Amit Peled, Daniel Piro, Shahar Alon, Nofar Rozenberg

BGU / MolOpt

Final Project in Software Engineering

Application Design Document (ADD)

Contents

[Use Cases 2](#_Toc124174024)

[1.1 Users Use-Cases 2](#_Toc124174025)

[1.2 Runs/Algorithms Use-Cases 5](#_Toc124174026)

[System Architecture 7](#_Toc124174027)

[2.1 Software Components 7](#_Toc124174028)

[2.2 Target Machine 7](#_Toc124174029)

[2.3 Components Interaction 7](#_Toc124174030)

[Data Model 9](#_Toc124174031)

[3.1 Description of Data Objects 9](#_Toc124174032)

[3.2 Data Objects Relationships 10](#_Toc124174033)

[3.3 Databases 11](#_Toc124174034)

[Behavioral Analysis 12](#_Toc124174035)

[4.1 Sequence Diagrams 12](#_Toc124174036)

[4.2 Events 19](#_Toc124174037)

[4.3 States 20](#_Toc124174038)

[4.3.1 User States 20](#_Toc124174039)

[4.3.2 Runs/Tasks States 20](#_Toc124174040)

[4.3.3 Algorithms States 20](#_Toc124174041)

[Object Oriented Analysis 21](#_Toc124174042)

[5.1 Class Diagrams 21](#_Toc124174043)

[5.2 Class Description 22](#_Toc124174044)

[5.3 Packages 22](#_Toc124174045)

[5.4 Unit Testing 23](#_Toc124174046)

[User Interface Draft 25](#_Toc124174047)

[Testing 30](#_Toc124174048)

Chapter 1

# Use Cases

In this chapter we are going to use the same actors described in the ARD:

**1.** **Guest:** Can only register and login to the system.

**2.** **Registered User:** Can upload data and see if the algorithm run successfully or not, watch his previous runs (date and is it ran successfully), see statistics of his algorithms runs.

**3.** **Admin:** Have ‘Registered User’s permissions.

Admin can see statistics and if algorithms ran for of all the users, see how many users are using the system at this moment, and he can remove users from the system. Admin can also give and remove ‘Admin’ permissions

## 1.1 Users Use-Cases

1. Use-Case: **Register**  
   Actor: Guest  
   Preconditions: A Registered User with the same email doesn’t exist in the system  
   Parameters: Email, Password.   
   Postconditions: A Registered User with the same email doesn’t exist in the system

Result: User with the relevant parameters is created  
Actions:

1. Guest: Inputs the relevant details
2. System: Checks for data validity
3. System: Finds that data is invalid

System: Present error message

1. System: Finds that data is valid

System: Create Registered User with the given details

1. Use-Case: **Login**  
   Actor: Guest  
   Preconditions: A Registered User with the same email does exist in the system  
   Parameters: Email, Password  
   Postconditions: The Guest now have ‘User’ permissions  
   Result: This user can perform any [Registered User related operations](#_2.1_User_Profiles)  
   Actions:
2. Guest: Inputs email and password.
3. System: Checks for data validity
4. System: Finds that data is invalid

System: Present error message

1. System: Finds that data is valid

System: allow this user Registered User’s permissions

1. Use-Case: **Logout**  
   Actor: Registered User  
   Preconditions: The registered user is currently logged in to the system.   
   Parameters: None.  
   Postconditions: The User has only guest permissions.

Result: The User is no longer logged in.  
Actions:

1. Registered User (Logged in): Selects Logout.
2. System: Checks if the user is currently logged in.
3. System: Finds that the user is not logged in

System: Present error message

1. System: Finds that the user is logged in

System: Logging out the user from the system.

1. Use-Case: **See All Runs of Algorithms**  
   Actor: Registered User   
   Postconditions: Details of previous runs of this are shown  
   Result: The user can see details of all his previous runs  
   Actions:
2. System: Brings all this user’s results
3. Use-Case: **See How Many Users Are Using the System**  
   Actor: Registered User  
   Result: User sees the number of users that active at that moment  
   Actions:
4. System: Brings the number of users that signed-in
5. Use-Case: **Edit Profile**

Actor: Registered User   
Postconditions: User details are shown  
Result: The user can edit his own user details.   
Actions:

1. System: Brings all this user’s details.
2. Registered User: Edit the relevant details.
3. System: Updates these details in the system.
4. Use-Case: **View User’s Runs Statistics**

Actor: Registered User   
Postconditions: User has already ran at least one algorithm in the system.  
Result: The user can see the statistics of his own runs.   
Actions:

1. System: Brings all this user’s runs details.
2. System: Calculate the statistics and show it to the user.
3. Use-Case: **Remove Registered User from The System**  
   Actor: Admin  
   Preconditions: A Registered User with the same email does exist in the system

Parameters: email  
Postconditions: A Registered User with the same email does not exist in the system  
Result: The user with that email is no longer registered to the system  
Actions:

1. Admin: inputs email
2. System: Checks for data validity
3. System: Finds that data is invalid

System: Present error message

1. System: Finds that data is valid

System: Removes the user associated to this email from database

1. Use-Case: **Add Admin to The System**Actor: Admin  
   Preconditions: A Registered User with the same email does exist in the system

Parameters: email  
Postconditions: The User with this email is defined as Admin   
Result: the user with that email can perform any [Admin related operations](#_2.1_User_Profiles)  
Actions:

1. Admin: inputs email
2. System: Checks for data validity
3. System: Finds that data is invalid

System: Present error message

1. System: Finds that data is valid

System: Set user as Admin

1. Use-Case: **Remove Admin from The System**  
   Actor: Admin  
   Preconditions: A Registered User with the same email does exist in the system and defined as Admin

Parameters: email  
Postconditions: The User with this email is not defined as Admin   
Result: the user with that email can perform only [Registered User related operations](#_2.1_User_Profiles)

Actions:

1. Admin: inputs email
2. System: Checks for data validity
3. System: Finds that data is invalid

System: Present error message

1. System: Finds that data is valid

System: Set user as not admin

1. Use-Case: **See All Users Algorithm Runs History**  
   Actor: Admin  
   Postconditions: The details of all the users’ runs are shown

Result: Admin sees all the details that saved for every user.

Actions:

1. System: Gets all algorithm saved details

## 1.2 Runs/Algorithms Use-Cases

**General Use-Case for running an algorithm in the system:**

Actor: Registered User   
Parameters: dataset  
Postconditions: Result of this data calculated, the result save in the data-base  
Result: The user can see the result  
Actions:

1. Registered User: Inputs dataset
2. System: Calculate the result
3. System: Save the result in the database
4. System: Show the result.
5. Use-Case: **Run Alignment Algorithm (without Auto option)**  
   Actor: Registered User   
   Parameters: .sdf files  
   Postconditions: .sdf file is valid.  
   Result: The user can see the result in a .mol2 file  
   Actions:
6. Registered User: Inputs .sdf file.
7. System: Run the alignment and calculate the results in a .mol2 file.
8. System: Save the results details in the database.
9. System: Show the result to the user (.mol2).
10. Use-Case: **Run Alignment Algorithm Using Auto Option**  
    Actor: Registered User   
    Parameters: .sdf files  
    Postconditions: .sdf file is valid.  
    Result: The user can see the result in a .csv file  
    Actions:
11. Registered User: Inputs .sdf file
12. System: Run the alignment and calculate the results in a .mol2 file
13. System: With the .mol2 file, the system will run automatically the Feature Extraction algorithm and calculate the output in .csv file.
14. System: Save the results details in the database
15. System: Show the result to the user (.csv).
16. Use-Case: **Run Machine Learning Algorithm**

**(XGBoost / Lasso Regressor / Decision Tree Regression / Binding Score)**   
Actor: Registered User   
Parameters: .csv file  
Postconditions: .csv file is valid.  
Result: The user can see the result in a new .csv file  
Actions:

1. Registered User: Inputs .csv file
2. Registered User: Choose which machine learning algorithm he wants to run.
3. System: Run the ML algorithm and calculate the results in a new .csv file
4. System: Save the results details in the database
5. System: Show the result to the user (.csv).

Chapter 2

# System Architecture

## 2.1 Software Components

**Web Server**: This component is responsible for hosting the web application and handling incoming HTTP requests from clients (e.g., browsers). In our case it’s Django-React project, the web server would likely be running a Python web server uWSGI.

**Application** **Server**: This component is responsible for running the backend logic of the web application and handling requests from the web server. In our Django-React project, the application server would be running the Django application.

**Database**: This component is responsible for storing and organizing the data used by the web application. In this case, the database would be a NoSQL database MongoDB.

**React** **Frontend**: This component is responsible for rendering the user interface of the web application and handling user interactions. In our Django-React project, the frontend would be built using ReactJS.

**Client**: This is the component that initiates requests to the web server, typically a web browser like Chrome or Firefox.

## 2.2 Target Machine

**Server-Side Files**: These are files that are stored on the server and are accessed and processed by the server-side components of the system (e.g., the web server, application server). Examples of server-side files include Python scripts, Django template files, and database configuration files.

**Client-Side Files**: These are files that are sent to the client (e.g., web browser) and are accessed and processed by the client-side components of the system (e.g., the React frontend). Examples of client-side files include HTML, CSS, and JavaScript files.

## 2.3 Components Interaction

**Client sends a request to the web server**: When a user interacts with the web application (e.g., by clicking a link or submitting a form), their client (e.g., web browser) sends an HTTP request to the web server.

**Web server receives the request and passes it to the application server**: The web server receives the request and passes it on to the application server, which is responsible for handling the backend logic of the web application.

**Application server retrieves or updates data in the database**: If the request requires data to be retrieved or updated in the database, the application server communicates with the database to perform these operations.

**Application server generates a response and sends it back to the web server**: Once the application server has completed its processing, it generates an HTTP response and sends it back to the web server.

**Web server sends the response to the client**: The web server receives the response from the application server and sends it back to the client, typically in the form of an HTML document.

**Client renders the response**: The client (e.g., web browser) receives the HTML document and renders it to the user, displaying the requested content or the results of the user's interaction with the web application.

Chapter 3

# Data Model

## 3.1 Description of Data Objects

Graphical user interface

Description automatically generatedGraphical user interface

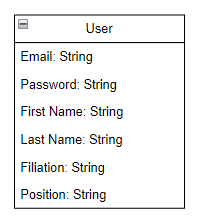
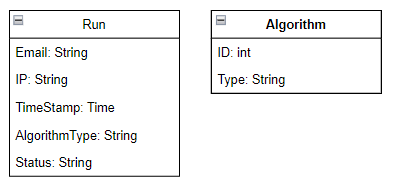
Description automatically generated with medium confidence

**User Object:** represents an account in the system.

**Run Object:** represents an algorithm execution artifact.

**Algorithm Object:** represents an algorithm that exists in the system.

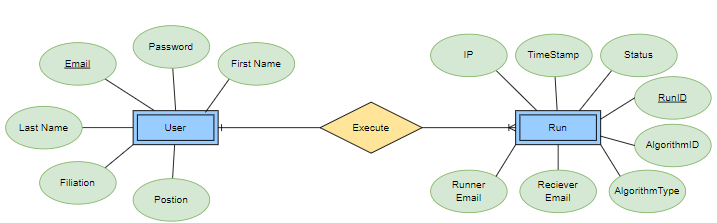
## 3.2 Data Objects Relationships



Diagram

Description automatically generated

## 3.3 Databases



**ERD**

**Database Tables**

Graphical user interface, table

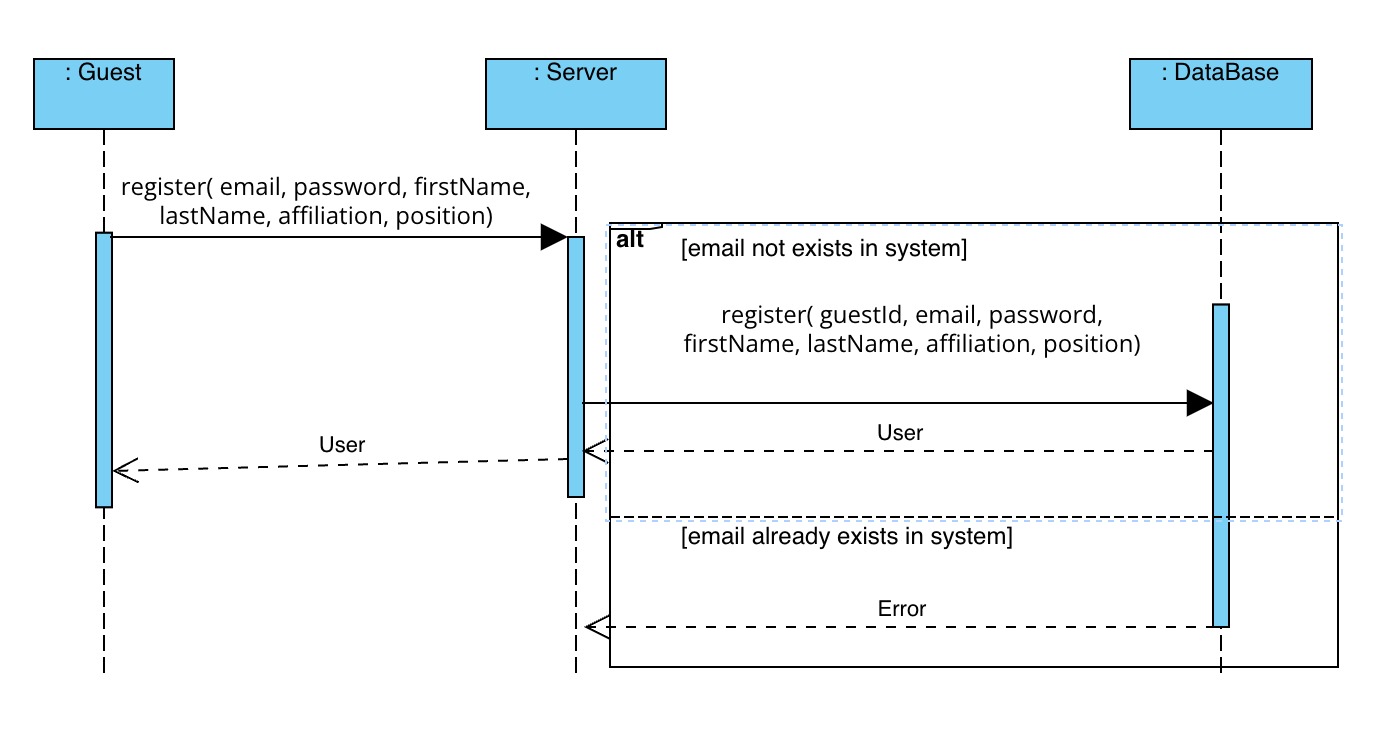
Description automatically generated

Chapter 4

# Behavioral Analysis

## Sequence Diagrams

1. **Register:**

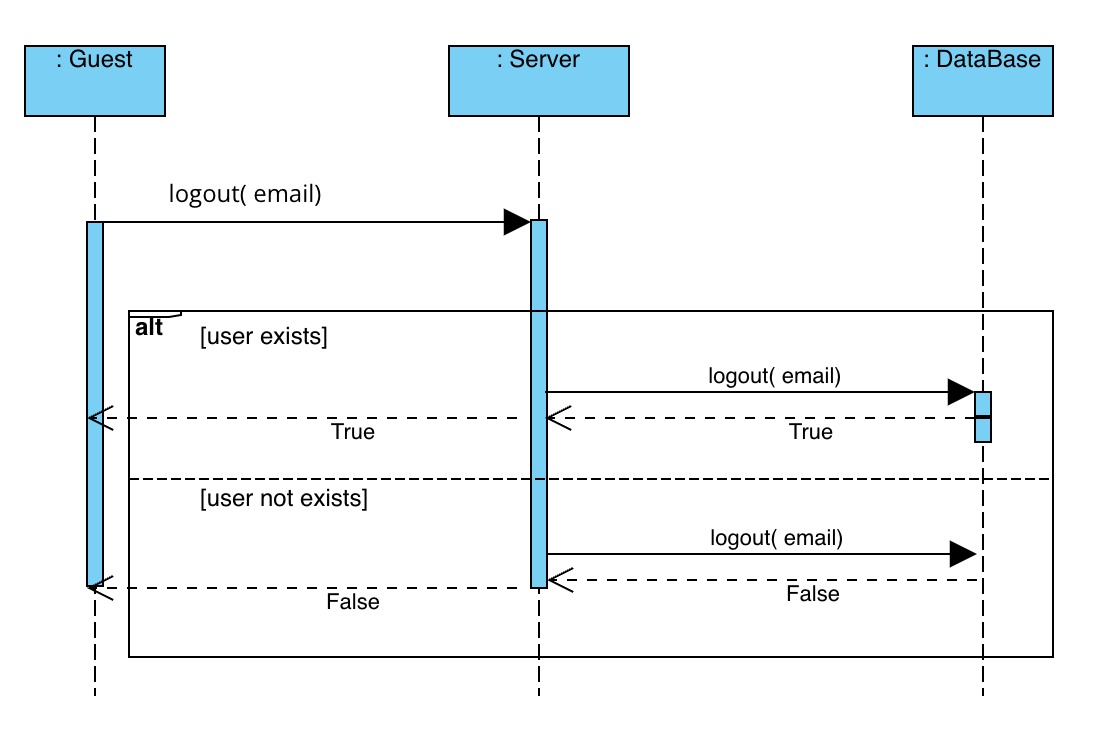


1. **Login:**

Chart, diagram, box and whisker chart

Description automatically generated with medium confidence

1. **Logout**

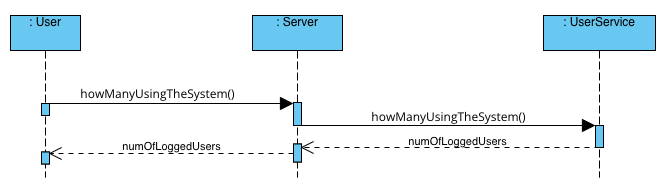


1. **See History of Algorithm Run:**

Diagram

Description automatically generated

1. **See How Many Users Are Using the System:**



1. **Remove Registered User from The System:**

Diagram

Description automatically generated

1. **Add Admin to The System:**

Diagram

Description automatically generated

1. **Remove Admin from The System:**

Diagram

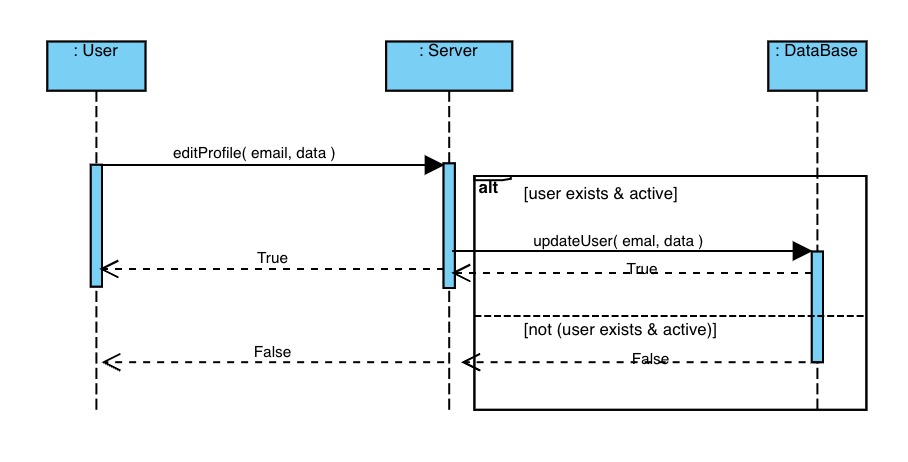
Description automatically generated

1. **See All Users Runs History:**

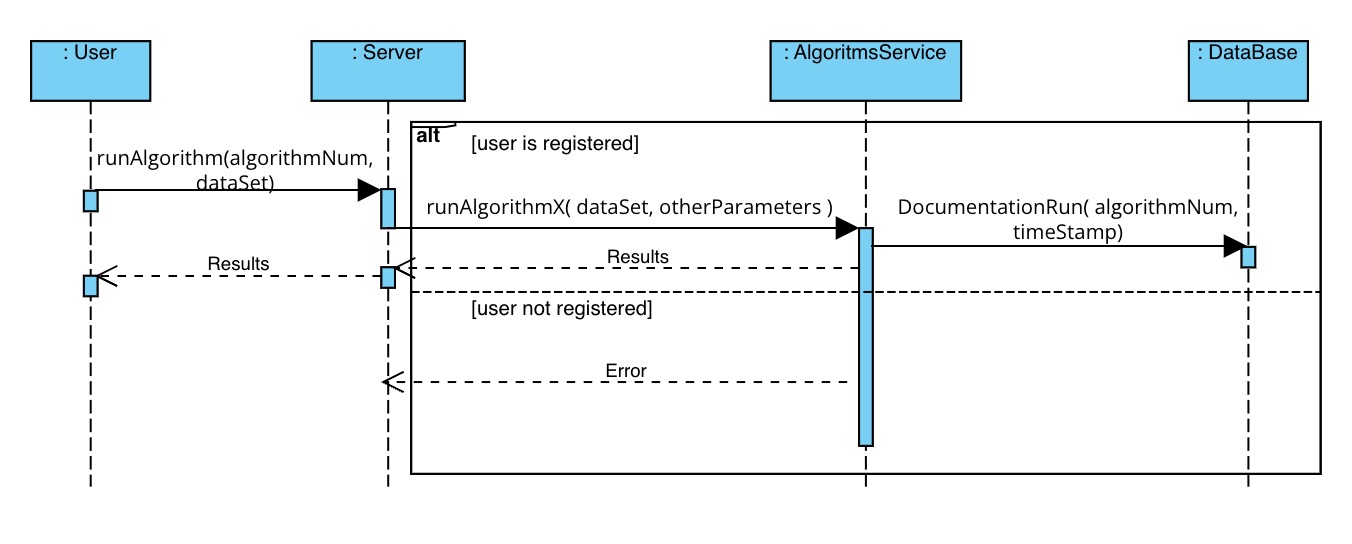
Diagram

Description automatically generated

1. **Edit User Profile**



1. **General Algorithm Run:**



1. **Run Alignment Algorithm**

Diagram

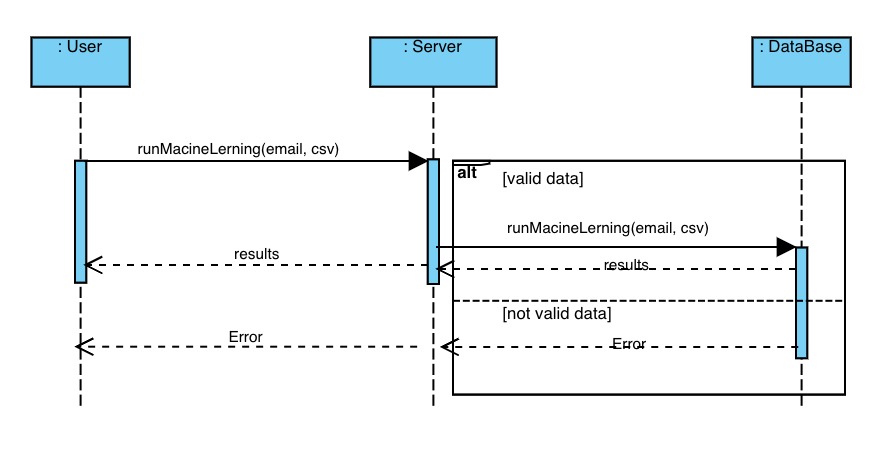
Description automatically generated

1. **Run Auto**

Diagram

Description automatically generated

1. **Run Machine Learning Algorithm**



## 4.2 Events

|  |
| --- |
| Event: Server connection lost |
| Event scenario: Upon losing the server connection, the system will write a message to the system logger, and will notify the user about it.  In addition, any current work that will be running in the system will be canceled, and any data at the system at that current state will be saved, and any action in the system will be disabled (until the server will be restored). |

|  |
| --- |
| Event: Server connection restored |
| Event scenario: Upon restoring the server connection, the system will write a message to the system logger, and will notify the user about it.  In addition, actions in the system will be enabled again, and any data from the state before the server connection loss, will be restored. |

|  |
| --- |
| Event: Database connection lost |
| Event scenario: Upon losing the database connection, the system will write a message to the system logger, and will notify the user about it.  In addition, any current work that will be running in the system will be paused, and any database related actions will be disabled. |

|  |
| --- |
| Event: Server connection restored |
| Event scenario: Upon restoring the database connection, the system will write a message to the system logger, and will notify the user about it.  In addition, any paused work in the system will resume running, and all the system actions will be enabled again. |

|  |
| --- |
| Event: Algorithm run stopped / rejected |
| Event scenario: Upon run cancelation/rejection, the system will write a message to the system logger and will notify the user about it. |

|  |
| --- |
| Event: Algorithm run success/failure |
| Event scenario: Upon run success/failure results, the system will write a message to the system logger and will notify the user about it.  In addition, the system will send an email to the running user. |

|  |
| --- |
| Event: User actions (Login, Logout, Register, etc.) |
| Event scenario: Upon any user action in the system, the system will write a message to the system logger and will notify the user about it. |

## 4.3 States

### 4.3.1 User States

In our system one of the main actors is the Users.

Each user in the system has few states:

* Logged out: which means that the person that use the system is currently a Guest (the lowest permission level in the system).
* Logged in: which means that the person that use the system got into the system with a certificated (subscribed) user, and he can use most of the system (except admin actions).
* Is-Admin: which means that the person that use the system got into the system with an elevated certificated (subscribed) user, and he can do any action that the website allows.

### 4.3.2 Runs/Tasks States

In our system another important and main actors is the Runs.

Each run in the system has few states:

* Pending: the algorithm run is ready to run but waiting for its turn to run.
* Running: the algorithm is currently being run in the system.
* Rejected: the algorithm run request was denied by the system.
* Canceled: the algorithm run request was stopped by the running user / admin.
* Success: the algorithm finished running and its result are good.
* Failed: the algorithm finished running and its result are bad.

### 4.3.3 Algorithms States

In our system another important and main actors is the Algorithms.

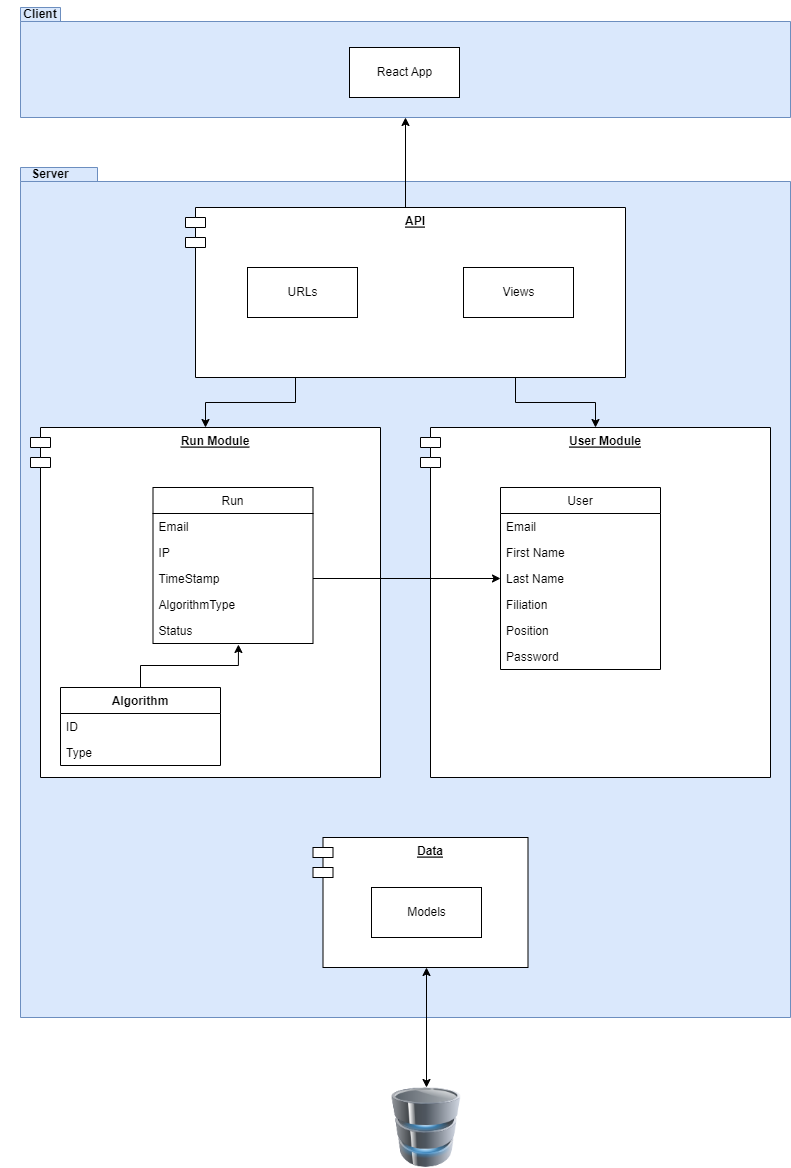
These actors are included in the system as const actors.

In other words, the algorithms have only one state and their state is unchangeable.

Chapter 5

# Object Oriented Analysis

## 5.1 Class Diagrams



## 5.2 Class Description

**Users Class**

This class is responsible for any representation of the users that using the system. Each object from the type of User in the system, presents different account in the system.

Each User has:

* Unique email.
* Valid password and personal details.
* Permission level (in the system).

**Runs Class**

This class is responsible for any execution of algorithm in the system.

It contains any details about:

* What is the algorithm that ran.
* Who ran the algorithm.
* Who will get the run’s results.
* What is the time that the run started.
* What are the results / status of the run.

Each Run has:

* Only one executer.
* Many users that can see the results (Admin, users that the executer chose, the executer himself)
* One running algorithms.
* Unique ID and Time Stamp (the starting time of the algorithm).

**Algorithms Class**

This class presents any algorithm that exists in the system.

Each Algorithm has:

* Unique ID.
* Type.

## 5.3 Packages

**Frontend Packages**

* React JS
* Material UI

**Backend Packages**

* Django (Python)
* MongoDB (Database)

**Sever Packages / Technologies**

* TBD

**Security Packages / Technologies**

* TBD

**Need to be modified / changed…**

## 5.4 Unit Testing

**Class Users:**

Test: register

* Good scenario

Description: The test will be given valid data and and register this email with the data to the system

Expected Result: The new user will be registered successfully.

* Bad scenario

Description: The test will be given invalid data OR email of an existing user and will call the register function with them.

Expected Result: The system will present an appropriate error.

Test: login

* Good scenario

Description: The test will be given email associated with registered user and the correct password

Expected Result: The user associated with this email will be logged.

* Bad scenario

Description: The test will be given email associated with registered user and the wrong password OR email not associated with registered user

Expected Result: The system will present an appropriate error.

Test: Edit User Profile

* Good scenario

Description: The test will be given email associated with valid data

Expected Result: The user associated with this email will be updated with the new data.

* Bad scenario

Description: The test will be given email associated with registered user and invalid data OR email not associated with registered user

Expected Result: The system will present an appropriate error.

**Class Runs:**

Test: Ducument Run

* Good scenario

Description: The test will be given valid data

Expected Result: The data will be saved

* Bad scenario

Description: The test will be given invalid data

Expected Result: The system will present an appropriate error.

Test: Run alignment Algorithm

* Good scenario

Description: The test will be given valid .sdf files

Expected Result: The algoritm run and return the results to the user

* Bad scenario

Description: The test will be given invalid .sdf files

Expected Result: The system will present an appropriate error.

Test: Run auto alignment Algorithm

* Good scenario

Description: The test will be given valid .sdf files

Expected Result: The alignment algorithm run than the feature extract algorithm run and return the results to the user

* Bad scenario

Description: The test will be given invalid .sdf files

Expected Result: The system will present an appropriate error.

Test: Run Machine Learning Algorithm

* Good scenario

Description: The test will be given valid .csv files

Expected Result: The machine learning algorithm run and return the results to the user

* Bad scenario

Description: The test will be given invalid .csv files

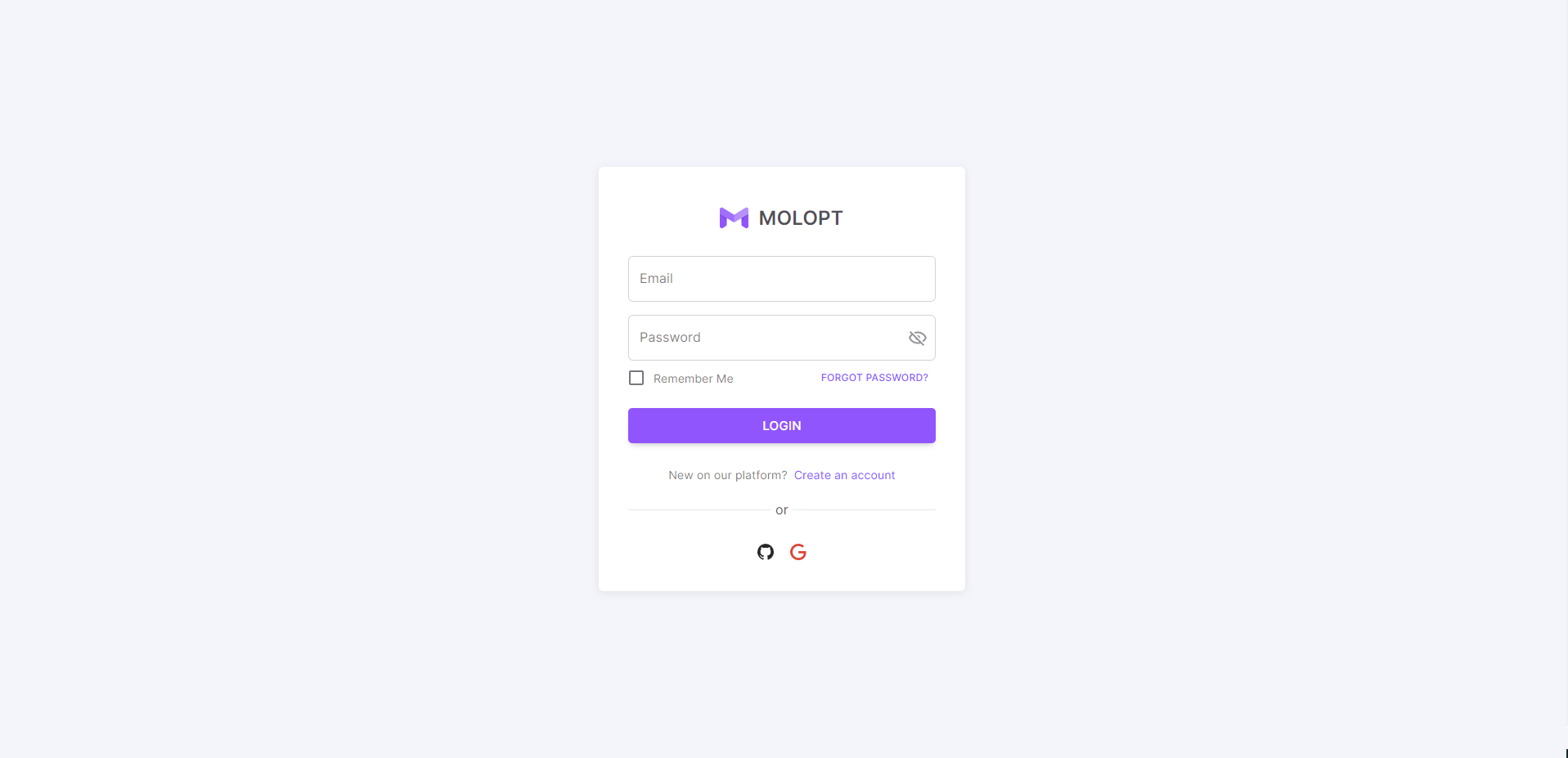
Expected Result: The system will present an appropriate error.

**Need to be modified / changed…**

Chapter 6

# User Interface Draft

**Login Page**

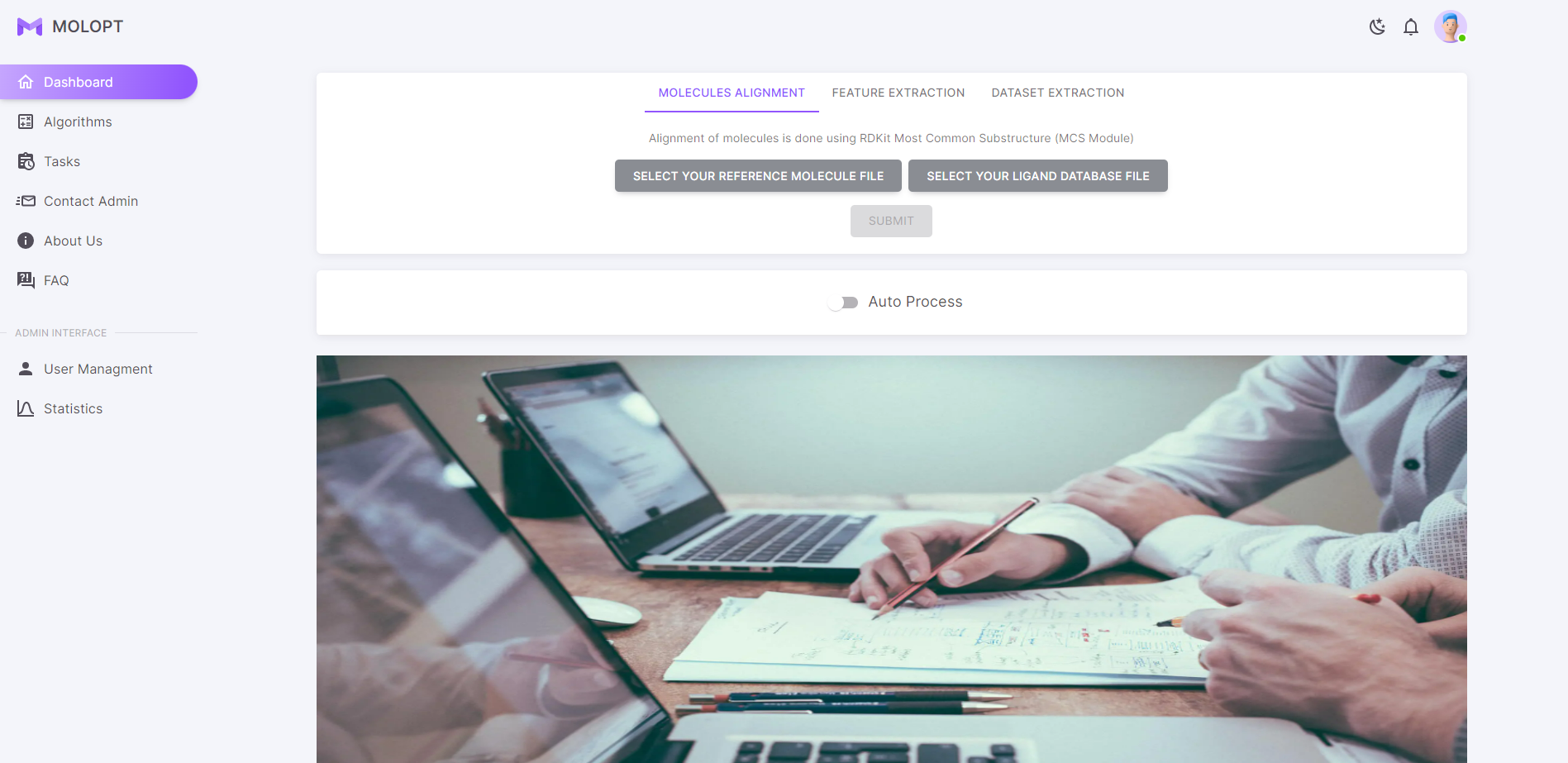


**Register Page**

Graphical user interface, application

Description automatically generated

**Home Page / Dashboard**



**User Profile Page**

Graphical user interface, application, Teams

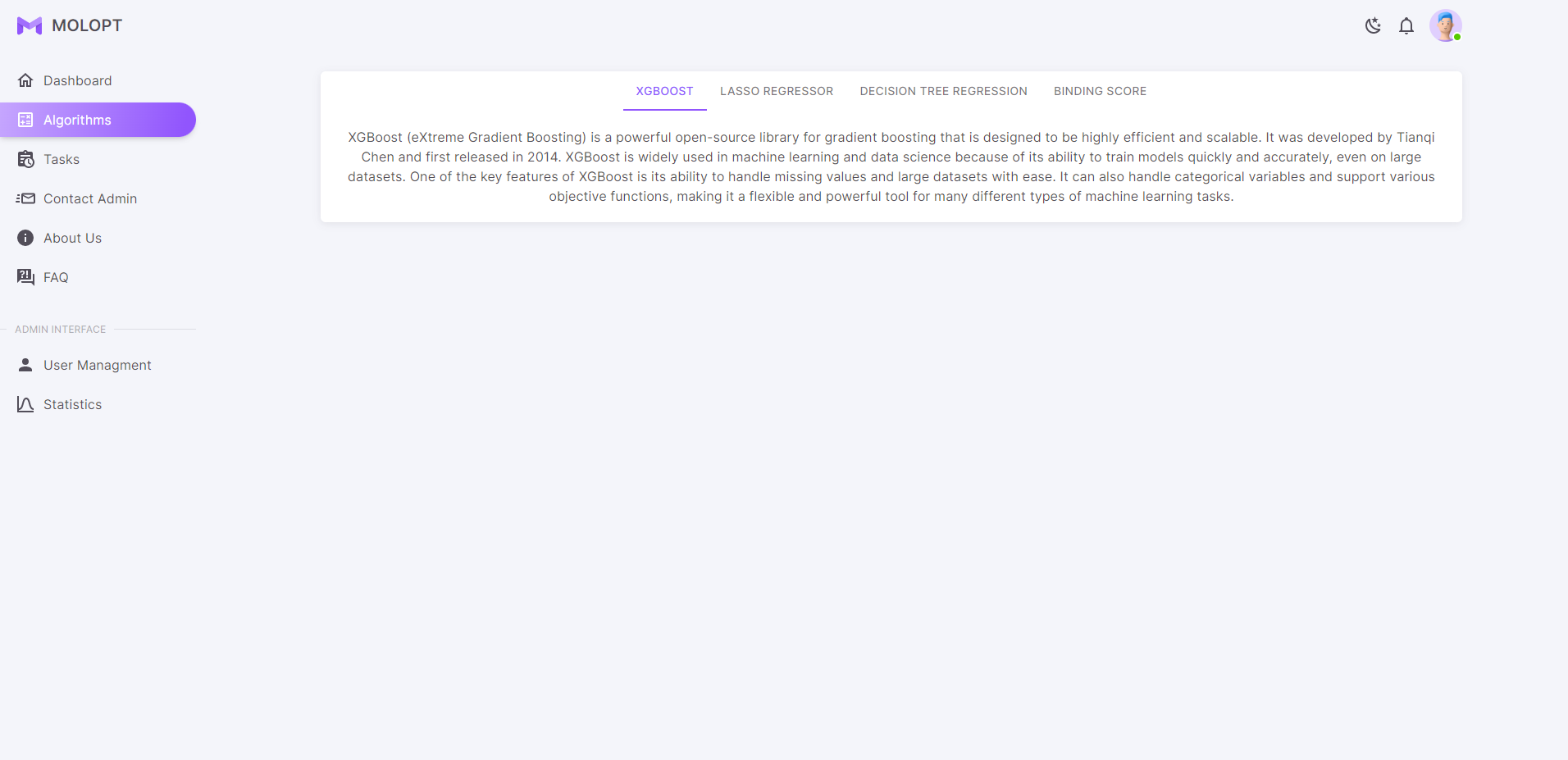
Description automatically generated

**Algorithms Runs (Tasks) / Algorithms Queue**

Graphical user interface, application, table

Description automatically generated

**Algorithms Info Page**



**Contact Admin Page**

Graphical user interface, text, application, Teams

Description automatically generated

**About Us Page**

**Will be added…**

**FAQ Page**

Graphical user interface, text, application, email

Description automatically generated

**User Management Page (For Admin)**

Graphical user interface, application

Description automatically generated

**Statistics Page**

**Will be added…**

Chapter 7

# Testing

|  |
| --- |
| Requirement #1: Guest must be able to register to the system. |
| Test:   * Good scenario   Description: The test will be given valid credentials and will call the register function with them.  Expected Result: The new user will be registered successfully.   * Bad scenario   Description: The test will be given invalid credentials OR credentials of an existing user and will call the register function with them.  Expected Result: The system will present an appropriate error. |

|  |
| --- |
| Requirement #2: Guest must be able to login the system if he is registered. |
| Test:   * Good scenario   Description: The test will be given valid credentials and will call the login function with them.  Expected Result: The new user will be logged in successfully.   * Bad scenario   Description: The test will be given invalid credentials OR credentials of a  non-existing user and will call the login function with them.  Expected Result: The system will present an appropriate error. |

|  |
| --- |
| Requirement #3: Guest must not be able to run the Mol-Optimizer. |
| Test:   * Good scenario   Description: The test will try to run an algorithm without logging in before the run execution.  Expected Result: The system will present an appropriate error.   * Note: This scenario will be tested on each algorithm in the system. |

|  |
| --- |
| Requirement #4: User must be able to upload dataset and run the Mol-Optimizer on it. |
| Test:   * Good scenario   Description: The test will be given a user which is registered and logged in and a valid files/dataset for running an algorithm and will execute the algorithm.  Expected Result: The user will run the algorithm successfully.   * Bad scenario   Description: The test will be given a user which is not registered OR not logged in OR an invalid files/dataset for running an algorithm and will execute the algorithm.  Expected Result: The system will present an appropriate error.   * Note: The bad scenario will be tested on each algorithm in the system. |

|  |
| --- |
| Requirement #5: User who is not Admin should not be able to see history of other users |
| Test:   * Good scenario   Description: The test will be given an admin which is registered and logged in and will call the “view history” function.  Expected Result: The admin will be able to see all the runs history.   * Bad scenario   Description: The test will be given a user which is not registered OR not logged in OR not an admin and will call the “view history” function  Expected Result: The system will present an appropriate error. |

|  |
| --- |
| Requirement #6: The system should record details of the runs of the algorithms |
| Test:   * Good scenario   Description: The test will be running an algorithm and will save the results.  Expected Result: The results will be available and matching to the algorithm that ran.   * Note: This scenario will be tested on each algorithm in the system. |

|  |
| --- |
| Requirement #7: User should be able to see history of the Mol-Optimizer runs he did. |
| Test: This has already been tested in Requirement #5 tests. |

|  |
| --- |
| Requirement #8: User should be able to see how many users are using the system at this current time. |
| * Good scenario   Description: The test will be given a user which is registered and logged in and will call the “see-amount-of-users” function.  Expected Result: The user will see the number of users in the system.   * Bad scenario   Description: The test will be given a user which is not registered OR not logged in and will call the “see-amount-of-users” function.  Expected Result: The system will present an appropriate error. |

|  |
| --- |
| Requirement #9: Admin should be able to see all users history details |
| Test: This has already been tested in Requirement #5 tests. |

|  |
| --- |
| Requirement #10: Admin should be able to remove Admin appointment of User |
| Test:   * Good scenario   Description: The test will be given an admin which is registered and logged in and will call the “remove admin” function with an existing admin in the system.  Expected Result: The wanted admin will be removed successfully.   * Bad scenario   Description: The test will be given a user which is not registered OR not logged in OR not an admin and will call the “remove admin” function with an existing admin in the system OR with a non-admin details.  Expected Result: The system will present an appropriate error. |

|  |
| --- |
| Requirement #11: Admin should be able to remove User from the system |
| Test:   * Good scenario   Description: The test will be given an admin which is registered and logged in and will call the “remove user” function with an existing user in the system.  Expected Result: The wanted user will be removed successfully.   * Bad scenario   Description: The test will be given a user which is not registered OR not logged in OR not an admin and will call the “remove admin” function with a non-existing user in the system.  Expected Result: The system will present an appropriate error. |

|  |
| --- |
| Requirement #12: The Mol-Optimizer should be able to run on every valid dataset. |
| Test: This has already been tested in Requirement #4 tests. |

|  |
| --- |
| Requirement #13: All tasks execution successfully |
| Test: This has already been tested in Requirement #6 tests. |

|  |
| --- |
| Requirement #14: All tasks complete within 10-15 min successfully |
| Test:  Each algorithm will be run, and time stamp will be taken before and after its execution.  Success will be achieved only if the longest algorithm will take less than 15 minutes (Otherwise – the test will fail). |

|  |
| --- |
| Requirement #15: System will support 1000 request per seconds |
| Test:  Stress tests will be created, and few random methods will be activated at the same time.  Success will be achieved only if all the requests will be responded as expected  (Otherwise – the test will fail). |

|  |
| --- |
| Requirement #16: On network/connection failure, system will notify user and rollback changes |
| Test:  First, a random function in the system will be activated.  Then, an intentional network failure will be done.  The test will check both the website recovery and that no change has occurred in the system.  If that will happen, Success will be achieved (Otherwise – the test will fail). |

|  |
| --- |
| Requirement #17: On connection issues, system will do auto retry |
| Test: This has already been tested in Requirement #16 tests. |

|  |
| --- |
| Requirement #18: System will not save any personal data nor expose personal data of users |
| Test:  The test will go on any credentials in the system and will check that all the sensitive data has been encrypted.  If all this data is encrypted Success will be achieved (Otherwise – the test will fail). |