Module Guide for Software Engineering

Team 1, BANDwidth
Declan Young
Ben Dubois
Nathan Uy
Aidan Mariglia

January 17, 2025

1 Revision History

Date	Version	Notes
01/07/2025	0.1	revision 0 design

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
Software Engineering	Explanation of program name
UC	Unlikely Change

Contents

List of Tables

List of Figures

3 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (Parnas et al., 1984). We advocate a decomposition based on the principle of information hiding (Parnas, 1972). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules layed out by Parnas et al. (1984), as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is implemented in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (Parnas et al., 1984). The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility, and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module decomposition that was constructed according to the likely changes. Section 6 specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 9 describes the use relation between modules. Section 10 includes the screenshots of the User Interface. Section 11 includes a link to the tasks related to the design, who it is assigned to and the expected completion time.

4 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 4.1, and unlikely changes are listed in Section 4.2.

4.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: The tests and test data that the Matlab models are being tested against

AC2: The categories in the leaderboard

AC3: How the models' performance is displayed to the user

AC4: How the model is executed e.g. execution progress reports, error handling, etc.

AC5: The programming language of the models submitted by the user

4.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: Input/Output devices (Input: file, mouse, keyboard, Output: monitor screen).

UC2: User log-in

UC3: Database Management System

UC4: API Communication

5 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 1. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Hardware Hiding Module

M2: App Module

M3: User Authentication Module

M4: Login Module

M5: Registration Module

M6: Database Management Module

M7: Web Server Module

M8: Intro Module

M9: Home Module

M10: Model Execution Module

M11: Test Data Module

M12: Test Algorithm Module

M13: Submit Module

M14: Submission Module

M15: Leaderboard Module

M16: Results Module

M17: Admin Control Panel Module

Level 1	**Level 2**
Hardware-Hiding Module	M1
Behaviour-Hiding Module	M4 M5 M8 M9 M12 M13 M14 M15 M16 M17
Software Decision Module	M2 M3 M6 M7 M10 M11

Table 1: Module Hierarchy

6 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table 2.

7 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas et al. (1984). The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. *Software Engineering* means the module will be implemented by the Software Engineering software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented.

7.1 Hardware Hiding Modules (M1)

Secrets: The data structure and algorithm used to implement the virtual hardware.

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the software. So, the system can use it to display outputs or to accept inputs.

Implemented By: OS

7.2 Behaviour-Hiding Module

Secrets: The contents of the required behaviours.

Services: Includes programs that provide externally visible behaviour of the system as specified in the software requirements specification (SRS) documents. This module serves as a communication layer between the hardware-hiding module and the software decision module. The programs in this module will need to change if there are changes in the SRS.

Implemented By: Software Engineering

7.2.1 Login Module (M4)

Secrets: Display logic.

Services: Render the login page and authentication messages

Implemented By: Login

Type of Module: Abstract Object

7.2.2 Registration Module (M5)

Secrets: Display logic.

Services: Render registration page and authentication messages

Implemented By: Registration

Type of Module: Abstract Object

7.2.3 Intro Module (M8)

Secrets: Display logic.

Services: Render intro page

Implemented By: Intro

Type of Module: Abstract Object

7.2.4 Home Module (M9)

Secrets: Display logic.

Services: Render home page

Implemented By: Home

Type of Module: Abstract Object

7.2.5 Test Algorithm Module (M12)

Secrets: The test algorithm

Services: Executes tests on the submitted models and evaluates results

Implemented By: TestAlgorithm

Type of Module: Abstract Data Type

7.2.6 Submit Module (M13)

Secrets: Display logic and error handling.

Services: Render the submit model page and accept only Matlab files

Implemented By: Submit

Type of Module: Abstract Object

7.2.7 Submission Module (M14)

Secrets: Display logic.

Services: Render a model's detailed result including graphical representations

Implemented By: Submission

Type of Module: Abstract Object

7.2.8 Leaderboard Module (M15)

Secrets: Sorting/Filtering Algorithm choice

Services: Displays results of previous submissions, provides sorting and filtering options

Implemented By: Leaderboard

Type of Module: library

7.2.9 Results Module (M16)

Secrets: Display logic.

Services: Render all user's submission results

Implemented By: Result

Type of Module: Abstract Object

7.2.10 Admin Control Panel Module (M17)

Secrets: Admin related logic

Services: Authorization and Execution of admin actions,

Implemented By: django.contrib.admin

Type of Module: Library

7.3 Software Decision Module

Secrets: The design decision based on mathematical theorems, physical facts, or programming considerations. The secrets of this module are *not* described in the SRS.

Services: Includes data structure and algorithms used in the system that do not provide direct interaction with the user.

Implemented By: Software Engineering

7.3.1 App Module (M2)

Secrets: All secrets of child modules

Services: Implement all requirements

Implemented By: All modules

Type of Module: Library

7.3.2 User Authentication Module (M3)

Secrets: Logic for authentication mechanism and password management.

Services: User login/logout, User registration, password recovery/reset.

Implemented By: django.contrib.auth

Type of Module: Library

7.3.3 Database Management Module (M6)

Secrets: The types and structure of retained data. The storage and access methods for retained data

Services: Stores and retrieves data which needs to be retained long term.

Implemented By: Postgresql and data library.

Type of Module: Library

7.3.4 Web Server Module (M7)

Secrets: Routing of modules

Services: Handle routing of requests

Implemented By: Web Server

Type of Module: Abstract Object

7.3.5 Model Execution Module (M10)

Secrets: Method of executing model.

Services: Executing use submitted models against the test suite.

Implemented By: Matlab

Type of Module: Library

7.3.6 Test Data Module (M11)

Secrets: The test data's format.

Services: Provides the data that the model will be tested against.

Implemented By: TestData

Type of Module: Record

8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	Modules
FR1	M7, M13
FR2	M7, M11, M12, M14
FR3	M10
FR4	M7, M5
FR5	M7, M4
FR6	M3, M6
FR7	M6
FR8	M3, M7, M15
FR9	M15
FR10	M15
FR11	M7, M10, M16
FR12	M7, M10, M16
LR1	M8, M9
LR2	M8, M9
UPR4	M8, M9
SLR1	M14
SLR2	M15
SLR3	M15
RR1	M10
IR1	M6
IMR1	M3

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M11, M12
AC2	M15
AC3	M13,M14
AC4	M10
AC5	M10

Table 3: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas (1978) said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 1 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

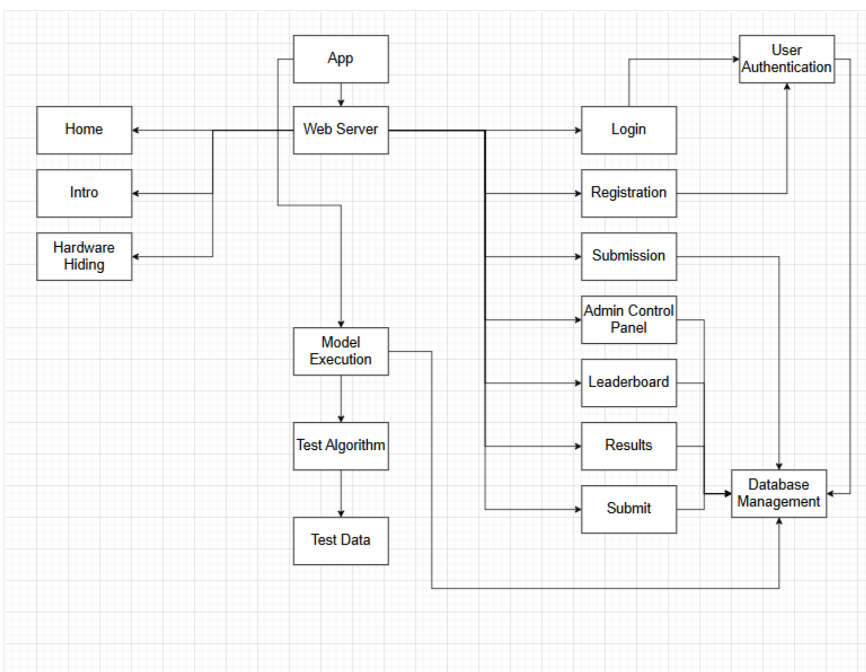


Figure 1: Use hierarchy among modules

10 User Interfaces

Refer to Appendix 1 - 10 (12) for screenshots of the various views for the design of the User Interface.

- 12.1: Intro Page
- 12.2: Registration Page
- 12.3: Login Page
- 12.4: Home Page
- 12.5: Submit Page
- 12.6: Results Page
- 12.7: Submission Page
- 12.8: Leaderboard Page
- 12.9: Leaderboard Page (with sorting)
- 12.10: Leaderboard Page (with filtering)

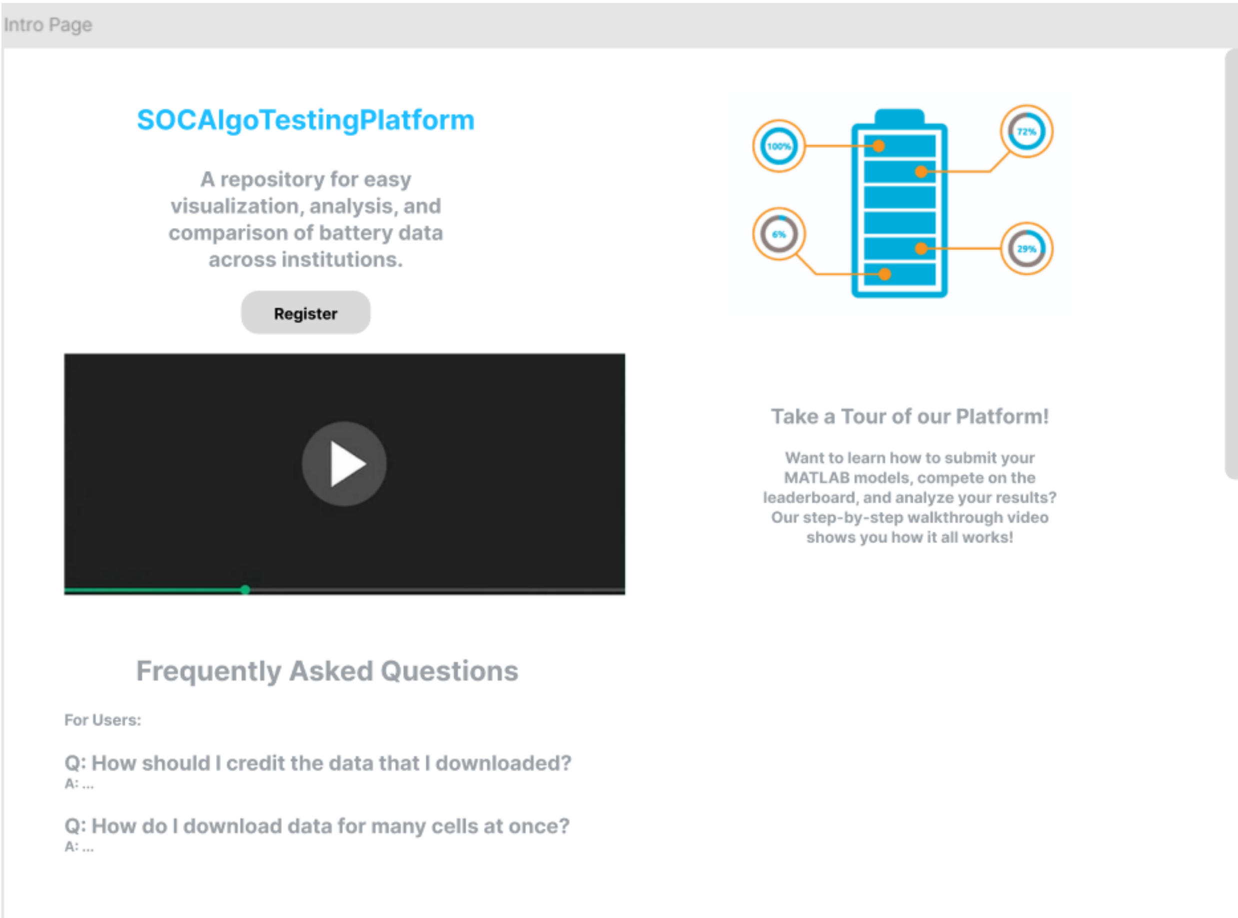
11 Timeline

Refer to the Project Board to view all tasks related to the design described in this document - https://github.com/users/AidanMariglia/projects/1.

All tasks explain the necessary work for completion as well as who is responsible for the task.

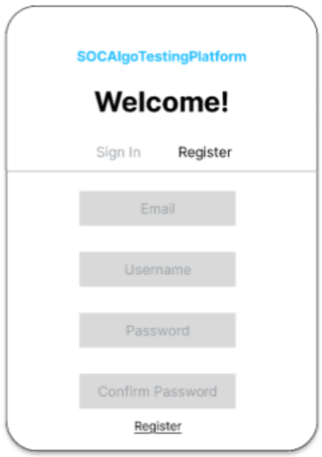
12 Appendix

12.1 Appendix 1

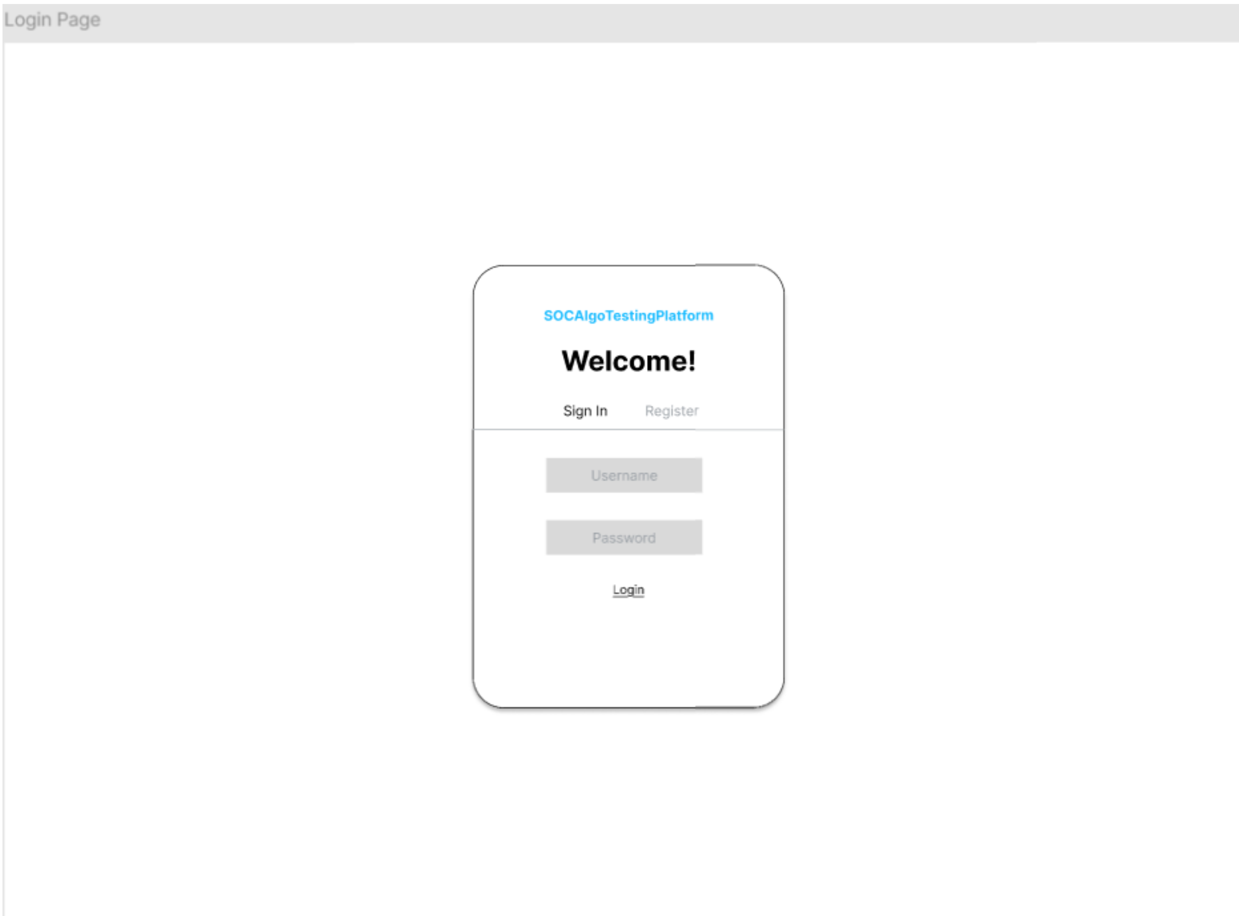


12.2 Appendix 2

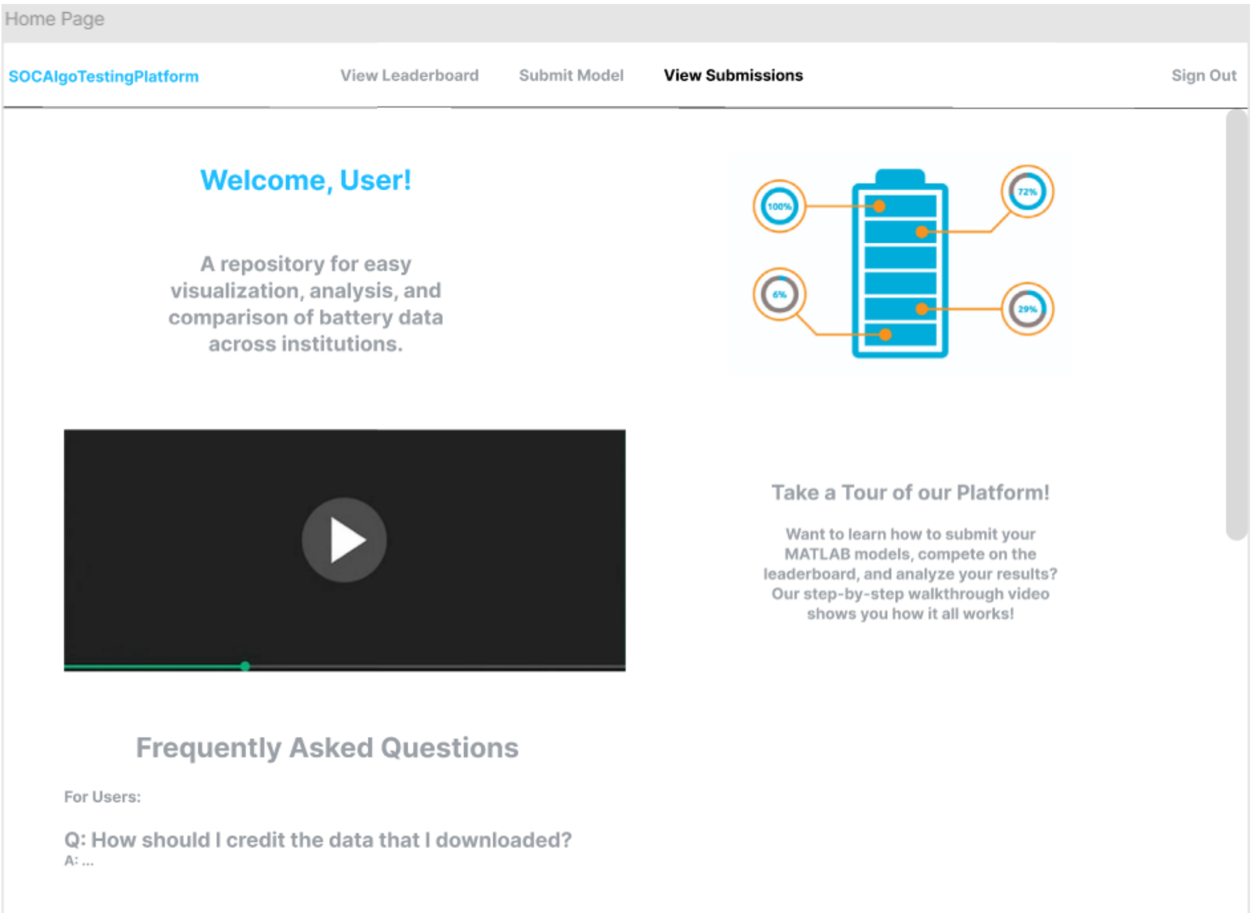
Reigstration Page



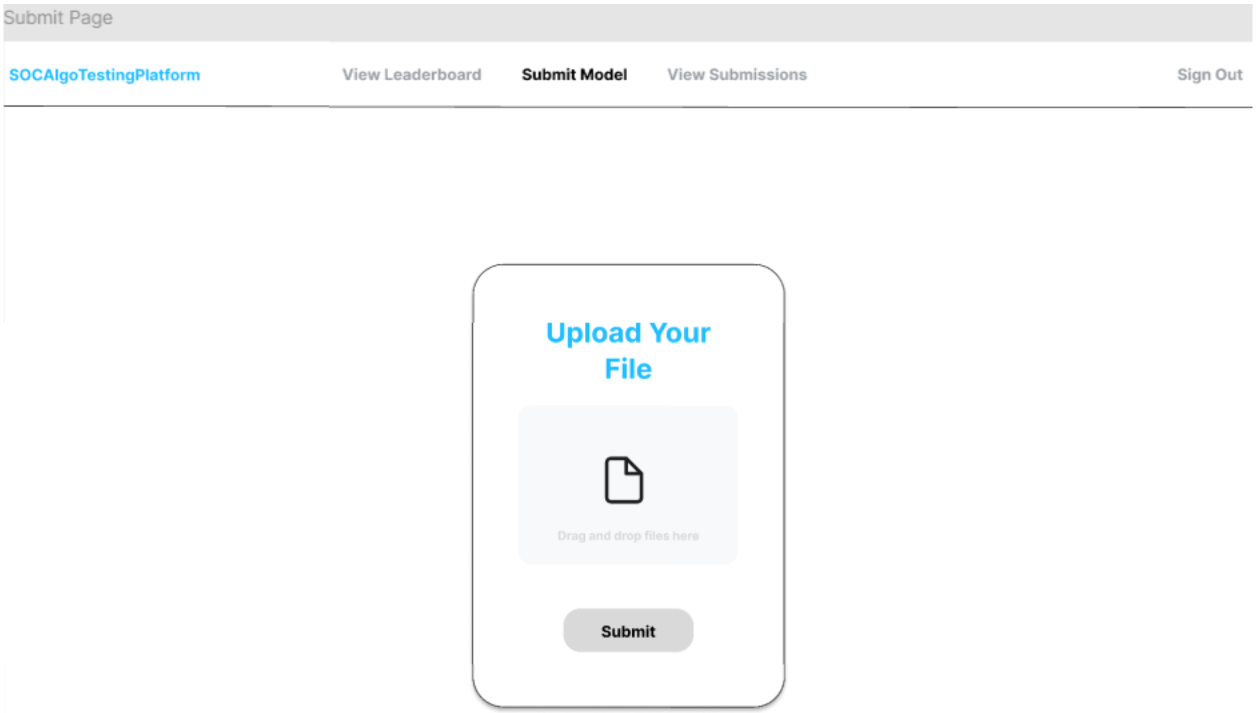
12.3 Appendix 3



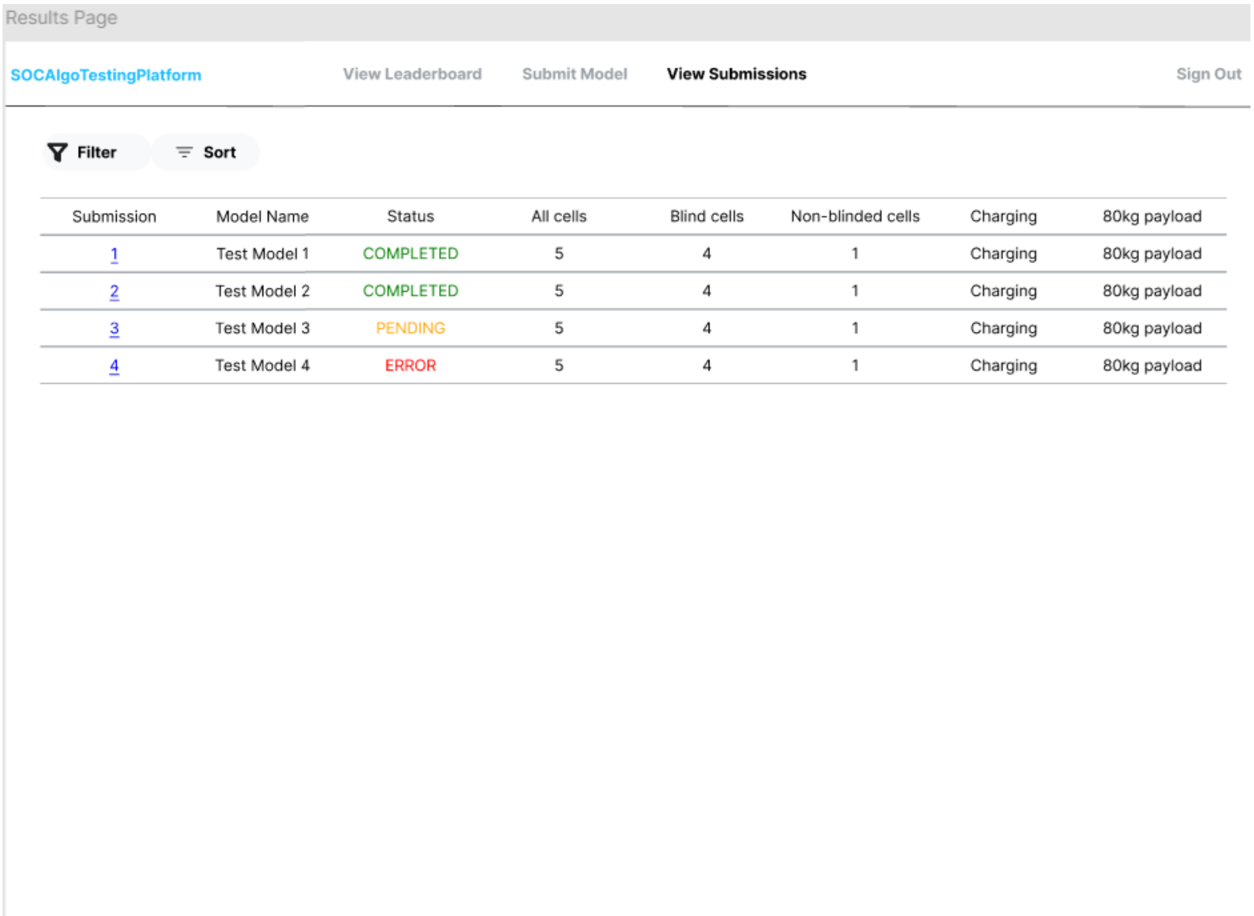
12.4 Appendix 4



12.5 Appendix 5

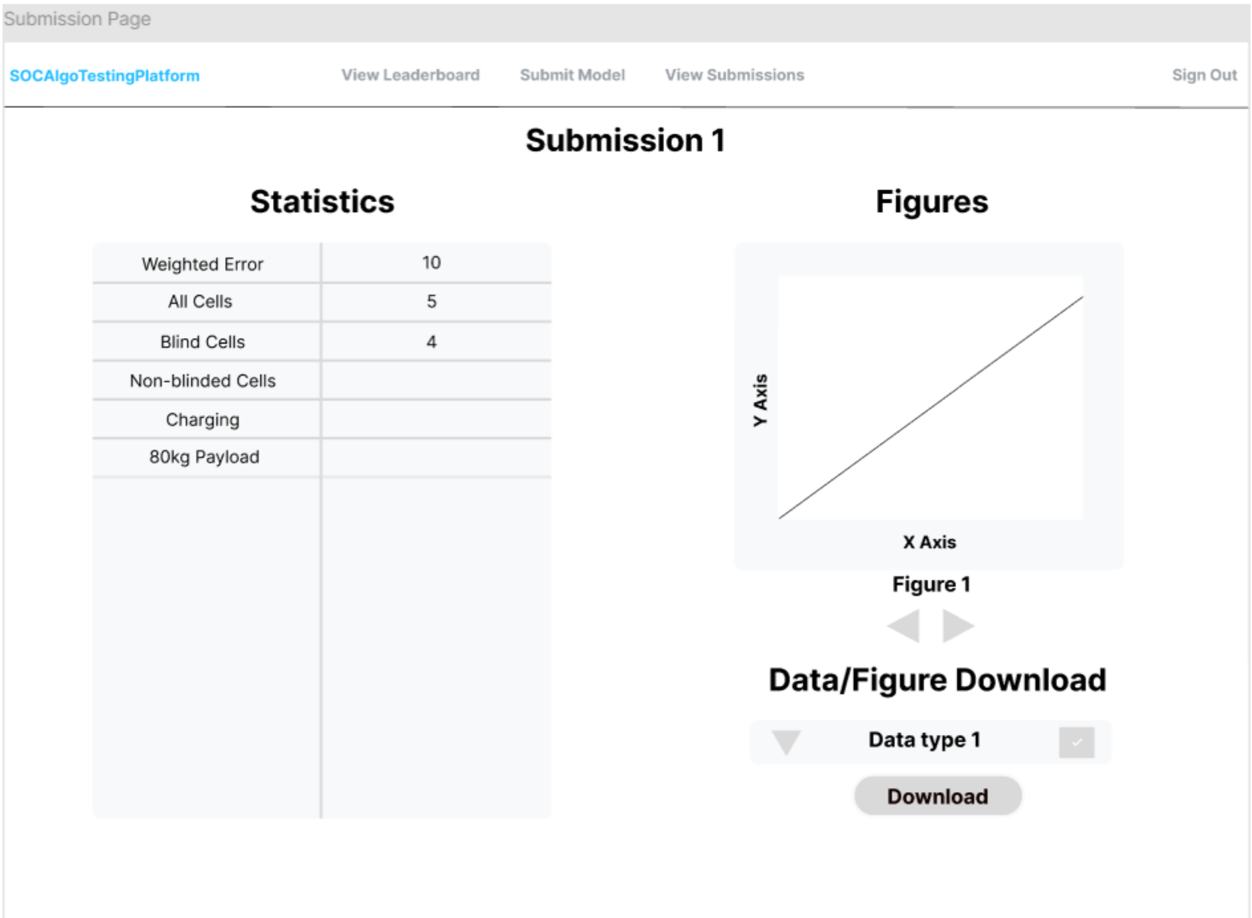


12.6 Appendix 6

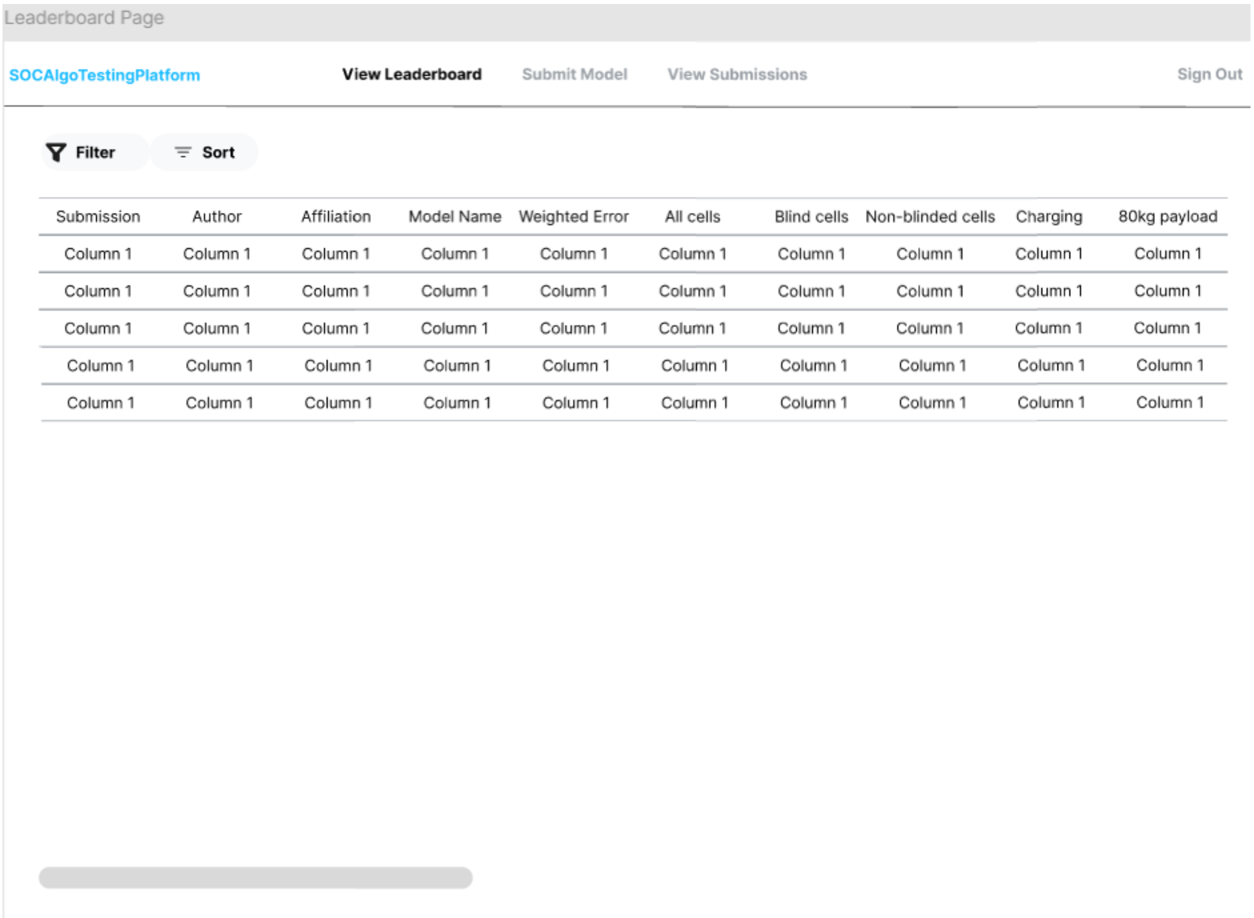


Submission	Model Name	Status	All cells	Blind cells	Non-blinded cells	Charging	80kg payload
1	Test Model 1	COMPLETED	5	4	1	Charging	80kg payload
2	Test Model 2	COMPLETED	5	4	1	Charging	80kg payload
3	Test Model 3	PENDING	5	4	1	Charging	80kg payload
4	Test Model 4	ERROR	5	4	1	Charging	80kg payload

12.7 Appendix 7



12.8 Appendix 8



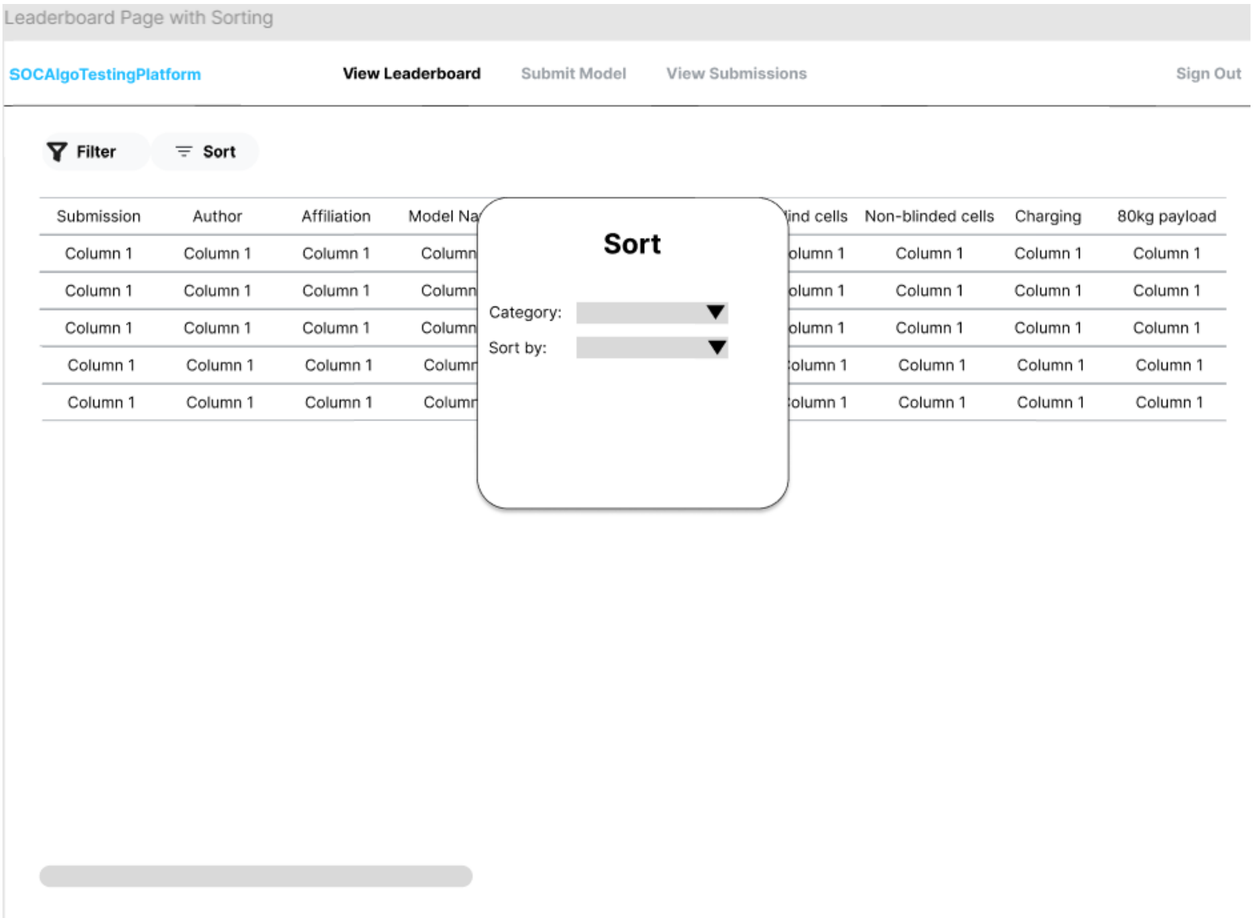
Leaderboard Page

SOCAlgoTestingPlatform View Leaderboard Submit Model View Submissions Sign Out

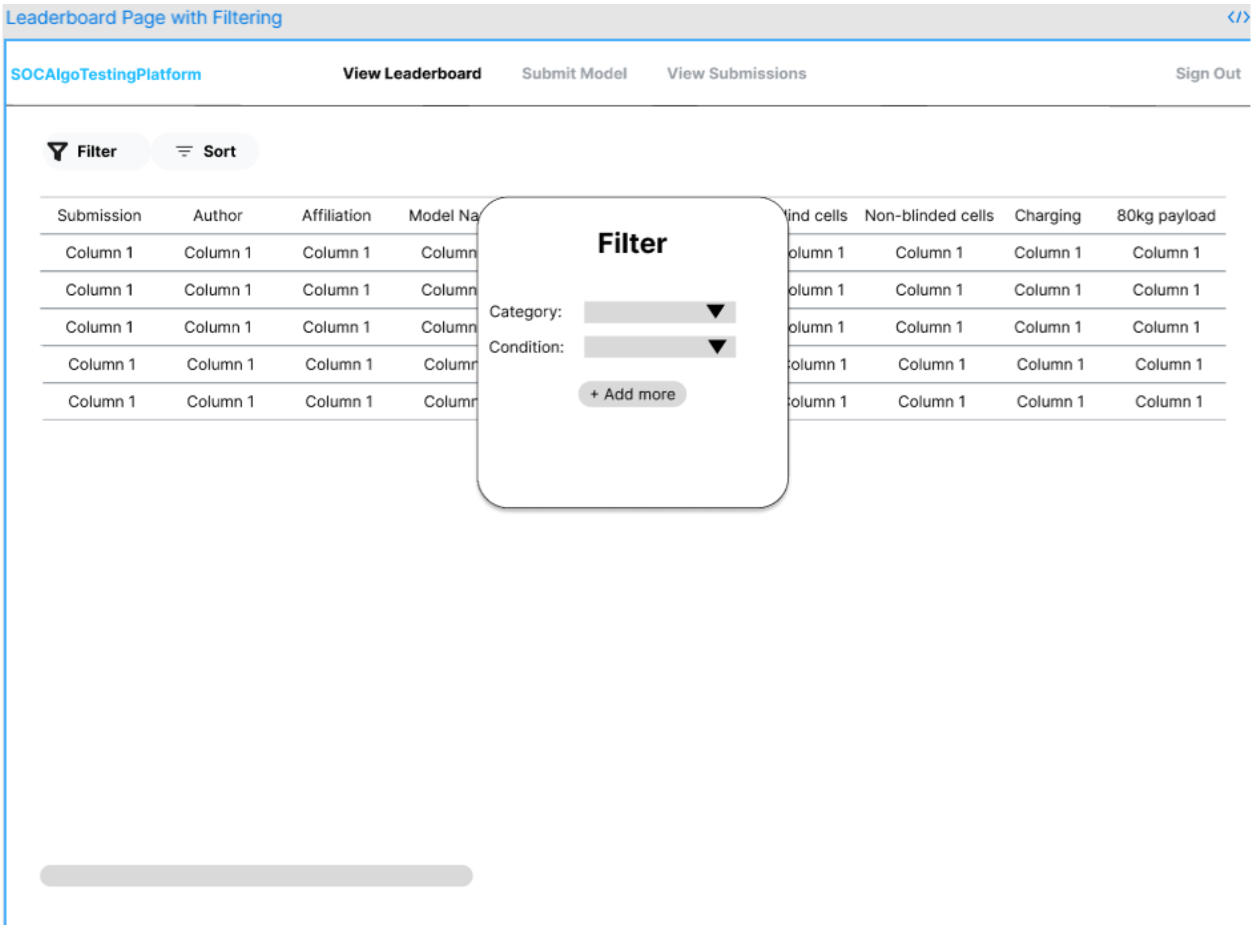
Filter Sort

Submission	Author	Affiliation	Model Name	Weighted Error	All cells	Blind cells	Non-blinded cells	Charging	80kg payload
Column 1	Column 1	Column 1	Column 1	Column 1	Column 1	Column 1	Column 1	Column 1	Column 1
Column 1	Column 1	Column 1	Column 1	Column 1	Column 1	Column 1	Column 1	Column 1	Column 1
Column 1	Column 1	Column 1	Column 1	Column 1	Column 1	Column 1	Column 1	Column 1	Column 1
Column 1	Column 1	Column 1	Column 1	Column 1	Column 1	Column 1	Column 1	Column 1	Column 1
Column 1	Column 1	Column 1	Column 1	Column 1	Column 1	Column 1	Column 1	Column 1	Column 1

12.9 Appendix 9



12.10 Appendix 10



References

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.

David L. Parnas. Designing software for ease of extension and contraction. In *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, pages 264–277, Piscataway, NJ, USA, 1978. IEEE Press. ISBN none.

D.L. Parnas, P.C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.