using “docker compose up -d -build” At the repo folder.

03

Default ports: frontend - :3000, backend - :12345, pyserver- :2021

04

Other

a working MongoDB compass listening on default port 27017 - more explaining on the repository on github

Next Steps

01

Expanding plant database and care history.

02

Improving photo replacement & remapping logic.

03

Optimizing Docker size and independence of external libraries.

04

Extending chatbot personalization.

Links

01

github repository

02

jira

03

Video Link