

AI Diagnostics Benchmark For Breast Cancer

Design Report

Student:

Albina Shynkar
Anne Leijzer
Arda Satici
Krisiyan Dimitrov
Tim Klampe

Supervisor:

Shreyasi Pathak

April 21st, 2023
University of Twente

Abstract

Following report has been written as part of the Design Project module of the University of Twente. The aim of the project is to train breast cancer detection models while keeping the patients' data private. This is achieved by sending the model to the data. Maintaining the confidentiality of sensitive information is a crucial element of this project. In collaboration with Ziekenhuis Groep Twente, we will obtain the necessary data to build the model.

In the upcoming sections of this report, the reader can find elaboration on the requirement specification and analysis, design choices, testing, and future improvements. At the end of the file, the reader will also be able to find appendices that contain screenshots from the working prototype.

Contents

1	Introduction	7
2	Project Proposal	8
2.1	Background	8
2.2	Proposed design	8
2.3	Objectives	8
2.4	Scope	8
2.5	Timeframe	9
2.6	Key Stakeholders	9
2.7	Procedure	9
3	Project Management	10
3.1	Agile Project Management Approach	10
3.2	Versioning and internal communication	11
3.3	Presentation with client	11
3.4	Task division	12
4	Requirement Specification	13
4.1	Requirement Formulation	13
4.2	Requirement Prioritization	13
4.3	Stakeholder Requirements	13
4.4	System Requirement	15
5	Global and Architectural Design Choices	16
5.1	Global design choices	16
5.1.1	Key design pillars	16
5.2	Software architecture	16
5.2.1	Frameworks and Libraries	16
5.2.2	Justification of the choices	17
6	Frontend	18
6.1	Lo-fi Prototype	18
6.2	Hi-fi Prototype	18
6.3	System Overview	18
6.3.1	Home Page	18
6.3.2	Login Page	19
6.3.3	Sign-up page	20
6.3.4	Upload model page	21

6.3.5	Leaderboard page	22
6.3.6	Profile page	22
6.3.7	About us page	23
6.3.8	Admin page	23
7	Backend	24
7.1	General system architecture	24
7.2	REST endpoints	25
7.2.1	Current State	25
7.2.2	Future development	25
7.3	Submitted project structure	25
7.3.1	Current state	25
7.3.2	Future development	25
7.4	Security	26
7.4.1	REST endpoint authentication, security	26
7.4.2	Account application process	26
7.4.3	HTTPS	26
7.4.4	Passwords	26
7.4.5	SQL injections	27
7.4.6	Docker	27
7.4.7	Further developments	27
7.5	Container handling	27
7.5.1	Choosing a base image	27
7.5.2	Building a container	28
7.5.3	Container evaluation	28
7.5.4	Re-running containers	28
7.5.5	Further development	28
7.6	Example dataset	29
7.7	Database	29
7.7.1	Current state	29
7.7.2	Future development	29
7.8	Email	29
7.8.1	Current state	29
7.8.2	Future development	29
7.9	Library compatibility	30
7.9.1	Current state	30
7.9.2	Future development	30
7.10	System configuration	30

7.10.1	Current state	30
7.10.2	Further development	31
7.11	Exporting predictions and calculating metrics	31
7.11.1	Current state	31
7.11.2	Future developer	31
7.12	System requirements and recommendations	32
7.12.1	General hardware	32
7.12.2	GPU	32
7.13	Operation System	32
7.13.1	Scalability	33
7.14	Common errors	33
7.14.1	pip install -r requirements.txt suddenly starts to error out	33
7.14.2	Docker container errors out during building or execution	33
8	Test Plan and System Testing	34
8.1	Test strategy	34
8.2	Front-end	34
8.3	Back-end	34
8.3.1	System Testing	35
8.3.2	Unit testing	35
8.4	Test roles & responsibilities	36
8.5	Test Reporting and Deliverables	36
9	Testing Results	37
9.1	Front-end Results	37
9.1.1	Automated Selenium Testing	37
9.1.2	User experience testing	37
9.2	Back-end Results	37
9.2.1	Database Testing	37
10	Risk Assessment	39
10.1	Executing unknown scripts	39
10.2	Uncertainties with scalability	39
11	Limitations	41
12	Future Improvements	42
13	Conclusion	43

14	References	44
15	Appendix	45
15.1	Appendix A: UML Diagrams	45
15.2	Appendix B: Lo-fi Design	46
15.3	Appendix C: Hi-fi Frontend Design	50

1 Introduction

Breast cancer has been a significant health concern for a long time. It is characterized by the abnormal increase of breast cells. The specific type of breast cancer diagnosed in a patient depends on the particular breast cells that undergo malignant transformation into cancer cells. (CDCBreastCancer, 2022a)

Several breast cancer screening methods have been developed for its diagnosis. The present project centers on the implementation of mammography as a detection method. Mammography involves taking X-ray images of the breast and is regarded as the optimal diagnostic tool for detecting breast cancer at an early stage. Routine mammography screening can significantly decrease the mortality rate associated with breast cancer. Presently, mammography is the preferred breast cancer screening method for most women who fall within the recommended age range. (CDCBreastCancer, 2022b)

The application of artificial intelligence can help accelerate the detection process. Nevertheless, training these artificial detection models takes rather a long time. Especially gathering the training data set can be a time-consuming process. For instance, it took our client 7 months to collect a complete training data set.

According to the article ‘Bring the model to the data: The Deep Learning Epilepsy Detection Challenge’, patient’s data is extremely sensitive and its usage therefore strictly regulated. Various research groups independently gather data for the diagnosis of breast cancer, with strict privacy regulations prohibiting the sharing of these datasets. Hospitals that possess the required data for research and educational purposes are hesitant to disclose it directly to external parties, out of fear of potential data breaches and fraudulent activities. Consequently, the unavailability of these datasets leaves research groups without access to them from adequately training their models. Without access to complete and relevant datasets, scientists are restricted from conducting high-quality research in this field.

To solve this challenge, the authors of this report (to be addressed as “the team” from now on) propose an alternative approach to be employed, whereby the breast cancer detection models will be sent to the data, as opposed to the transfer of the data to the various models. This model-to-data architecture preserves the confidentiality of sensitive datasets while enabling the requisite model training to occur.

The project will involve the development of a website that will serve as a platform for model developers to submit their code. The system will facilitate the automatic creation of a docker image to enable seamless submission of the code to the organization that possesses the required data. Upon completion of the script, the developer will receive the output results. Furthermore, a leaderboard will be available to allow users to compare the performance of various models.

2 Project Proposal

2.1 Background

Different research groups collect their own data for breast cancer diagnosis. They do not share the data sets because of privacy concerns. Hospitals, where the researchers might have access to the data for learning purposes, prefer to not share the data directly to prevent leakage of the data and fraud. This makes it hard for research groups to train their models without the needed access to data sets. Without the appropriate/complete data, scientists cannot perform qualitative breast cancer research.

2.2 Proposed design

To resolve the issue mentioned above a web application is proposed, which will allow a detection model training without providing external access to the data set. The project will consist of a web application where the user can upload their detection model. A Python back-end communicates over a REST API with a web-based front-end. The data scientist will be able to submit the model on the website, which will be packaged into a docker container and executed. Then, after the model evaluation is completed, the evaluation result should be shown to the data scientists. A leaderboard will give an overview of which models have the highest accuracy, precision, and recall.

2.3 Objectives

- Keep the data set private
- Send model to data, and return the results
- Reproducible environment for code evaluation
- Platform with a leaderboard to compare results

2.4 Scope

The end result of the project will be a platform that will take the model to the data (MTD). This paradigm will make use of container software and cloud computing. Therefore, the data will be kept within the hospital, and the privacy of sensitive data will be ensured.

This will be achieved by pursuing the following phases: planning phase, designing phase, implementation phase, testing phase, and finalizing phase. At the end of the project, a descriptive report and a poster presentation will be presented next to the final product and its documentation.

2.5 Timeframe

	Description of Work	Start and End Dates
Phase One	Project proposal, planning, design, requirements engineering	6 February 2023 - 24 February 2023
Phase Two	Prototyping, implementation, and testing	24 February 2023 - 31 March 2023
Phase Three	Preparation of final product, poster presentation, and finalizing report	31 March 2023 - 21 April 2023

Table 1: Descriptions of the tasks and the timeframes

2.6 Key Stakeholders

Client	Shreyasi Pathak, Ziekenhuis Groep Twente (ZGT)
Users	They will submit their detection model(s) to get them trained.
Training data provider	ZGT hospital
Future maintainer	IT staff of ZGT hospital. Their responsibility is to sustain and expand our implementation.

Table 2: Explanations of the key stakeholders

2.7 Procedure

Every Monday, the developer team has meetings with the client in order to discuss the process and get feedback regarding the work done. Everything discussed during the meetings is noted down by the team in order to be able to get the highest benefit from the client. Communication within the team is held via Discord and Whatsapp. Communication with the client is held via email and Discord.

3 Project Management

In this chapter the process of the creation of a proposal for the system to the client is described and all meetings with the stakeholder are analyzed, different choices that were proposed to the client have been discussed as well as the presentation of the final system.

3.1 Agile Project Management Approach

For the purpose of this project, the Agile methodology was chosen. Agile is an iterative methodology that consists of several development cycles called sprints. The duration of the sprints is two weeks. This duration is synchronized with the peer review session provided by the organization of the design project module.

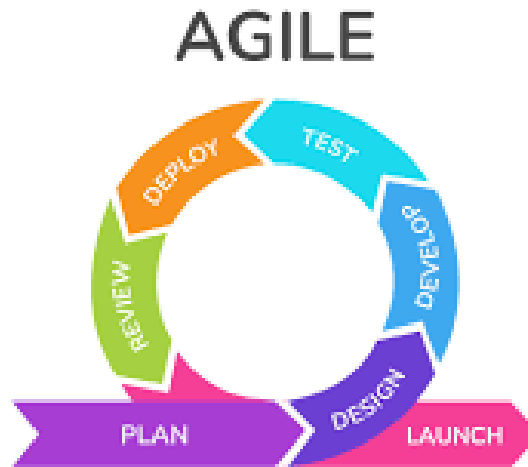


Figure 1: Agile project cycle

In order to make sure that the whole team is well-updated on the progress we have daily scrum meetings. During these meetings, we discuss the current progress, the tasks that still need to be completed, and any issues or uncertainties if necessary.

Also, we meet with our supervisor on a weekly basis as mentioned in Section 2. When a demo is presented or conceptual matters are discussed, stakeholders (from ZGT Almelo) are present too.

3.2 Versioning and internal communication

For communication within the team Whatsapp and Discord are used. For versioning GitLab is utilized. GitLab enables collaborative work on code and ensures that all modifications are recorded. In addition, boards with issues are utilized in GitLab to monitor tasks, assign responsibilities, and track progress.

3.3 Presentation with client

A total of four presentations will be given, including one poster presentation and three system presentations. The module coordinator will organize the poster presentation, which will include a poster, a brief project presentation, and a demonstration. The remaining three presentations will be presented to the hospital, our supervisor's chair, and our supervisor's research group. These presentations will cover the complete system in detail and a demonstration of how to use the system will be provided. Following the presentations, enough time will be allocated for discussing any questions that may arise.

3.4 Task division

	Albina Shynkar	Anne Leijzer	Arda Satici	Krisiyan Dimitrov	Tim Klampe
Lo-fi prototype		✓		✓	
Hi-fi prototype		✓		✓	
Home page	✓		✓		
Login/ Sign up	✓	✓	✓		
Upload model page	✓	✓	✓	✓	✓
Leaderboard page	✓		✓		✓
About us page	✓		✓	✓	
Admin page	✓		✓		
Backend functions					✓
Google login		✓			
Mail service		✓			✓
Connecting back and front-end		✓			✓
Testing model				✓	
Front-end testing		✓			
Back-end testing			✓		✓
Poster		✓			
Reflection assignment	✓	✓	✓	✓	
Report	✓	✓	✓	✓	✓
Presentation	✓	✓	✓	✓	✓

Table 3: Overview of task division

4 Requirement Specification

This chapter describes the process and approaches to gathering requirements, including formulating and prioritizing procedures tailored to meet the client's specific needs.

4.1 Requirement Formulation

The Agile project management practices have been adopted for the requirement specification. Iterative loops have enabled the team to establish clear requirements that match the client's needs.

The requirements served as guidance for the development phase. During the development phase, the requirements were updated when necessary. However, the good sign is that the core requirements from stakeholders remained unchanged during the product development. This suggests that the core functionalities of the system were well understood and elaborated.

4.2 Requirement Prioritization

After the requirements are specified by the team members, and discussed with the client, requirement prioritization was applied to the requirements.

The name of the technique used is MoSCoW Prioritization method. Below you can find the explanations of MoSCoW components taken from ProductPlan.com.

Must Have: Non-negotiable product needs that are mandatory for the project.

Should have: Important initiatives that are not vital, but add significant value.

Could have: Nice to have initiatives that will have a small impact if left out.

Will not have: Initiatives that are not priority for this specific time frame.

4.3 Stakeholder Requirements

After conducting meetings with our stakeholders, and applying the techniques mentioned before, an extensive list of requirements has been formulated.

Must

1. As a data scientist, I must be able to submit an evaluation model on the website.
2. As a data scientist, I must be able to provide a name when submitting the evaluation model.
3. As a data scientist, I must be able to provide a description when submitting the evaluation model.
4. As a data scientist, I must receive the evaluation results of the submitted model in return.

5. As a data scientist, I must receive the trained model in return.
6. As a stakeholder, I must be able to see a leaderboard with the scores of all evaluation models.
7. As a data scientist, I must be able to submit any Python file that is executable in a standard Linux environment to be used in training the model and evaluation.
8. As a stakeholder, I must be ensured that a malicious input would not negatively influence the system.
9. As a sensitive data provider (hospital), I must be ensured that the dataset is kept private.
10. As a stakeholder, I must be able to create an account.
11. As a data scientist, I must be able to log in to my account.

Should

1. As a data scientist, I should have an overview of the version of the runtime environment.
2. As a data scientist, I should be able to provide my version of the runtime environment.
3. As a stakeholder, I should be able to see who submitted a certain model.
4. As a stakeholder, I should be able to see the date and time of the model submission.
5. As a stakeholder, I should be able to edit my account.
6. As a data scientist, I should be able to download my trained model after evaluation analysis.
7. As a data scientist, I should be able to observe the evaluation progress.
8. As a data scientist, I should be notified when my model is done with the evaluation.
9. As a user, I should have a page with information about how to use the web application.

Could

1. As a stakeholder, I could be able to add comments on a model evaluation.
2. As a stakeholder, I could be able to respond to comments on a model evaluation.

Will not have

It was decided to not identify what does not have to be done or avoided to be implemented as the team concentrates fully on what has to be done in order for the system to meet client's expectations in the first place.

4.4 System Requirement

Besides the stakeholder requirements, according to what is expected from the system, the system requirements are formulated based on the stakeholder requirements. These requirements are the identification of the functionalities the system has.

- The system must be able to run a model evaluation with the submitted code on the provided dataset in a docker container.
- The system must store the user's results in their account.
- The system must be intuitive to use.
- The system must be user-friendly.
- The system must work on multiple operating systems: Windows, Linux, and Mac.

5 Global and Architectural Design Choices

This chapter discusses global and architectural design choices and their justification. The system structure is being discussed on a global level, global and preliminary design choices are described and the general system overview is provided.

5.1 Global design choices

In order to develop the system a decision has been made to ensure that the main objectives are satisfied. This decision introduces the key design principles the system is built upon that from now on will be addressed as key *design pillars*.

5.1.1 Key design pillars

Security: During, but not exclusively, the design phase of the web application, the security aspects of the application were the most crucial design pillar for us. Since the platform includes sensitive data, security techniques are applied in every section of the application as mentioned in Chapter 8.

Usability: During, but not exclusively, the design phase of the web application, the ease of understanding the user interface was one of the prioritized aspects of the application. Since a web application that is easy to use and understand improves the user experience, the usability of the web application is improved with exploratory usability testing (see Section 7.2).

5.2 Software architecture

In this section an overview of the software design choices for both the frontend and backend is provided. Each of these is introduced and an explanation on the way they are utilized is also provided.

5.2.1 Frameworks and Libraries

FRONTEND

For the purpose of this project *React* is the preferred technology. *React* is an open-source JavaScript library for building web applications. It can create reusable UI components allowing to implement complex, interactive and scalable systems.

@react-oauth/google dependency is used in the Login and Logout functionalities of our web application. With the help of this functionality, the users can log in and log out with their Google accounts.

In order to improve the look and user experience of the web application, the development team used *Mantine*. It is an open-source *React* components and hooks library that contains a large set of components including buttons, forms, navigation bars, tables etc. It is used to make the web application modern and user-friendly.

For the implementation of front-end, *Typescript* is used and it is the main programming language used in this project. It is an extension of JavaScript. *CSS* is used in the stylings of the webpages. For our automation testing of our web-application the chosen technology is *Selenium*.

BACKEND

Python is the language used in the implementation of the back-end. Docker container creation, database creation and inserting data in the database, running the algorithm, and creating the results of the training are all done with the help of Python. In order to manage the database and HTTP requests, SQLite is used. Besides SQLite, within our application, Docker is used to help us when building and delivering the functionalities as quickly as possible.

5.2.2 Justification of the choices

FRONTEND

In our frontend, the main application that we are using is *React* and it is chosen because of the fact that it is very fast and flexible. Also, it is well-documented. For the google-login feature, *React-oauth/google* is used because it is the latest version of the authentication library. Initially, react-google-login was implemented. But then it was noticed that this library will be deprecated so *React-oauth/google* is used. For the frontend, *Mantine* is used and is compatible with *React*, and it is well-documented, fast, and the components of *Mantine* are very useful for the platform. *Selenium* has been used a lot in the team's previous projects, and because of the fact that *Selenium* has open source availability and advantages when implementing automated tests, *Selenium* was chosen.

BACKEND

Python is by far the most popular programming language. Therefore, Python supports many frameworks and libraries that are useful for the project. Also, since we are using Docker containers and Machine Learning model training, Python is the best choice for us. Having libraries like 'numpy, tensorflow' makes our job easier.

6 Frontend

This chapter describes the progress of frontend implementation, discusses decisions taken regarding it, gives an overview of every page of the implemented frontend.

6.1 Lo-fi Prototype

After establishing the requirements the initial Lo-fi prototype was produced. To create the first visualizations of the system Figma was used. Figma is a web-based application for designing interfaces.

Three different interfaces with unique flows were created. Two of the interfaces consisted of multi-page applications, while the third was a single-page application with multiple pop-ups. (Screenshots of the various interfaces can be found in Appendix B) The different designs were shared with the client. After a meeting to discuss the possible interface, the decision was made to use the second design. This multi-page interface was the most intuitive and captured all the requirements in a user-friendly way. During the meeting, it was decided that a page with information about the project should be included. This page should explain the goal, the team, and how to use the application.

6.2 Hi-fi Prototype

With the initial Lo-fi prototype as a reference, the Hi-fi prototype was created. The Hi-fi prototype of the system helped to gather feedback from the stakeholders. The purpose of this feedback was to identify areas that required modifications before proceeding with the implementation phase.

Upon receiving feedback, the decision was made to remove the message and notification buttons as these features were not crucial to the stakeholders. The team was granted full design autonomy since there were no specific house styles or color schemes that needed to be adhered to.

6.3 System Overview

Finally, after incorporating all the feedback, the implementation started. The front end of the system consists of a total of 7 pages. On the following pages, an overview of the web application will be given.

6.3.1 Home Page

When accessing the web application, the home page will be the first page that appears. It provides an overview of the platform along with a description of its functionalities. In addition, users can easily access the login/signup page through the corresponding buttons.

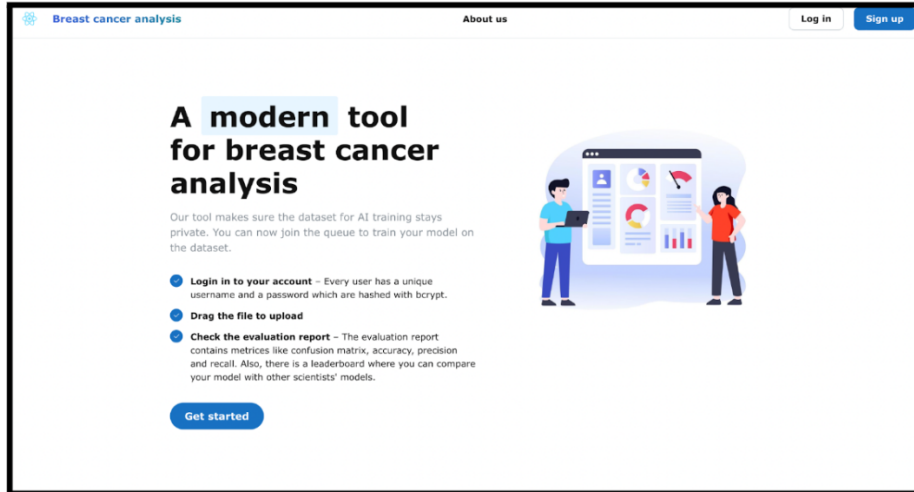


Figure 2: Home page

6.3.2 Login Page

The Login page has the feature of 'logging in with Google', which makes the lives of the users easier. If the user does not want to log in with Google, then the user can log in with their account by introducing their email and password. If the user does not have an account yet one can be created on the Sign Up page.

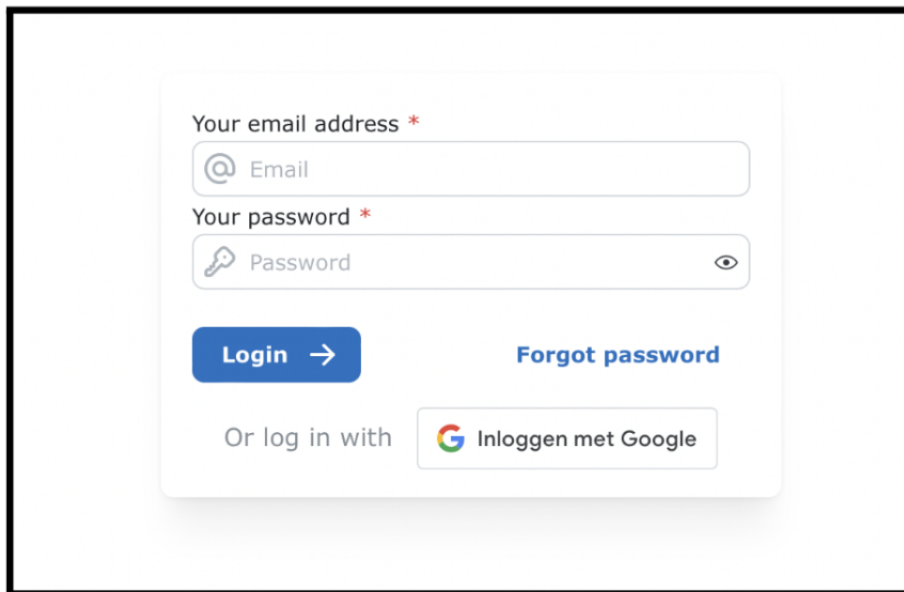


Figure 3: Login page

6.3.3 Sign-up page

The sign-up page allows the user to create a new account by providing personal information such as their name, email address, password, and details about their institution. Once the user finishes creating their account and submits the form, a request will be sent to the platform's administrative staff for approval or rejection.

Sign up form

Note: this form is sent to the board which will check whether you have proper rights to access this tool. This might take a while.

Your name *

Your surname *

Your email address *

Your password *

Confirm your password *

Institution *

Address *

Phone number *

I've read and accept the [terms and conditions](#)

Send request →

Figure 4: Sign-up page

6.3.4 Upload model page

This is the page where the users can upload their models for training. The users should enter a model name, the programming language, and they upload a zip file with the requirements.txt file and their model that they want to get trained. Once the model is uploaded, it will appear in the box on the right. This box shows the progress of all uploaded models. The user can see where in the queue their model is.

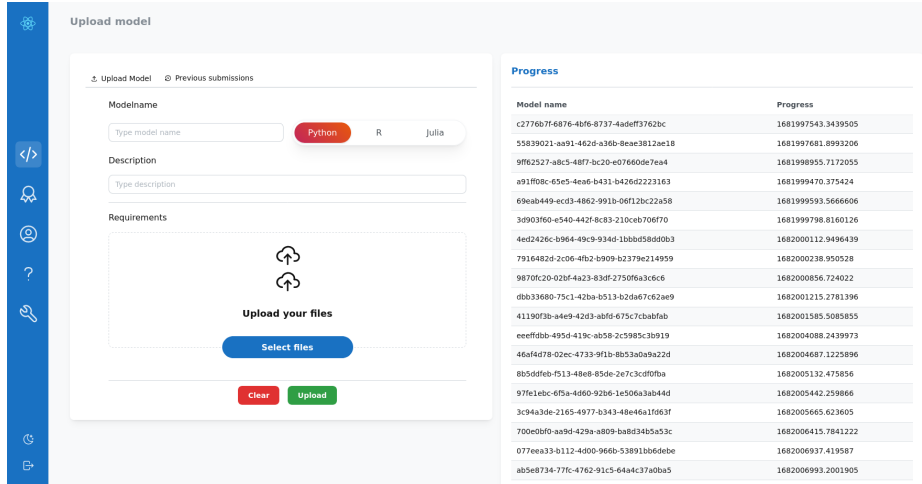


Figure 5: Upload Model page

The box on the left has two tabs. In the first tab the user can upload their model. In the second tab the user can observe all their previous submissions and rerun them if wanted.

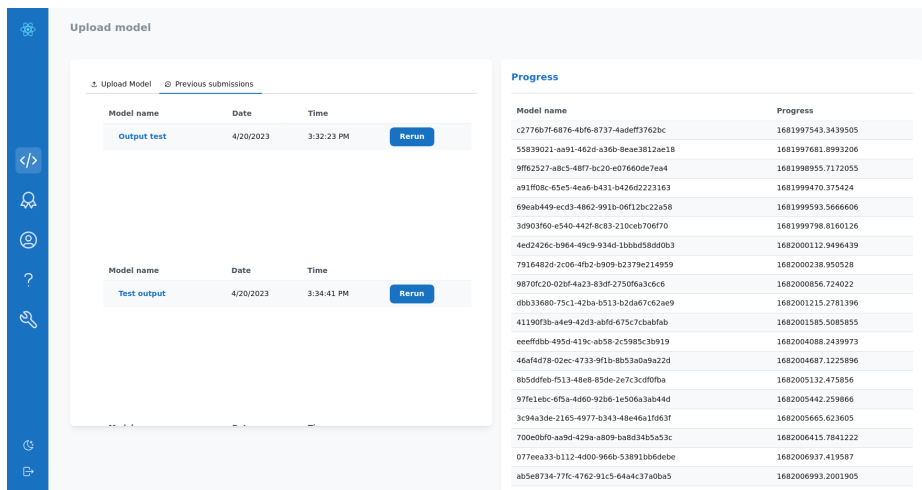


Figure 6: Upload Model page

6.3.5 Leaderboard page

The leaderboard page has the features of the model ranking according to Accuracy, Precision and Recall. Also, when the user clicks on the data shown on the leaderboard, a detailed overview of the model appears. If the model belongs to the user the option to download the model will also show up.

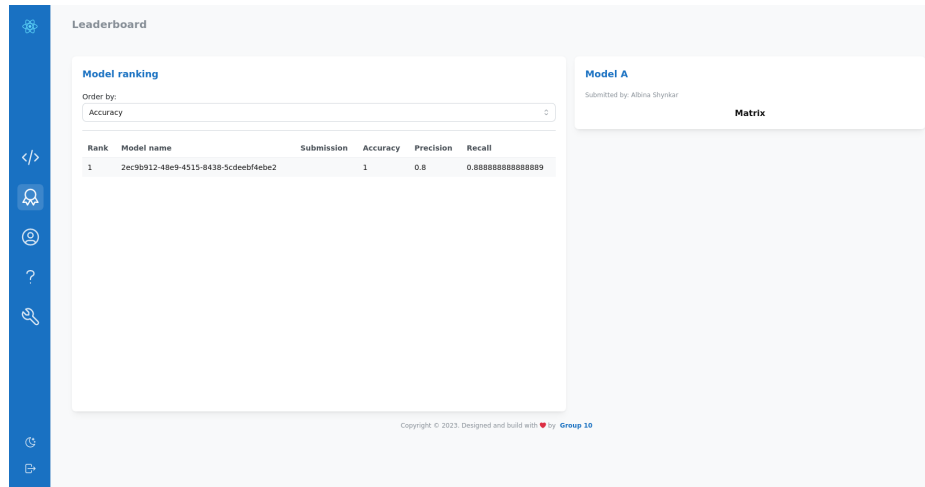


Figure 7: Leaderboard page

6.3.6 Profile page

The profile page includes an overview of the models of that user. The model name and the status of the model training are shown. Also, this page has features like changing the user settings and institution information.

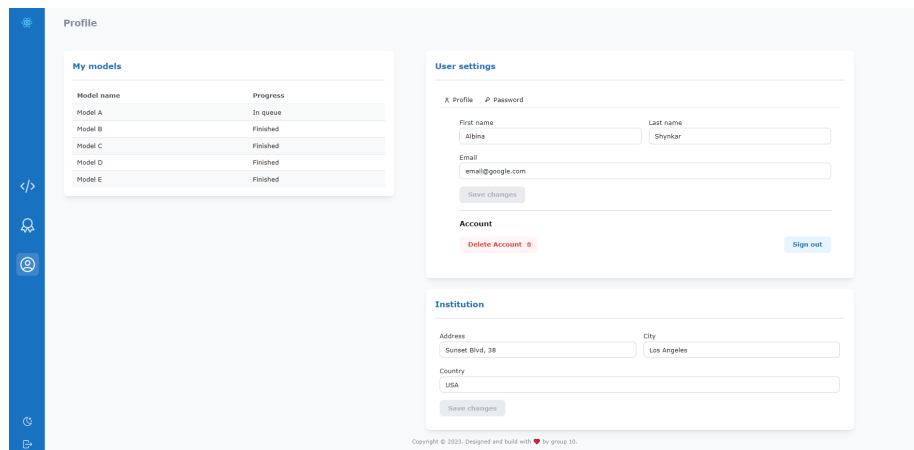


Figure 8: Profile page

6.3.7 About us page

The About Us page includes 2 sections. The first section focuses on how to use the application. This section includes instructions about how a user can sign up/login and how to evaluate a model. The second section explains the goal, proposed design and the functionalities of the platform.

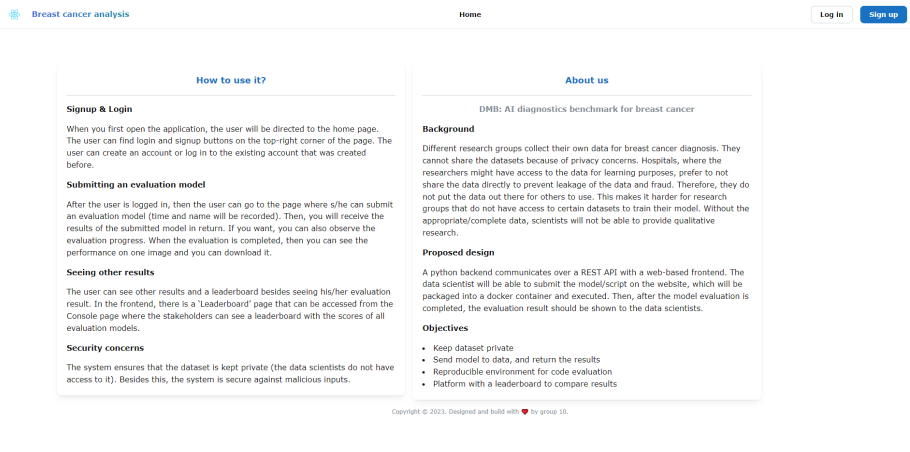


Figure 9: About us page

6.3.8 Admin page

The Admin page includes 2 sections. The first section is to keep track of the model trainings and the second section is to control the users within the platform. The admin can rename a model or remove a model from the system, and grant access to the signed-up users after checking the credentials.

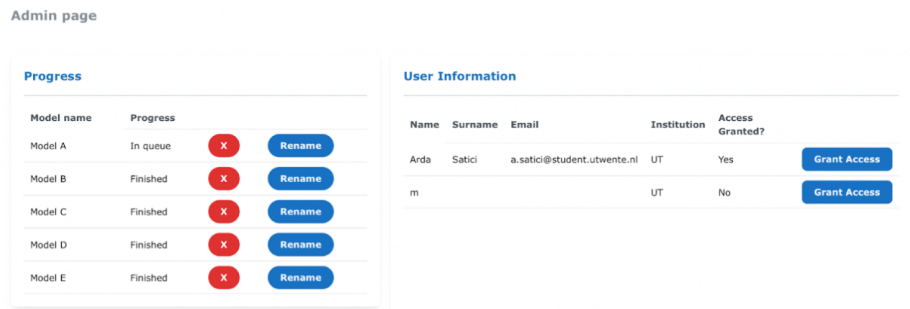


Figure 10: Admin page

7 Backend

This section is meant as a technical description of the backend. It is intended as a reference for deployment and further development. Please see the source code for more detailed information and elaborations on source code. For all sections, first an overview of the implemented status including eventual shortcomings is presented. Then recommendations for continued development are listed.

As of now, the backend is not fully finished yet. Most functionality is implemented, but not extensively tested so instability is to be expected.

7.1 General system architecture

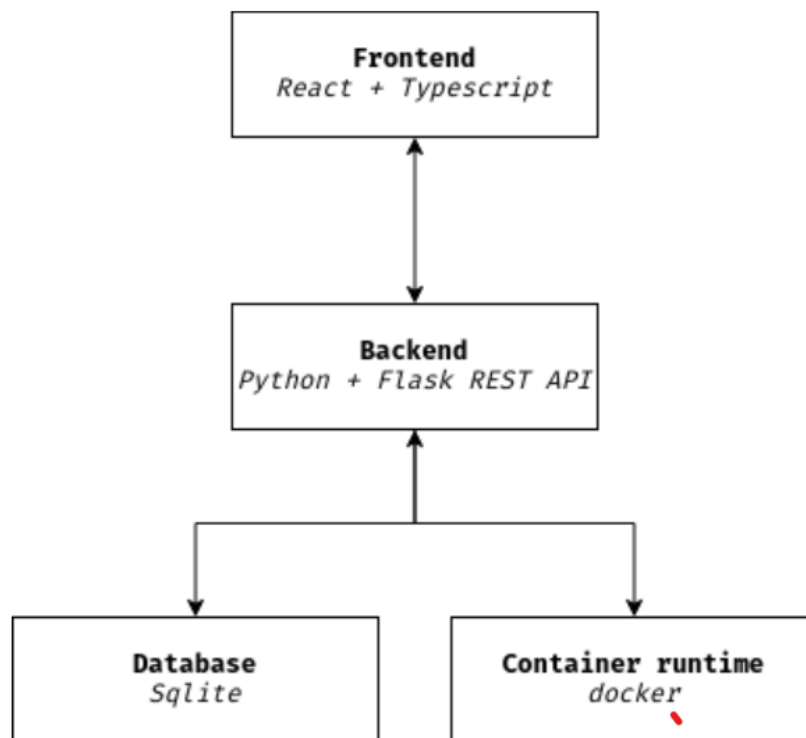


Figure 11: General System Architecture

In the following the general system architecture will be elaborated for each block in the above diagram.

The **frontend** communicates with the backend over a REST API. The frontend has been described in a previous section.

The **backend** has been programmed in python, together with flask as a web application framework. This combination has been chosen as we have no special performance requirements that would justify not using python, and flask is the well documented standard in building REST API's in python.

As the **database** we chose Sqlite, as we have no special performance requirements towards our data storage. Sqlite allows us an extremely simple and robust solution, that does not require any special deployment attention. Using Sqlite in python is also very easy and straightforward.

In order to run and evaluate uploaded source code we need to provide a runtime environment. We chose docker, as this **container runtime** allows us to efficiently build dynamic environments according to user's needs and then use available host resources to run compute intensive machine learning tasks. Docker is the standard in container runtimes, and while there are alternatives we chose docker as it is the most established and most documented solution.

7.2 REST endpoints

7.2.1 Current State

As mentioned in the introduction, we expose a REST API using python and flask. This allows us to develop the front and back end separately and only later then connect the two elements. For an overview of the endpoints and their parameters please see the source code.

7.2.2 Future development

The REST API should be updated to accommodate future developments, as of now the solution is sufficient to effectively work together with the frontend.

7.3 Submitted project structure

7.3.1 Current state

As of now we require a rather specific project structure in order to successfully run submitted code in our system. We require a file called train.py to be present in the root of the submitted project, this is the entrypoint of our execution. We will start this file, the user has to ensure that the complete submitted project can be executed through starting this specific file.

Then we also require that the user submits a requirements.txt in which the project dependencies are specified. We do not impose any restrictions on what python packages can be installed through this file.

7.3.2 Future development

We have started to investigate how Conda can be used to install package dependencies. Due to time constraints we were not able to fully get the implementation to work, but we think that this would be a viable way to streamline the installation of system dependencies. Conda would also allow the installation of project dependencies not only written in python, but also in R or Julia. So implementation effort for one language could be carried over in other language support.

7.4 Security

7.4.1 REST endpoint authentication, security

Security with regards to the REST API shows two different things that need to be ensured: Only authenticated users shall be allowed to interact with the system and those users should only be allowed to access information that is connected to their own user account. Only the admin user is allowed to see all data in the system.

Generally, authentication is done with the help of flask pyjwt. We issue JWT tokens, that are then subsequently used to authenticate against endpoints. JWT token also allows us to check if the requested resource belongs to the current user, or if the user is trying to access information from another user.

Preparatory work has been done to enable admin user functionality, but this work has not been finished due to time constraints. Right now, all users have the same rights.

7.4.2 Account application process

In order to make the system only accessible for users that have a genuine use-case, an account application process is envisioned. This means that the user first creates an account and supplies personal information. This information is then reviewed by hospital staff, which then decides case-by-case whether the account is to be activated or not. Only after the activation of the useraccount, code can be submitted and executed. This account application process has not been fully implemented due to time constraints, preparatory backend work has been started.

7.4.3 HTTPS

In order to secure the communication between frontend and backend, an HTTPS certificate needs to be configured in the system. One way to achieve this is by configuring flask to use HTTPS. Another way would be to make use a reverse proxy that handles the certificates.

After the backend is accessible through HTTPS, the frontend needs to be updated to switch all endpoints to HTTPS.

For most of the development workflow we have been using self signed HTTPS certificates, but due to increasing complexity of handling certificates from an unknown certificate authority we have decided to switch to HTTP for the time remaining. Switching the current system to HTTPS should not require much time and effort.

7.4.4 Passwords

As we have a user authentication workflow, we need to be careful about handling passwords of users. We send the password from the frontend to the backend over a HTTPS connection, so security in

transport is ensured (given a proper certificate has been configured). In the backend we hash the password with bcrypt together with a salt and then save it in the database. This means that even if the database is published, no passwords can be recovered.

7.4.5 SQL injections

SQL injection not an issue, as we are using prepared statements throughout the backend.

7.4.6 Docker

One of the biggest security risks in this project is of course that we are blindly running code that users upload to our platform. Absolute security is impossible to achieve in this scenario, but we can achieve reasonable security by implementing a few design choices.

First, the container has no internet connection during running. This already mediates the risk of remote control of the running code or also just uploading the dataset to an external server. Secondly, the only input is through read-only mounted filesystems that contain the dataset, and thirdly the only output is through a shared file system in which the predictions of the user are being saved. This approach tightly controls the dataflow into and out of the container.

7.4.7 Further developments

We have started the implementation of google sign in, but have not finished the work. The frontend should be in a good state, but the backend needs to be extended for this authentication workflow. We recommend researching if rootless docker can be used for this project. Currently we are still interacting with the docker daemon running with root privileges, which is a huge security risk. As an alternative to docker, podman could maybe be used as well.

In addition to switching to a rootless docker install, further hardening strategies (both for docker install and base operating system) can be used to reduce attack surface.

Our JWT token authflow right now does not use refresh tokens. This is pretty easy to implement, we are just running out of time.

7.5 Container handling

For the creation and execution of docker containers we are using the docker sdk for python.

7.5.1 Choosing a base image

After we receive the source code of the user, we need to first choose our base image we will use for the deployment. First we check if we have a Nvidia GPU available in the host. If yes, this allows us to use CUDA for accelerating machine learning related tasks. In order to achieve the best possible

performance we choose a nvidia/cuda container. This container is pre-configured to use the host GPU when attached to the container. If no Nvidia GPU is available, we use a base image for one of the programming languages. So for python we will use a python main container, and the same for R and Julia as well. This will still allow us to execute machine learning code, although not at the same performance as when CUDA acceleration is available.

7.5.2 Building a container

After we have chosen a base image, we can build the container. For this we create a dockerfile that specifies the base image, as well as operations to install both system packages or missing libraries as well as the dependencies of the project.

7.5.3 Container evaluation

Once we have built the container, we can run it. The workflow looks as follows:

1. Attach GPU, if applicable
2. Attach and mount filesystems
3. Start target file to start the submitted machine learning project
4. Submitted project loads dataset through a provided path
5. Track results and container status

7.5.4 Re-running containers

As we save the submitted ZIP file in the database, we can quite easily re-run already submitted code. This could be useful in situations where a previous submission is to be tested against an updated environment. As of now this functionality is not implemented yet, but extending the current workflow should not result in too much extra effort.

7.5.5 Further development

The whole workflow with the python backend and the docker runtime is working very well, but certain things should be looked at.

First, containers are not yet deleted after execution. Docker is able to reuse parts of containers during the build process, which speeds up everything by a lot. Working on identifying what parts to delete and which to retain is something that the available time has not allowed us to.

Second, no live output is available from the running container yet. This means, during a multiple day training run of a machine learning script we are not able to give any updates to the user. Live

output is possible, in the short timeframe I was just not able to implement it using the mentioned SDK.

7.6 Example dataset

In order to ensure that the submitted code will be able to even work with our dataset, developers need to know the format beforehand. For this we make a small subset of the data available to download. This allows the developer to write logic to first read and then also run with the present dataset.

7.7 Database

7.7.1 Current state

Like previously mentioned, we are using sqlite as our database. Sqlite can be easily installed on nearly every operating system, and it requires literally no setup. The whole database is contained in one single file, which makes backups and transfers of data very easy.

7.7.2 Future development

Even if the system is to be massively scaled I think that the sqlite database will hold up quite well. The system could be easily migrated to Postgres or another database, but as of now I don't think we can create load scenarios where this would be necessary.

7.8 Email

7.8.1 Current state

When the user uploads their model for training it is sent to the server with the necessary test data. The duration of the training process can range from a few hours to a few days. As the user is unable to directly monitor the progress of their model, it was decided that email notifications would be sent to provide status updates. The user will receive an email notification when their model begins training, and upon completion of the process, they will receive another email to indicate that their model has finished training.

In addition to that the user will also receive an email when he created an account, as well as another email when the account application was accepted.

7.8.2 Future development

We only have a preliminary solution that should showcase how sending emails integrates into our workflow. We are using a personal gmail account of one of our group members that sends out simple text based emails. None of the code is intended for production use and should be replaced before

deployment. A separate email server should be used that sends from a hospital owned domain. Instead of simple text based emails, HTML templates could be used for emails.

7.9 Library compatibility

7.9.1 Current state

As of now, the user submits next to the code also the dependencies which then get installed using pip. Even if all dependencies are correctly installed with pip, some system level dependencies might still be unfulfilled. As an example one can take the python package opencv. This package requires some libraries on the operating level to be present. Now those libraries are pre-installed by default in e.g. Ubuntu, but not necessarily in a docker container that is built against a very minimal system configuration. Anticipating these missing dependencies is very hard, and from our perspective can only be solved with systematic testing.

7.9.2 Future development

So in order to ensure compatibility with commonly used machine learning libraries and packages, systematic testing needs to be done. This means more concretely, that for all docker base images that are being used by the system, libraries need to be tested if they run or are missing dependencies. This process could be automated to continuously check compatibility with newer versions of involved software, but this will require substantial development effort.

A non-exhaustive list of packages and libraries that should be checked for compatibility includes:

- Tensorflow
- Keras
- OpenCV
- Numpy

7.10 System configuration

7.10.1 Current state

The backend has quite some hidden complexity, and requires configuration to correctly work on the host system. As hardcoding values in the source code does not really provide a nice user experience, we are exposing certain configuration options with a configuration file. This file is called `.env` and can be found in the project repository. We allow for the configuration of the following values: The **DATASET_PATH** is the path on the host system that points towards the dataset we want to use for our training and evaluation. **OUTPUT_PATH** points towards the directory we write the predictions to. **CUDA_BASE_IMAGE** is the base image we use to deploy containers with

CUDA acceleration. This needs to be set in accordance with the available CUDA version of the host system, as well as the current image version available to download. **PYTHON_BASE_IMAGE** is the base image we use when we deploy a pure python container without using GPU acceleration. This also needs to be updated to support newer python versions. **SEND_EMAIL** configures if an email is to be sent out in various parts of the system or not. **ADMIN_USERNAME** points towards the user that is to have admin privileges. This is not fully implemented yet and does not expose a true admin functionality yet.

7.10.2 Further development

The above described configuration options seemed logical during the development process, but that doesn't mean that they shouldn't be adjusted during further development. Especially the admin config flag is/was more of a temporary solution, depending on the design of the admin user this flag could be removed. If multiple datasets are to be supported, more configuration options are to be implemented.

7.11 Exporting predictions and calculating metrics

7.11.1 Current state

Let's assume we have reached the point that our submitted machine learning code has finished training and evaluating. Now we still need to export the predictions from inside the docker container to the backend, and then subsequently use the predictions to calculate some metrics. The main issue we have is how to export the metrics. The started container has no internet connection, so sending over some API is not an option. Instead we are mounting the `/mnt/localssd` filesystem, and then write in a specific path on that filesystem. This allows us to read out the predictions and then calculate metrics. Which filesystem to mount can be configured, as described in an earlier section.

Now, because not we but the user needs to integrate logic in his/her program to actually write the predictions in a format we support, we propose the use of a framework. You can find this framework in the repository of the backend (`framework.py`). The framework consists of a single file with a single function that accepts the predictions of the submitted program and then writes everything out to a specified path.

The current implementation is security hardened and ensures that no arbitrary code execution is possible.

7.11.2 Future developer

Right now the predictions can be written to a mounted volume from inside a docker container, read by the backend and then used in metrics calculations.

All data (assuming it is in the form of a numpy array) can be written to the disk, but the backend does not use the actual ground truth values for the calculations but test data instead. This means that the ground truth still needs to be incorporated into this logic. I have added comments to the source code in the appropriate place describing how to export and then use said ground truth.

Also more error handling needs to be implemented, to check and handle the correct form of the saved data. Also logic needs to be implemented that checks if the framework is part of the submission. Only when the predictions are correctly exported we are able to calculate metrics.

7.12 System requirements and recommendations

7.12.1 General hardware

76.8.1 General hardware Generally speaking, we have a system that is primarily going to build and run a lot of docker containers. Right now, all building and running happens sequentially so there is no concrete for high-count CPUs, large amounts of system memory or storage. But the general rule ‘more is better’ still applies here, so a modern multi-core CPU and sufficient amounts of RAM will increase the build and subsequently also the execution time.

When building containers for machine learning tasks a lot of dependencies need to be installed, so each container is going to be between 3 and 7 GB large. We therefore recommend at least 100GB of storage to be available. As building a docker container from scratch can be quite IO intensive we recommend using SSDs instead of HDDs.

7.12.2 GPU

Right now only supports NVIDIA GPUs for GPU accelerated tasks, the only requirement here is the installation of a Nvidia driver on the host system. The system is tested both against a consumer level card (RTX 3060) as well as a professional card (RTX A6000).

AMD GPUs most likely can be supported in the same fashion as NVIDIA GPUS, but this has not been further investigated as most machine learning tasks work best on CUDA.

7.13 Operation System

The backend is only developed to work on a Linux based system, but imposes no strict requirements on what concrete distribution is to be used. The only hard requirement we have is a working docker installation on the host, and the user executing the backend should be part of the docker group. This ensures that the user can interact with the docker runtime without root privileges.

7.13.1 Scalability

As of now, the system builds and evaluates containers sequentially. This is perfectly adequate if we do not expect more than one user to interact with the system at the same time, but with multiple users this will become a bottleneck quickly. The system could be extended to work with multiple GPUs at the same time, but due to a lack of available time and hardware we did not implement this.

7.14 Common errors

This section is meant to showcase some common error types that have occurred during the development process and to provide some assistance for future developers.

7.14.1 `pip install -r requirements.txt` suddenly starts to error out

When `pip install -r requirements.txt` suddenly starts to error out after working fine for a while, most likely docker is running out of storage. This can be solved by a system prune: `docker system prune -a -volumes`

7.14.2 Docker container errors out during building or execution

Whenever a container fails to build or execute, the SDK does not really provide a lot of information about the concrete error. As of now, the only reliable way to investigate the error cause is to manually start the last build container and try to replicate the steps taken by the user.

8 Test Plan and System Testing

This section covers all the aspects of testing the implemented system, like test strategy, description of the testing performed for frontend and backend, test roles and responsibilities as well as reporting and deliverables.

8.1 Test strategy

For the testing of the platform for AI diagnostics benchmark for breast cancer, the following types of testing will be used:

- Unit testing
- Integration testing
- System testing
- Exploratory usability testing

Moreover, when applying those tests, there will be different roles assigned to the responsible people (Refer to Section 8.4.

When testing the platform, the following objectives will be kept in mind: the testing should be able to test the functionality, reliability, performance, and security of the system.

8.2 Front-end

The system we are building is a web application. Thus any sequence of clicking buttons must result in a meaningful response from the website. In order to justify the correctness of the system automated system testing will be applied. The technology that will be used is called Selenium. Selenium is an open-source framework that allows its users to test their web applications by running automated tests. This framework provides a variety of features, including the ability to simulate user actions such as clicking buttons, entering text into fields, and navigating between pages. Multiple programming languages and web browsers are supported by Selenium. In this project, Python and a Chrome driver will be used. The result of the testing can be found in Section 9.

8.3 Back-end

- Unit testing → gitlab pipeline that builds everything when pushed
- Integration testing → test the api
- System testing → create a simple image classification model(s) to test the model evaluation results extraction.

8.3.1 System Testing

The AI diagnostics benchmark for breast cancer web applications can handle a large variety of machine learning models. In theory, any script that provides some results can be trained via this website. This insight served as motivation to create, at the moment of writing, 2 simple image classification models that will be used to justify the correctness of the back-end part and in particular the ability to extract results from the docker container.

The second model is more closely related to the topic of this project.¹ It classifies ultrasound images as normal (no cancer), benign, and malignant from a given dataset. In this section, we will not go into detail on how exactly the machine learning script works. The essence is that in the end, it produces the following results: accuracy, precision, recall, and a confusion matrix (see Stakeholder requirements).

Both models were able to run in the created runtime environment and finished evaluation in the expected time. No special considerations need to be taken into account when executing the models, so we can conclude that the system provides adequate support for the two use cases.

8.3.2 Unit testing

To ensure the stability and correctness of our backend code, we utilize pytest for automated unit testing. For example, we have test functions that validate the behavior of our database queries, ensuring that data is inserted, updated, and retrieved correctly. We also have test functions that validate the behavior of our REST API endpoints, ensuring that they return the expected results for different input parameters.

When we run our pytest suite, we receive a detailed report of the test results, including which tests passed, which tests failed, and any error messages that occurred. This allows us to quickly identify and address any issues that arise, whether they be logic errors in our code or problems with our underlying infrastructure.

We did not achieve full code coverage for unit testing, as especially the docker related logic is very difficult to test.

¹The model is taken from an “Computer Vision” assignment from “Data Science and Artificial Intelligence: Seeing through the hype” elective offered at Technical Computer Science bachelor program at University of Twente. Minor changes are applied for the relevance of this project.

8.4 Test roles & responsibilities

Each team member will be assigned the responsibility of testing specific components of the system. The testing workload will be distributed among all members, with each individual responsible for developing the necessary code for their assigned tests and documenting the testing outcomes.

Roles	Responsibilities
Testing Leader	The testing leader is the one who manages all the testing procedure
Testing Implementer	The testing implementer is the one who works on developing the code required for the testing.
Tester	The tester is responsible for testing the system.
External Supervisor	The external supervisor is the one who helps the other people with the possible problems occurring during the testing.
Writer	The writer records the testing results, and works on writing a testing report to deliver to the stakeholder.

Table 4: Test Roles and Responsibilities

8.5 Test Reporting and Deliverables

- A detailed explanation of the tested components
- A report which touches on the un-tested parts and explains the possible issues about those parts.
- A detailed analysis of testing coverage (Especially for the unit-testing)

9 Testing Results

This section gives an insight into the results of the testing that was performed for the system. It discusses frontend and backend testing results separately.

9.1 Front-end Results

9.1.1 Automated Selenium Testing

The front-end has been extensively tested with automated Selenium tests.

First routing between the different pages was tested, ensuring all buttons navigate to the correct web pages. This testing process consists of three parts: testing the Login button, testing the Sign-up button, and testing the buttons in the navigation sidebar. This was done by checking if the URLs match the corresponding page.

Upon completion of these tests, it can be concluded that all buttons successfully navigated to their corresponding pages.

Next, the sign-up and login pages were tested. On the sign-up page, the different fields have been tested. Extra focus was laid on the password fields. The primary objective of this testing was to verify that the password met the required syntax and matched the "confirm password" field.

9.1.2 User experience testing

Since it was not feasible to physically meet with the client until the end of the module, traditional user experience testing could not be carried out. Nevertheless, there have been multiple online feedback moments with both the supervisor and the hospital staff.

First, the multiple initial Lo-Fi designs were discussed with the supervisor and a choice was made on what design to proceed with. Afterwards, the Hi-fi design was presented and approved by the supervisor, after some discussed alterations were made. Lastly, the front end of the system was shown, on multiple occasions, to the supervisor and hospital to receive constant feedback, which was incorporated to make necessary improvements.

9.2 Back-end Results

Back-end testing is done very extensively on our platform. And the testing is divided into 2 parts.

9.2.1 Database Testing

- Before the test cases are executed, some variables are created in order to use during testing. Then, we test the user creation and insertion into the database. We check if the user is created with the correct name, email, and password.

- Then, the containers are tested. We check if registering the containers with id and userid are successful, and then the switch of status of container is tested.
- At the end, building logs and execution logs are tested.
- The result of the database testing is successful, all tests are passing and we believe that the database insertion/deletion functionality is tested sufficiently.

10 Risk Assessment

Even though the system has been tested on multiple levels there are a number of risks worth addressing:

10.1 Executing unknown scripts

The most concerning risk of the web application is the fact that we allow individuals to upload scripts, unknown to us, on the website and the backend executes them without checking whether the execution will harm and/or take advantage of the system.

The docker container identifies two phases: the building phase and the execution phase. During the building, the passed requirements are taken care of. That is, among more installing required libraries. During the second phase, the provided script is executed but before that, the docker container is disconnected from the internet. This measure prevents data extraction from the website. There still remains the risk of malicious scripts attempting to break out of the docker container and obtain unassigned privileges. We did research on the topic and this approach requires quite an effort and therefore seems unlikely to happen considering that the reward would be obtaining anonymous mammographic breast images of patients. Nevertheless, this possibility must be taken into account.

The risk mentioned above was addressed during a meeting with the supervisor and the stakeholders. The discussion led to the conclusion that users should apply for the right to train their models. In the application form, it will be made clear that by applying the user agrees with the general terms and conditions of which the user will be held accountable for any damage and malicious use of the application. Information about users is stored, e.g. the institution they are part of. The reputation of the researchers is at stake and this will minimize the cases of cyber attacks on the web application.

Another possible solution we thought of was to have a third party validate the true intentions of the scripts. However, that means we share another's work without permission violating the copyright principles described in the GDPR.

10.2 Uncertainties with scalability

Besides the risk of executing unknown scripts, scalability is also one of the risks that our platform has. We were not able to test our platform with a very large ZIP-file, or very long queue-length. There is a risk that adding a ZIP-file with the size larger than the capacity of the website can make the application slower (or maybe even crash).

Since we were not able to test this, this risk remains as an 'uncertainty' for us, and this risk can be resolved in future when the web-application is tested with the real-data by the client.

Another possible scalability problem is about how wide the web-application will be used. We, as the developer team, are discussing things like ‘how scalable is it to deploy the web-application to other healthcare institutions?’, ‘will their devices support the web-application?’, and ‘will we be able to get more datasets?’. According to our meetings with our client and the supervisor, the web-application may be supported by other institutions but again, this will remain as an uncertainty for us for now.

It is important to mention this risk in this document so that the client is aware of this risk so that they will not have a loss in revenue in case they deploy before they really conclude this risk.

11 Limitations

Despite the fact that the system covers all ‘must’ requirements there are some limitations that need to be addressed. At the moment of writing the system supports only scripts written in Python. Therefore only Python models can be trained on the web application.

The team has chosen to add the functionality of authorizing users to train their models after they have agreed to the terms and conditions of use. This was discussed and agreed upon with the client. In this way, some security is applied. Currently, only the web page exists but the logic is still in progress.

Also, it was discussed with the client that it should be possible for users to leave comments on models on the leaderboard. This feature was given a lower priority and thus due to time constraints, it has not been implemented yet.

It was also planned to add a live progress bar so users (data scientists) can see how far the models are with the training. As of now, it is quite difficult to extract a stream of the current output from the docker container during runtime, so currently only output, after the evaluation is finished, is displayed.

Ideally, the web application would allow users to log in with their Google accounts. However, at the moment of writing this has not been fully implemented. For now, the team uses our own login management.

For the sake of convenience, the team aimed to implement a Drag & Dropbox so that users can easily upload models. Currently, this feature works as expected on the MAC operating system. There are some issues on Windows that require further investigation. Linux has not been tested yet.

12 Future Improvements

The system MVP has been implemented and tested according to planning. However, there is still room for improvement. At the moment of writing the web application supports only models that are written in Python. In the future, as the requirements specification suggests, the system should be able to train models in programming languages R and Julia. In the process of development, the team realized that the concept of the system is quite generic and therefore can be used to train models specialized in a different kind of analysis. Those are dependent on the needs of the ZGT hospital.

Also, the system has the potential to improve communication within the network of data scientists that are specialized in breast cancer detection. The web application can connect those researchers by allowing them to comment on models listed on the leaderboard. We can even go a step further and allow functionality for direct message communication on the web application itself. This feature would improve the collaboration between data scientists.

In order to secure the system, only users that are granted permissions can train their models. In this way, accountability is ensured. However at the moment of writing, due to time constraints, this feature is still to be implemented. Therefore it is considered a future improvement.

As mentioned in Section 9, the live progress bar (expected duration time) has not been implemented yet. Training of a single model can take up to several days. Even though users are notified by email when the training is finished it would be convenient for the data scientists to see how far along the way the model training is, so they can adjust their schedule accordingly.

The current logging mechanism supports only manual account management. The Google o/auth is still in progress at the moment of writing. In the future, this feature will be implemented.

Currently, the Drag Drop feature of uploading model files is supported on Mac but not on Windows. In the future, further investigation must be done to identify and solve the issue.

Additionally, it would be a very nice addition if a sort functionality is added for the table on the Leaderboard page. In this way, the users can sort the leaderboard according to the features of accuracy, precision and recall.

13 Conclusion

This project will contribute to a whole new part of healthcare. Hospitals have a lot of data, nevertheless, their primary goal is to help patients. Doctors do not have the time to work full-time on developing a platform to share their knowledge with other research organizations. This web application can be the first step to making healthcare data open for doctors while still keeping the confidentiality of the patient.

The multi-page web application allows users to upload breast cancer detection models. These models then get packaged into a docker container which is then sent to the server. At the server the docker container, with the detection model, is executed. Once the model evaluation is completed, the evaluation results are sent back to the frontend. The user can observe their model's results on the webpage. It can also see an overview of all the uploaded models and their evaluations. On the leaderboard page, a ranking of all the models can be found.

This project has been designed to be used by the ZGT Hospital. The future goal would be to expand this platform to multiple institutions. This raises various questions: how scalable is this project to deploy at another hospital? How will we give access to other hospitals? Where do we get more datasets from? And many other questions. . . These questions need to be kept in mind when planning on using the platform for a larger audience.

Even though there are still numerous unanswered questions, this project has established a solid foundation for a promising platform in the healthcare world. The web application can train breast cancer detection models while maintaining the training data private, thus fulfilling the main requirement. The ZGT Hospital can deploy the platform within their institution and has a strong starting point to scale this project to be used by several hospitals.

14 References

1. CDCBreastCancer. (2022a, March 9). What Is Breast Cancer? Centers for Disease Control and Prevention. https://www.cdc.gov/cancer/breast/basic_info/what-is-breast-cancer.htm
2. CDCBreastCancer. (2022b, November 17). What Is Breast Cancer Screening? Centers for Disease Control and Prevention. https://www.cdc.gov/cancer/breast/basic_info/screening.htm
3. Reinke, A. (2021). Bring the model to the data: The Deep Learning Epilepsy Detection Challenge. *EBioMedicine*, 66. <https://doi.org/10.1016/j.ebiom.2021.103323>

15 Appendix

15.1 Appendix A: UML Diagrams

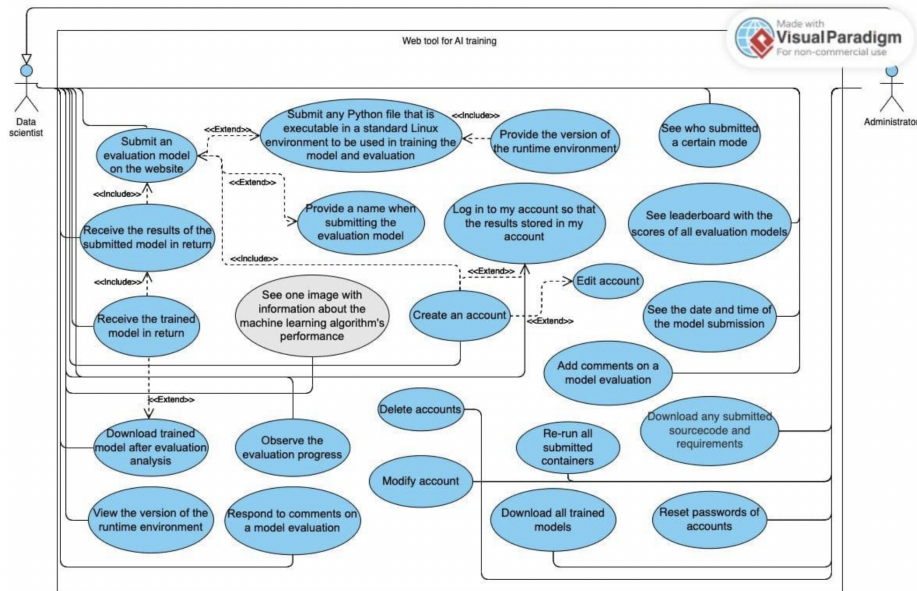


Figure 12: Use Case Diagram

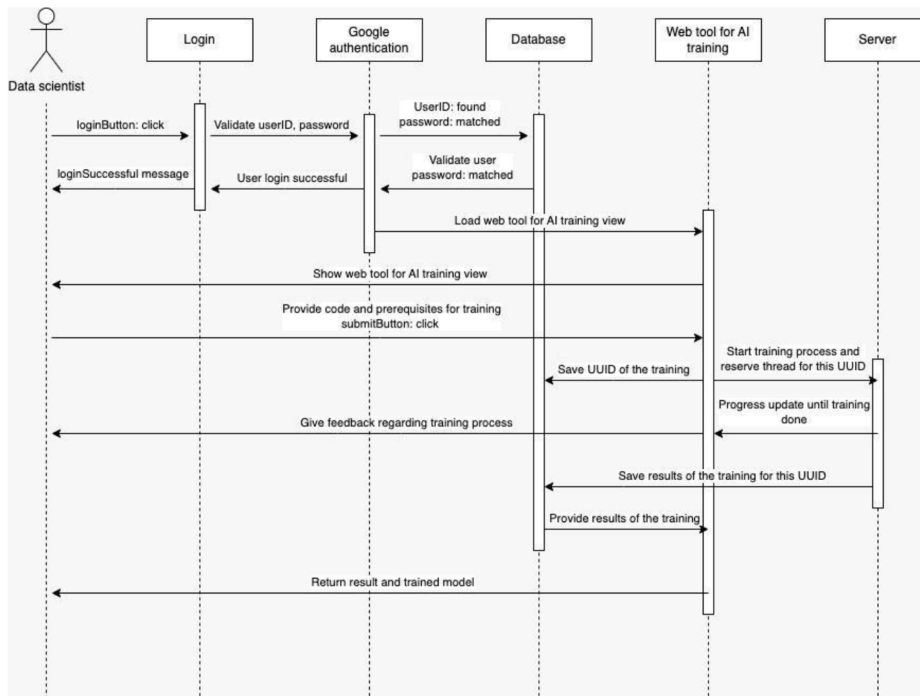


Figure 13: Sequence Diagram

15.2 Appendix B: Lo-fi Design

Design 1: multi-page application 1

Benchmark for breast cancer analysis
train your model with our dataset

Model name

No file chosen

What evaluation results are you interested in?

Confusion matrix
 Precision
 Recall

Figure 14: Submission Page

Benchmark for breast cancer analysis
train your model with our dataset

Leaderboard

1. Model A	Feb 12 2012
2. Model B	Feb 12 2012
3. Model C	Feb 12 2012

Order by: [Accuracy](#) [Precision](#) [Recall](#)

Model evaluation results: Model A

Confusion matrix

		True Class	
		Positive	Negative
Predicted Class	Positive	TP	FP
	Negative	FN	TN

- Accuracy: 78%
- Precision: 79%
- Recall: 83%

Figure 15: Leaderboard Page

Design 2: multi-page application 2

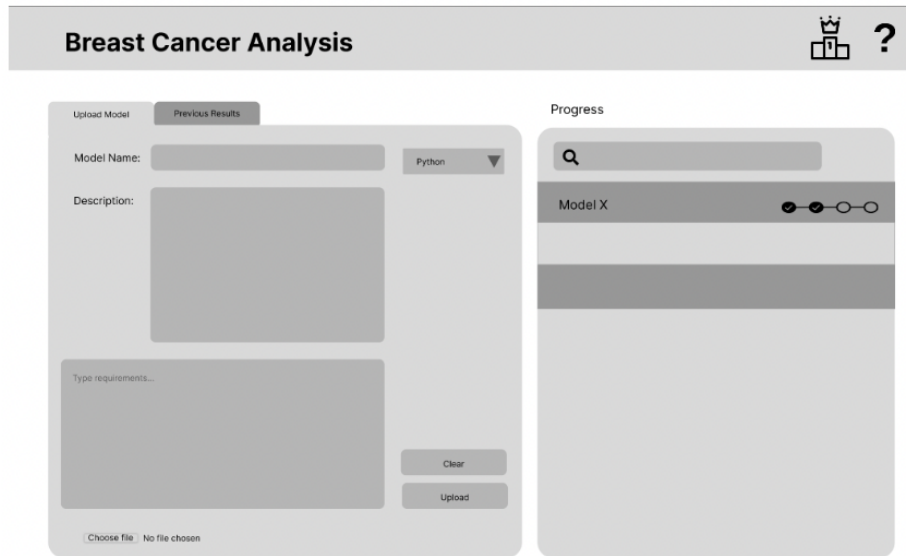


Figure 16: Submission Page

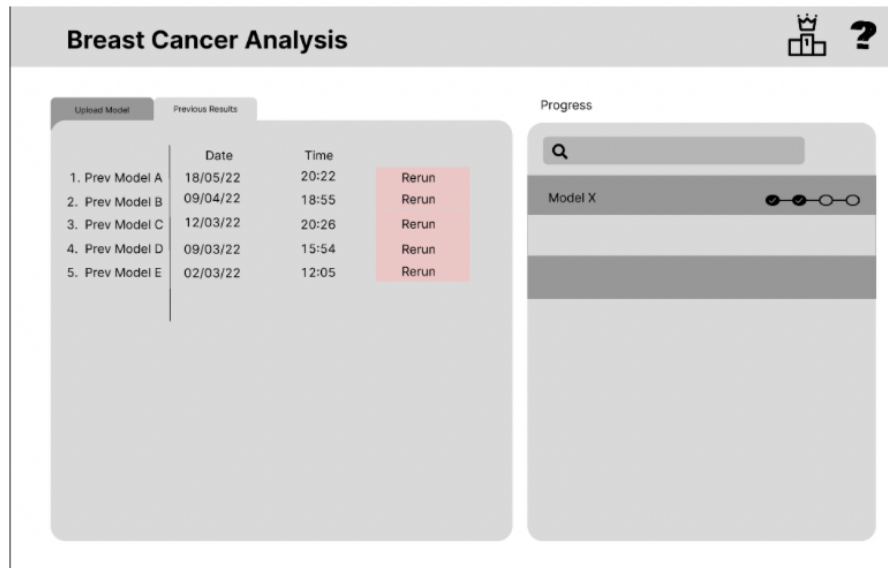




Figure 17: Result Page

Breast Cancer Analysis  

Leaderboard

Search

1. Model A	15/02/23
2. Model B	15/02/23
3. Model C	15/02/23
4. Model D	15/02/23
5. Model E	15/02/23
6. Model F	15/02/23
7. Model G	15/02/23

Order by:

Accuracy Precision
 Recall Matrix


Model A

Accuracy: 81%
Precision: 75%
Recall: 78%

	Positive (1)	Negative (0)
Positive (1)	TP	FP
Negative (0)	FN	TN

[Download Model](#)

Figure 18: Leaderboard Page

Breast Cancer Analysis 

How To Use

About Us




Figure 19: About Us Page

Design 3: single-page application

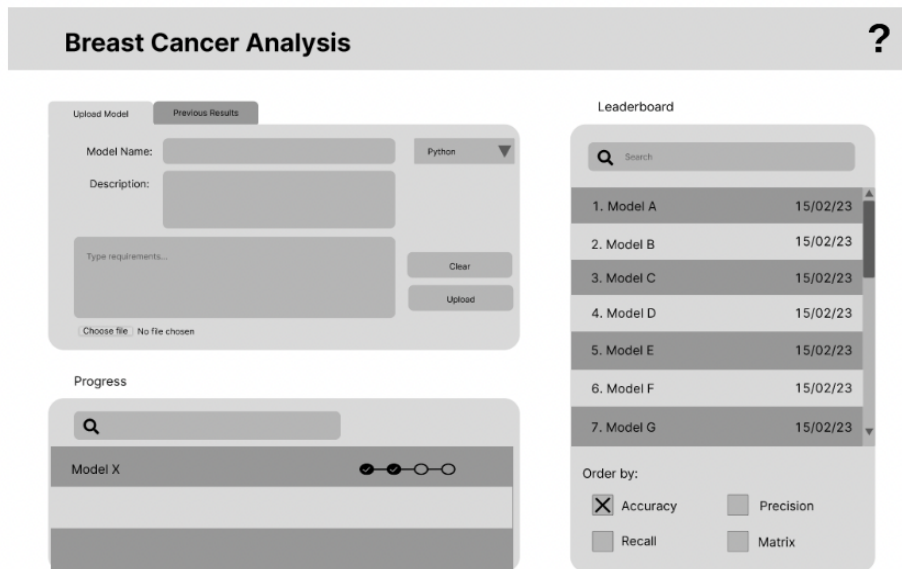


Figure 20: Console Page

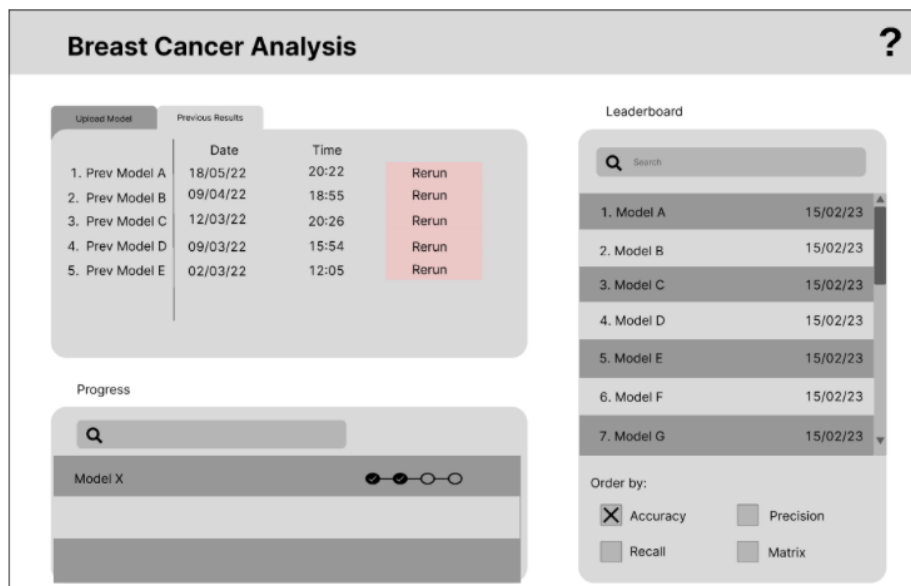


Figure 21: Console Page-2

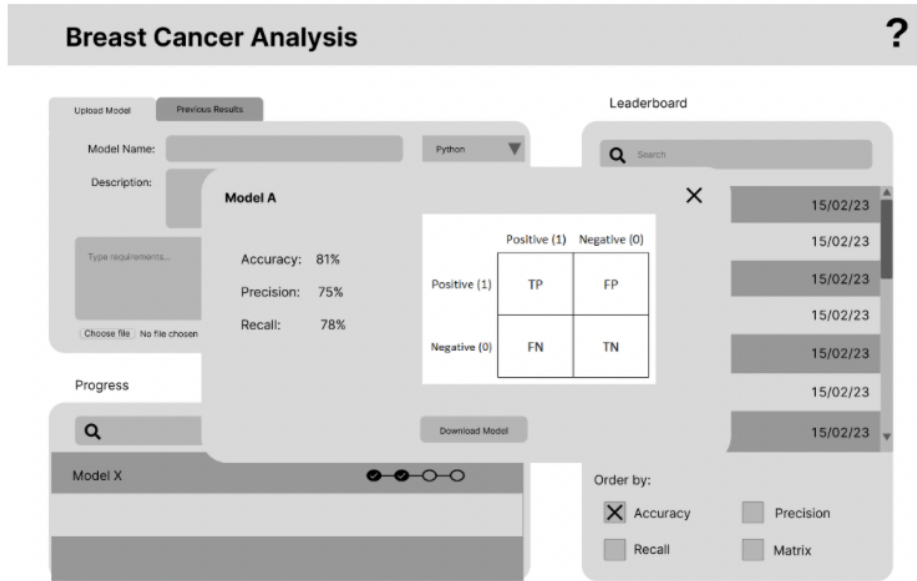


Figure 22: Upload Model Page Design

15.3 Appendix C: Hi-fi Frontend Design

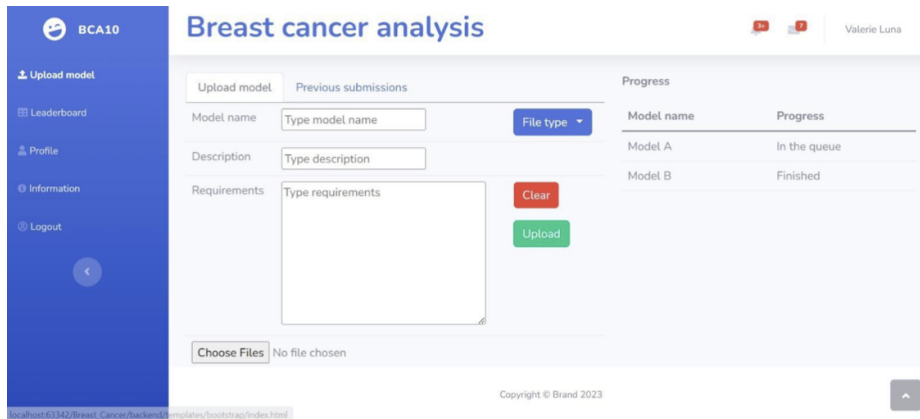


Figure 23: Upload Model Page Design

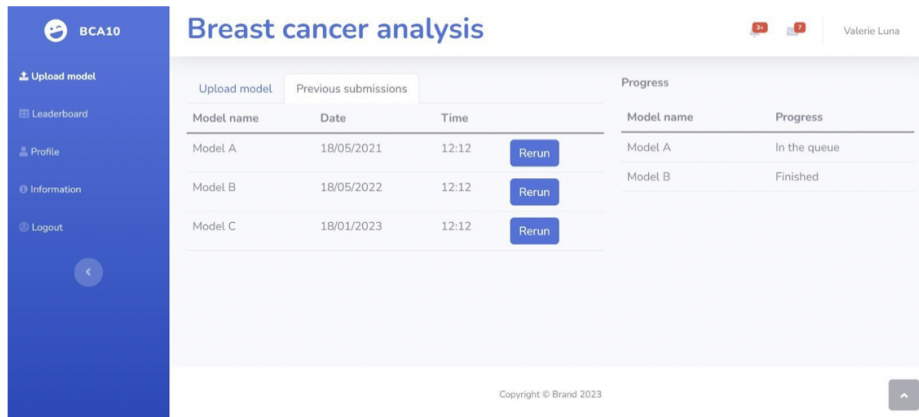


Figure 24: Upload Model Page Design - Previous Submissions Tab

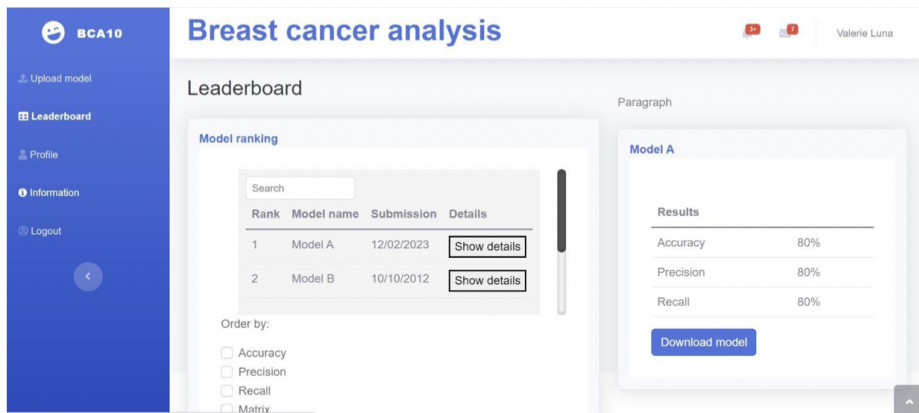


Figure 25: Leaderboard Page

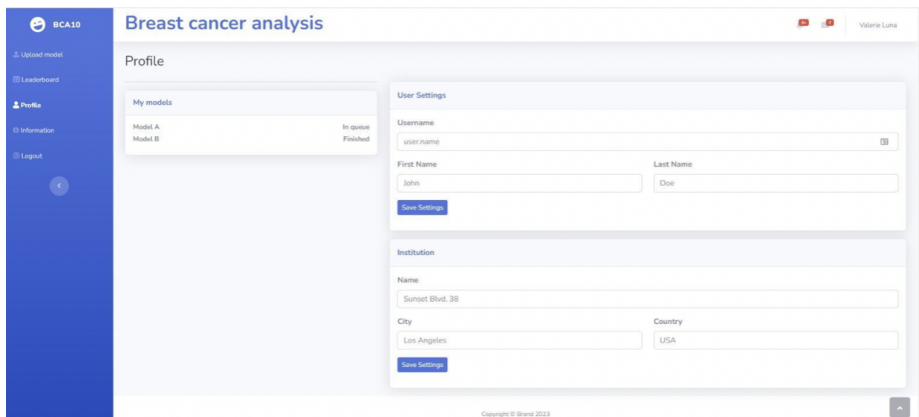


Figure 26: Profile Page

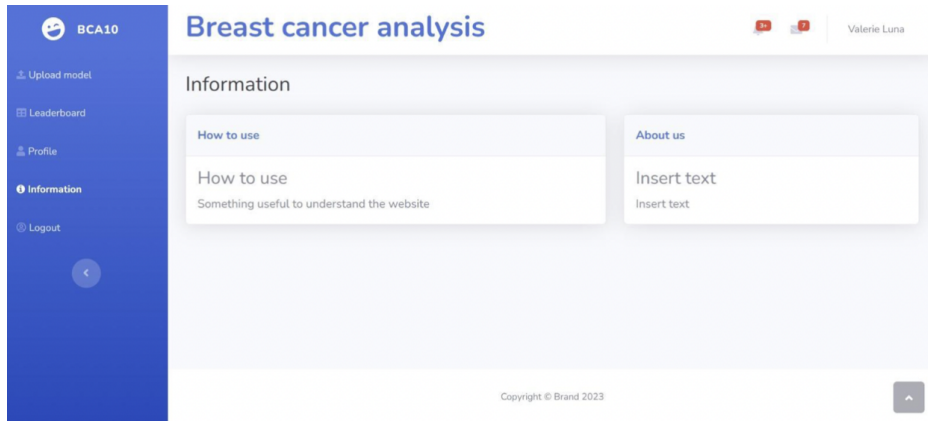


Figure 27: Information Page

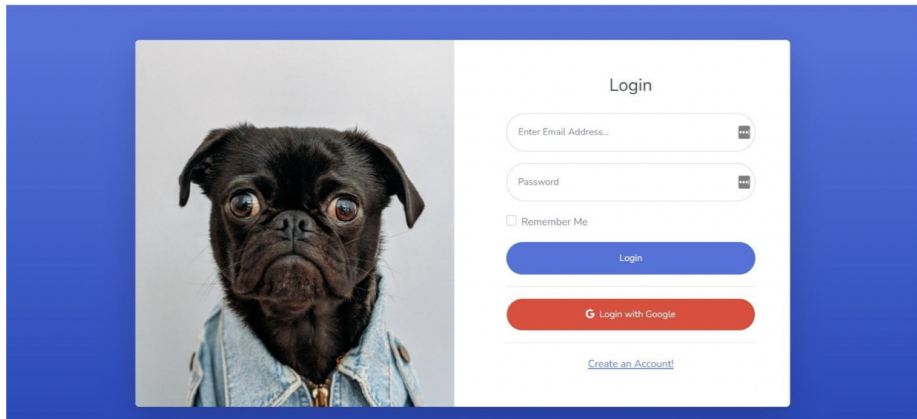


Figure 28: Login Page