POLITECNICO
MILANO 1863

DREAM - Data-dRiven PrEdictive FArMing in Telangana

Careddu Gianmario

La Greca Michele Carlo

Zoccheddu Sara

Prof. Elisabetta Di Nitto

Implementation and Testing Document

Version 1.0

5th February 2022

Deliverable: IT

Title: Implementation and Testing Document

Authors: Careddu Gianmario
La Greca Michele Carlo
Zoccheddu Sara

Version: 1.0

Date: 5th february 2022

Download page: https://github.com/gccianmario/Zoccheddu-LaGreca-Careddu

---

TABLE OF CONTENTS

1. INTRODUCTION

1.1. Purpose

The purpose of this document is to describe the implementation and testing of DREAM.
The code can be found at the repository hosted on GitHub, reachable at this link: https://github.com/gccianmario/Zoccheddu-LaGreca-Careddu

1.2. Revision history

- 5 February 2022, version 1.0
    - First IT implementation

1.3. Document Structure

- **Chapter 1** gives an introduction about the purpose of the document and reports a summary of the structure of the document

- **Chapter 2** listes the functionalities that are implemented in the software and describes the programming languages and development framework adopted to implement DREAM, analyzing both advantages and disadvantages of such choices.

- **Chapter 3** describes the source code structure, both for the frontend and the backend.

- **Chapter 4** is devoted to the testing part, and structure and procedure are explained.

- **Chapter 5** provides details on the installation procedure.

- **Chapter 6** shows the effort spent by each member of the group.

- **Chapter 7** includes the reference documents.

2. DEVELOPMENT

2.1. Implemented functionalities

The actual DREAM implementation involves the most important functionalities for what concerns the figures of farmers and policy makers, leaving out the figure of agronomists which are only described in the RASD and DD documents. This decision is based on the fact that the farmers are the core and the reason for the whole application, whereas policy makers are needed in order to monitor the general situation from above. As described in the DD, the development has been divided in cycles, we have finished the first cycle with the slight variation of postponing the harvest report functionalities in favor of the tip request functionalities shared with policy makers.
All the listed functionalities have been implemented in both frontend and backend, except when specified.

The functionalities that have been implemented for the users of type Farmer are:

- **Signing up**
- **Logging in**
- **Sending a HR to farmers**
- **Receiving a HR by a farmer**
- **Using HR-messages**
- **Use of the TRs**
- **Post a tip**
- **Post a question**
- **Voting inside the forum**
- **Post an answer to a question**

Regarding the users of type Policy maker, the functionalities that have been implemented are:

- **Signing up**
- **Logging in**
- **Sending a TR (only backend)**
- **Changing status of TRs**
- **Using TR-messages**

The admin users have access to a console where they can manage all the data present in the database. The most important features are:

- **Managing authorization codes**
- **Moderation of content and accounts**

2.2. Adopted development frameworks

As already explained in the DD document (section 2), the architectural design is based on three layers: front-end layer, back-end layer and data layer. The communication between the web application in the front-end and the server located in the back-end is handled by making use of a RESTful architecture and therefore this communication is stateless. All the advantages of using this kind of architectures are well explained in the DD document.
FInally, we remind that the implementation follows a MVC (Model-View-Controller) pattern. These and other design choices are described in section 2.6 and 2.7 of the DD document.

2.2.1. Programming language(s)

For what concerns the back-end, the adopted programming language was Python. We chose this programming language for its dynamicity and flexibility, and because it is one of the most used in web development. Dynamically typed, it is a programming language that focuses on rapid and robust development and it can be used for projects of practically any size.[11][12][13][14]

The main advantages of choosing Python are:

- **Simplicity:** Python is easy to use and read. Python has syntax that allows developers to write programs with fewer lines than some other programming languages.

- **Flexibility**: Python is a highly flexible programming language that can integrate a number of languages and implementations in the programming process.

- **Versatility:** Python can be treated in a procedural way, an object-oriented way or a functional way.

- **Libraries:** Python makes available a huge number of libraries that are suitable for almost anything.

- **Productivity**: Another advantage of Python is that this powerful programming language can increase productivity. Its integration features and control capabilities can enhance the productivity of enterprise software applications.

Python has some limitations as well. The main drawbacks of using this language are:

- **Speed limitations**: Python being an interpreted programming language is slower than other programming languages.

- **Threading Issues**: The Global Interpreter Lock (GIL) of Python doesn't allow executing more than one thread at a given time. This creates some limitations for the language.

- **Not native to mobile environment**: Android and iOS don’t support mobile computing in Python as an official programming language.
- **Memory consumption:** Python’s memory consumption is very high.

Regarding the front-end, we decided to adopt JavaScript, by making use of React, which is a declarative, efficient, and flexible JavaScript library useful for building user interfaces. More details about React will be described later in this section.
Regarding pros and cons of Javascript, the main advantages are:

- **Speed:** JavaScript tends to be very fast because it is often run immediately within the client's browser. So long as it doesn't require outside resources, JavaScript isn't slowed down by calls to a backend server. Also, major browsers all support JIT (just in time) compilation for JavaScript, meaning that there's no need to compile the code before running it.

- **Simplicity:** JavaScript's syntax was inspired by Java's and is relatively easy to learn compared to other popular languages like C++.

- **Popularity:** JavaScript is everywhere on the web, and with the advent of Node.js, is increasingly used on the backend.

- **Interoperability:** Unlike PHP or other scripting languages, JavaScript can be inserted into any web page. JavaScript can be used in many different kinds of applications because of support in other languages like Pearl and PHP.

- **Server Load:** JavaScript is client-side, so it reduces the demand on servers overall, and simple applications may not need a server at all.

- **Rich interfaces:** JavaScript can be used to create features like drag and drop and components such as sliders, all of which greatly enhance the user interface and experience of a site.

On the other hand, the drawbacks of JavaScript are:

- **Client-Side Security -** Since JavaScript code is executed on the client-side, bugs and oversights can sometimes be exploited for malicious purposes. Because of this, some people choose to disable JavaScript entirely.

- **Browser Support -** While server-side scripts always produce the same output, different browsers sometimes interpret JavaScript code differently. These days the differences are minimal, and you shouldn't have to worry about it as long as you test your script in all major browsers.

2.2.2. Python framework: Django REST Framework

2.2.2.1. Overview

In order to implement the backend, we decided to use Django REST Framework (DRF). Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. It's free and open source. The Django REST Framework is a package built on top of Django to create web APIs [15][16][17].
The main reasons for using Django are:

- **Many features included:** Django has a huge number of robust and steady free libraries that helps the developer in reusing as much code as possible.
- **Security**: Django includes many security features, such as prevention of common attacks like SQL injections and cross-site request forgery.
- **Scalable**: Django uses a "shared-nothing" architecture, which means it is possible to add hardware at any level – database servers, caching servers or web/application servers. The framework cleanly separates components such as its database layer and application layer and it ships with a simple-yet-powerful cache framework.

However, some drawbacks need to be mentioned:

- **Speed**: its architecture is combined with Python, which is not the fastest language around, so this may lead to slow websites.
- **Lack of convention**: in comparison to frameworks like Ruby on Rails, everything has to be explicitly defined, which leads to configuration boilerplate that may slow down the development process. On the other hand, relying on configuration is a common practice for the Python ecosystem.
- **Inability to simultaneously handle multiple requests**: unlike many modern web frameworks, Django is incapable of enabling individual processes to handle multiple requests simultaneously.

2.2.2.2. Structure

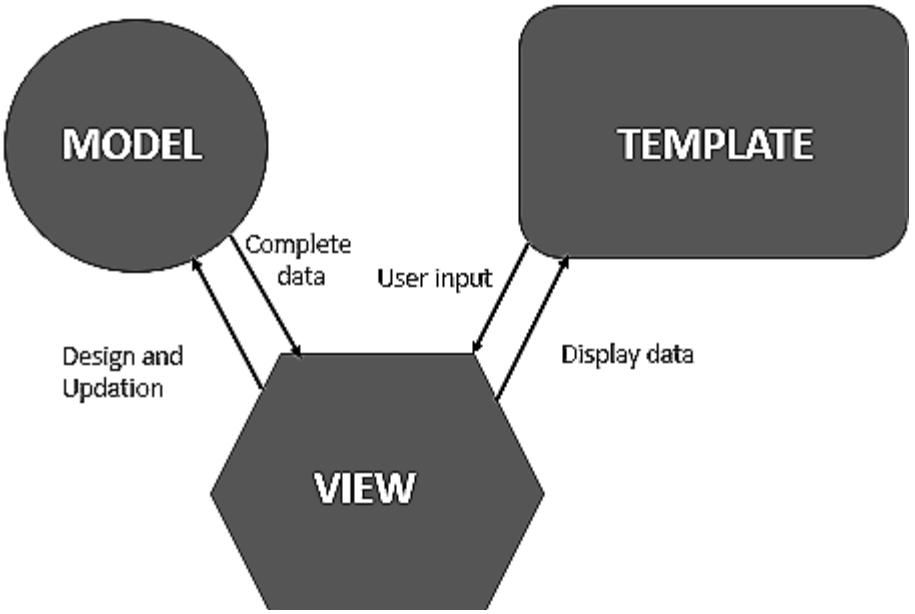
Django appears to be a MVC framework. Actually, Django does not exactly follow the MVC pattern, but it is based on MVT (Model-View-Template) architecture instead. MVT is a software design pattern for developing a web application [18].

The MVT Structure has the following three parts:

Model: The model is going to act as the interface of your data. It is responsible for maintaining data. It is the logical data structure behind the entire application and is represented by a database (generally relational databases such as MySql, Postgres).

View: The View is the user interface — what you see in your browser when you render a website. It is represented by HTML/CSS/Javascript and Jinja files.

Template: A template consists of static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted.



However with the DjangoREST framework the concept of template is replaced by the frontend application and the concept of view by the definition of functions associated to urls. This allows to move the computational complexity for displaying the data from the server to the client.

2.2.3. JavaScript library: React

React (also known as React.js or ReactJS) is a free and open-source front-end JavaScript library for building user interfaces based on UI components. It is the V(view part) in the MVC (Model-View-Controller) model [19][20][21].
The main reasons to use React are:

- **Suitable for creating web application**
  To create a dynamic web application specifically with HTML strings, React JS It provides less coding and gives more functionality. It makes use of the JSX(JavaScript Extension), which is a particular syntax letting HTML quotes and HTML tag syntax to render particular subcomponents. It also supports the building of machine-readable codes.

- **Reusable components**
  A ReactJS web application is made up of multiple components, and each component has its own logic and controls. These components are responsible for outputting a small, reusable piece of HTML code which can be reused wherever you need them. The reusable code helps to make your apps easier to develop and maintain. These Components can be nested with other components to allow complex applications to be built of simple building blocks. ReactJS uses a virtual DOM based mechanism to fill data in HTML DOM. The virtual DOM works fast as it only changes individual DOM elements instead of reloading complete DOM every time.

- **Improved performance**
  React uses Virtual DOM, thereby creating web applications faster. Virtual DOM

compares the components' previous states and updates only the items in the Real DOM that were changed, instead of updating all of the components again, as conventional web application

- **Unidirectional data flow**
  React follows a unidirectional data flow. This means that when designing a React app, developers often nest child components within parent components. Since the data flows in a single direction, it becomes easier to debug errors and know where a problem occurs in an application at the moment in question.

- **Scope for testing codes**
  ReactJS applications are extremely easy to test. It offers a scope where the developer can test and debug their codes with the help of native tools.

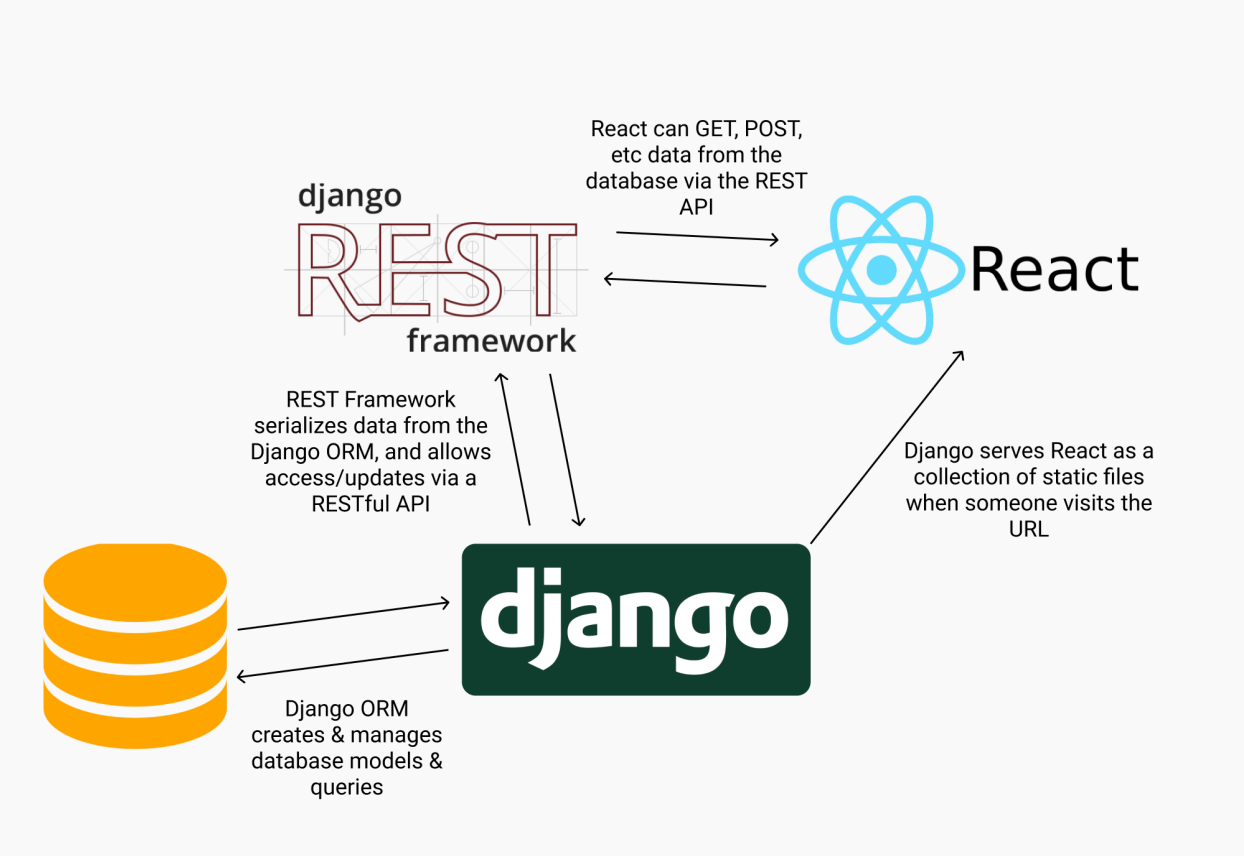
- **Interoperability**
  ReactJS components can be compiled with ReactDOM to run in browsers, but also with ReactNative to be executed in iOS and Android operating systems. The change from web to mobile is not automatic, but it can be done reusing the majority of the code.

However, there are some drawbacks that need to be listes:

- **Poor Documentation**
  React technologies are updating and accelerating so fast that there is no time to make proper documentation. To overcome this, developers write instructions on their own with the evolving of new releases and tools in their current projects.

- **View Part**
  ReactJS Covers only the UI Layers of the app and nothing else. So you still need to choose some other technologies to get a complete tooling set for development in the project.

- **Freedom and performances**
  React doesn't put any limitation to developers on what can be done client side, this is of course an advantage in term of development speed but if abused can lead to severe performance degradation client side.



2.2.4. API integrations: Postman

In order to test our application while implementing it, we made use of Postman Postman is an application used for API testing. It is an HTTP client that tests HTTP requests, utilizing a graphical user interface, through which we obtain different types of responses that need to be subsequently validated. Some of Postman's advantages include the collection feature and the possibility to create different testing environments. Postman is highly recommended for manual API testing, specifically for REST APIs [22][23].

3. SOURCE CODE

3.1. Frontend

As said before, the frontend was implemented by making use of ReactJS. React is based on the concept of atomic design. According to this design, the user interface is a hierarchical structure made of independent modules, arranged by role and structure, from the simplest component, the atom, to an entire page. The “Atomic Design” hierarchy correlates with levels of abstraction and reusability. A page, for instance, is a very concrete entity. It is not reusable and will not appear more than once in an app. Simpler components, atoms, molecules, and even organisms, may very well appear numerous times in a single app [24][25].
The main folders are found inside the *src* package. In particular, in the *components* package it is possible to find:

- **atoms (not used due to material UI component)**
  Atoms are the smallest possible components, such as buttons, titles, inputs or event color pallets, animations, and fonts. They can be applied on any context, globally or within other components and templates, besides having many states, such as this example of button: disabled, hover, different sizes, etc

- **molecules**
  They are the composition of one or more components of atoms. Here we begin to compose complex components and reuse some of those components. Molecules can have their own properties and create functionalities by using atoms, which don’t have any function or action by themselves. Molecules are relatively simple groups of UI elements functioning together as a single unit. For example, a form label, search input, and a button that forms a search form molecule. As mentioned earlier, this hierarchy continues all the way to full pages.

- **pages**
  Pages are the navigational parts of the application and it’s where the components are distributed in one specific template. The components get real content and they’re connected with the whole application. At this stage, we can test the efficiency of the

design system to analyze if all the components are independent enough or if we need to split them in smaller part

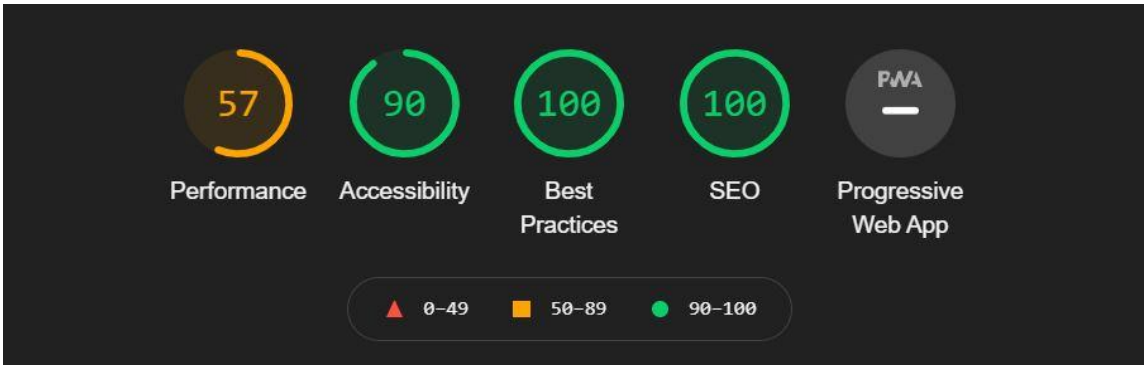
- **templates**
  In this state we stop composing components and begin to set their context. Moreover, the templates create relationships between the organisms and other components through positions, placements and patterns of the pages but it doesn't have any style, color or component rendered. That's why it looks like a wireframe.

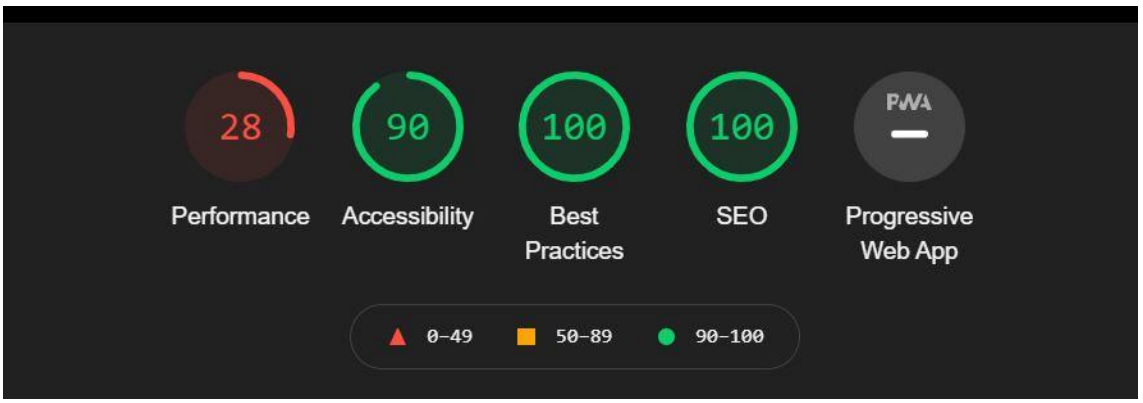
3.1.1. MUI library

Material UI is an open source library for React that provides a robust, customizable, and accessible set of foundational and advanced components, in particular MUI X, MUI Core and templates have been used for the frontend implementation.

3.1.2. Lighthouse benchmarking

The frontend has been analyzed with lighthouse, which is an open-source tool to evaluate website quality,, the following are the results obtained first for desktop version then for mobile. As expected by the design performances are not really high but the accessibility is high in both the platforms.





3.2. Backend

As said before, the backend of the application is implemented by means of the Django framework. The backend folder contains several subfolders, which are described below [26].

3.2.1. Core

The core contains the configuration files of the project. It is composed of the following files:

- *__init._*

This is an empty file. The function of this file is to tell the Python interpreter that this directory is a package and involvement of this __init.py_ file in it makes it a python project.

- *settings.py*

It contains the Django project configuration. The setting.py is the most important file, and it is used for adding all the applications and middleware applications. It is the main setting file of the Django project. It also contains **sqlite3** as the default database. Finally, it contains some pre-installed apps and middleware that are there to provide basic functionality.

- *serializers.py*

Serializers allow complex data such as querysets and model instances to be converted to native Python data types that can then be easily rendered into JSON, XML or other content types. In this file are defined general serializers.

- *urls.py*

URL is a universal resource locator, it contains all the endpoints that we should have for our website. It is used to provide the address of the resources (images, webpages, websites…)

- *wsgi.py*

WSGI stands for Web Server Gateway Interface, it describes the way servers interact with the applications. This server takes care of hosting the application once it is developed.

- *asgi.py*

ASGI works similar to WSGI but comes with some additional functionality. ASGI stands for Asynchronous Server Gateway Interface. It is now replacing its predecessor WSGI.

3.2.2. Applications

Following the components defined in DD section 2.2.3, the backend has been divided in the following django applications; a further subdivision has been performed to separate the model definition and the exposure of the data model through the Api. Some components instead have been merged. In practice each django app is described by a folder in the main directory. In the following all the developed apps and the mapping with the DD components is presented:

- **chat (Chat component):** this folder implements the functionality of exchanging messages between users;
- **forum** and **forum_api (Posting component, Voting component):** these folders contain all the implementation related to the forum, such as posting questions, answers and tips, and expressing likes and dislikes;
- **report** and **report_api (Report component)i**: these folders are devoted to the implementation of the harvest report;
- **request** and **request_api (Request component)***:* these folders contain the implementation of the help request and the tip request functionalities;

- **users (Security component, Access component)** : this folder delas with the functionality of authentication of the users and validations of the tokens associated to them.

Each application folder is characterized by the following files:

- *_init_.py*
  This file is an empty file and does not need any modifications. It just represents that the app directory is a package.

- *admin.py*
  Admin.py file is used for registering the Django models into the Django administration.
  It is used to display the Django model in the Django admin panel. It performs these major tasks: registering models, creating a Superuser, logging in and using the web application.

- *apps.py*
  Apps.py is a file that is used to help the user include the application configuration for their app. Users can configure the attributes of their application using the apps.py file.

- *models.py*
  Models.py represents the models of web applications in the form of classes. It is considered the most important aspect of the App file structure. Models define the structure of the database. It tells about the actual design, relationships between the data sets, and their attribute constraints.

- *serailizer.py*
  This file contains specific serializers for the application they refer to. They work as described for the Core folder.

- *tests.py*
  Tests.py allows the user to write test code for their web applications. It is used to test the working of the app.

- *urls.py*

Urls.py links the user's URL request to the corresponding pages it is pointing to.

- *views.py*
  Views provide an interface through which a user interacts with a Django web application. It contains all the views in the form of classes. We use the concept of *Serializers* for making different types of views(such as List Views, and Detail Views)

3.2.3. manage.py

This file is used as a command-line utility for our projects. It is used for debugging, deploying, and running web applications. The file contains the code for running the server, makemigrations or migrations, and several other commands as well, which we perform in the code editor. The command *runserver* is used to start the test server for our web application, provided by the Django framework. The command *makemigrations* is used to apply new migrations across projects and apps that have been carried out due to the changes in the database. Finally, *migrate* is used for making the changes to the modules in the database.

3.2.4. db.sqlite3

It is a local database file where all the generated data will be stored. It is a light database that Django offers.

3.2.5. Endpoints from the base url

It is important to note that some RESTFull best practice has been omitted on purpose to reduce the url length and the number of endpoints exposed.

```
admin/
api/ (13)
      categories/
      posting/question/
      posting/tip/
      posting/answer/
      reading/tips
      reading/questions
```



```
        voting/tip/like
        voting/tip/dislike
        voting/tip/remove-vote
        voting/answer/like
        voting/answer/dislike
        voting/answer/remove-vote
        reading/answers/

api/user (3)
  register/
   logout/blacklist/
   info/

api/request/ (7)
   sending_hr_farmer/
   all_hr/
   changing_status_hr_farmer/
   changing_status_tr/
   sending_tr/
   all_tr_farmer/
   all_tr_policymaker/

api/chat/ (4)
  load-tr-messages/
  send-tr-message/
  load-hr-messages/
  send-hr-message/

api/token/
api/token/refresh/
```


4. TESTING

As websites grow they become harder to test manually. Not only is there more to test, but, as interactions between components become more complex, a small change in one area can impact other areas, so more changes will be required to ensure everything keeps working and errors are not introduced as more changes are made. One way to mitigate these problems is to write automated tests, which can easily and reliably be run every time you make a change.

4.1. Backend testing

4.1.1. Overview

Django provides a test framework with a small hierarchy of classes that build on the Python standard `unittest` library. Despite the name, this test framework is suitable for both unit and integration tests. The Django framework adds API methods and tools to help test web and Django-specific behavior. These allow it to simulate requests, insert test data, and inspect the application's output.

To write a test, it is possible derive from any of the Django (or `unittest`) test base classes (`SimpleTestCase`, `TransactionTestCase`, `APITestCase`, `LiveServerTestCase`) and then write separate methods to check that specific functionality works as expected (tests use "assert" methods to test that expressions result in True or False values, or that two values are equal, etc.). When a test run starts, the framework executes the chosen test methods in the derived classes. The test methods are run independently, with common setup and/or tear-down behavior defined in the class [8].

4.1.2. Structure

To better understand the testing procedure, a brief description of the testing structure is necessary.

Inside every Django application, a file test.py is created, and all the procedures useful for the test of that specific application are placed in this file.

By importing `APITestCase`, it is possible to have a separate class for testing a specific use case, with individual test functions that test aspects of that use-case (for example, a class to test that a model field is properly validated, with functions to test each of the possible failure cases).

The new class defines methods that it is possible to use for pre-test configuration (for example, to create any models or other objects that are needed for the test) [8].

4.1.3. Procedure

Inside the `test.py` file of each Django application, a class `MessagesTestes(APITestCase)` is created from the already existing `APITestCase` class. This class contains all the elements to execute tests on that specific application. REST framework includes the following test case classes, that mirror the existing Django test case classes, but use APIClient instead of Django's default Client.

Inside this class, the first function that is implemented is `setUp()`. This function defines all the elements that are going to be used in the tests, such as model instances, methods and attributes. All these elements are saved in a temporary database instance that will live only when the test is run, and destroyed once the test is over.

After this function, several other functions are implemented, referring to functionality inside the application that are going to be tested.

Each of these functions will work in a similar way. A test url will be defined, referring to the endpoint that Django rest framework provides to the frontend. A request will be defined, and an user (defined in the setUP() function) will be authenticated through the method force_authentication. Then, a response will be fetched by calling the relative methods in the application view, containing the result of the operation.

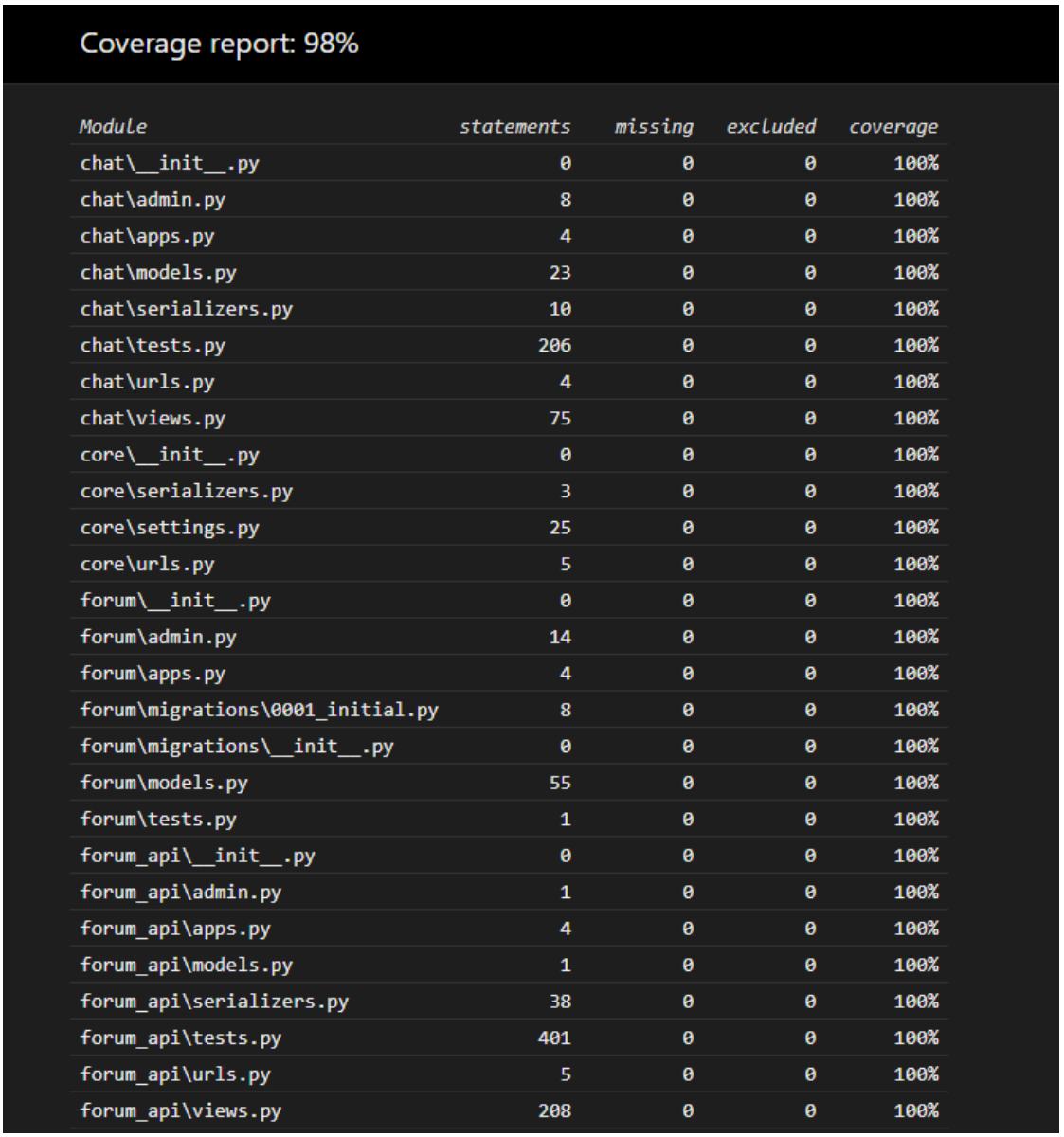
These functions use Assert functions to test whether conditions are true, false or equal (AssertTrue, AssertFalse, AssertEqual). If the condition does not evaluate as expected then the test will fail and report the error to your console. Since the response will contain the status code of the operation just performed, the assertion will be performed based on it [9].

4.1.4. Coverage

Coverage is used for measuring the effectiveness of tests, showing the percentage of the codebase covered by tests. Coverage can help offer suggestions on what should be tested.

After the installation, is it possible to run tests using this tool by executing the command: `coverage run manage.py test`

Coverage allows to check the state of the tests with reports, by executing the command: `coverage html`



Coverage report: 98%

Module	statements	missing	excluded	coverage
chat__init__.py	0	0	0	100%
chat\admin.py	8	0	0	100%
chat\apps.py	4	0	0	100%
chat\models.py	23	0	0	100%
chat\serializers.py	10	0	0	100%
chat\tests.py	206	0	0	100%
chat\urls.py	4	0	0	100%
chat\views.py	75	0	0	100%
core__init__.py	0	0	0	100%
core\serializers.py	3	0	0	100%
core\settings.py	25	0	0	100%
core\urls.py	5	0	0	100%
forum__init__.py	0	0	0	100%
forum\admin.py	14	0	0	100%
forum\apps.py	4	0	0	100%
forum\migrations\0001_initial.py	8	0	0	100%
forum\migrations__init__.py	0	0	0	100%
forum\models.py	55	0	0	100%
forum\tests.py	1	0	0	100%
forum_api__init__.py	0	0	0	100%
forum_api\admin.py	1	0	0	100%
forum_api\apps.py	4	0	0	100%
forum_api\models.py	1	0	0	100%
forum_api\serializers.py	38	0	0	100%
forum_api\tests.py	401	0	0	100%
forum_api\urls.py	5	0	0	100%
forum_api\views.py	208	0	0	100%

manage.py	14	2	0	86%
report\admin.py	5	0	0	100%
report\apps.py	4	0	0	100%
report\models.py	22	0	0	100%
report_api__init__.py	0	0	0	100%
report_api\admin.py	1	0	0	100%
report_api\apps.py	4	0	0	100%
report_api\models.py	1	0	0	100%
report_api\serializers.py	7	0	0	100%
report_api\tests.py	41	0	0	100%
report_api\urls.py	4	0	0	100%
report_api\views.py	13	0	0	100%
request__init__.py	0	0	0	100%
request\admin.py	8	0	0	100%
request\apps.py	4	0	0	100%
request\models.py	33	0	0	100%
request\tests.py	1	0	0	100%
request_api__init__.py	0	0	0	100%
request_api\admin.py	1	0	0	100%
request_api\apps.py	4	0	0	100%
request_api\models.py	1	0	0	100%
request_api\serializers.py	18	0	0	100%
request_api\tests.py	348	0	0	100%
request_api\urls.py	4	0	0	100%
request_api\views.py	140	0	0	100%
users__init__.py	0	0	0	100%
users\admin.py	20	0	0	100%
users\apps.py	4	0	0	100%
users\models.py	88	32	0	64%
users\serializers.py	68	0	0	100%
users\tests.py	151	0	0	100%
users\urls.py	4	0	0	100%
users\views.py	37	7	0	81%
Total	**2153**	**41**	**0**	**98%**

Coverage tests consist in testing all the files inside the Django applications, reporting if they had success or not, and giving information about possible errors in the code. The percentage of each file is given by the complete parts that have already been tested. Regarding DREAM backend, the total test percentage is 98%, and the 2% refers to parts of the system that are either not used explicitly or configuration elements [10].

4.1.5. Frontend integration testing

The frontend has been developed in parallel, when new features were available in the backend they have been gradually integrated in the frontend and tested on the flight. We have decided to not write integration testing in React, but testing by simulating manually the user experience, trying to simulate the “most atypical” user action and observing the system behavior.

5. INSTALLATION PROCEDURE

5.1. Project initialization

Following instructions are useful to initialize the DREAM project on a local machine.

- Git is required for cloning the project from GitHub. The git installation procedure is available in the following link: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git
- To clone the project from GitHub: `git clone https://github.com/gccianmario/Zoccheddu-LaGreca-Careddu`

5.2. Django installation

Following instructions are useful to correctly install the Django backend.

- Python is required. Information about the installation are provided in the following link: https://www.python.org/downloads/
- Firstly, it is necessary to go in the directory using the CMD terminal in Windows: `../Zoccheddu-LaGreca-Careddu\IT\dream-backend`
- To install Django: `pip install django`
- To install Django rest framework: `pip install djangorestframework`
- To install cors headers: `pip install django-cors-headers`
- To install coverage: `pip install coverage`
- To run the server: `python manage.py runserver`

Note: some packages may be already installed.

5.3. Django testing

Following instructions are useful to execute the testing of the backend.

- Firstly, it is necessary to go in the backend directory: `../Zoccheddu-LaGreca-Careddu\IT\dream-backend`
- To install coverage: `pip install coverage`
- To run tests: `coverage run manage.py test`
- To view a detailed report in the console, with informations about the testing percentage of the code: `coverage report -m`
- To view a html report, it is possible to run the following command, and it will be available in the folder `/htmlcov` (to visualize it, it is necessary to open the file index.html in a web browser): `coverage html`

5.4. React installation

- Download and install NodeJs from the following link: https://nodejs.org/en/, choose the option to automatic add to the PATH variable npm otherwise a further step of configuration will be needed for the next part
- Go to `../Zoccheddu-LaGreca-Careddu\IT\dream-frontend` with CMD terminal in Windows and run the following commands:
  `npm install` (to install all the dependencies)
  `npm start` (to run the web application on localhost:3000 by default)

5.5. Deployment

For the correct operation of the system in local environment the django backend has to run on http://127.0.0.1:8000/
Instead the React application has to run on http://localhost:3000/

5.6. Django admin and extra informations

The console admin can be used for data insertion and modification at the following link: http://127.0.0.1:8000/admin/
Maximum care has to be taken since the action performed in the console admin can lead to inconsistent state of the database especially for the creation of new users and their authentication.

It is possible to access to the server using the admin through the previous url, inserting the following credentials:

- Email address: *admin@gmail.com*
- Password: *admin*

(all the registered user has admin as password to simplify testing)

6. EFFORT SPENT

	Total time
Careddu Gianmario	86h
La Greca Michele Carlo	58h
Zoccheddu Sara	55h

7. REFERENCES

1. https://thefactfactor.com/facts/pure_science/biology/crops/2082/
2. https://smartbear.com/learn/performance-monitoring/what-is-mobile-first/
3. https://en.wikipedia.org/wiki/Don%27t_repeat_yourself
4. https://developers.google.com/web/updates/2019/02/rendering-on-the-web?hl=zh-cn
5. https://simplicable.com/new/thin-client-vs-thick-client
6. https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller
7. https://developer.mozilla.org/en-US/docs/Web/HTTP/Status
8. https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Testing
9. https://www.django-rest-framework.org/
10. https://docs.djangoproject.com/it/4.0/topics/testing/advanced/
11. https://thepythonguru.com/pros-cons-of-using-python-for-web-development/
12. https://www.w3schools.com/python/python_intro.asp
13. https://www.netguru.com/blog/python-pros-and-cons
14. https://www.freecodecamp.org/news/the-advantages-and-disadvantages-of-javascript/
15. https://www.benchmarkit.solutions/lets-understand-the-pros-and-cons-of-using-django/
16. https://www.djangoproject.com/
17. https://www.netguru.com/blog/django-pros-and-cons
18. https://www.geeksforgeeks.org/django-project-mvt-structure/
19. https://www.simform.com/blog/react-architecture-best-practices/#react-architecture-diagram
20. https://medium.com/geekculture/react-js-architecture-features-folder-structure-design-pattern-70b7b9103f22
21. https://www.javatpoint.com/pros-and-cons-of-react
22. https://www.encora.com/insights/what-is-postman-api-test
23. https://dzone.com/articles/postman-for-api-testing-pros-cons-and-alternative
24. https://danilowoz.com/blog/atomic-design-with-react
25. https://blog.bitsrc.io/sharing-react-components-from-atoms-and-molecules-to-pages-2d0d722b1dba
26. https://techvidvan.com/tutorials/django-project-structure-layout/

