AY 2021/2022

Politecnico di Milano

Implementation Document

Ottavia Belotti Alessio Braccini Riccardo Izzo

Professor
Elisabetta Di Nitto

Version 1.0
January 23, 2022

Contents

1 Introduction

The code can be found in the official project repository on GitHub at the link: `https://github.com/AlessioBraccini/SE2-Belotti-Braccini-Izzo`.

1.1 Purpose

This document aims to describe how the implementation and integration testing took place. Implementation is the last step of the DREAM application development cycle. Testing, instead, means check that the critical parts of the application works in a correct way, as described in the DD document.

1.2 Definitions, Acronyms, Abbreviations

- API: Application Programming Interface
- DBMS: DataBase Management System
- DD: Design Document
- HTTP: HyperText Transfer Protocol
- JS: JavaScript
- REST: REpresentational State Transfer
- RASD: Requirements Analysis and Specification Document
- UI: User Interface
- URL: Uniform Resource Locator
- WSGI: Web Server Gateway Interface
- ASGI: Asynchronous Server Gateway Interface

1.3 Revision History

- Version 1.0:

1.4 References

- Django Framework: `https://www.djangoproject.com/`
- REST Framework: `https://www.django-rest-framework.org/`
- Vue.js: `https://vuejs.org/`
- Axios: `https://axios-http.com/docs/intro`

2 Development

2.1 Implemented Functionalities

Given the three types of user, we decided to implement th functionalities for Policy Maker users and Agronomist user. In particular:

Policy Maker

- Farmers ranking
- Visualization of humidity sensors and water irrigation systems data as graphs
- Retrieving agronomists' reports about steering initiatives

Agronomist

- Farmers ranking tailored on the agronomist's district
- Uploading reports concerning steering initiatives
- Creation and updating of daily plans
- Help requests inbox
- Weather widget

2.2 Adopted Development Frameworks

Model-View-Controller paradigm

2.2.1 Programming Language

The programming language of choice for the DREAM backend is Python.

For the client side we choose to use Javascript, a text-based programming language, that allows to build interactive web pages. Alongside Javascript, that allow the interaction with the user of the elements present in the page, we used HTML and CSS to give structure and style to the page.

- **Pros:**
  - \+ Allow fast Development
  - \+ Readability
  - \+ Widely spread among WebApps
- **Cons:**
  - \- Slow

2.2.2 Django Framework

2.2.3 Django REST Framework

2.2.4 Vue.js

Vue.js is an open-source model–view–viewmodel front end JavaScript framework that allow to create user interfaces and single-page applications. Vue.js features an incrementally adaptable architecture that focuses on declarative rendering and component composition. The core library is focused on the view layer only. Advanced features required for complex applications such as routing, state management and build tooling are offered via officially maintained supporting libraries and packages.

We used this framework to build up the client-side rendering of pages because its easy usage as it integrate in a single .vue file, also called components, the HTML, CSS and javascript part.

In order to communicate with the backend we use the javascript library Axios. It is a promise-based HTTP Client for node.js and the browser. On the server-side it uses the native node.js http module, while on the client it uses XMLHttpRequests. It transforms in an automatic way the json reply that arrives from the server in vue ready XMLHttpRequests.

2.3 API Integration

2.3.1 OpenWeatherMap API

To allow the user to retrieve the weather information we use this external api service that let us know in real time the weather condition of a specific territory. This return us not only the basics information but also more specific ones.

2.4 DataBase

Postgresql

3 Source Code

3.1 Backend Structure

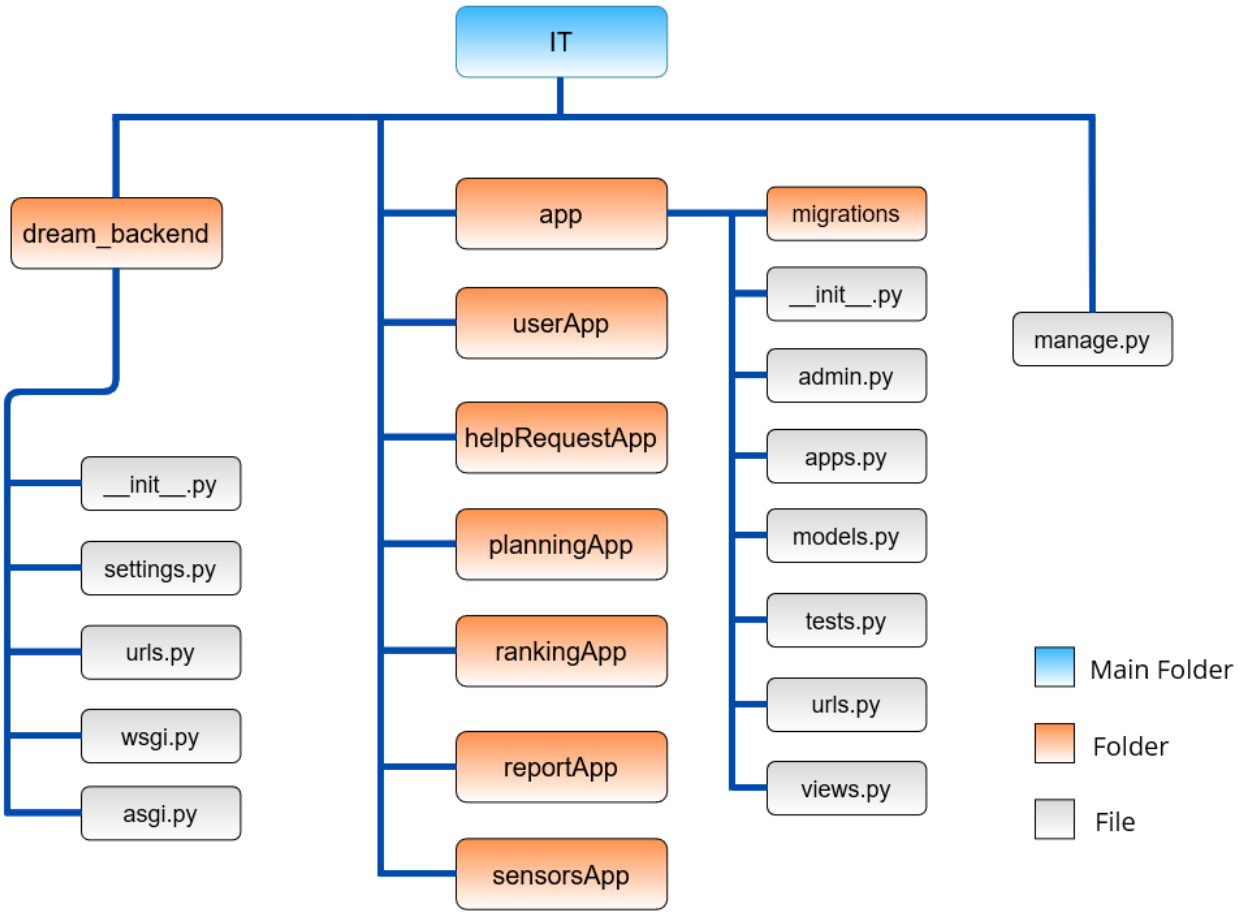

Figure 1: Backend structure

- **IT**: root directory, container for the project
- **dream_backend**: folder that contains the configuration files of the project

- **__init.py__**: it tells the Python interpreter that the directory is a Python package
- **settings.py**: main setting file for the Django project, used to configure all the applications and middleware, it also handles the database settings
- **urls.py**: URL declarations for the Django project, it contains all the endpoints that the website should have
- **wsgi.py**: entry-point for WSGI-compatible web servers to serve your project, it describes the way how servers interact with the applications
- **asgi.py**: entry-point for ASGI-compatible web servers to serve your project, ASGI works similar to WSGI but comes with some additional functionality
- **migrations**: Django's way of propagating changes to the models into the database schema, when changes occur this folder is populated with the records of them
- **admin.py**: used for registering the Django models into the Django administration, it allows to display them in the Django admin panel
- **apps.py**: common configuration file for all Django apps, used to configure the attributes of the app
- **models.py**: it defines the structure of the database, it allows the user to create database tables for the app with proper relationships using Python classes. It tells about the actual design, relationships between the data sets and their attribute constraints
- **tests.py**: used to test the overall working of the app through unit tests
- **views.py**: provide an interface through which a user interacts with a Django website, it contains the business logic of the app
- **manage.py**: command-line utility for executing Django commands; these includes debugging, deploying and running

3.2 Frontend Structure

The front-end web application is contained into the dream_frontend folder of the IT directory. Of course, following the four tier architecture described in the Design Document, the front-end web application can also be deployed to a dedicated web server, which will then make requests to a different backend server. Here is represented the structure of the web app:

4 Testing

4.1 Backend Testing

5 Installation

5.1 Requirements

5.2 Backend Installation

5.3 Frontend Installation

6 Effort Spent

Student	Time for implementation
Ottavia Belotti	80h
Alessio Braccini	80h
Riccardo Izzo	80h