AquaSens: Intelligent Irrigation Recommendation System

Final Mini Project in Artificial Intelligence Foundations

Student: Houssem Eddine Chaouch
Software engineering cycle, 4eme année GL BI DS
Sousse Polytechnic School

Academic Year 2024–2025
Module: Foundations of AI
Supervisor: Dr. Fatma Sbiaa

1 General Introduction

This report presents **AquaSens**, an intelligent irrigation recommendation web application developed as a final mini project for the *Fondements de l'IA* module in the engineering cycle (4th year GL BI DS) at Ecole Polytechnique Sousse, under the supervision of Mme Fatma Sbiaa. The work is carried out in the context of smart agriculture and water-saving technologies, with the objective of applying fundamental artificial intelligence techniques to a real-world decision support problem. The system combines a modern React frontend, a Node.js/Express backend, a MongoDB database and a Python-based machine learning microservice to help farmers estimate their irrigation needs.

2 Context and Objectives of the Application

Agricultural water management is a critical issue in Tunisia and in many Mediterranean regions, where climate variability and limited water resources require precise irrigation planning. The main objective of AquaSens is to provide farmers with personalized irrigation level recommendations (Low, Medium or High) and practical agronomic advice based on soil, weather, crop and regional information. More specifically, this mini project in the *Fondements de l'IA* module aims to:

- Design and implement a full-stack web application that respects the software engineering principles taught in the GL BI DS engineering curriculum.
- Integrate a decision tree machine learning model through a dedicated Python/Flask microservice in order to predict irrigation needs from heterogeneous field data.
- Expose a secure web interface where authenticated users can submit prediction requests through an intuitive form and consult their history of predictions.

3 Database Design and Characteristics

The backend uses MongoDB to store user accounts and prediction history through Mongoose schemas. Two main collections are defined: `User` and `Prediction`, each with specific fields and constraints.

3.1 User Collection

The `User` schema stores authentication information for each farmer or user of the system. Its main fields are:

- `name`: user full name, required string.
- `email`: unique identifier for login, required string.
- `password`: hashed password using `bcrypt`.
- `role`: user role, with default value `FARMER`.

Passwords are never stored in clear text; instead, the system hashes them with `bcrypt.hash` and verifies them with `bcrypt.compare` during login. Successful signup or login returns a JSON Web Token (JWT) containing the user id and role, which will be used to secure protected routes.

3.2 Prediction Collection

The `Prediction` schema records each machine learning prediction triggered by a user. Its main fields are:

- `userId`: reference to the `User` who made the request.
- `input`: an object containing all input features (soil, weather, crop, etc.).
- `result`: categorical irrigation level (Low, Medium, High).
- `decisionPath`: an array describing the most important decision tree splits used for the prediction.
- `recommendation`: an object that stores agronomist recommendations (action and advice list).
- `createdAt`: automatic date of prediction creation.

MongoDB is well suited to store flexible objects such as the input feature vector and the recommendation structure, while keeping a simple schema for user-specific history.

4 Machine Learning and Model Architecture

The machine learning component is implemented in Python using a Flask microservice. A decision tree classifier is trained on a dataset of irrigation records, then exported with `joblib` and loaded at runtime together with a feature encoder.

4.1 Features and Preprocessing

The model expects a set of numerical and categorical features:

- Numerical: soil pH, soil moisture, organic carbon, temperature, humidity, rainfall, sunlight hours, wind speed and previous irrigation amount.
- Categorical: soil type, crop type, crop growth stage, season, region and mulching used.

Incoming JSON data from the Node.js backend is converted to a pandas `DataFrame`. Categorical variables are encoded with the fitted encoder; the Boolean field `Mulching_Used` is mapped to `0/1`. The model outputs an integer class that is mapped to irrigation levels using a target map, for example $0 \to Low$, $1 \to Medium$, $2 \to High$.

4.2 Decision Path and Recommendations

In addition to the predicted class, the system extracts a human-readable decision path from the decision tree. For each visited node, the feature name, threshold and actual value are recorded, which explains why the model recommended a given irrigation level. A separate recommendation function returns:

- An overall action (for example "Immediate irrigation required" for High level).
- A list of practical advice such as using drip irrigation, monitoring soil moisture or delaying irrigation when water is sufficient.

This combination of prediction and explanation increases transparency and pedagogical value for users who are not experts in AI.

5 System Architecture

The overall architecture follows a service-oriented pattern with three main layers: frontend, backend API and ML microservice. The frontend is built with React and TypeScript, the backend with Node.js/Express and MongoDB, and the ML part with Flask.

5.1 Backend API

The Node.js API exposes endpoints for authentication and prediction management:

- `POST /api/auth/signup`: registers a new user, hashes the password, and returns a JWT.
- `POST /api/auth/login`: verifies credentials and issues a JWT token.
- `POST /api/predictions`: authenticated route that forwards the input features to the Flask service, receives the prediction, stores it in MongoDB, and returns it to the frontend.
- `GET /api/history`: returns the list of past predictions for the authenticated user.
- `GET /api/history/:id`: returns the details of a specific prediction.

A middleware verifies the JWT token, extracts the user id and role, and protects all history and prediction routes by rejecting unauthorized requests.

5.2 Frontend Interface

The React frontend offers three key pages:

- Login and signup forms with validation using Zod schemas for email format and password length.
- A prediction form that collects all required soil, weather and crop parameters, validates value ranges (for example pH between 0 and 14), and sends an HTTP POST request to the API.
- A prediction history view that displays past results in a table, with colored badges for Low/Medium/High irrigation needs and navigation to a detailed view.

Axios interceptors attach the JWT token to each request and redirect the user to the login page if the session becomes invalid or the token expires.

6 Results and Evaluation

The model is evaluated on a held-out test set using standard classification metrics such as accuracy and confusion matrix. The irrigation levels are generally well separated, and qualitative inspection shows that High irrigation recommendations correspond to low soil moisture or high temperature, whereas Low irrigation recommendations occur when soil moisture is already sufficient. The decision path output confirms that the model uses intuitive splits on key variables such as soil moisture, rainfall and crop growth stage. However, the performance depends on the representativeness of the training data, and the system may require retraining when deployed in new regions or for new crops.

7 Conclusion and Perspectives

This mini project demonstrates how a complete intelligent irrigation recommendation system can be built by combining a React frontend, a Node.js REST API, a MongoDB database and a Python-based machine learning service. The application secures user access, stores prediction history, and provides transparent explanations of model decisions through decision paths and textual recommendations, which aligns with the learning objectives of the *Fondements de l'IA* module. Future improvements may include integrating real meteorological APIs, supporting additional crops and soil types, using more advanced models (Random Forests or Gradient Boosting), and deploying the system in a cloud environment for real-world use.