

POLITECNICO DI MILANO
Computer Science and Engineering

Implementation and Test Deliverable

DREAM - Data-dRiven PrEdictive FArMing in
Telangana

Software Engineering 2 Project
Academic year 2021 - 2022

February 6, 2022
Version 1.0

Authors:
Kinga Marek
Józef Piechaczek
Mariusz Wiśniewski

Professor:
Damian Andrew Tamburri

Contents

Contents	1
1 Introduction	2
1.1 Purpose	2
1.2 Scope	2
1.3 Definitions and Acronyms	3
1.4 Revision History	4
1.5 Reference Documents	4
1.6 Document Structure	4
2 Product Functions	6
2.1 Implemented Functions	6
2.2 Excluded Functions	8
3 Development Frameworks	11
3.1 Programming Languages	11
3.2 Frameworks and libraries	13
3.3 Relational database management system	17
4 Code Structure	18
4.1 High Level Overview	18
4.2 Client	19
4.3 Server	20
4.4 System Tests	21
5 Testing	22
5.1 Integration Testing	22
5.2 System Testing	32
6 Installation Guide	37
6.1 Docker	38
6.2 Server	38
6.3 Client	39
6.4 Running System Tests	39
7 Effort Spent	40
References	43

Chapter 1

Introduction

1.1 Purpose

The purpose of this document is to present the work done during the implementation of the Data-driven Predictive Farming in Telangana (*DREAM*) application's initial prototype. It follows the directives and instructions specified in the *I&T assignment goal, schedule, and rules* [3]. A thorough backdrop together with problem definition may be found in the *Requirement Engineering and Design Project: goal, schedule, and rules* document [4].

1.2 Scope

Implementation and Test Deliverable (*ITD*) provides an overview of the languages and frameworks utilized during development, as well as a synopsis of the functionalities accessible in the current version of the program.

Furthermore, described here are: the created code, a document outlining the structure of the code, the development frameworks used, installation instructions, and information on how software tests were performed.

This document, together with the Requirement Analysis and Specification Document (*RASD*) [2] and the Design Document (*DD*) [1], is intended to inform on the completion of the *DREAM* application.

1.3 Definitions and Acronyms

1.3.1 Definitions

Expression	Definition
Data transfer object	Object that encapsulates data. Used to transport data between the API layer and Business logic layer.
Database migration	A feature of Entity Framework that allows for incremental changes of database schema as well as definition of the database schema based solely on the classes defined in the code.
Web driver	A remote control interface that allows for user agent introspection and control. It provides a platform- and language-neutral wire protocol that allows out-of-process applications to remotely command web browser behavior.

1.3.2 Acronyms

Acronyms	Expression
API	Application PRogramming Interface
DD	Design Document
DOM	Document Object Model
DREAM	Data-dRiven PrEdictive FArMing in Telangana
ITD	Implementation and Test Deliverable
JWT	JSON Web Token
LINQ	Language-Integrated Query
R	Requirement
RASD	Requirements Analysis and Specifications Document
SQL	Structured Query Language
UI	User Interface

1.4 Revision History

Date	Revision	Notes
06.02.2022	v.1.0	First release.

1.5 Reference Documents

- [1] Kinga Marek, Józef Piechaczek, and Mariusz Wiśniewski. *Design Document, DREAM - Data-dRiven PrEdictive FArMing in Telangana*. Jan. 2022.
- [2] Kinga Marek, Józef Piechaczek, and Mariusz Wiśniewski. *Requirements Analysis and Specifications Document, DREAM - Data-dRiven PrEdictive FArMing in Telangana*. Dec. 2021.
- [3] Elisabetta Di Nitto, Matteo Rossi, and Damian Tamburri. *A.Y. 2021-2022 Software Engineering 2 I&T assignment goal, schedule, and rules*. Oct. 2021.
- [4] Elisabetta Di Nitto, Matteo Rossi, and Damian Tamburri. *A.Y. 2021-2022 Software Engineering 2 Requirement Engineering and Design Project: goal, schedule, and rules*. Oct. 2021.

1.6 Document Structure

1. **Introduction:** describes the main aim of the work as well as its scope. It also specifies specific definitions, acronyms, and abbreviations that will be used throughout the text.
2. **Product Functions:** discusses the requirements and functionalities that are actually implemented in the initial prototype of the program, as well as the reasoning for including them and rejecting others.
3. **Development Frameworks:** explains the languages and frameworks used during development, together with motivation for their choice. Furthermore, it provides information about the adopted middleware and any API employed that was not included in the DD.
4. **Code Structure:** elaborates on the structure of the code, including the organization of the different modules and the use of the different programming languages.
5. **Testing:** describes the testing procedures, the key test cases, as well as the results of performed tests.
6. **Installation Guide:** provides instructions on how to install the program and how to run it. It lists all the prerequisites that needs to be satisfied to run the application successfully.

7. **Effort Spent:** gives a detailed summary of the time spent during the development of the program by each contributor.

8. **References**

Chapter 2

Product Functions

This chapter explains the criteria and functions that are actually implemented in the program's initial prototype, as well as the logic for adding them and rejecting others.

2.1 Implemented Functions

According to the assignment description, groups of three students should simulate the gathering of data from the other elements of the system by implementing the features supplied to two of the three stakeholders in the project. The product's initial prototype focuses on functions relevant to farmers and policy makers.

All the requirements that are satisfied with the initial prototype of the program are listed below.

ID	Requirement
<i>Authentication</i>	
R1.	The system must uniquely identify each user by his e-mail.
R2.	The system must allow an unregistered user to create an account with a chosen role.
R4.	The system must ensure that a farmer inserts his farm data during the registration process.
R6.	The system must allow a registered user to log in to the application.
R8.	The system must allow a logged-in user to sign out of the application.
R9.	The system must allow a registered user to delete his account.
<i>Policy maker</i>	
R10.	The system must allow a policy maker to assign a note to a farmer.
R11.	The system must ensure that every farmer initially has a neutral note.

- R12.** The system must have a list of predefined farmer's problem types.
 - R13.** The system must allow a policy maker to specify a problem type when assigning a negative note.
 - R17.** The system must ensure that an automatic help request followed by additional farm visits are created, if a farmer receives a negative note.
R17 is implemented only partially - without creating additional farm visits.
 - R19.** The system must allow a policy maker to see a list of all farmers in Telangana.
 - R20.** The system must allow a policy maker to see a list of all farmers with a specific note.
 - R21.** The system must allow a policy maker to see a list of all farmers in a given mandal.
 - R22.** The system must allow a policy maker to find a farmer's summary by his name and surname.
 - R23.** The system must allow a policy maker to see a farmer's summary.
-

Farmer

- R24.** The system must allow a farmer to see his own farmer's summary.
 - R25.** The system must allow a farmer to see personalized suggestions.
 - R26.** The system must allow a farmer to manage his monthly production data.
 - R27.** The system must allow a farmer to see a list of help requests created by him.
 - R28.** The system must allow a farmer to find a help request created by him by the topic.
 - R29.** The system must allow a farmer to see a specific help request created by him.
 - R30.** The system must allow a farmer to delete a specific help request created by him.
 - R31.** The system must allow a farmer to create a new help request.
 - R32.** The system must allow a farmer with a positive note to see a list of all help requests in his mandal.
 - R33.** The system must allow a farmer with a positive note to find a help request in his mandal by the topic.
 - R34.** The system must allow a farmer with a positive note to see a specific help request in his mandal.
 - R35.** The system must allow a farmer with a positive note to respond to a specific help request in his mandal.
 - R36.** The system must allow a farmer with a positive note to delete a help response, only if it was created by him.
 - R37.** The farmer is able to see farmer's summary of a farmer with a negative note whose help request he received.
 - R38.** The system must allow a farmer to see a list of all forum threads in Telangana.
 - R39.** The system must allow a farmer to find a forum thread by the topic.
-

- R40.** The system must allow a farmer to see a specific forum thread with its comments.
 - R41.** The system must allow a farmer to create a comment in a forum thread.
 - R42.** The system must allow a farmer to delete a comment in a forum thread, only if it was created by him.
 - R43.** The system must allow a farmer to create a forum thread.
-

2.2 Excluded Functions

Functions relevant to the agronomist's role and responsibilities (including farm visits) were determined not to be added in the product's original prototype. All the requirements that are not satisfied are mentioned below.

Although the information from external systems is presented in the client application, the functionalities related to interaction with external systems were excluded. Therefore, the data presented to the users are a collection of static, mocked data. The server application also contains definitions of API endpoints for external systems' data retrieval, but the functionality was not developed due to absence of any real data sources.

The only excluded function relevant to the farmers and policy maker is the reset password functionality. The omission was motivated by the low business value and the preference for more thorough testing in return.

ID	Requirement
R3.	The system must ensure that an agronomist chooses the area of responsibility during the registration process.
R5.	The system must ensure that two casual farm visits are scheduled for each year of the farm's existence.
R7.	The system must allow a registered user to reset his password.
R14.	The system must ensure that a farm is visited more often in the event of any problems.
R15.	The system must have a specified number of additional visits caused by each predefined problem type.
R16.	The system must ensure that only the most recently specified problem type is taken into account when determining the number of visits.
R18.	The system must ensure that all the additional farm visits created due to obtaining a negative note are deleted after a farmer's negative note is updated with a positive or a neutral one.
R44.	The system must allow an agronomist to manage his area of responsibility.
R45.	The system must allow an agronomist to see the list of suggestions for mandals in his area of responsibility.

- R46.** The system must allow an agronomist to manage the list of suggestions for mandals in his area of responsibility.
 - R47.** The system must allow an agronomist to see a list of all farmers in his area of responsibility.
 - R48.** The system must allow an agronomist to see a list of all farmers with a specific note in his area of responsibility.
 - R49.** The system must allow an agronomist to see a list of all farmers in a given mandal in his area of responsibility.
 - R50.** The system must allow an agronomist to find a farmer's summary in his area of responsibility by his name and surname.
 - R51.** The system must allow an agronomist to see a farmer's summary in his area of responsibility.
 - R52.** The system must allow an agronomist to see a list of all help requests in his area of responsibility.
 - R53.** The system must allow an agronomist to find a help request in his area of responsibility by the topic.
 - R54.** The system must allow an agronomist to see a specific help request in his area of responsibility.
 - R55.** The system must allow an agronomist to respond to a specific help request in his area of responsibility.
 - R56.** The system must allow an agronomist to delete a help response, only if it was created by him.
 - R57.** The system must allow an agronomist to see his daily plans.
 - R58.** The system must allow an agronomist to submit a daily plan's execution state, by rejecting or confirming each visit, on the same date or after the daily plan's date has passed.
 - R59.** The system must allow an agronomist to provide a comment for a visit he is confirming.
 - R60.** The system must ensure that after an agronomist submits his daily plan, for all the causal visits marked as confirmed, new ones are created in approximately half of a year.
 - R61.** The system must allow an agronomist to delete a visit before its date.
 - R62.** The system must ensure that a deleted visit is marked as rejected.
 - R63.** The system must ensure that in case of rejecting a casual visit, a new one is created in maximally 5 days.
 - R64.** The system must allow replanning any different visit than a casual one without any constraints.
-

External systems

- R65.** The system must read and update weather forecasts every day.
 - R66.** The system must read and store data from humidity sensors every day.
 - R67.** The system must read and store data from water irrigation systems every day.
 - R68.** The system must ensure that a farmer who creates a help request cannot be its recipient.
-

Chapter 3

Development Frameworks

This section of the article describes the programming languages and frameworks utilized throughout development, as well as the reasons behind their selection in forms of their advantages and disadvantages.

3.1 Programming Languages

3.1.1 TypeScript

TypeScript[11] is a strongly typed programming language build on JavaScript and maintained by Microsoft. According to Stack Overflow 2020 Developer survey, it was the second most loved programming language. Its several downsides and upsides are described below.

Advantages

- Readability - thanks to strong typing, the code is more self-documenting, making it more suitable for large-scale projects.
- Early spotted bugs and types error - data types error are not only detected at runtime like in dynamic typing with JavaScript, but also during compile time.
- Rich IDE support - information about types combined with IDEs support can produce a significant productivity boost, offering features like autocompletion, suggestions or just better code navigation.

Disadvantages

- TypeScript is just a subset of JavaScript - experienced JavaScript developers may happen to run into some issues trying to use functions that are not available in TypeScript.
- Requires writing more code - especially in case of generic functions, the types definitions can look bloated and defined inline by default.

- Code needs to be transpiled into JavaScript - it can be done on the fly, but still can be a bottleneck for larger code bases.

3.1.2 C#

C#[5] is a general purpose, strongly typed, object-oriented programming language designed by Microsoft. It runs on .NET execution system called the common language runtime (CLR) that not only provides runtime services, but also extensive libraries designed for building many kinds of apps. Below, you can find some of its advantages and disadvantages.

Advantages

- Active community - due to its popularity and various applications, an abundance of guidelines, tutorials, and documentation can be found on the Internet.
- Extensive tooling and libraries - developers can choose among many libraries available as *nuget*[10] packages and choose a convenient IDE well-suited for C# development, such as *Rider* or *Visual Studio* [7].
- Open source and cross-platform - over the last years, C# and .NET went through a huge transformation from proprietary, Windows-only to an open source, cross-platform technology. The change introduced significant performance boost and language improvements like implicit usings and new lambda capabilities [19].

Disadvantages

- Stability issues for new releases - together with great transformations of C# and .NET mentioned in the advantages list, some previously supported frameworks and functionalities were abandoned. In addition, the new releases of .NET Core introduced many breaking-changes [8].
- No standalone compiler, being tied to .NET's Common Language Runtime - the code can not be compiled into a machine code. Instead, it uses proprietary byte code that is executed on a virtual machine called *Common Language Runtime* that needs to be available on the machine to run a C# application [5].

3.1.3 Python

Python [14] is a high-level general-purpose programming language that is interpreted. With the usage of considerable indentation, its design philosophy prioritizes code readability. Its language elements and object-oriented approach are intended to assist programmers in writing clear, logical code for small and large-scale projects. Its advantages as well as drawbacks are presented below.

Advantages

- High-level language that boosts productivity. When compared to other languages, the language features rich support libraries and clean object-oriented designs that enhance programmer efficiency by a significant margin.
- Interpreted language - Python runs the code immediately, line by line. In the event of an error, it suspends further execution and communicates the problem. Due to the fact that the language shows only one error at a time, it makes debugging simple.
- Portability - programming languages, such as C/C++, generate a need to rewrite the code in order to run the application on other platforms. Python, on the other hand, is not the same. Due to the usage of interpreter, code written in this language is platform-independent.

Disadvantages

- Execution speed - Python is a dynamically typed interpreted language. Because the language is interpreted, each line of code must be explicitly interpreted for execution. This is time-consuming, and hence slows down the execution process.
- Memory consumption - a necessary sacrifice for Python using dynamic typing is very high memory usage. It makes it practically unusable in scenarios when the program's performance plays an important role.
- Runtime errors - being a dynamically typed language means that the data type of variable can change any time. This may lead to unexpected runtime errors.

3.2 Frameworks and libraries

3.2.1 Entity Framework

Entity Framework[21] is an open source, object-relational mapper for .NET that supports many types of databases. It provides support for LINQ queries, updates, schema migrations and change of tracking [13].

Advantages

- No SQL knowledge necessary - the developers are not required to write SQL queries to interact with the database since Entity Framework takes the burden of translating the LINQ queries into SQL.
- Defines a common syntax that can be used both for manipulating collections stored in application's memory or database, regardless of the database engine used.

Disadvantages

- Possible lack of support for the latest or more sophisticated database engine features - entity framework may not immediately reflect the latest features added to database engines.
- Sometimes, lack of knowledge about the way how LINQ queries are translated may lead to creation of very expensive SQL queries.

3.2.2 FluentValidation

FluentValidation[9] is a .NET library designed for definition of strongly-typed validation rules. It allows for setting up dedicated validator classes that define validation rules using fluent API.

Advantages

- Easy creation of readable validation rules with meaningful error messages.
- Decouples validation rules and models.
- Can be added as a middleware that is executed even before the request with given model reaches controller.

Disadvantages

- Rules may be hard to debug.

3.2.3 AutoMapper

AutoMapper[12] is a library build for automating object-to-object mapping. The library aims at simplifying the task of mapping one object to another. It automatically maps fields with the same name leveraging reflection and for more complex cases it allows configuration of custom projections.

Advantages

- Saves time automating mapping of fields with the same value.
- Mappings configurations may be tested automatically using functions from the library.

Disadvantages

- Refactoring issues - after renaming a field that was used in the automatic mapping (name matching), the mapping will break without any compile time errors.
- Hard to debug.

3.2.4 React

React [18] is a declarative, efficient, and flexible JavaScript/TypeScript library for building user interfaces created by Facebook. It lets the user compose complex UIs from small and isolated pieces of code called “components”.

Advantages

- React allows users to reuse components, which saves time and readability.
- Code is easier to maintain and more flexible due to its modular structure.
- Improved performance due to virtual DOM.

Disadvantages

- High pace of development – environment continually changes.

3.2.5 Ant Design

Ant Design [20] is an open source React UI library written in TypeScript. It comes with a set of high-quality React components and theme customization capability.

Advantages

- Polished look and feel.
- Built using static typing.
- Excellent form handling.

Disadvantages

- High pace of development.

3.2.6 Redux

Redux [6] is an open-source JavaScript library for managing and centralizing application state. It has a simple, limited API designed to be a predictable container for application state.

Advantages

- Predictable state, which makes it possible to implement tasks like undo/redo.
- Easy to debug and test.

Disadvantages

- Requires a lot of boilerplate code.

3.2.7 Selenium WebDriver

Selenium WebDriver [16] is one of the most popular development tools utilized in Web UI automation. It has no direct interaction with the web components on a webpage. A browser-specific driver serves as a conduit between the test script and the web browser. Then, Selenium locators are used to locate components on a page so that relevant methods for interacting with the element may be employed.

Advantages

- Support for a variety of programming languages - offers native bindings for JavaScript, Python, Java, C#, Ruby, and Kotlin.
- Platform independent - compatible with Windows, Linux, macOS, Android (with Selenium Appium or Robotium), and iOS (with iOS-driver or Appium).
- Cross-browser - supports all the major browsers, such as Chrome/Chromium, Safari, Opera, Firefox, and Edge.

Disadvantages

- Only used for web-based apps - it cannot be used to automate desktop application testing since it cannot recognize desktop app objects. It is solely intended for testing web applications with the browsers listed above.
- High test maintenance - the framework's reliance on a single, rigid element identifier may lead to fragile tests. In a situation when an element's identifier in the user interface changes, the tests may break.
- Steep learning curve - requires knowledge and understanding of at least one of aforementioned programming languages. It does not allow for codeless testing, as offered by other existing tools.

3.2.8 pytest

As a comprehensive Python testing tool, *pytest* [17] may be used for numerous sorts of software testing at various levels of abstraction. It is now one of the most widely used testing frameworks. It has sophisticated capabilities like "assert" rewriting, a third-party plugin model, and a powerful yet simple fixture model.

Advantages

- Flexible and easy to use fixtures allow for test parametrization and help minimizing the boilerplate code.
- Extensible - pytest can be effortlessly extended with a variety of hooks and plugins. They could enhance its functionalities by introducing parallel test execution or just explicitly marking dependencies and the order in which the test cases should be run.
- Compact test suites - pytest established the idea for the tests to be simple Python functions, rather than forcing developers to incorporate their tests within major test classes.
- Can be easily integrated with *Selenium WebDriver*, described in section 3.2.7.

Disadvantages

- Incompatibility with other testing frameworks - because pytest requires tests to be written using its own proprietary techniques, it unavoidably sacrifices convenience for compatibility. In other words, building tests for pytest binds the programmer to it, and the only way to use another testing framework is to rewrite the majority of the code.

3.3 Relational database management system

3.3.1 Postgres

Postgres, or PostgreSQL[15], is a reliable, free and open source database management system with over 30 years of active development and strong community.

Advantages

- Comprehensive documentation and active community.
- No license purchase required.
- Support for geographic objects, enabling use as a geospatial data store.

Disadvantages

- Major version updates may require long downtimes for large databases.
- High deployment cost on major cloud providers (e.g., Microsoft Azure).

Chapter 4





Code Structure

This chapter delves into the code's structure, including the structuring of the modules and the usage of several programming languages. It gives a high-level overview of the project directory as well as a deep look into each part of the system.

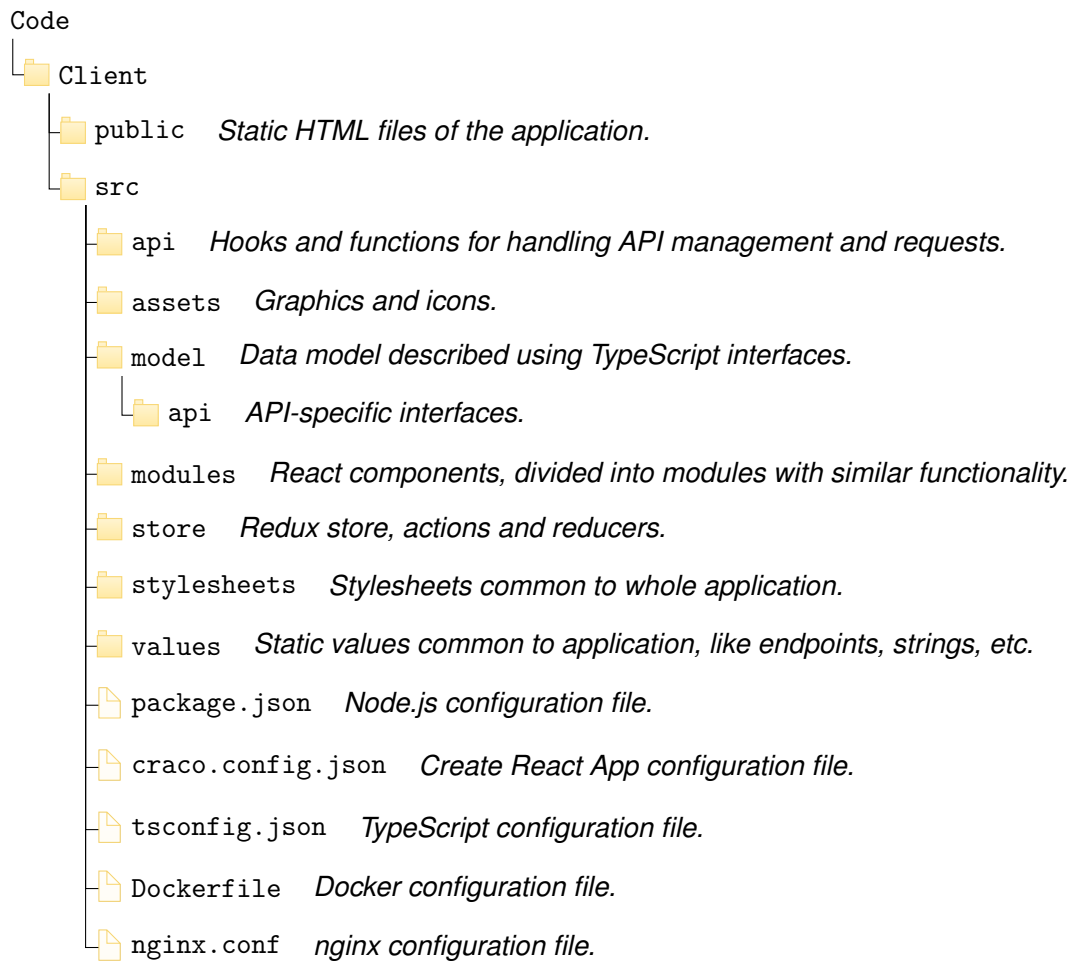
4.1 High Level Overview

For the reasons described in chapter 3 the server application was implemented using C# programming language and the client application using TypeScript. The high level overview of the code directory contents reflects the division into the client and server application code base. Additionally, a folder for the system tests written in Python was introduced. Finally, the directory contains a docker compose file required for launching the application using docker containers.

Code

-  Client *Source code of the Client-side application.*
-  Server *Source code of the Server-side application.*
-  system_tests *Source code of the system tests.*
-  docker-compose.yml *Docker Compose configuration file.*

4.2 Client



4.3 Server

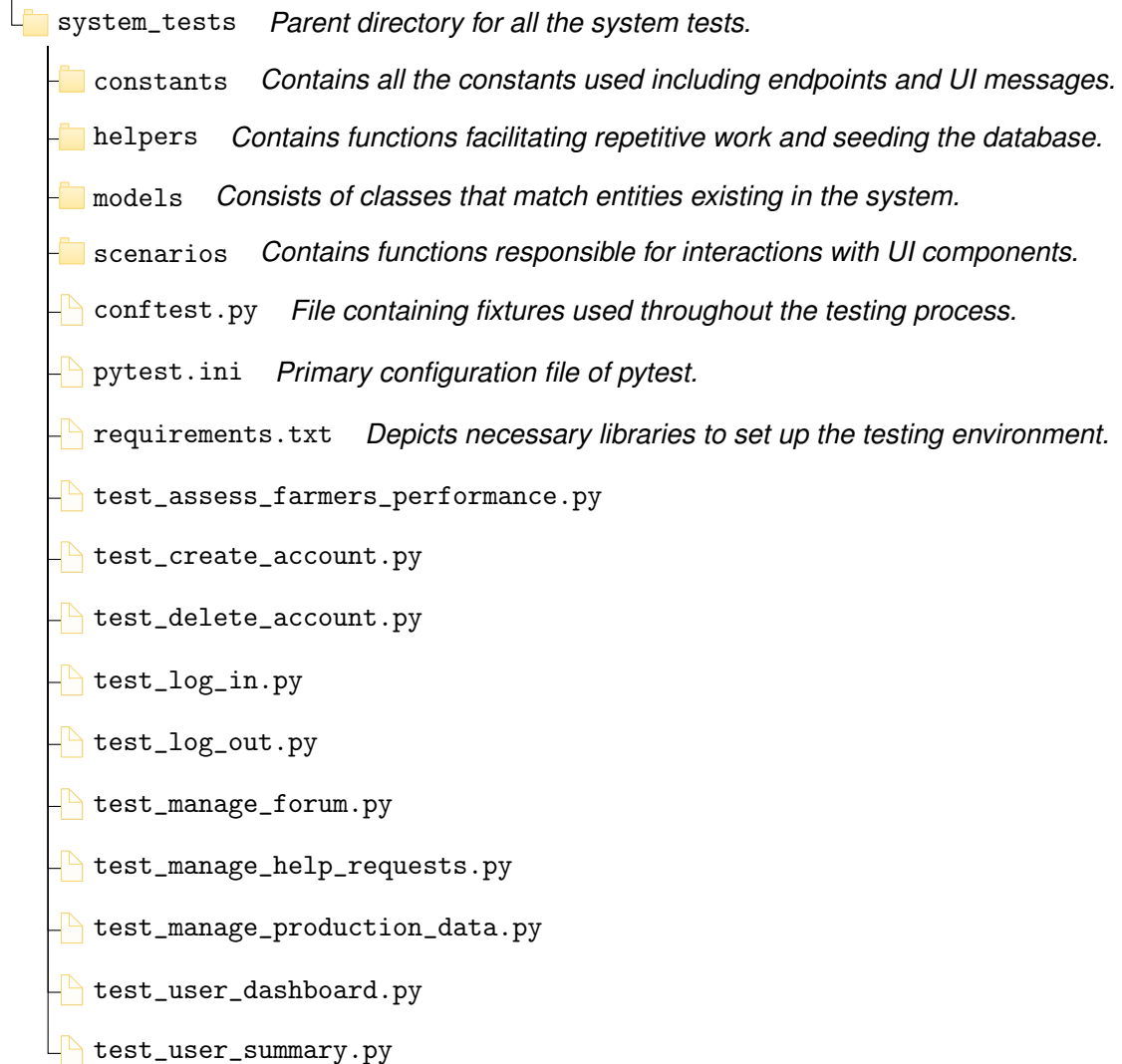
Code



4.4 System Tests

The directory tree in which the system tests were organized is depicted below. All the tests were written in *Python* using *Selenium WebDriver* (see section 3.2.7) and *pytest* (section 3.2.8). A thorough description of system tests together with their results is to be found in section 5.2.

Code



Chapter 5

Testing

As stated in the DD, the project verification approach was planned to be centered on three different types of tests: unit, integration, and system tests. In the end, however, unit tests were omitted in favor of gaining more time for detailed integration and system tests.

Furthermore, *GitHub Actions* was the platform utilized for continuous integration. The analysis of the correctness of the code, its format, building it on a container designed for this purpose, and the automated firing of system tests with each new code delivery substantially enhanced project quality management. As a result, the faulty or unstable code was unable to be pushed to the main repository branch. All the developed tests were triggered by each push to a release branch, pull request to the main branch, or just manually.

5.1 Integration Testing

The integration testing were performed manually and were focused on Server's API testing using *Swagger* interface. The test cases are described below. For the sake of brevity, the test cases descriptions of GET endpoints were omitted since they were tested together with corresponding POST, DELETE and PUT endpoints.

Additionally, each endpoint apart from the one in *account* path was tested to ensure that the server returns 401 status code for all requests performed without a valid JWT token.

5.1.1 POST /api/account/registration/farmer

Test Case Objective	Test Case Description	Expected Result
Do not allow registering farmer with email already assigned to another account.	Register two farmer's accounts with the same email.	Server responds with 400 error code and description of the error.

Do not allow registering farmer with mandal name not existing in the GET mandals endpoint.	Register a farmer inserting a random string inside the mandal field.	Server responds with 400 error code and description of the error.
Do not allow registering farmer with waterIrrigation-SystemId already used, entered by another user.	Register two accounts with the same waterIrrigationSystemId	Server responds with 400 error code and description of the error.
Do not allow registering farmer with sensorSystemId already used, entered by another user.	Register two accounts with the same sensorSystemId	Server responds with 400 error code and description of the error.
Do not allow registering farmer with data missing in one of the following FarmAddressLine1, FarmCity, FarmPostalCode, Mandal, Name, Surname, Password or Email.	Register with missing data in one of the required fields.	Server responds with 400 error code and description of the error.
Allow registering a farmer that entered valid data.	Registering a farmer using valid data. Try to log in using email and password used during registration.	Server responds with status code 200 for both registration and log in request.

5.1.2 POST /api/account/registration/policy-maker

Test Case Objective	Test Case Description	Expected Result
Do not allow registering a policy maker with email already assigned to another account.	Register two accounts with the same e-mail.	Server responds with 400 error code and description of the error.
Do not allow registering farmer with data missing in one of the following fields: Name, Surname, Password, or Email.	Register with missing data in one of the required fields.	Server responds with 400 error code and description of the error.

Allow registering a policy maker that entered valid data.	Register a policy maker using valid data. Log in using email and password used during registration.	Server responds with status code 200 for both registration and log in request.
---	---	--

5.1.3 POST /api/account/login

Test Case Objective	Test Case Description	Expected Result
Do not allow log in with e-mail that does not exist.	Log in with e-mail that does not exist.	Server responds with 400 error code and description of the error.
Do not allow log in with invalid credentials.	Log in with invalid credentials.	Server responds with 400 error code and description of the error.
Allow login in with valid credentials.	Log in with valid credentials	Server responds with 200 error code and JWT token inside the response body.

5.1.4 DELETE /api/account/{id}

Test Case Objective	Test Case Description	Expected Result
Do not allow account deletion with invalid userId in request route.	Delete account, entering invalid userId.	Server responds with 404 error code and description of the error.
Do not allow account deletion with invalid credentials.	Delete account, entering invalid credentials.	Server responds with 400 error code and description of the error.
Allow account deletion when entering valid credentials and userId in request route.	Delete account, entering valid credentials and userId in request route. Try to log on the deleted account.	Server responds with status code 200 to the delete request. For the second request, the server responds with status code 404.

5.1.5 POST/api/farmer/{farmerId}/note

Test Case Objective	Test Case Description	Expected Result
Do not allow assigning a note that is not one of the specified enum values: Neutral, Positive, Negative.	Use header with a JWT token belonging to a policy maker, perform a request to assign a note filling note field with a random string.	Server responds with 400 error code and description of the error.
Do not allow assigning a negative note without specifying a valid problem type.	<ol style="list-style-type: none">1. Use header with a JWT token belonging to a policy maker, perform a request to assign a negative note filling a problem type with a random string.2. Use header with a JWT token belonging to a policy maker, perform a request to assign a negative note leaving a problem type empty.	In both cases, the server responds with 400 error code and description of the error.
Do not allow assigning a note with invalid farmerId in request route.	Use header with a JWT token belonging to a policy maker, perform a request to assign a note specifying a farmerId that does not exist.	Server responds with 404 error code and description of the error.
Do not allow assigning a note by a user that is not a policy maker (or a not logged-in user).	<ol style="list-style-type: none">1. Use header with a JWT token belonging to a farmer and perform a request to assign a note using valid input data.2. Perform a request to assign a negative note, leaving a problem type empty (no JWT token in headers).	In the first case, the server responds with 403 error code and 401 in the second.
Allow policy maker to assign a note to a farmer.	Use header with a JWT token belonging to a policy maker, perform a request to assign a note to existing farmer using valid input. Perform GET to retrieve the list of notes for the same farmer.	The server responds with status code 200 and ID of created note. The note with given ID exists in the list in the GET endpoint for notes.

Create an automatic help request to another farmer with a positive note when assigning a negative note.	Use header with a JWT token belonging to a policy maker, perform a request to assign a positive note to existing farmer using valid input. Then, assign a negative note to another farmer that is in the same mandal as the one with a positive note. Perform GET to /API/requests with RecipientUserId set to UserId of farmer with positive note.	The last, GET request returns a requests list in which one of the requests has field IsAutomatic set to true, the topic set to the problemTypeName specified during negative note assignment and the description that informs that it is an automatic help request created due to negative note assignment.
---	---	---

5.1.6 POST /api/farmer/farm/production-data

Test Case Objective	Test Case Description	Expected Result
Do not allow adding a production data with invalid production type.	Use a header with a JWT token belonging to a farmer and perform a request to add production data using a random string as a production type.	Server responds with 400 error code and description of the error.
Do not allow adding a production data by a user that is not a farmer.	Use a header with a JWT token belonging to a policy maker and perform a request to add production data using valid input data.	Server responds with 403 error code and description of the error.
Allow adding a production data with valid input data.	Use header with a JWT token belonging to a farmer and perform a request to add production data using a valid input data. Perform GET to retrieve the list of production data of the farmer.	Server responds with status code 200 and ID of added production data. The production data with given ID exists in the list of the GET response.

5.1.7 POST /api/forum/thread

Test Case Objective	Test Case Description	Expected Result
Do not allow creating a forum thread by a user who is not a farmer.	Use a header with a JWT token belonging to a policy maker and perform a request to add forum thread using valid input data.	Server responds with 403 error code and description of the error.
Do not allow creating a forum thread without a topic specified.	Use header with a JWT token belonging to a farmer and perform a request to add forum thread without a topic specified.	Server responds with 400 error code and description of the error.
Allow creating a forum thread by a farmer.	Use header with a JWT token belonging to a farmer and perform a request to perform a request to add forum thread using a valid input data. Perform GET to retrieve the forum thread by ID.	Server responds with status code 200 and ID of forum thread. The GET request responds with a forum thread created in the POST request.

5.1.8 DELETE /api/forum /comment /{commentId}

Test Case Objective	Test Case Description	Expected Result
Do not allow deleting a forum comment by a user different from its author.	Use header with a JWT token belonging to a farmer and perform a request to delete a forum comment created by another user.	Server responds with 403 error code and description of the error.
Do not allow deleting a comment with invalid commentId in request route.	Use header with a JWT token belonging to a farmer, perform a request to delete a comment specifying a commentId that does not exist.	Server responds with 404 error code and description of the error.

Allow deleting a forum comment by its author.	Use header with a JWT token belonging to a farmer and perform a request to delete a forum comment created by him. Perform GET to retrieve the forum thread related to deleted comment.	Server responds with status code 200 to the delete request. The comment does not exist in the comments list returned to the GET response
---	--	--

5.1.9 POST /api/requests

Test Case Objective	Test Case Description	Expected Result
Do not allow creating a help request with one of the following fields empty: topic, description.	Use a header with a JWT token belonging to a farmer and perform a request to create a help request with one of the required fields empty.	Server responds with 400 error code and description of the error.
Do not allow creating a help request by a user who is not a farmer.	Use header with a JWT token belonging to a policy maker and perform a request to create a help request using valid input data.	Server responds with 403 error code and description of the error.
Allow creating a help request by farmer with valid input data.	Use header with a JWT token belonging to a farmer and perform a request to create a help request using valid input data. Perform GET to retrieve the created help request by ID.	Server responds with status code 200. The GET request returns the previously created help request.

5.1.10 POST /api/requests/{requestId}/response

Test Case Objective	Test Case Description	Expected Result
Do not allow creating a help response with empty message field.	Use header with a JWT token belonging to a farmer and perform a request to create a help response with message field empty.	Server responds with 400 error code and description of the error.

Do not allow creating a help response with invalid requestId in request route.	Use a header with a JWT token belonging to a farmer that and perform a request to create a help response specifying a requestId that does not exist.	Server responds with 404 error code and description of the error.
Do not allow creating a help response by a user that was not a recipient of given help request.	Use header with a JWT token belonging to a farmer and perform a request to create a help request using valid input data. Create a new farmer account and log in. Use a header with a JWT token obtained from the log in endpoint and perform a request to create a help response using valid input data to a help request created in the first step.	Server responds with 400 error code and description of the error.
Allow creating a help response by a farmer with valid input data.	Use a header with a JWT token belonging to a farmer and perform a request to create a help response using valid input data. Perform GET to retrieve the created help request by ID.	Server responds with status code 200. The GET request returns given help request with previously created help response.

5.1.11 PUT /api/requests/{requestId}

Test Case Objective	Test Case Description	Expected Result
Do not allow updating a help request with one of the following fields empty: topic, description.	Use a header with a JWT token belonging to a farmer and perform a request to update a help request with one of the required fields empty.	Server responds with 400 error code and description of the error.
Do not allow updating a help request by a user who is not an author of the help request.	Use header with a JWT token belonging to a farmer who is not an author of the help request and perform a request to update a help request using valid input data.	Server responds with 403 error code and description of the error.

Do not allow updating an automatic help request.	Use a header with a JWT token belonging to a farmer and perform a request to update an automatic help request created on behalf of that farmer using valid input data.	Server responds with 400 error code and description of the error.
Do not allow updating a help request with invalid requestId in request route.	Use header with a JWT token belonging to a farmer that and perform a request to update a help request specifying a requestId that does not exist.	Server responds with 404 error code and description of the error.
Allow updating a help request with valid input data by its author.	Use header with a JWT token belonging to a farmer and perform a request to update a help request created by him using valid input data. Perform GET to retrieve the updated help request by ID.	Server responds with status code 200. The GET request returns the help request with updated fields.

5.1.12 PUT /api/requests/response/{responseld}

Test Case Objective	Test Case Description	Expected Result
Do not allow updating a help response with empty message field.	Use a header with a JWT token belonging to a farmer and perform a request to update a help response, putting empty string in the message field.	Server responds with 400 error code and description of the error.
Do not allow updating a help response by a user who is not an author of the help response.	Use a header with a JWT token belonging to a farmer who is not an author of the help response, and perform a request to update a help response using valid input data.	Server responds with 403 error code and description of the error.
Do not allow updating a help response with invalid responseld in request route.	Use a header with a JWT token belonging to a farmer that and perform a request to update a help response specifying a responseld that does not exist.	Server responds with 404 error code and description of the error.

Allow updating a help response with valid input data by its author.	Use a header with a JWT token belonging to a farmer and perform a request to update a help response using valid input data. Perform GET to retrieve the help request to which the response was assigned.	Server responds with status code 200. The GET request returns the help request with response list that contains the modified response with updated fields.
---	--	--

5.1.13 DELETE /api/requests/{id}

Test Case Objective	Test Case Description	Expected Result
Do not allow deleting a help request by a user who is not its author.	Use header with a JWT token belonging to a farmer and perform a request to delete a help request that was not created by that farmer.	Server responds with 403 error code and description of the error.
Do not allow deleting an automatic help request.	Use header with a JWT token belonging to a farmer on whose behalf an automatic was help request was created. Perform a request to delete that help request.	Server responds with 400 error code and description of the error.
Do not allow deleting a help request with invalid id in request route.	Use header with a JWT token belonging to a farmer that and perform a request to delete a help request specifying an id of a help request that does exist.	Server responds with 404 error code and description of the error.
Allow deleting a help request by its author.	Use a header with a JWT token belonging to a farmer and perform a request to delete a help request that was created by the same farmer. Perform GET to retrieve the help request list.	Server responds with status code 200. The GET request returns the help request list. The deleted request is not present in the returned list.

5.1.14 DELETE /api/requests/response/{id}

Test Case Objective	Test Case Description	Expected Result
Do not allow deleting a help response by a user who is not its author.	Use a header with a JWT token belonging to a farmer who is not an author of the help response, and perform a request to delete a help response using valid input data.	Server responds with 403 error code and description of the error.
Do not allow deleting a help response with invalid id in request route.	Use a header with a JWT token belonging to a farmer and perform a request to delete a help response specifying an id of a help response that does exist.	Server responds with 404 error code and description of the error.
Allow deleting a help response by its author.	Use a header with a JWT token belonging to a farmer and perform a request to delete a help response that was created by the same farmer. Perform GET to retrieve the help request to which the response was assigned.	Server responds with status code 200. The GET request returns the help request with responses list. The deleted response is not present in the responses list.

5.2 System Testing

System tests were designed to ensure that the whole system is working properly. They were performed in a controlled environment, with both the client and server running in the background through the means of the dockerized environment.

System tests were written in *Python* using *pytest* and *Selenium WebDriver* frameworks. These two allowed to test the entire system in a fully automated manner. The following sections describe the implemented test cases, which match the system's use cases previously defined in RASD.

5.2.1 Account creation

Use cases: *Create account, Insert role specific data.*

Account creation process was tested for both actors: farmers and policy makers. The test cases follow a series of steps: opening of the registration form, filling in the common fields, selecting a role (farmer or policy maker), specifying role specific data (optional, for farmers only), and lastly submitting the form. The tests verify if a correct success notification appeared on the screen.

Additionally, error handling was also verified through testing if it is possible to create two accounts providing the exact same data. Similarly to the positive test cases, a popup of the appropriate notification was verified.

Test results

```
test_create_account.py::test_create_policy_maker_account PASSED [ 15%]
test_create_account.py::test_create_two_policy_makers_with_the_same_data_with_error PASSED [ 18%]
test_create_account.py::test_create_farmer_account PASSED [ 20%]
test_create_account.py::test_create_two_farmers_with_the_same_data_with_error PASSED [ 22%]
```

5.2.2 Log in

Use cases: *Log in.*

This group of tests focuses on verification of the logging in and logging out functionality. As the previous group, the tests go through the scenarios for both actors. The steps look as follows: opening the log in dialog, filling in email and password, and waiting for a redirection to user's dashboard.

Furthermore, negative scenarios were tested as well. They include an attempt to log in by an unregistered user and to log in without providing email nor password. The appearance of appropriate error messages was checked.

Test results

```
test_log_in.py::test_log_in_policy_maker PASSED [ 29%]
test_log_in.py::test_log_in_farmer PASSED [ 31%]
test_log_in.py::test_log_in_unregistered_user_with_error PASSED [ 34%]
test_log_in.py::test_log_in_no_data_with_error PASSED [ 36%]
test_log_out.py::test_log_out_policy_maker PASSED [ 38%]
test_log_out.py::test_log_out_farmer PASSED [ 40%]
```

5.2.3 Delete account

Use cases: *Delete account.*

The functionality of deleting an account for both the policy maker and farmer was also tested. The test cases consist of: opening of user's summary, opening delete account form, filling in the form, and then submitting it. Tests assert that the success message appeared, and the user was logged out.

Test results

```
test_delete_account.py::test_delete_account_policy_maker PASSED [ 25%]
test_delete_account.py::test_delete_account_farmer PASSED [ 27%]
```

5.2.4 Forum

Use cases: *View forum, Create forum thread, View forum thread, Add forum comment, Delete forum comment.*

This group of tests verifies the functionalities related to the forum. The actions of opening the forum view, creating a new thread, viewing a forum thread, adding comments in it, and also deleting these comments were all tested.

In case of opening the forum view or a particular thread, redirections to correct addresses were asserted. Creation of a forum thread or just addition of a comment were verified by checking if they eventually appeared either in the forum view or inside a particular thread. For the case of deletion of a comment, the tests verified if it is possible to go through the whole process without receiving any warning.

Moreover, the appearance of error messages in case of attempts to create a forum thread or adding a forum comment without providing any data was also checked.

Test results

```
test_manage_forum.py::test_open_forum_view PASSED [ 43%]
test_manage_forum.py::test_create_forum_thread PASSED [ 45%]
test_manage_forum.py::test_create_forum_thread_no_data_with_error PASSED [ 47%]
test_manage_forum.py::test_view_forum_thread PASSED [ 50%]
test_manage_forum.py::test_add_comment_to_forum_thread PASSED [ 52%]
test_manage_forum.py::test_add_comment_to_forum_thread_no_comment_with_error PASSED [ 54%]
test_manage_forum.py::test_delete_forum_comment PASSED [ 56%]
```

5.2.5 Assess farmer's performance

Use cases: *Assess farmer's performance.*

These tests focus on the policy maker's functionality that allows him to assess farmer's performance via changing his note. They verify if it is possible to open the summary of a chosen farmer by asserting that the policy maker was redirected to a valid address. Subsequently, the assignment of a positive, neutral, and negative note were all verified. In case of a negative note, the tests ensured that it was mandatory to select a problem type. In all the cases, appearing messages were verified. They were either positive for successful scenarios, or negative if no problem type was specified.

Test results

```
test_assess_farmers_performance.py::test_open_farmers_view PASSED [ 2%]
test_assess_farmers_performance.py::test_open_farmers_summary PASSED [ 4%]
test_assess_farmers_performance.py::test_give_positive_note PASSED [ 6%]
test_assess_farmers_performance.py::test_give_neutral_note PASSED [ 9%]
test_assess_farmers_performance.py::test_give_negative_note PASSED [ 11%]
test_assess_farmers_performance.py::test_give_negative_note_no_problem_type_with_error PASSED [ 13%]
```

5.2.6 Help requests

Use cases: *View list of my requests, Request help, View my request, Delete request, View list of requests, Create response, Delete response.*

This group of tests was responsible for the verification of all the functionalities related to help requests. Tests verified if a farmer can see a list of his help requests, a list of incoming help requests (in case he has a positive note) as well as each particular request either created by him or an incoming one by verifying if he was redirected to an appropriate address. Subsequently, the processes of creation of a help request or responding to an incoming one as well as deletion of a response were tested.

Test results

```
test_manage_help_requests.py::test_open_my_help_requests_view PASSED [ 59%]
test_manage_help_requests.py::test_create_help_request PASSED [ 61%]
test_manage_help_requests.py::test_create_help_request_no_data_with_error PASSED [ 63%]
test_manage_help_requests.py::test_view_my_help_request PASSED [ 65%]
test_manage_help_requests.py::test_open_provide_help_view PASSED [ 68%]
test_manage_help_requests.py::test_view_help_request PASSED [ 70%]
test_manage_help_requests.py::test_respond_to_help_request PASSED [ 72%]
test_manage_help_requests.py::test_delete_help_response PASSED [ 75%]
test_manage_help_requests.py::test_delete_help_request PASSED [ 77%]
```

5.2.7 Manage production data

Use cases: *Manage production data.*

The process of management of the production data by the farmer was also thoroughly tested. The tests asserted that a farmer can open his production data view, which results in address change. Furthermore, the processes of addition, editing, and deletion of either one entry or all production data on one page were tested. The asserts verified if the messages shown in system's notifications in case of deletion were appropriate. When checking the addition of a new entry or editing of an existing one, the tests verified if the latest production data entry was correct.

Test results

```
test_manage_production_data.py::test_open_production_data_view PASSED [ 79%]
test_manage_production_data.py::test_add_production_data PASSED [ 81%]
test_manage_production_data.py::test_add_production_data_no_data_with_error PASSED [ 84%]
test_manage_production_data.py::test_delete_production_data PASSED [ 86%]
test_manage_production_data.py::test_edit_production_data PASSED [ 88%]
test_manage_production_data.py::test_delete_all_production_data PASSED [ 90%]
```

5.2.8 User dashboard

Use cases: *Display personal suggestions.*

This group of tests checked if it was possible for both actors to enter their specific dashboards. In case of the farmer, asserts verified that his summary, recent production data, help requests, and suggestions tailored for him were displayed, whereas in case of the policy maker they checked if two separate lists of farmers with positive and negative notes were presented.

Test results

```
test_open_user_dashboard.py::test_open_dashboard_policy_maker PASSED [ 93%]
test_open_user_dashboard.py::test_open_dashboard_farmer PASSED [ 95%]
```

5.2.9 Farmer's summary

Use cases: *View farmer's summary, Get information about soil humidity, Get information about weather forecasts, Get information about water usage.*

Similarly to the previous case, the verification of the user's summary was also tested for both actors. The tests verified that when opening the summary view, users were redirected to an appropriate address. Additionally, for the farmers, the existence of components displaying information about the soil humidity, water usage, and weather forecasts was checked.

Test results

```
test_open_user_summary.py::test_open_summary_policy_maker PASSED [ 97%]
test_open_user_summary.py::test_open_summary_farmer PASSED [100%]
```

Chapter 6

Installation Guide

This section contains instructions for installing and running the application successfully. It specifies all the conditions that must be met in order for the program to execute correctly.

The simplest way to run the application with just a single, simple command is by using docker compose. The prerequisites and the simple command is described in the section 6.1.

Moreover, the application can also be run without docker containers. In that case, however, the installation steps are much longer and complex. It requires:

- Postgres database installation (recommended version: 14.1)
 - Configured to accept requests at port 5432.
 - If necessary, adjust the connection string in *Code/Server/API/appsettings.json* (the Server address in the *appsettings* should be set to 127.0.0.1 instead of *postgres* as in the docker approach).
- Following the steps described in 6.2 for Server installation.
- Following the steps described in 6.3 for Client installation.

Depending on the installation approach choice, the client application will be available on:

- Port 1337 for the installation with docker containers.
- Port 3000 for the installation without docker containers.

6.1 Docker

Before executing the command depicted in 6.1 install the latest Docker Desktop version from the [official website](#). The minimum version of docker required is 4.4.4 (73704). After successful installation, open a terminal in the *Code* directory and run the command from the listing 6.1.

```
1 docker-compose up -d --build
```

Listing 6.1: Docker compose.

6.2 Server

The only prerequisite is installation of the .NET 6.0 SDK. Then, the installation requires execution of commands in a terminal opened inside the root project's directory presented in listings: 6.2, 6.3, 6.4, 6.5. It is important to preserve the order of listings.

```
1 cd Code/Server/API/  
2 dotnet restore  
3 dotnet build --configuration Release --no-restore
```

Listing 6.2: Install dependencies and build *API*.

```
1 cd Code/Server/BusinessLogic/  
2 dotnet restore  
3 dotnet build --configuration Release --no-restore
```

Listing 6.3: Install dependencies and build *BusinessLogic*.

```
1 cd Code/Server/DataAccess/  
2 dotnet build --configuration Release --no-restore
```

Listing 6.4: Install dependencies and build *DataAccess*.

```
1 cd Code/Server/  
2 dotnet publish --configuration Release  
3 cd ./API/bin/Release/net6.0/publish/  
4 dotnet run ./API.dll
```

Listing 6.5: Publish and run server application.

6.3 Client

Before executing npm commands, it is necessary to install node.js instance. Recommended versions: 14.17 or 16.13. Afterwards, open terminal in the root directory of the project, run the npm commands from the listing 6.6 should be executed.

```
1 cd Code/Client/  
2 npm install  
3 npm start
```

Listing 6.6: Install dependencies and build *Client*.

6.4 Running System Tests

The whole system testing process requires downloading and installation of [Python 3.10](#). This is the version used throughout the development process of the initial prototype of the system.

The second step is to install all the necessary packages. Those are listed inside *requirements.txt* file. See listing 6.7 for specific commands to execute.

```
1 cd Code/system_tests/  
2 pip install requirements -r requirements.txt
```

Listing 6.7: Install requirements.

Additionally, since *Selenium WebDriver* is employed, it is necessary to install a web browser. The one used in system tests is *Google Chrome*. Its latest stable release could be downloaded via the [official website](#) or a package manager as shown in the listing 6.8. Throughout the development process, system tests were executed in the headless mode (without visible browser's UI shell). This could be changed inside *conftest.py* configuration file.

```
1 sudo apt install google-chrome-stable
```

Listing 6.8: Install Google Chrome.

Finally, after all the previous steps are completed, it is possible to run the system tests using *pytest* as depicted in the listing 6.9. If it is needed to run only a particular subset of available test cases, please refer to [pytest documentation](#).

```
1 pytest
```

Listing 6.9: Run system tests.

Chapter 7

Effort Spent

Mariusz Wiśniewski

Topic	Hours	Date
Document structure	1	07.01.2022
Document: Introduction: purpose, scope, and structure	2	20.01.2022
System tests: environment setup	2	24.01.2022
Continuous integration	3	28.01.2022
Document: Product functions	1	27.01.2022
Document: Product functions justification	1	29.01.2022
Client: policy maker views implementation	4	30.01.2022
Client: farmer's summary view implementation	3	31.01.2022
Server: SQL queries to seed the database	1.5	31.01.2022
Document: Development frameworks citations	0.5	01.02.2022
System tests	8	01.02.2022
Document: Code structure	1.5	02.02.2022
Document: Testing and system testing description	2	03.02.2022
Client: change note modal	1.5	03.02.2022
System tests: production data, help requests, delete account, user summary	8	04.02.2022
Client: farmer's dashboard	3	04.02.2022
System tests: resolving todos, forum and dashboard tests, general cleanup	9	05.02.2022

Document: Installation guides, system testing, development frameworks	2	06.02.2022
Document: development frameworks, code structure and system testing	4	06.02.2022
Document: general adjustments	1	06.02.2022
System tests: resolving todos	3	06.02.2022
Total	62	

Józef Piechaczek

Topic	Hours	Date
Client: Project configuration	5	25.01.2022
Client: Homepage and Redux connectivity	4.5	26.01.2022
Client: Redux connectivity, Authentication components	7	26.01.2022
Client: Authentication components	4.5	27.01.2022
Client: Dashboard, Redux connectivity	3.5	28.01.2022
Client: Hooks for API integration, API definition	6	29.01.2022
Client: Routing, Requests and forum views	6.5	30.01.2022
Client: Requests and forum views, API integration - mandal search, account creation	7	31.01.2022
Client: Production data, request detailed views, API integration – login/logout	7	01.02.2022
Client: Data model, API integration for policy-makers – dashboard, farmers	4	04.02.2022
Client: API integrations for farmers, Docker configuration	7	05.02.2022
Client: API integrations for farmers – deleting of responses and help requests, visual changes	2	06.02.2022
Document: Frameworks and code structure description	2	06.02.2022
Total	66	

Kinga Marek

Topic	Hours	Date
Server: Docker compose for the database	1	30.12

Server: Project configuration and Data Access Layer implementation	2	31.12
Server: Component interfaces setup based on DD	2	04.01
Server: Swagger, automated migrations, user registration	3	28.01
Server: Dockerfile and docker compose setup, user login	3	29.01
Server: mandals and problem types seeders and endpoints, forum-related endpoints, help requests-related endpoints	5.5	30.01
Server: Exception middleware issues, production data-related endpoints, account deletion, help requests-related endpoints	8	31.01
Server: Suggestions seeder and endpoint, resolving exception middleware issues	6	01.02
Server: Resolving issues with http(s)	2	01.02
Server: GET endpoint for collecting data related to farmers	1.5	02.02
Document: Programming languages, C#	2	03.02
Server: Fixes required for frontend integration, C#	4	04.02
Document: Server code structure (plus time searching for how to generate multi-line directory labels)	2	04.02
Document: Chapter 3: TypeScript, FluentValidation, AutoMapper	3	05.02
Server: Minor fixes due to client integration	2	05.02
Document: Chapter 5: Integration Testing and server tests	5	05.02
Server: Validation fixes after integration tests	1	06.02
Document: Installation guide, code structure, review, resolving todos	8	06.02
Total	61	

References

- [5] *A tour of the C# language*. Jan. 2021. URL: <https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/> (visited on 02/03/2022).
- [6] Dan Abramov and Andrew Clark. *Redux*. June 2015. URL: <https://redux.js.org> (visited on 02/01/2022).
- [7] Microsoft Corporation. *.NET Tools & Editors*. URL: <https://dotnet.microsoft.com/en-us/platform/tools> (visited on 02/03/2022).
- [8] Microsoft Corporation. *Breaking changes reference overview*. Sept. 2021. URL: <https://docs.microsoft.com/en-us/dotnet/core/compatibility/breaking-changes> (visited on 02/03/2022).
- [9] Microsoft Corporation. *Entity Framework*. URL: <https://docs.microsoft.com/en-us/ef/> (visited on 02/04/2022).
- [10] Microsoft Corporation. *Nuget.org*. Jan. 2021. URL: <https://www.nuget.org/packages> (visited on 02/03/2022).
- [11] Microsoft Corporation. *TypeScript*. Oct. 2012. URL: <https://typescriptlang.org> (visited on 02/01/2022).
- [12] .NET Foundation. *AutoMapper*. URL: <https://automapper.org/> (visited on 02/06/2022).
- [13] .NET Foundation. *FluentValidation*. URL: <https://fluentvalidation.net/> (visited on 02/04/2022).
- [14] Python Software Foundation. *Python*. URL: <https://www.python.org/> (visited on 02/06/2022).
- [15] The PostgreSQL Global Development Group. *PostgreSQL*. <https://www.postgresql.org>. (Visited on 01/04/2022).
- [16] Jason Huggins. *The Selenium Browser Automation Project*. 2004. URL: <https://selenium.dev/> (visited on 01/24/2022).
- [17] Holger Krekel et al. *pytest 6.2*. 2004. URL: <https://github.com/pytest-dev/pytest> (visited on 01/24/2022).
- [18] Meta Platforms, Inc. *React*. May 2013. URL: <https://reactjs.org> (visited on 02/01/2022).
- [19] David Ramel. *.Net 6 is here*. Aug. 2021. URL: <https://visualstudiomagazine.com/articles/2021/11/08/net-6-arrives.aspx> (visited on 02/03/2022).

REFERENCES

- [20] Ant Design Team. *Ant Design*. 2015. URL: <https://ant.design> (visited on 02/01/2022).
- [21] Wikipedia contributors. *Object–relational mapping* — *Wikipedia, The Free Encyclopedia*. 2021. URL: https://en.wikipedia.org/w/index.php?title=Object%E2%80%93relational_mapping&oldid=1047542743 (visited on 05/01/2022).