Amit Peled, Daniel Piro, Shahar Alon, Nofar Rozenberg

BGU / MolOpt

MolOptimizer - Final Project in Software Engineering

Application Requirement Document (ARD)

**Abstract**

This document is the first of three documents describing the final project in

the software engineering program. Specifically, this document describes the

requirements of the software project from the standpoint of the customer.

This project is a software engineering project which created as part of the   
Ben Gurion University (BGU) software engineering program.

For our final project in software engineering degree,  
 the four of us (Amit, Daniel, Shahar and Nofar) assembled   
with Samuel Viswas and Dr. Barak Akabayov from the BGU Chemistry department to create an amazing global molecules optimizer web-application   
which called “MolOptimizer” (MolOpt).

MolOpt was Developed at Akabayov Research Laboratories,   
at the department of Chemistry in Ben Gurion University of the Negev,   
Under the supervision of Dr. Barak Akabayov

The main purpose of MolOpt and the main reason why it was developed,   
is to be used by researchers working in the field of   
Small Fragment based inhibitors and is intended to be helpful  
 in optimization of ligand databases.

At First, MolOpt was a basic flask-based web package   
which can be used for Alignment of large ligand datasets,   
extracting large volume of Chemical Descriptors and training Machine Learning models   
to predict binding scores, which only ran locally on the labs computers.

Our project was mainly focused on taking this toolkit and make it a real web-application   
that has server connections, big database storage, user management   
and parallel & optimized algorithm runs.

Contents

[Guidelines 3](#_Toc130286233)

[Introduction 4](#_Toc130286234)

[1.1 The Problem Domain 4](#_Toc130286235)

[1.2 Context 4](#_Toc130286236)

[1.3 Vision 4](#_Toc130286237)

[1.4 Stakeholders 4](#_Toc130286238)

[1.5 Software Context 5](#_Toc130286239)

[User Scenarios 6](#_Toc130286240)

[2.1 User Profiles – The Actors 6](#_Toc130286241)

[2.2 Use-cases 6](#_Toc130286242)

[2.3 Sequence Diagrams 11](#_Toc130286243)

[Functional Requirements 18](#_Toc130286244)

[Non-functional Requirements 19](#_Toc130286245)

[4.1 Implementation Constraints 19](#_Toc130286246)

[4.2 Platform Constraints 20](#_Toc130286247)

[4.2.1 SE Project Constraints 20](#_Toc130286248)

[4.3 Special Restrictions & Limitations 20](#_Toc130286249)

[Risk Assessment & Plan For Proof Of Concept 21](#_Toc130286250)

# Guidelines

MolOptimizer will be a global tool (web-application) for chimists by researchers working in the field of Small Fragment based inhibitors.

This project implemtation requiers us to work the following subjects/issues:

* Taking the local toolkit (toolkit that works only locally on BGU lab’s computers) and make into a global platform.
* Changing this toolkit to be a great web-application that its heart is going to be the MolOpt toolkit.
* Enabling user management.
* Enabling parallel use – both for single user to runs multiple algorithms, and for multiple users to work on the app at the same time.
* Making this web-application run on the server.
* Improving the toolkit file’s allignment.
* Adding security service when using the app.
* Developing web-application which will be easy to use (“user friendly”), generic and easy to be add on in the future

(keeping the option of adding new functionality or new algorithms to the app easily)

* Adding system cheking for algorithms run (for example – validitiy checks).
* Optimizing algorithms runs – make algorithms run in parallel or run in background.
* Adding error handeling and rubotness.
* Adding logging system.

Chapter 1

# Introduction

MolOpt is a web-application that supports multiple users’ management, big data storage in databases (for the use of the users), parallel & optimized runs of complicated machine learning algorithms.

Its main goal is to supply the most convenient platform for researchers working in the field of Small Fragment based inhibitors and is intended to be helpful in optimization of ligand databases (for example: help developing new medicines).

MolOpt is a Python and React JS secured app based with connection to external servers and databases – which means that the app will be online and free to access for all the researchers working in the field of Small Fragment based inhibitors.

## 1.1 The Problem Domain

The project is related to the domain of Chemistry.

In more details, it is related to field of small fragment-based inhibitors.

## 1.2 Context

The system will be running on the web and will be available from any browser.

## 1.3 Vision

The main goal of our project is to present MolOpt to any chemist working in the field and make it global and accessible to everyone.

## 1.4 Stakeholders

* **Requirements Requesters:** Samuel Viswas and Dr. Barak Akabayov.
* **Application developers:** Amit Peled, Daniel Piro, Shahar Alon and Nofar Rozenberg.
* **Customers:** chemist researchers working in the field of Small Fragment based inhibitors
* **Related organization:** BGU.

## 1.5 Software Context

As described in the introduction, MolOptimizer is going to be a web-application in the field of Chemistry.

The app is going to support some features:

1. User management – any guest will be able to register himself into the app and use it. Some users will also be able to give/remove permissions to other users (mainly using Python and React JS as the backend and frontend sides of the app).
2. Algorithms runs – the heart of MolOpt is the algorithms for helping with the optimization of l0igand databases.

In the app, user that will want to run a specific algorithm, will be able to upload files (of a certain type) and run the algorithms with its files input, and get the results afterwards.

1. Database management – the app will store all the users’ information and part of the algorithms details (using MongoDB).
2. Security – MolOpt will allow secured usage of the entire application, any error handling or logs will be stored in the database and will be available for users depends on their permission level.
3. Parallel usage – using scheduling algorithms running in the backend side of the app (using Python).

Chapter 2

# User Scenarios

## 2.1 User Profiles – The Actors

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.

## 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

**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:

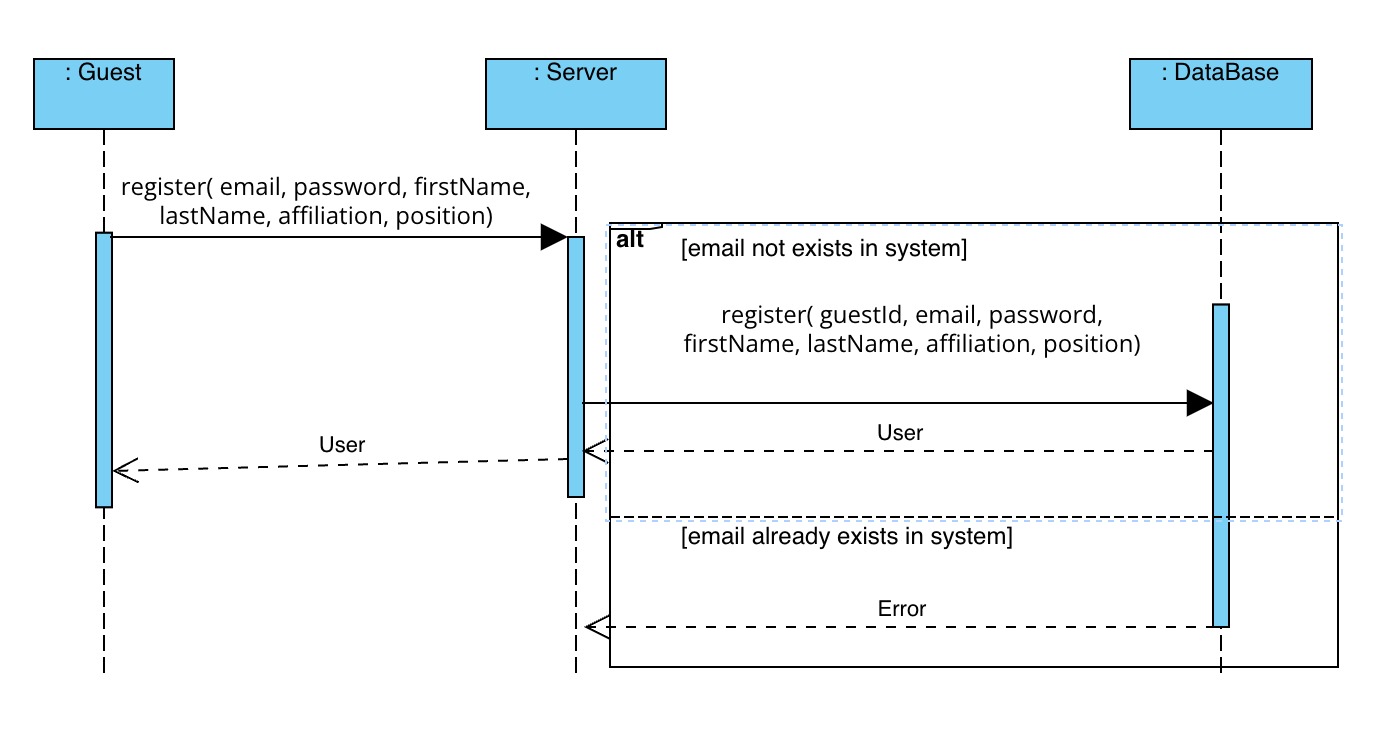
1. Registered User: Inputs dataset
2. System: Calculate the result
3. System: Save the result in the database
4. System: Show the result.
5. Use-Case: **Run Alignment Algorithm (without Auto option)**  
   Actor: Registered User   
   Parameters: .sdf files  
   Postconditions: .sdf file is valid.  
   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).

## Sequence Diagrams

1. **Register:**

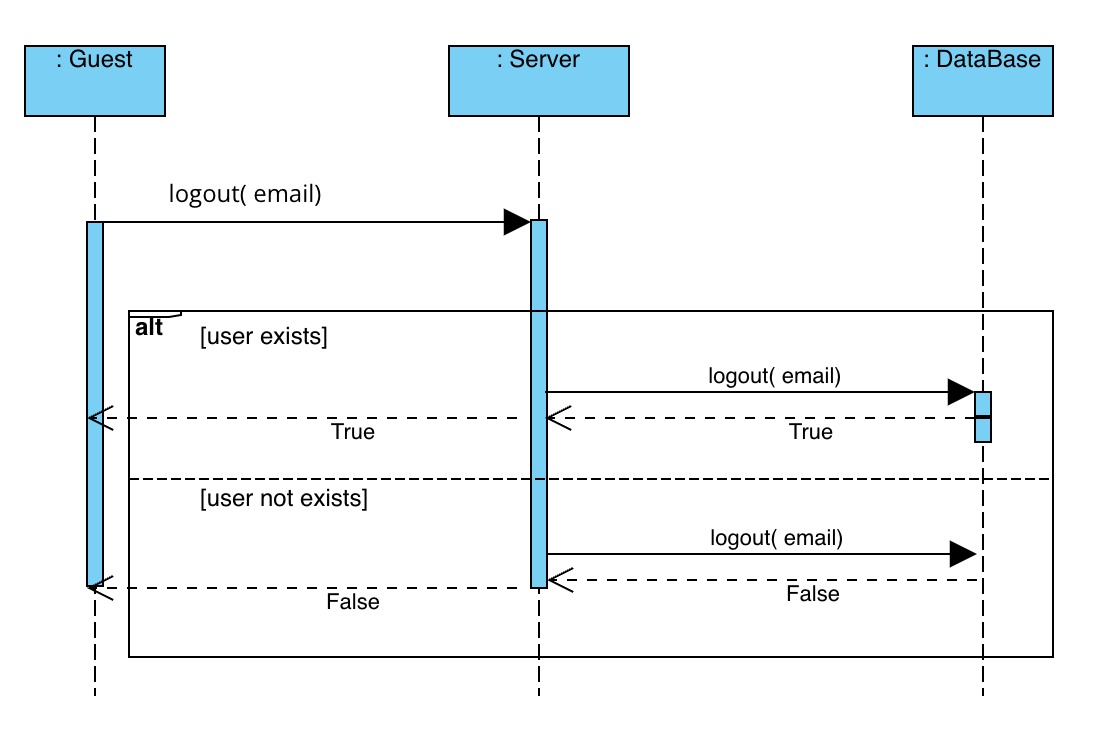


1. **Login:**

Chart, diagram, box and whisker chart

Description automatically generated with medium confidence

1. **Logout**

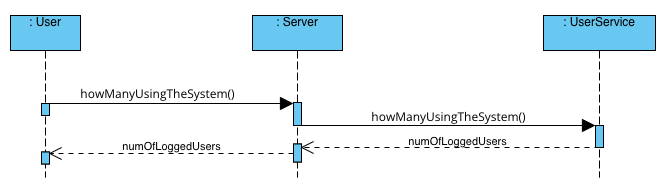


1. **See History of Algorithm Run:**

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

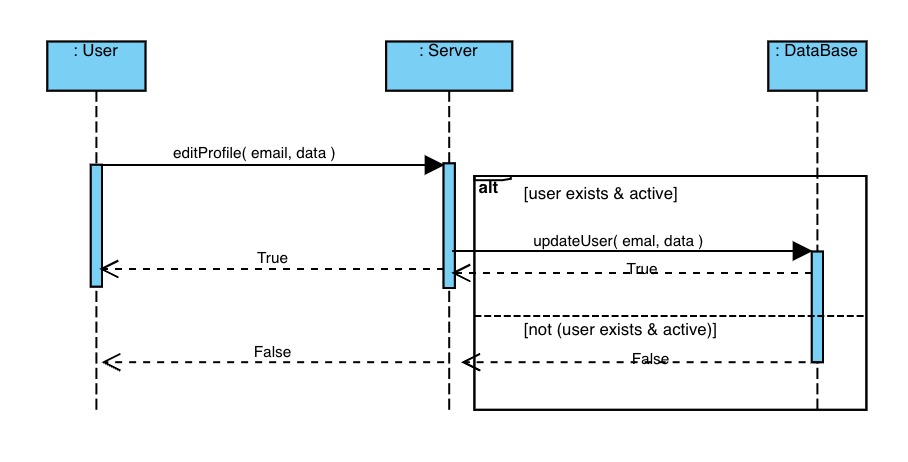
Description automatically generated

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

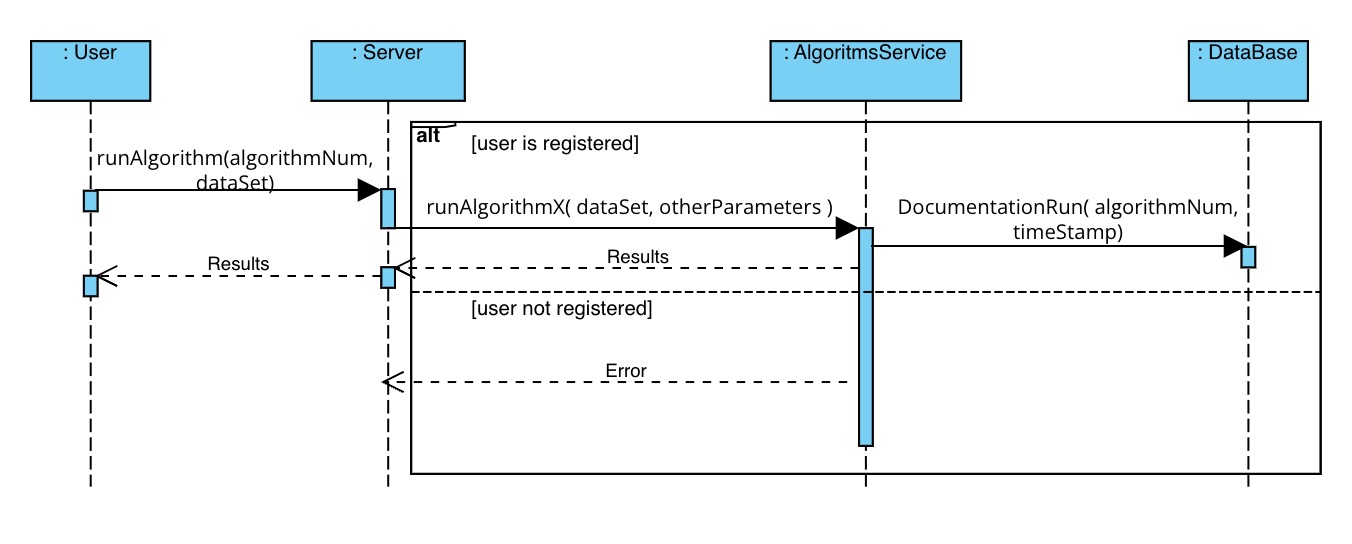
Diagram

Description automatically generated

1. **Edit User Profile**



1. **General Algorithm Run:**



1. **Run Alignment Algorithm**

Diagram

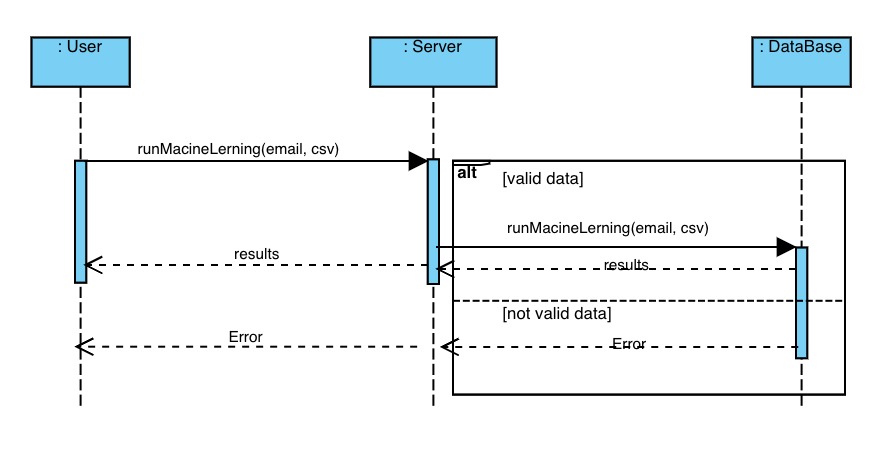
Description automatically generated

1. **Run Auto**

Diagram

Description automatically generated

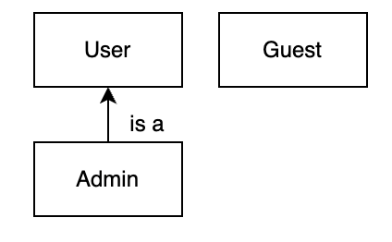
1. **Run Machine Learning Algorithm**



Chapter 3

# Functional Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Description | Priority | Risk |
| 1. | Guest must be able to register to the system. | MH | Low |
| 2. | Guest must be able to login the system if he is registered. | MH | Low |
| 3. | Guest must not be able to run the Mol-Optimizer. | MH | Low |
| 4. | User must be able to upload dataset and run the  Mol-Optimizer algorithms on it. | MH | High |
| 5. | User should be able to see the results of Mol-Optimizer runs on the dataset he uploaded | MH | Medium |
| 6. | User who is not Admin should not be able to see history of other users | MH | Low |
| 7. | The system should record details of the runs of the algorithms. | MH | Low |
| 8. | User should be able to see history of the Mol-Optimizer runs he did. | MH | Low |
| 9. | User should be able to see how many users are using the system at this current time. | NTH | Low |
| 10. | Admin should be able to see all users history details | MH | Medium |
| 11. | Admin should be able to remove Admin appointment of User | MH | Low |
| 12. | Admin should be able to remove User from the system | MH | Low |
| 13. | The Mol-Optimizer should be able to run on every valid dataset. | MH | Low |



\* User types hierarchy:

* **Top priority –** Admin.
* **Standard priority –** User (Subscriber).
* **Low priority –** Guest.

Chapter 4

# Non-functional Requirements

This project required to have optimal performance and run on every available web platform.

The project quality is tested by many parameters:

* Allowing big number of users using the project at the same time.
* Giving the users reliable, “fast-running” and optimized application to use.
* Giving the users the most secure web-application.
* Make sure it runs on any available platform (in our case – web browsers).

## 4.1 Implementation Constraints

**Performance (Speed, Capacity, Throughput, etc.).**

Task execution:

Acceptable: All tasks execution successfully.

Unacceptable: 90% of tasks execute successfully, 10% need to retry.

Worse: None of the task execute successfully.

Task completion speed:

Acceptable: All tasks complete within 10-15 min successfully.

Unacceptable: All tasks complete within 20 min successfully.

Worse: All tasks complete within more than 20 min successfully.

Users’ capacity:

Acceptable: Website support 1000 request per seconds.

Unacceptable: Website support 1000 request per 3 seconds or more.

Worse: Website won’t support 1000 request.

**Reliability & Stability.**

Network failure:

Acceptable: Notify user and rollback changes.

Unacceptable: Rollback changes without notify user.

Worse: Allow partial changes to update, corrupt data.

Data recovery:

Acceptable: System will do auto retry on connection issues.

Unacceptable: System will notify user on failure only.

Worse: System abord execution without any notification.

**Safety & Security.**

Privacy:

Acceptable: System will not save any personal data nor expose personal data of users.

Unacceptable: System will save personal data for authentication.

Worse: System will save personal data for no reason.

Encryption:

Acceptable: System will encrypt all user data.

Unacceptable: System will encrypt only login details.

Worse: System will not encrypt any data.

Permission:

Acceptable: System will abstract any data based on permission.

Unacceptable: System will show all data but disable access to them based on permission.

Worse: System will allow all users access to all website content.

**Usability & Availability.**

Acceptable: Website will work on all known browser types.

Unacceptable: Website will work only on chrome and Firefox.

Worse: Website will work only on chrome.

## 4.2 Platform Constraints

Server:

* Server will be implemented using Python with PyCharm as IDE.
* Server will use NumPy, panda as external lib for algorithms visual.
* Server will use multi threads for async execution.

Client:

* Client will be implemented using ReactJS with visual studio code as IDE.
* Client will use bootstrap as main library design.

### 4.2.1 SE Project Constraints

* System will be interactive and support user input via browser.
* System will simulate data via mocking on early develop stages.

## 4.3 Special Restrictions & Limitations

* User would have access to computer with internet connection.
* User would have email for authentication
* User would know to read English
* User would know to operate keyboard and mouse on basic level.
* User would know the file type the system support and how to provide them.
* System would support accessibility for needed users.

Chapter 5

# Risk Assessment & Plan For Proof Of Concept

In our proof-of-concept prototype we plan to implement the system with basic functionality to get sense of what our system should look like, and which features will be harder to create.

In the prototype we will use new technologies and program languages we got a chance to use before, hence in the process of developing the prototype we will get better understanding of their limitations.

The prototype implementation will contain:

* Client Side:

The client side will contain the pages relevant to the functionality implemented on the server side as well as placeholders for the rest of the functionality that not yet implemented. That way we can provide visualization of the project and receive more accurate feedback about it.

* Server Side:

The server side will contain first iteration of the user system implementation and the option to run the mol-opt algorithms as well as integrating the with the client side.

* Data Layer:

Base planning of the database as well as integrating it with the server side.