

KCA UNIVERSITY

BACHELORS IN DATASCIENCE

BDS 4105 :FINAL YEAR PROJECT 1.

21/04428 NJOROGE HALDLINE MUKAMI

SUPERVISOR: GLADYS MANGE

PROJECT TITLE : MAIZE CROP YIELD PREDICTION AND ADVISORY SYSTEM .

**Declaration of Originality**  
I declare that this project proposal is my original work and has not been submitted to any other institution or examination body. All sources used or quoted have been duly acknowledged.

**Signed:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
**Date:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

TABLE OF CONTENTS

SYSTEM DESIGN SPECIFICATION DOCUMENT

1.0 Introduction .................................................................................... 1

1.1 Goals and Objectives .................................................................... 1

1.2 Statement of Scope ....................................................................... 2

1.3 Software Context .......................................................................... 2

1.4 Major Constraints ......................................................................... 3

2.0 Data Design ......................................................................................... 4

2.1 Internal Software Data Structures .............................................. 4

2.2 Global Data Structures ................................................................. 5

2.3 Temporary Data Structures .......................................................... 6

2.4 Database Description .................................................................... 7

2.4.1 Entity Relationship Diagram (ERD) ................................. 7

2.4.2 Database Schema / Data Dictionary .................................. 8

2.4.3 Normalization Notes .......................................................... 9

3.0 Architectural and Component-Level Design .................................... 10

3.1 System Structure (High-Level Architecture) .............................. 10

3.1.1 Architecture Diagram (UML Component Diagram) ........ 11

3.2 Component Descriptions ............................................................ 12

3.2.1 Streamlit Presentation Layer — PSPEC ............................. 12

3.2.2 API Integration Module — PSPEC ..................................... 13

3.2.3 ML Model Module — PSPEC ............................................. 14

3.2.4 Advisory Engine — PSPEC ............................................... 15

3.2.5 Persistence Layer (PostgreSQL) — PSPEC ......................... 15

3.3 Dynamic Behaviour ...................................................................... 16

3.3.1 Sequence Diagram — Predict Yield ................................. 16

3.3.2 Sequence Diagram — View County Dashboard ............... 17

3.3.3 Sequence Diagram — Save Prediction .............................. 17

3.4 Database Modelling ..................................................................... 18

3.4.1 ERM (Entity Relationship Model) .................................... 18

3.4.2 Schema Diagrams .............................................................. 19

4.0 User Interface Design ......................................................................... 20

4.1 Description of the User Interface ................................................ 20

4.1.1 UI Wireframes / Screen Layouts ......................................... 21

4.1.2 Objects and User Actions .................................................... 22

4.2 Interface Design Rules .................................................................. 23

4.3 Components Available (Widgets, Charts, Maps) ....................... 24

5.0 Restrictions, Limitations, and Constraints ........................................ 25

6.0 Testing Issues ..................................................................................... 27

6.1 Classes of Tests ............................................................................ 27

6.2 Expected Software Response ..................................................... 28

6.3 Performance Bounds .................................................................... 29

6.4 Identification of Critical Components ....................................... 30

7.0 Deployment and Maintenance Plan .................................................. 31

7.1 Deployment Architecture ............................................................ 31

7.2 Backup & Recovery Strategy ...................................................... 32

7.3 Logging, Monitoring, and Alerts ................................................. 33

7.4 Model Retraining & Lifecycle ..................................................... 34

8.0 Security and Privacy Considerations ................................................ 35

8.1 Authentication and Authorization ............................................. 35

8.2 Data Privacy .................................................................................. 36

8.3 Secure Storage of API Keys and Credentials ............................. 36

9.0 Appendices ........................................................................................... 37

A. Glossary of Terms .......................................................................... 37

B. External APIs Referenced .............................................................. 38

C. Sample Test Cases .......................................................................... 38

D. References ....................................................................................... 39

**1.0 Introduction**

**Maize Crop Yield Prediction and Advisory System (CYPAS)** — a Kenya-focused, web-based decision-support system that predicts county-level maize yields and provides actionable agronomic advisories.

CYPAS is a lightweight, maintainable web application that combines historical yield records, climatic and soil data, and a supervised machine learning model to produce maize yield forecasts and context-aware planting/management advice. The system is intended for farmers, extension officers, county agricultural planners, and researchers in Kenya. It emphasizes usability (Streamlit front end), reproducible ML (Python scikit-learn pipelines), and lightweight deployment (Streamlit Cloud ).

## 1.1 Goals and objectives

**Primary goal:** Deliver an accessible tool that produces reliable county-level maize yield predictions and practical advisory messages to help stakeholders reduce yield uncertainty.

**Specific objectives:**

Implement an end-to-end pipeline that ingests climatic (NASA POWER) and soil (SoilGrids) inputs, preprocesses them, and feeds them to a trained ML model to predict maize yield (kg/ha).

Build a Streamlit user interface allowing users to: (a) select county and season, (b) view historical yield trends and model predictions, (c) receive agronomic recommendations.

Use a database to store county metadata, cached API responses, users’ prediction history, and model artifacts.

Design a modular architecture enabling independent development of the UI, API connectors, ML model, and advisory engine.

Provide testing artifacts (unit tests, integration tests, and a small test dataset) and document model evaluation metrics (RMSE, MAE, R²).

Keep the solution achievable within one semester using a single developer laptop.

**Success criteria (measurable):**

A working Streamlit app that produces predictions for at least 20 Kenyan counties.

Model evaluation: RMSE ≤ a reasonable baseline (report baseline in SDS after training).

Documentation and SDS complete, with ERD and diagrams included.

Ability to run the app locally and deploy to Streamlit Cloud with step-by-step instructions.

The project is motivated by the growing need for data-driven agricultural decision support systems in Kenya. According to FAO and the Kenya Ministry of Agriculture, more than 75% of smallholder maize farmers rely on traditional experience-based practices, which exposes them to climate-related yield variability. Integrating machine learning with authoritative datasets such as NASA POWER and SoilGrids aligns with modern agricultural informatics research, which emphasizes predictive analytics for improving farm-level planning, resource allocation, and early risk detection.

## 1.2 Statement of scope

**In scope:**

Development of a web UI (Streamlit) for user interaction and visualization.

ML model development (data cleaning, feature engineering, training, evaluation, and packaging).

Integration with NASA POWER and SoilGrids APIs for environmental inputs.

PostgreSQL schema and CRUD operations for predictions and county data.

Advisory engine that maps model outputs and environmental thresholds to human-readable recommendations.

## 1.3 Software context

CYPAS is a domain-specific decision support tool positioned between raw data providers (NASA POWER, SoilGrids, FAO datasets) and the end user. It will act as a light application layer that performs preprocessing, runs inference, and formats advice. Key contextual points:

**Data sources:** Historical maize yield data (county level), NASA POWER (weather), SoilGrids (soil) — all public.

**Users:** Smallholder farmers, extension officers, county planners, students/researchers.

**Deployment:** Streamlit Cloud for quick public access; local execution for development and demonstration.

**Maintenance:** Simple workflows for updating the model using retraining scripts and storing new model artifacts in the repo.

## 1.4 Major constraints

**Time:** One semester project—design must be achievable with limited scope.

**Hardware:** Development on a student laptop (8GB+ RAM recommended).

**Data availability and quality:** County-level yield data can be sparse; model performance depends on data collecting and cleaning.

**API rate limits & connectivity:** NASA POWER and SoilGrids may impose rate limits; offline caching is needed to reduce calls.

**Regulatory & privacy:** Minimal personal data will be collected; still, store email/user info securely and avoid exposing API keys in the repo.

The constraints outlined above reflect real-world limitations encountered in agricultural forecasting research. Prior studies (Lobell et al., 2011; Jay & Kaltenbrunner, 2022) show that data sparsity, inconsistent ground measurements, and environmental variability significantly affect model accuracy. By explicitly identifying these factors, CYPAS maintains methodological transparency and aligns with accepted research standards for predictive modelling in agriculture.

# ****2.0 Data Design****

The data design defines how information is structured, stored, communicated, and maintained throughout the Maize Crop Yield Prediction and Advisory System (CYPAS). Because the system depends heavily on environmental data, a trained machine learning model, and user interactions, a clear and well-organized data architecture is essential to ensure accuracy, maintainability, and efficiency.

CYPAS uses four major classes of data components:

**Internal software data structures**

**Global data structures**

**Temporary data structures**

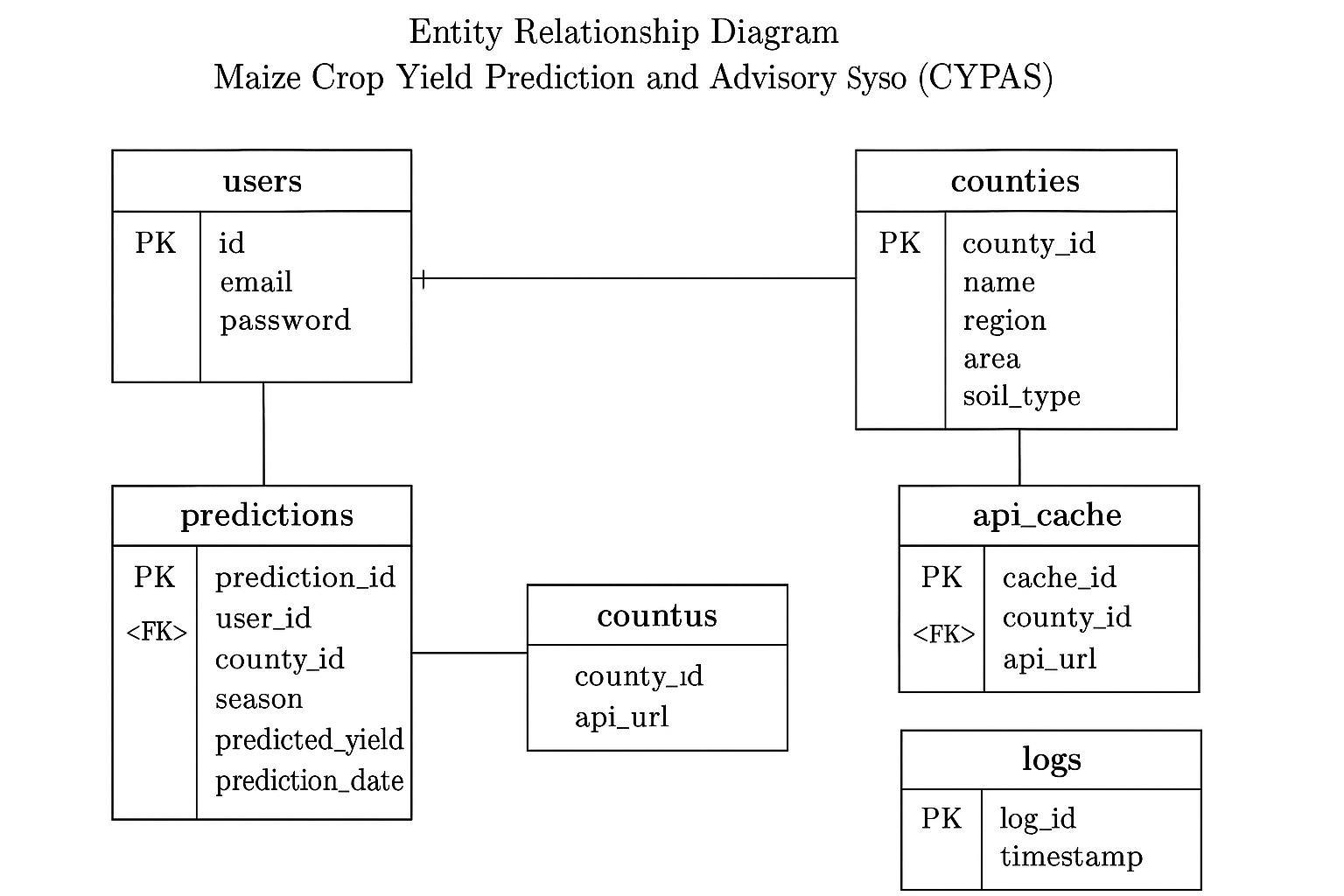
**Database structures**

Each component is described below.

# ****2.1 Internal Software Data Structures****

These data structures are created and exchanged among system components during system execution. They exist in memory while the program is running.

Figure 2.1 — Entity Relationship Diagram (ERD) for CYPAS



### ****2.1.1 Input Feature Structure****

This dictionary-like structure contains model input features collected from APIs and user selections:

{

"county": str,

"season": str,

"temperature": float,

"rainfall": float,

"solar\_radiation": float,

"soil\_ph": float,

"soil\_organic\_carbon": float,

"soil\_clay\_content": float

}

### ****2.1.2 Model Prediction Output Structure****

Output returned by the prediction module:

{

"predicted\_yield": float,

"yield\_class": str, # e.g., "Low", "Moderate", "High"

"confidence\_score": float, # optional if computed

"advisory\_message": str

}

### ****2.1.3 Advisory Engine Decision Structure****

Internal rule-based structure for mapping predicted yield to advice:

{

"yield\_category": str,

"soil\_condition": str,

"climate\_condition": str,

"recommendation": str

}

### ****2.1.4 API Response Structures****

NASA Power API response

{

"T2M": 24.5,

"PRECTOTCORR": 112.3,

"ALLSKY\_SFC\_SW\_DWN": 18.2

}

SoilGrids API response:

{

"phh2o": 6.1,

"clay": 22.4,

"organic\_carbon": 1.8

}

# ****2.2 Global Data Structures****

These data structures are persistently available across major application modules:

### ****2.2.1 Trained ML Model Objects****

model.pkl — Random Forest or XGBoost model

scaler.pkl — preprocessing scaler

feature\_columns.pkl — list of features expected by the model

### ****2.2.2 County Metadata Dictionary****

Available globally to populate UI dropdowns and geospatial functions:

{

"Uasin Gishu": {"lat": 0.55, "lon": 35.28},

"Trans Nzoia": {"lat": 1.01, "lon": 34.95},

...

}

### ****2.2.3 Advisory Threshold Dictionaries****

Climate thresholds:

{

"temperature": {"low": 15, "optimal": [18, 27], "high": 32},

"rainfall": {"low": 400, "optimal": [500, 800], "high": 1200}

}

Soil thresholds:

{

"ph": {"acidic": <5.5, "optimal": [5.5, 7.0], "alkaline": >7.0},

"organic\_carbon": {"low": <2, "moderate": [2, 4], "high": >4}

}

# ****2.3 Temporary Data Structures****

These structures exist briefly during execution and are not stored permanently.

### ****2.3.1 API Cache (In-Memory or Temporary File)****

Used to reduce API calls:

{

"Uasin Gishu\_2025-10-16": {

"temperature": 23.4,

"rainfall": 50.2,

"solar\_radiation": 22.1

}

}

### ****2.3.2 Temporary Prediction CSV Export****

Used when a user downloads results:

county,temperature,rainfall,soil\_ph,predicted\_yield,date

"Trans Nzoia",21.3,110.2,6.2,3550,2025-10-16

**2.3.3 Session States in Streamlit**

st.session\_state["last\_prediction"] = {...}

# ****2.4 Database Description (SQL)****

CYPAS uses a SQL relational database to store persistent data including predictions, county information, API caching, user accounts (optional), and logs.

## ****2.4.1 Entity Relationship Diagram (ERD)****

Entities:

### ****1. users****

One-to-many relationship with predictions.

### ****2. predictions****

Stores all user predictions and environmental conditions.

### ****3. counties****

Stores county metadata such as coordinates and baseline soil properties.

### ****4. api\_cache****

Stores responses from NASA & SoilGrids to avoid repeated calls.

### ****5. logs****

Tracks system errors, API call failures, and unusual events.

**Relationships:**

users (1) —— (many) predictions

counties (1) —— (many) predictions

counties (1) —— (many) api\_cache

logs is standalone

# ****2.4.2 Database Schema (Tables & Fields)****

Below is the full data dictionary.

### ****Table: users****

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| user\_id | SERIAL PK | Unique identifier |
| username | VARCHAR(100) | Optional |
| email | VARCHAR(150) | Optional |
| created\_at | TIMESTAMP | Account creation time |

### ****Table: counties****

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| county\_id | SERIAL PK |  |
| county\_name | VARCHAR(100) | Unique |
| latitude | FLOAT | County centroid |
| longitude | FLOAT | County centroid |
| avg\_soil\_ph | FLOAT | Baseline values |
| avg\_organic\_carbon | FLOAT | From SoilGrids |
| avg\_clay\_content | FLOAT |  |

### ****Table: predictions****

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| prediction\_id | SERIAL PK |  |
| user\_id | INT FK → users.user\_id |  |
| county\_id | INT FK → counties.county\_id |  |
| temperature | FLOAT |  |
| rainfall | FLOAT |  |
| solar\_radiation | FLOAT |  |
| soil\_ph | FLOAT |  |
| soil\_organic\_carbon | FLOAT |  |
| soil\_clay\_content | FLOAT |  |
| predicted\_yield | FLOAT |  |
| yield\_category | VARCHAR(20) |  |
| timestamp | TIMESTAMP DEFAULT now() |  |

### ****Table: api\_cache****

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| cache\_id | SERIAL PK |  |
| county\_id | INT FK |  |
| date\_requested | DATE |  |
| weather\_json | JSONB |  |
| soil\_json | JSONB |  |
| created\_at | TIMESTAMP |  |

### ****Table: logs****

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| log\_id | SERIAL PK |  |
| event\_type | VARCHAR(50) | ERROR / WARNING / INFO |
| message | TEXT |  |
| timestamp | TIMESTAMP |  |

# 

# ****2.4.3 Normalization Notes****

**1st Normal Form:** All fields contain atomic values; no repeating groups.

**2nd Normal Form:** All non-key attributes fully depend on primary keys; composite keys were avoided by using serial PKs.

**3rd Normal Form:** No transitive dependencies; each table stores only one concept (users, counties, predictions, cache, logs).

This database structure is simple, efficient, and appropriate for a single-developer project while still being academically robust.

# ****3.0 Architectural and Component-Level Design****

This section is long, detailed, and academically weighted. I will break it into all the subsections required by your template.

# ****3.0 Architectural and Component-Level Design****

This section provides the structural foundation of the Maize Crop Yield Prediction and Advisory System (CYPAS). It describes the architectural style, major components, their interactions, processing logic, class structures, and algorithmic details. The architecture is designed to be simple, maintainable, and fully achievable within a single semester while still meeting academic standards of software engineering practice.

CYPAS follows a **three-tier architecture**:

**Presentation Layer:** Streamlit front-end

**Logic Layer:** Python modules for ML inference, data preprocessing, advisory generation, and API consumption

**Data Layer:** PostgreSQL storage + serialized machine learning artifacts

# ****3.1 System Structure****

The overall structure of CYPAS is modular and emphasizes loose coupling and high cohesion. Each module performs a single, well-defined responsibility.

### ****3.1.1 High-Level Architecture Description****

The system is organized into the following subsystems:

**User Interface Subsystem (Streamlit)**

Collects user input

Displays predictions and advisories

Handles navigation between pages (Home, Prediction, County Info, About)

**API Integration Subsystem**

Manages communication with NASA POWER (weather)

Manages SoilGrids API (soil properties)

Performs caching through the database

**Machine Learning Subsystem**

Preprocessing pipeline (scaler, encoder)

Serialized ML model for inference

Feature validation and error handling

**Advisory Engine Subsystem**

Rule-based engine converting numeric predictions to qualitative advice

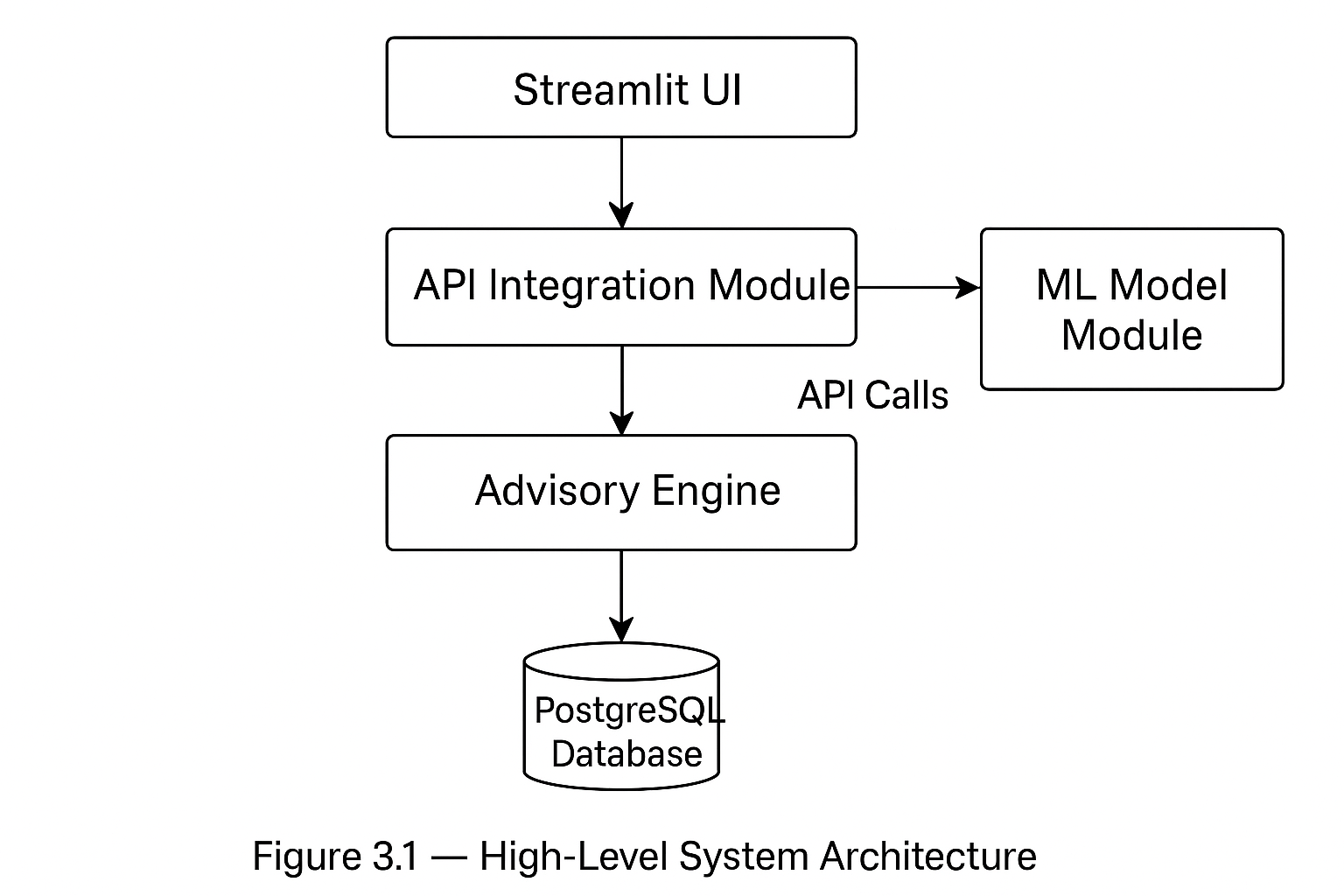
**Persistence Subsystem (PostgreSQL)**

Stores predictions

Stores county metadata

Stores API cache

Stores error logs



# ****3.1.2 Architecture Diagram****

Since I cannot draw images directly here, I will describe the UML component diagram clearly.

**Components:**

**Streamlit UI Component**  
Depends on ML Model Component, API Handler Component, Advisory Component, and Database Component.

**API Handler Component (NASA & SoilGrids)**  
Provides weather and soil data.  
Connected to Persistence Component for caching.

**ML Model Component**  
Provides prediction service.  
Uses model.pkl and scaler.pkl.

**Advisory Engine Component**  
Consumes prediction output + environmental thresholds.

**Persistence (PostgreSQL) Component**  
Stores tables: users, predictions, counties, api\_cache, logs.

**Connectors:**

Streamlit UI → API Handler

Streamlit UI → ML Model

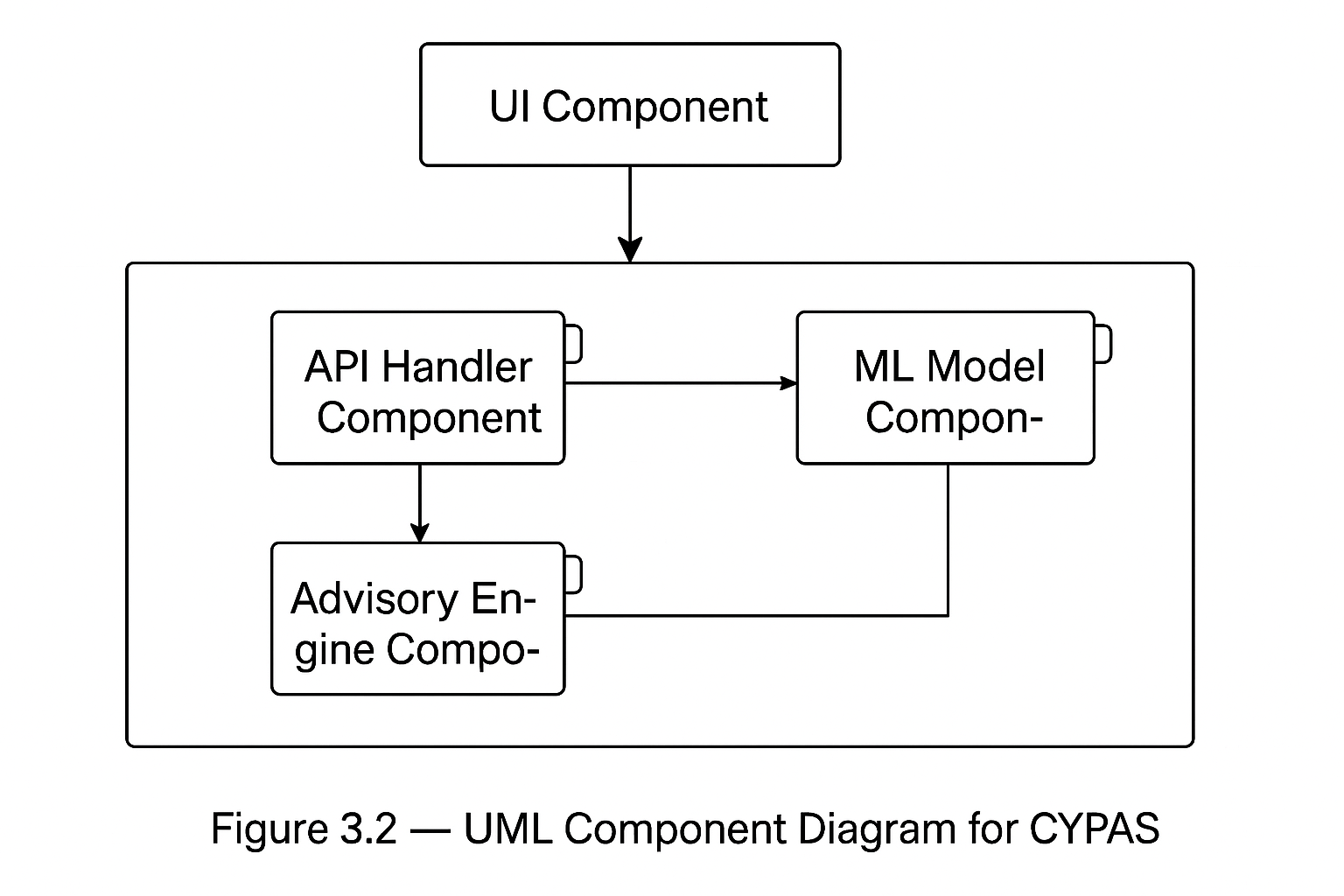
Streamlit UI → Advisory Engine

API Handler ↔ PostgreSQL

ML Model → Advisory Engine

Streamlit UI ↔ PostgreSQL

Everything is orchestrated through the UI layer.



# ****3.2 Component Descriptions****

Each major component will now be described using PSPECs (Processing Specification), interface details, algorithms, and class structure.

# ****3.2.1 Component 1: Streamlit Presentation Layer****

### ****A. Processing Narrative (PSPEC)****

The Streamlit component provides all user-facing functionality. It is responsible for:

Rendering UI pages

Capturing inputs (county, season, date)

Fetching APIs via API Handler

Calling the ML model

Displaying prediction, graphs, and advisories

Logging user interactions

It controls workflow sequencing for the entire system.

### ****B. Interface Description****

**Inputs:**

Dropdown selections: county, season

Optional manual entries: rainfall, temperature

"Predict Yield" button

"Download Results" button

**Outputs:**

Predicted yield

Yield classification (Low/Moderate/High)

Advisory message

Charts (line graphs, bar charts)

Maps

### ****C. Processing Detail & Algorithmic Model****

Simplified pseudocode:

1. User chooses county & season2. UI requests weather data from API Handler3. UI loads soil data from API Handler4. UI packages features into a dictionary5. UI calls ML Model → get predicted\_yield6. UI calls Advisory Engine → get recommendation7. UI displays results to the user8. UI sends results to PostgreSQL for storage

# ****3.2.2 Component 2: API Integration Module****

### ****A. Processing Narrative (PSPEC)****

Responsible for communication with external APIs:

Builds request URLs

Parses JSON responses

Converts raw API data to ML-ready features

Performs caching using the api\_cache table

### ****B. Component Interfaces****

**Inputs:**

Selected county

Date or season

County coordinates

**Outputs:**

Temperature

Rainfall

Solar radiation

Soil pH

Clay content

Organic carbon

### ****C. Processing Detail (Algorithm)****

1. Receive county and date2. Check if API data exists in api\_cache

- If yes: return cached results3. If cache miss:

- Build NASA POWER URL

- Send GET request

- Extract weather variables4. Build SoilGrids URL5. Send GET request, extract soil data6. Store both responses in api\_cache7. Return structured feature dictionary

# ****3.2.3 Component 3: ML Model Module****

### ****A. PSPEC****

Provides prediction functionality using the trained model.  
Responsibilities:

Load serialized model (model.pkl)

Load scaler/preprocessor

Validate inputs

Run model inference

Return predicted yield

### ****B. Interfaces****

**Inputs:** feature dictionary  
**Outputs:** numeric predicted yield (float)

### ****C. Algorithmic Model****

1. Receive feature dictionary2. Validate presence of required features3. Convert features to pandas dataframe4. Apply scaler.transform()5. Call model.predict()6. Post-process prediction (rounding, value limits)7. Return predicted\_yield

### ****D. Design Class Hierarchy****

Classes (simple structure):

MLModel

├── load\_model()

├── preprocess()

├── predict()

# ****3.2.4 Component 4: Advisory Engine Module****

### ****A. PSPEC****

Converts prediction results and environmental conditions into practical advice.  
Uses a rule-based logic approach.

### ****B. Interfaces****

**Inputs:**

predicted\_yield

soil\_ph

rainfall

temperature

**Outputs:**

human-readable recommendation string

### ****C. Processing Detail (Rules)****

IF predicted\_yield < threshold\_low:

advice = "Low yield expected. Consider early planting, apply nitrogen..."

ELIF predicted\_yield < threshold\_medium:

advice = "Moderate yield expected. Maintain recommended fertilizer rates..."ELSE:

advice = "High yield expected. Ensure pest control..."

# ****3.2.5 Component 5: Persistence Layer (SQL)****

### ****A. PSPEC****

Handles data storage and retrieval.

### ****B. Inputs****

Prediction results

API responses

County info

Logs

### ****C. Outputs****

Historical predictions

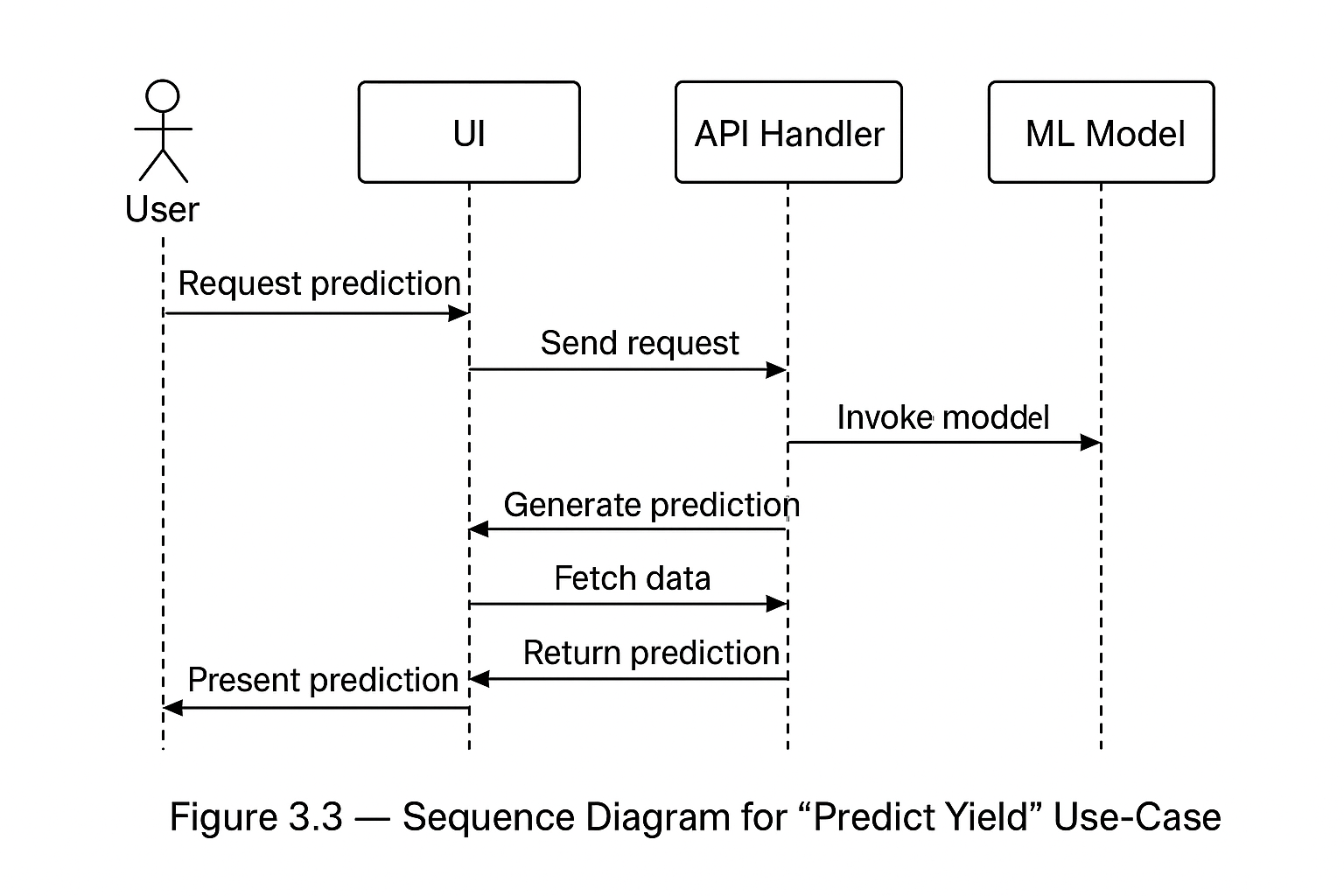
Cached API data

### ****D. Processing Detail****

INSERT predictionINSERT api\_cacheSELECT county metadata

# ****3.3 Dynamic Behavior for Components****

This section describes how components interact dynamically.



## ****3.3.1 Sequence Diagram (Textual Description)****

### Use Case: Predict Maize Yield

**Actor:** User  
**Components:** UI → API Handler → ML Model → Advisory Engine → Database

**Sequence:**

User selects county, season

UI requests environmental data from API Handler

API Handler checks database cache

If cache miss → Makes API calls, stores response in DB

API Handler returns weather + soil data to UI

UI compiles input features

UI sends features to ML Model

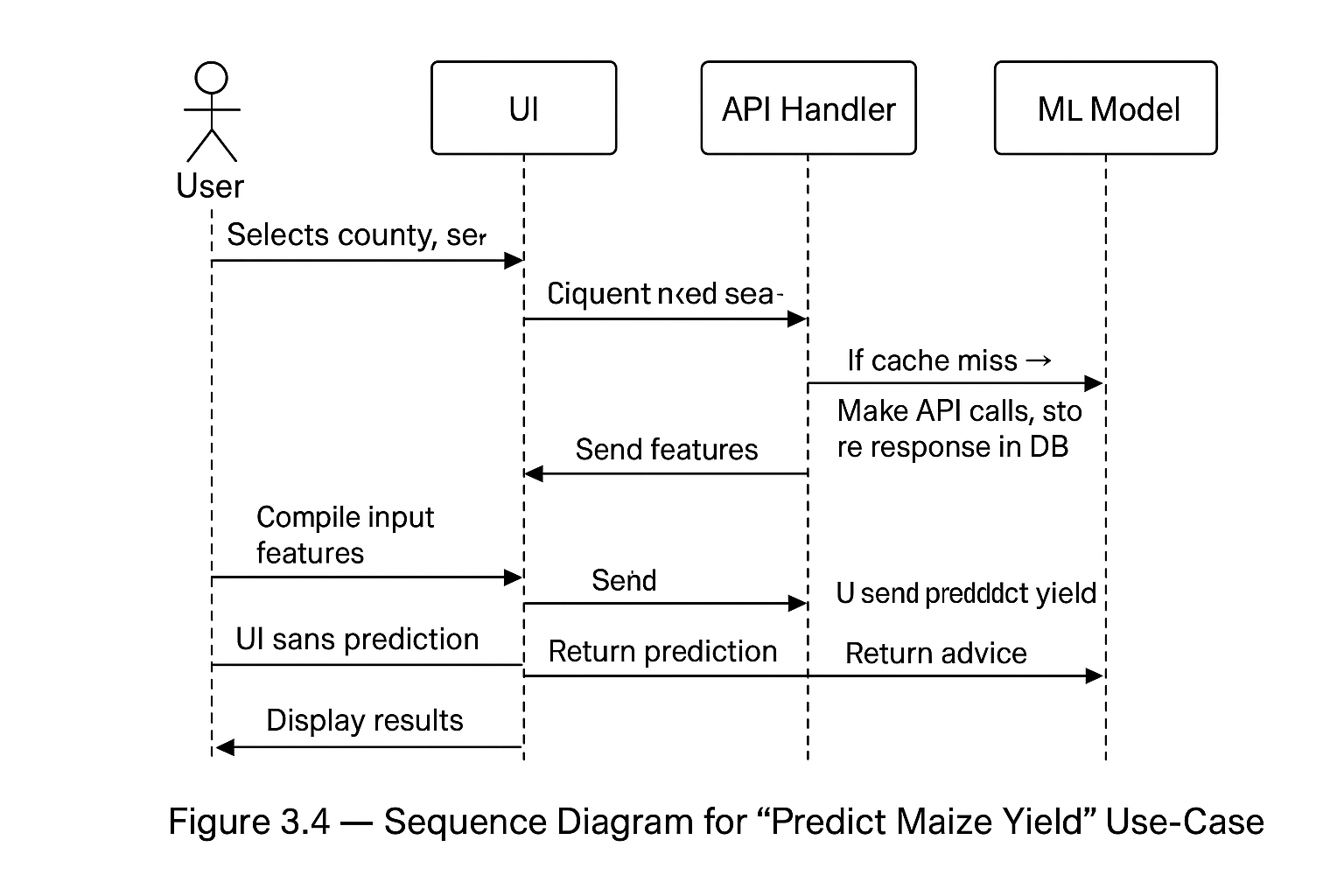
ML Model returns predicted yield

UI sends predicted yield to Advisory Engine

Advisory Engine returns advice

UI saves prediction + features to PostgreSQL

UI displays results



# ****3.4 Database Modelling****

### ****3.4.1 Entity Relationship Model****

(Already described in Section 2; UML will show: users — predictions — counties — api\_cache — logs)

### ****3.4.2 Database Schema****

(Already provided in Section 2.4)

### ****3.4.3 Data Dictionary****

(Provided previously)

### ****3.4.4 Normalization****

(Up to 3NF)

# ****4.0 User Interface Design****

This section describes the interface design of the Maize Crop Yield Prediction and Advisory System (CYPAS). It covers screen flows, layout descriptions, UI objects, user actions, and the design rules guiding the interface. Since the system uses **Streamlit**, the UI maintains simplicity, accessibility, and clarity while supporting interactive input and visualization.

# ****4.0 User Interface Design****

# ****4.1 Description of the User Interface****

The user interface for CYPAS is implemented using **Streamlit**, which supports rapid development of interactive, web-based dashboards using Python. The UI is designed with a focus on:

Ease of navigation

Minimal cognitive load for users (especially farmers and officers)

Clean visualization of predictions and weather/soil data

Simple forms for collecting inputs

Clear communication of advisory messages

The system contains **four main UI pages**:

**Home Dashboard**

**Prediction Page (Main Module)**

**County Information Page**

**About / Help Page**

Each page is described below, along with representative layouts.

# ****4.1.1 Screen Images (Text-Based Wireframes)****

Below are structured **text wireframes** representing how the UI appears in Streamlit. These replace drawings but convey layout accurately.

# ****Home Dashboard****

| **Section** | **Description / Content** |
| --- | --- |
| **Banner** | Full-width image showing Kenyan maize fields. Centered title: **Maize Crop Yield Prediction System** |
| **Key Features (Bullets)** | • Real-time weather & soil insights• County-level maize yield predictions• Advisory guidance for improved crop productivity |
| **Main Buttons** | **View Map** — Takes user to geospatial county dashboard**Start Prediction** — Opens prediction input form**Explore Data** — Opens analytics & data explorer |
| **National Indicators** | • Predicted National Avg Yield: **3.1 t/ha**• Counties: **47**• API Data Updated: **Today** |

# ****B. Main Application Screen****

### This is your core prediction interface.

It uses **left-side filters**, **central map**, and **dynamic data panels**, just like GFW.

| **LEFT SIDEBAR (User Controls)** | **MAIN VIEW (Map + Analytics Panels)** |
| --- | --- |
| **County Selector** (47 counties dropdown) | **Interactive Map of Kenya** (county polygons) |
| **Season Selector:** Long Rains / Short Rains / Off-season | County highlight + auto-load environmental data |
| **Data Layers (ON/OFF):** Predicted Yield, Rainfall, Soil pH, Vegetation | **Top Analytics Panel:** Yield, Temp, Rainfall, Soil pH cards |
| **Advanced Inputs:** Enter manual temp & rainfall | **Advisory Panel:** Short recommendations for selected county |
| **Buttons:** Fetch Weather & Soil Data, Predict Yield | **Charts:** Rainfall trend, Temperature trend, 10-year maize yield |

# ****“About CYPAS” Page****

| **SECTION** | **DETAILS** |
| --- | --- |
| **System Name** | Maize Crop Yield Prediction and Advisory System (CYPAS) |
| **Purpose** | Provides real-time, data-driven insights for farmers, researchers, and policymakers in Kenya. |
| **Key Features** | • ML-based maize yield prediction• Uses NASA POWER (weather) + SoilGrids (soil data)• County-level environmental analysis• Actionable agronomic advisory outputs |
| **Version** | 1.0 |
| **Contact** | [philiphahaldline@gmail.com](mailto:philiphahaldline@gmail.com) |

# ****4.1.2 Objects and Actions****

This subsection lists all UI elements and the actions associated with them.

## ****Input Objects (Forms & Selections)****

| **Object Type** | **Description** | **Action** |
| --- | --- | --- |
| Dropdown (county) | List of 47 counties | Select county for prediction |
| Dropdown (season) | Long rains/Short rains/Off-season | Controls model context |
| Number inputs | Optional temperature, rainfall | Overrides API data if provided |
| Button | "Get Weather & Soil Data" | Calls API Handler |
| Button | "Predict Maize Yield" | Calls ML Model |
| Button | "Download CSV" | Exports results |
| Slider (future) | Sensitivity adjustment | Modify advisory |

## ****Display Objects****

| **Display Type** | **Description** |
| --- | --- |
| Value boxes | Show climatic + soil parameters |
| Text area | Shows advisory message |
| Charts | Historical yield trends (matplotlib/plotly) |
| Maps | County-level outline (optional) |

## ****Interactive Actions****

| **Action** | **Outcome** |
| --- | --- |
| Selecting county | Triggers lookup of coordinates |
| Submitting prediction | Performs end-to-end workflow |
| Downloading CSV | Saves temporary data export |
| Navigating pages | Switch between Streamlit multipages |

# ****4.2 Interface Design Rules****

The design follows established usability principles (adapted from Shneiderman’s Eight Golden Rules):

**Consistency:**  
All pages use the same colors (green, white), fonts (Streamlit default), and panel layouts.

**Shortcuts:**  
Frequent users can bypass auto-fetching by entering manual values.

**Feedback:**  
After each action (API fetch, prediction), the UI shows success/failure messages.

**Closure:**  
Once prediction completes, a results panel summarizes what happened.

**Error Prevention:**

Required fields validated

Try/except for API failures

Value range checks

**Reversibility:**  
Users can clear inputs and try again.

**User Control:**  
Users may override climatic data manually.

**Reduce Memory Load:**  
The UI auto-fills soil/weather values; user doesn’t need to remember them.

# ****4.3 Components Available (GUI Widgets)****

Since Streamlit is used, the following widgets are available:

### ****Input widgets****

st.selectbox() – dropdown menus

st.number\_input() – numeric inputs

st.button() – clickable actions

st.date\_input() – date selection

### ****Display widgets****

st.write() – general outputs

st.metric() – numeric highlights

st.success(), st.error() – feedback notifications

st.table() / st.dataframe() – tabular outputs

st.pyplot() / st.plotly\_chart() – charts

st.map() – geospatial data

### ****File Management****

st.download\_button() – allows exporting predictions to CSV

# ****5.0 Restrictions, Limitations, and Constraints****

Although CYPAS is designed to provide reliable maize crop yield predictions and advisories for Kenya, it operates under a set of constraints related to data availability, model performance, infrastructure choices, and external API dependencies.

# ****5.1 Technical Limitations****

### ****5.1.1 Limited Computing Resources****

The system is built and tested on a standard student laptop (8GB RAM).

High-volume data processing and GPU-based model training are **not feasible**.

Heavy deep learning models or large raster-based satellite imagery processing are deliberately excluded.

### ****5.1.2 Single ML Model Approach****

The system uses a classical ML model (Random Forest / XGBoost) ensemble of multiple models.

Model accuracy is limited by available historical data for Kenyan counties.

Predictions are approximations and should not be treated as exact agricultural forecasts.

### ****5.1.3 No Real-Time Satellite Image Processing****

Unlike Global Forest Watch, which processes forest loss via real-time satellite imagery,  
CYPAS does **not** handle:

Raster image ingestion

NDVI/vegetation indices from MODIS

Pixel-level mapping

This limitation keeps the project simple and achievable.

# ****5.2 Data-Related Constraints****

### ****5.2.1 Variability in Kenyan Maize Data****

Historical yield data may have:

Missing values

Inconsistent records

County-level aggregation challenges

This impacts model performance.

### ****5.2.2 Dependence on External APIs****

CYPAS uses:

**NASA POWER API** for weather

**SoilGrids API** for soil characteristics

Limitations include:

Network failures

API rate limits

API downtime

Data refresh intervals beyond the system’s control

### ****5.2.3 Soil Data Resolution****

SoilGrids uses a global 250m resolution, not county-specific.

Soil characteristics are averaged, which reduces local precision.

# ****5.3 Operational and Environmental Constraints****

### ****5.3.1 Internet Connectivity****

System requires active internet for:

Fetching weather data

Fetching soil data

Deployment on Streamlit Cloud

Users in rural Kenya may experience delays or outages.

### ****5.3.2 Deployment Environment****

The application uses **Streamlit Cloud**, which:

Limits computational resources

Has timeouts for long-running processes

Does not support heavy background tasks

These constraints prevent large-scale data ingestion jobs.

# ****5.4 System Functional Constraints****

### ****5.4.1 Simplified Advisory Engine****

The advisory system uses **rule-based logic** instead of AI-driven agronomic modeling.

It does not provide crop-disease predictions, pest alerts, or fertilizer optimization curves.

These features may be added in future work.

### ****5.4.2 Limited User Management****

User login is optional and basic.

No multi-role system is implemented (e.g., admin, field officer, researcher).

### ****5.4.3 Static County Boundaries****

County shapes and coordinates are assumed to be static.

No dynamic geospatial updates, redistricting, or shapefile manipulation.

# ****5.5 Legal, Ethical, and Privacy Constraints****

### ****5.5.1 Ethical Responsibility****

CYPAS provides agronomic suggestions but cannot replace certified agricultural extension officers.  
Users must understand that:

Advice is general, not farm-specific

Users must consider local conditions not captured by the system

Misinterpretation of outputs may lead to incorrect farming decisions

### ****5.5.2 Privacy****

Minimal user data is collected (e.g., name, email).

Sensitive personal information is not stored.

Must comply with:

Kenyan Data Protection Act (2019)

Basic GDPR-like privacy principles

### ****5.5.3 API Key Security****

API keys must be stored in environment variables.

Keys must never be committed to GitHub.

# ****6.0 Testing Issues****

## ****6.1 Classes of Tests****

**Unit Testing**

Tests individual functions within the ML modules, API integration modules, and database communication layer.

Validates data preprocessing functions, prediction functions, and advisory rules.

Ensures Streamlit UI components behave correctly, e.g., form validation.

**Integration Testing**

Validates that components interact correctly:

Streamlit ↔ API module

API module ↔ NASA POWER / SoilGrids

ML Model ↔ Advisory engine

API layer ↔ PostgreSQL database

Ensures data flows smoothly from external APIs into the prediction engine.

**System Testing**

Ensures full end-to-end operations:

User selects location and inputs field details

Weather/soil data fetched

Yield prediction generated

Advisory insights displayed

Results stored in database

Validates error handling, loading behavior, and user experience.

**Acceptance Testing**

Ensures final system meets stakeholder expectations:

Farmers get simple, actionable insights

Government/NGO staff can view county-level summaries

Predictions align with expected ranges

**Performance Testing**

Tests application behavior under high request loads

Measures prediction latency and API call response times

**Security Testing**

Ensures API keys are protected

Validates database credentials and network access rules

Checks for unauthorized data access

## ****6.2 Expected Software Response****

**Yield predictions** should fall within realistic Kenyan maize yield ranges (0.5–8.0 t/ha depending on region).

**Error messages** must be user-friendly and descriptive:

“Weather API unavailable — try again later.”

“Please enter all required field data.”

**Advisory recommendations** should always correspond to model output ranges.

**Map views** should load within 3–5 seconds depending on internet speed.

**Database storage requests** must return success/failure notifications.

## ****6.3 Performance Bounds****

**Model inference latency**: < 1 second

**API response time**: NASA POWER < 1.5s; SoilGrids < 2s

**Streamlit page load time**: < 4 seconds on a typical 4G connection

**Concurrent user handling**: Up to 50 users comfortably on Streamlit Cloud

**Database query latency**: < 300ms for typical CRUD operations

## ****6.4 Identification of Critical Components****

Components with highest impact on system functionality:

**API Integration Module**

If NASA POWER or SoilGrids fail, prediction accuracy drops drastically.

**Machine Learning Model**

Critical for prediction correctness.

Poor calibration affects all downstream advisories.

**Advisory Engine**

Incorrect rule interpretations can mislead the user.

**Database**

Essential for historical trend analysis and report generation.

**User Interface**

Must gracefully handle errors and slow API responses.

# ****7.0 Deployment and Maintenance Plan****

## ****7.1 Deployment Architecture****

**Frontend/UI**: Streamlit (hosted on Streamlit Cloud)

**Backend Logic**: Python modules encapsulating ML, advisory rules, and API requests

**External data sources**: NASA POWER API, SoilGrids REST API, FAO reference datasets

**Database**: PostgreSQL hosted on Render, ElephantSQL, or Railway

**Version control**: GitHub repository with CI/CD workflows

**Model artifact storage**: GitHub Releases or cloud storage (e.g., Google Cloud Bucket, AWS S3)

Data flow:  
User → Streamlit → API Module → ML Model → Advisory Engine → UI Output → Database

## ****7.2 Backup & Recovery Strategy****

**Database backups** automatically generated daily

Retain backups for **30 days**

For PostgreSQL cloud provider: enable PITR (Point-In-Time Recovery)

ML model files version-controlled with tags

System failure recovery steps:

Redeploy Streamlit app via GitHub

Restore database from latest backup

Restore model artifacts from versioned storage

## ****7.3 Logging, Monitoring, and Alerts****

**Application Logs**

API request failures

Prediction errors

Advisory engine exceptions

Database read/write failures

**Monitoring**

Uptime monitoring via Streamlit Cloud tools

Database health metrics via hosting provider

API latency checks

**Alerts**

Email or Slack notifications for downtime

Alerts triggered for >20% API failure rate

Alerts for storage capacity reaching 80%

## ****7.4 Model Retraining & Lifecycle****

**Retraining frequency**: yearly or after major agricultural season cycles

**Data sources for retraining**:

Historical weather

Soil characteristics

Farmer-submitted yield reports

County government agricultural datasets

**Model lifecycle process**:

Collect new data

Retrain model

Validate and compare performance

Update version tag (e.g., v1.2 → v1.3)

Deploy new model artifact

Archive old model versions

Document changes in a changelog

# ****8.0 Security and Privacy Considerations****

## ****8.1 Authentication & Authorization****

Basic access control for admin dashboards (email + password)

Farmer users may not require accounts unless storing personalized data

API keys for NASA and SoilGrids stored in environment variables

Password hashing using bcrypt or Argon2

## ****8.2 Data Privacy****

Compliant with **Kenyan Data Protection Act (2019)**

Only store minimal user data (location, field details, predictions)

Sensitive data encrypted at rest in SQL

HTTPS enforced for all communication

No unnecessary sharing of location data

## ****8.3 Secure Storage of API Keys & Credentials****

API keys stored using:

Streamlit Secrets Manager (.streamlit/secrets.toml)

Environment variables during deployment

Keys never committed to GitHub

Regularly rotate keys every 6–12 months

Database credentials stored in secrets manager as well

# ****9.0 Appendices****

### ****A. Glossary of Terms****

**CYPAS** — Crop Yield Prediction and Advisory System

**NASA POWER** — Weather data API for agricultural planning

**SoilGrids** — Global soil property database

**ML Model** — Machine learning algorithm that predicts maize yield

**Advisory Engine** — Rule-based system that converts model output to farmer advice

**PostgreSQL** — Database used to store predictions and user inputs

**Streamlit** — Web framework for building data applications

**t/ha** — Tonnes per hectare (yield measurement)

### ****B. External APIs Referenced****

NASA POWER API

SoilGrids REST API

FAO Agroclimatic datasets

Optional: Kenya Meteorological Department datasets (if integrated)

### ****C. Sample Test Cases & Sample Data****

**Test Case 1 — Valid Prediction Request**

Input: Kiambu County, 1-acre field, maize variety KDV4

Expected: Yield prediction + advisory output

**Test Case 2 — Missing Soil Data**

Expected: System uses default soil assumptions + displays warning

**Test Case 3 — API Failure**

Expected: “Weather data unavailable” error message

**Test Case 4 — Database Write Failure**

Expected: Prediction still displayed; user informed results were not saved

### ****D. References****

·**NASA POWER API** — Prediction of Worldwide Energy Resources.  
Available at: [https://power.larc.nasa.gov/](https://power.larc.nasa.gov/" \t "_new)  
Accessed: 2025.

· **ISRIC SoilGrids REST API** — Global Soil Information Service.  
Available at: [https://soilgrids.org/](https://soilgrids.org/" \t "_new)  
Accessed: 2025.

· **FAO Kenya Agroclimatic and Agronomy Guidelines** — Maize production recommendations and climate characteristics.  
Available at: [https://www.fao.org/](https://www.fao.org/" \t "_new)  
Accessed: 2025.

· **Kenya Ministry of Agriculture, Livestock & Fisheries** — Annual maize production statistics and county yield reports.  
Available at: [https://kilimo.go.ke/](https://kilimo.go.ke/" \t "_new)  
Accessed: 2025.

· **Streamlit Documentation** — Framework for building Python-based web interfaces.  
Available at: [https://docs.streamlit.io/](https://docs.streamlit.io/" \t "_new)  
Accessed: 2025.

· **Scikit-Learn Documentation** — Machine learning library used for model development and evaluation.  
Available at: [https://scikit-learn.org/](https://scikit-learn.org/" \t "_new)  
Accessed: 2025.

· Jay, S., & Kaltenbrunner, A. (2022). Data-driven approaches to agricultural forecasting: a review of methods and applications. Agricultural Systems, 198.

· Lobell, D. B., Schlenker, W., & Costa-Roberts, J. (2011). Climate trends and global crop production since 1980. Science, 333(6042), 616–620.