

Decision Support System

Table of Contents

1. INTRODUCTION	5
2. OBJECTIVES & SCOPE	6
3. REQUIREMENTS ANALYSIS	6
3.1. Pipeline	6
3.2. Application	7
4. PIPELINE	7
4.1. Data Ingestion	7
4.2. Data preparation	7
4.3. Storing Features	8
4.4. Training Model	8
4.5. Evaluate Model	8
5. APPLICATION	9
5.1. Backend	9
5.2. Frontend	10
5.3. Prototype	10
5.3.1. Login	10
5.3.2. Input Parameters	11
5.3.3. Predictions	11
5.3.4. Dashboard	12
6. DEPLOYMENT	13
7. TECHNOLOGY STACK & JUSTIFICATION	14
7.1. Development Environment	14
7.2. Pipeline Technology Selection	14
7.3. Deployment Technology	15
7.4. Backend technology	16
7.5. Frontend Technology	17
8. DEVELOPEMENT PLAN	17
8.1. Sprint 1 24/02 – 14/03 Setup & Research	17
8.2. Sprint 2 17/03 – 28/03 Building End-To-End Prototype	18
8.3. Sprint 3 31/03 – 11/04 Application Development	18
8.4. Sprint 4 21/04 – 28/04 Data Ingestion & Preprocessing	19
8.5. Sprint 5 29/04 – 09/05 Model Training & Evaluating	19
8.6. Sprint 6 12/05 – 23/05 Model Deployment	19
8.7. Sprint 7 26/05 – 06/06 Testing & Finetuning	20
9. TESTING & VALIDATION	20
10. SECURITY CONSIDERATIONS	20

11.	CHALLENGES & SOLUTIONS	20
12.	RISKS	21
13.	REFERENCE LIST	22
	ATTACHMENTS	23

Table of Figures

FIGURE 1. PROJECT STRUCTURE	9
FIGURE 2. LOGIN PAGE PROTOTYPE	11
FIGURE 3. INPUT PARAMETER PAGE PROTOTYPE FOR ENTERING PARAMETERS FOR MODEL INFERENCE	11
FIGURE 4. PREDICTIONS PAGE PROTOTYPE WHICH WILL DISPLAY PREVIOUS PREDICTIONS MADE ON THE APPLICATION.	12
FIGURE 5. DASHBOARD PAGE PROTOTYPE TO GAIN DEEPER INSIGHTS INTO AN INDIVIDUAL PREDICTION.	13
FIGURE 6. APPLICATION FLOW FOR MAKING PREDICTIONS.	24
FIGURE 7. USE CASE DIAGRAM OF THE APPLICATION.	25

1. Introduction

In this document will outline my internship assignment. The internship will take place at Mobilab&Care, located in Geel at Thomas More. Currently, the organisation has developed a machine learning model capable of detecting cardiovascular disease, but there is no structured way of sharing this model or making it easily accessible to external parties such as doctors and researchers. In addition to sharing the existing model, a pipeline will be prepared that will allow the organisation to retrain models more easily in the future when new data becomes available, ensuring the platform remains up-to-date. When my assignment will have been realised, the stakeholders will experience the following benefits: cardiologists will be able to use the application as a clinical decision support tool for patient treatment, patients will benefit from faster and more accurate diagnoses, and researchers from partnering institutions will have streamlined access to the trained machine learning models.

2. Objectives & Scope

The main goal of this project is to build a recommendation system for clinicians and researchers. This system will use machine learning models to help with decision-making by providing predictions based on input parameters. To achieve this, an automated machine learning pipeline will be set up to handle the training, evaluation, and deployment of models, reducing the need for manual work.

Alongside the pipeline, an application will be developed to let users select models, enter input parameters, and receive predictions. The focus is on making the system easy to use, flexible, and scalable, so new models can be added and used without too much effort.

To determine the scope of the project, I will use MoSCoW analysis. This framework helps prioritize key features and focus development efforts effectively.

Must Have:

- Automated machine-learning pipeline.
- An application that interacts with the model via an API for real-time inference.
- Authentication for the application.

Should Have:

- SHAP analysis for interpretability.
- Lime analysis for individual predictions.
- Model overview page for key model metrics.

Could Have:

- Database for storing predictions and patient profiles.
- Support for multiple models.
- Ability to export predictions as a downloadable file.
- Model management page for adding/removing models

Won't Have:

- Custom user authentication outside of Auth0.
- Admins can edit user roles from the admin page.

3. Requirements Analysis

To gain a better understanding of the requirements of the project all functional requirements are listed below. These include all the requirements of both the machine learning pipeline and the application.

3.1. Pipeline

1. **Data Ingestion:** Load clean CSV data for preprocessing without further cleaning.
2. **Data Preparation:** Convert categorical data using one-hot encoding and scale numerical features for model compatibility.
3. **Storing Features:** Save column names to ensure the correct mapping between training and prediction data.

4. **Model Training:** Train classification models, checking for missing values and using parallel processing to speed up training. Hyperparameter tuning will be applied to optimize performance.
5. **Model Evaluation:** Evaluate models using key metrics like AUC, precision-recall, accuracy, F1 score, and confusion matrix. The best model is selected for deployment.

3.2. Application

1. **Model Interaction:** The application will serve as a way for clinicians to interact with the trained model. Granting them the possibility to give the model parameters to make a prediction
2. **Prediction Analysis:** The application will give an overview of a prediction once one is made. This is meant for the user to get a deeper understanding of how the model made its choice.
3. **Model Metrics:** An overview will be created to view the selected model's metrics such as precision, recall, accuracy, SHAP, and training shape.
4. **Model Management:** The application will offer a way to add a new model to the system through a file upload and will also be able to remove a certain model.
5. **User Authentication:** Users of the application should have a valid account before they are able to access the application.

4. Pipeline

The goal of the pipeline is to automate the process of training and deploying a classification model. This pipeline will be specifically designed for data scientists working on the models that will power the app. The pipeline will handle various tasks, including data ingestion, preprocessing, model training, hyperparameter tuning and evaluation.

4.1. Data Ingestion

The data ingestion process is the first step in the pipeline. The expected input format for the data is CSV. It's important to note that the ingested data is assumed to be already clean, meaning no additional cleaning or augmentation will be performed at this stage. Data preprocessing for this project will focus solely on making the data ready for model training.

4.2. Data preparation

Although the data is already considered clean, there are still necessary preprocessing steps to prepare it for training. First, categorical data must be converted into a format that can be understood by machine learning algorithms. This is typically done through one-hot encoding, which splits categorical features into binary columns for each unique category.

Additionally, scaling of numerical features is required to ensure that all variables are on a comparable scale. This helps prevent any single feature with larger values from disproportionately influencing the model. Common scaling techniques include standardization, which centres the data around a mean of 0 with a standard deviation of 1, or min-max scaling, which normalizes the values to a specific range, typically between 0 and 1. These steps ensure that the model can learn efficiently and accurately.

4.3. Storing Features

Once the data is prepared, the next step is to save the column names into a file. This is important for maintaining the correct mapping between the features used for training and those used during prediction. By storing the column names, we ensure that when new data is passed into the model for inference, the correct features are selected and aligned with the trained model. This helps maintain consistency and prevents errors that might occur due to mismatched or missing columns during predictions.

4.4. Training Model

The models that will be trained for this project will be classification models, as our objective is to make clinical recommendations. Before we begin the training process, it is essential to check if there are any missing values in the dataset. If missing values are found, it is necessary to adjust our model selection to choose algorithms that handle missing data effectively, such as decision trees or models that can naturally handle missing values. If no missing values are present, we can proceed with a wider range of classification models, including logistic regression, random forests, or support vector machines.

To enhance efficiency, we will make use of parallel processing, allowing us to train multiple models simultaneously and optimize the model selection process. After the model has been trained, we will focus on hyperparameter tuning. This step is critical to refine the model's performance by adjusting hyperparameters to ensure that the model achieves its highest possible accuracy and predictive power. Hyperparameter tuning helps optimize the model and can be performed through methods such as grid search or random search to fine-tune settings like learning rate, regularization strength, or the number of trees in ensemble methods.

4.5. Evaluate Model

To evaluate the performance of the classification model, several key metrics will be used. These metrics include the Area Under the Curve (AUC), precision-recall curve, accuracy, F1 score, and confusion matrix. The AUC metric provides insight into the model's ability to distinguish between classes, while the precision-recall curve evaluates the trade-off between precision and recall, which is especially important when dealing with imbalanced classes. Accuracy gives a general sense of the model's overall performance, while the F1 score is useful for balancing precision and recall, especially in cases of class imbalance. The confusion matrix helps in understanding the true positives, true negatives, false positives, and false negatives, providing a detailed view of the model's predictive performance.

Once all these metrics have been evaluated, the model that achieves the best performance according to these metrics will be selected for deployment, enabling it to make predictions on new, unseen data. In addition to performance metrics, we will also assess the model for any potential bias. Evaluating model bias is essential to ensure that the model is fair and does not favour one class over another or produce unjust outcomes, which could lead to biased predictions in real-world applications. This step will help ensure the model's reliability and fairness before it is deployed for inference.

Additionally, we will perform SHAP analysis to interpret the model's decisions. This will help us understand the global feature importance, as it gives us the importance of all parameters for every prediction that is made. (Lundberg, 2018)

5. Application

The application will serve as the primary interface for clinicians to interact with the trained machine learning model. It will function as a recommendation system, allowing clinicians to input relevant patient parameters and receive a prediction. Additionally, the application will provide interpretability features through SHAP/LIME analysis, helping users understand the key factors influencing each prediction. This transparency is crucial for building trust in the model and ensuring that clinicians can make informed decisions based on the system's output. A more detailed overview of the different use cases of the application can be found in [Attachment E](#).

The application will consist of two main components: the backend (found in `main.py`), which will handle model inference and logic, and the frontend (found in the static directory), which will provide an intuitive interface for users to interact with the system.

The folder structure for the application will be as seen on Figure 1, ensuring a structured environment that is easy to work with. An activity diagram of the application which explains its basic operation is also provided (attachment 5-application).

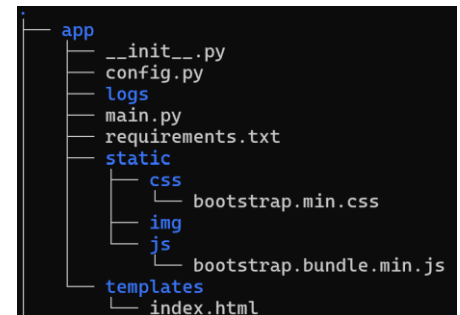


Figure 1. Project Structure

To ensure a good user experience, the application is split up into multiple pages. For consistency across all pages, a standard layout will be implemented. This includes a navbar and a place for the main content from the different pages.

Perhaps the most important page of the application being the input parameters page. This page serves a form to the user on which the user can enter all the parameters required for the model and choose a model from a dropdown list that will handle inference. After submitting the form, the prediction from the selected model is displayed below the form also providing a button to go to the dashboard, which can also be found in the navbar. The dashboard only displays information when a prediction is made, giving more insights about the prediction made. The dashboard serves a LIME plot of the last prediction made; this is used to gain a better understanding of which parameters had the biggest impact on the decision of the model. Thus, adding transparency to the models' predictions. A table with all the parameters used for the prediction is also displayed.

Another page that was added is the model page. This page allows the user to select a model upon which the page serves some key insights into the models' metrics. These include accuracy, precision, recall, training shape, receiver operator curve, precision/recall curve, and global SHAP analysis of the selected model. The final page is the admin page, here the admin can approve or reject new registrations from users. This makes the process of access control more streamlined, meaning the administrators don't require any external dashboard to manage access to the application.

5.1. Backend

The backend is responsible for facilitating communication between the application and the deployed machine learning model. It acts as an intermediary that processes user requests, validates input data, sends it to the model for inference, and returns the results to the frontend. It is important here to use the extracted features from data preprocessing to ensure the parameters are mapped correctly. A diagram highlighting the entire flow of the backend can be found in [Attachment D](#).

To ensure that only authorized users can access the system, the backend will handle authentication and authorization, verifying credentials and enforcing role-based access control. Additionally, input validation will be implemented to check for missing values, incorrect data types, or out-of-range values before sending the data to the model.

Once the input data is processed, the backend will communicate with the deployed model to generate predictions based on the provided patient parameters. Alongside these predictions, the system will also request SHAP analysis to explain the most influential factors in the model's decision-making process. This ensures that clinicians not only receive predictions but also gain insights into why the model arrived at a particular outcome. The backend will structure this information and return it in a format that the frontend can easily display.

To maintain reliability and security, the backend will include logging and monitoring mechanisms that track API requests and responses, allowing for debugging, performance analysis, and anomaly detection. Given the sensitive nature of the data being processed, industry-standard security measures will be implemented, including encryption for data transmission, request validation to prevent abuse, and strict access control mechanisms. The overall design of the backend will prioritize efficiency, scalability, and security to ensure a seamless interaction between the user and the deployed model.

5.2. Frontend

The frontend serves as the primary interface through which clinicians interact with the system. It will provide an intuitive and minimalistic design, allowing users to easily input patient data, obtain model predictions, and explore the explanations behind each decision. Before accessing the application, clinicians will need to log in through an authentication system that verifies their credentials. Once authenticated, they will be redirected to the main interface, where they can begin using the model as a recommendation tool.

The central part of the application will feature an input form where users can enter patient biomarkers and submit them for analysis. Upon submission, the backend will process the input and return a prediction, which will be displayed clearly on the interface. In addition to the prediction, the frontend will also provide a SHAP analysis dashboard that presents an interpretable breakdown of the model's decision-making process. This dashboard will highlight the most influential biomarkers, allowing clinicians to understand why a particular prediction was made.

Since the application is designed for clinical use, the frontend will prioritize a clean and efficient user experience. The interface will be accessible and responsive, ensuring usability across different devices, including desktops and tablets. The focus will be on streamlining interactions, minimizing unnecessary complexity, and providing a user-friendly environment where clinicians can quickly obtain insights without needing technical expertise. By integrating an intuitive design with meaningful model explanations, the frontend will serve as an essential tool for improving clinical decision-making and fostering trust in the model's predictions.

5.3. Prototype

This low fidelity prototype shows the basic structure of the application. Which includes all pages to be made.

5.3.1. Login

The login page is simple and should be used to authenticate clinicians to keep patient data safe. A prototype of this page can be seen on Figure 2

Login

Email


Password

Login

Figure 2. Login page prototype

5.3.2. Input Parameters

This page, seen on Figure 3, is used to interact with the model. Here, predictions can be made and saved, by filling in all the biomarkers and pressing Inference. A link to the dashboard, to gain more insight into the prediction, is also provided.

 Model Predictions Dashboard

Model

Biomarker <input type="text"/>	Biomarker <input type="text"/>	Biomarker <input type="text"/>
Biomarker <input type="text"/>	Biomarker <input type="text"/>	Biomarker <input type="text"/>

Inference

Hospitalisation 84%

Dashboard

Figure 3. Input parameter page prototype for entering parameters for model inference

5.3.3. Predictions

This page serves previous predictions made by the clinician. There is also the option to filter on: patient, date, and prediction. This is a possible Expansion and can only be made when a database is available. A prototype of this page can be seen on Figure 4.

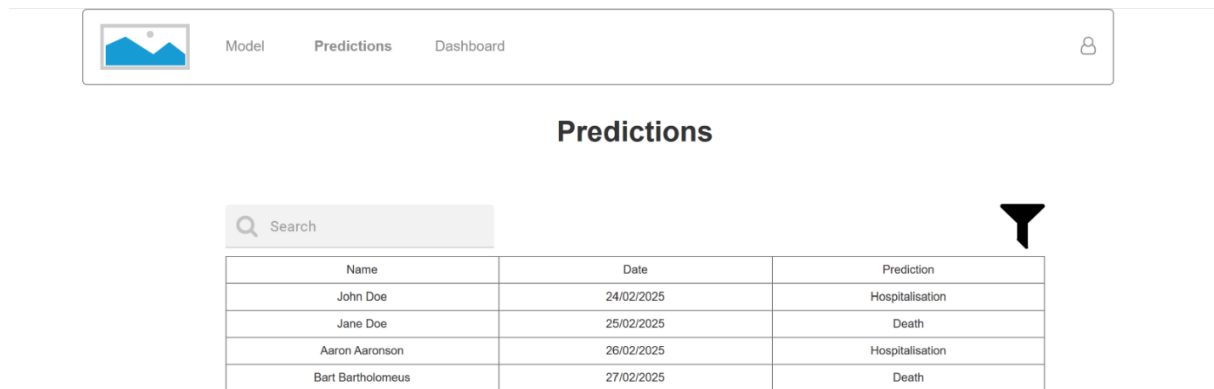


Figure 4. Predictions page prototype which will display previous predictions made on the application.

5.3.4. Dashboard

The dashboard houses an overview of the SHAP/LIME features of a certain prediction, a prototype of this page is illustrated on Figure 5. This gives the clinician an overview of how the model made its prediction and shows us an overview of the parameters used to make this prediction. This will also include filtering on patient, date, prediction.

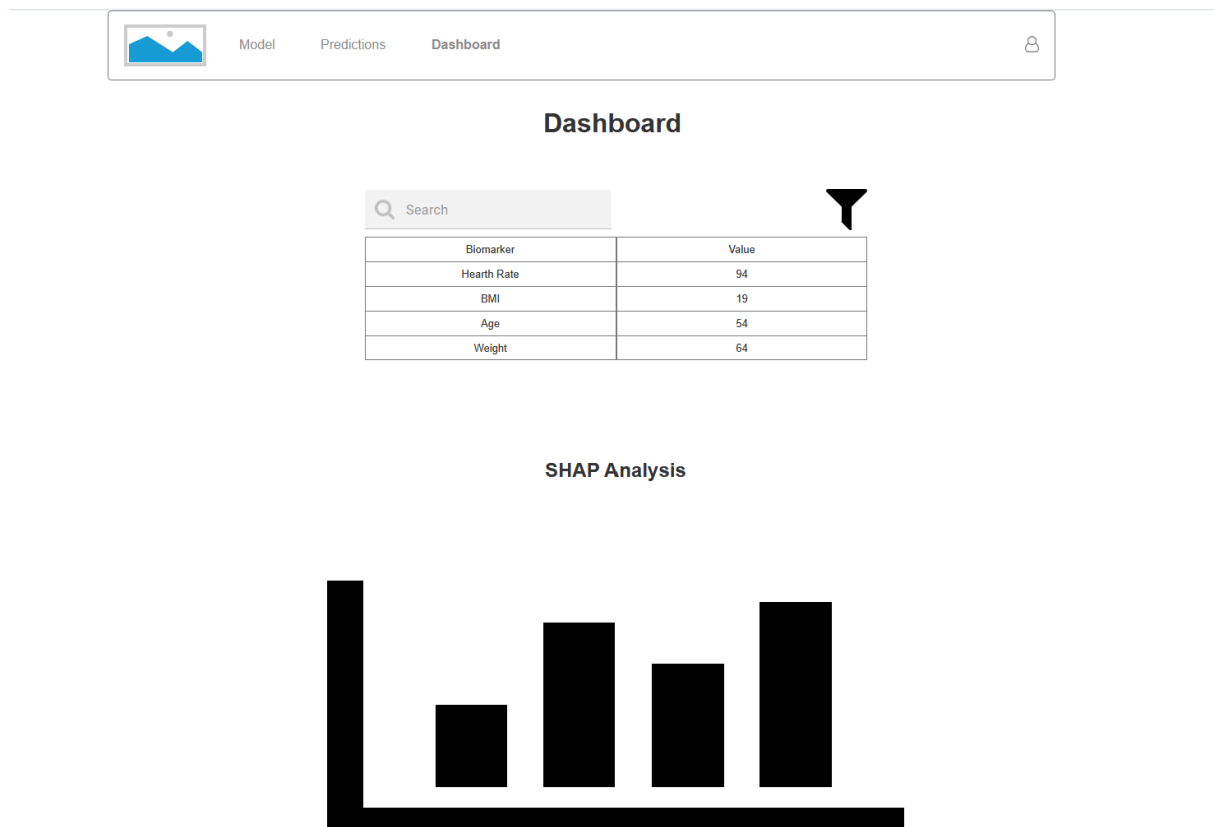


Figure 5. Dashboard page prototype to gain deeper insights into an individual prediction.

6. Deployment

To make the application accessible to third parties, it is essential to deploy it on a server. For this, I will utilise Kubernetes microservices. This enables me to maintain full separation of concerns compared to a monolithic approach, where all components would be tightly coupled into a single deployment.

Each major part of the application — the web application, the database, and the machine learning inference service — will be packaged into its own container and deployed independently. Kubernetes will orchestrate these containers, ensuring scalability, resilience, and easier updates in the future. Using this approach also allows for better load balancing, simplified service discovery through Kubernetes Services, and fault isolation, meaning that if one component fails, it will not directly bring down the entire system.

For local development and testing, the application will initially be run using Docker Compose. This allows a fast and lightweight simulation of the production environment, without the overhead of a full Kubernetes cluster. When migrating to a production environment, these containers can be adapted for Kubernetes deployment using manifests such as Deployments, Services, Ingress configurations, and Persistent Volume Claims (PVCs) for shared storage.

7. Technology Stack & Justification

To develop this solution, a solid technology stack is needed to ensure robustness. Below I have outlined my decision-making process for selecting technologies. I do this by making use of a weighted ranking method for most parts.

7.1. Development Environment

For the development of both the machine learning pipeline and the application, I have chosen to use a virtual machine (VM) managed with Vagrant and running Ubuntu Server. This setup ensures a controlled, isolated, and reproducible environment that mirrors production environments more closely. I selected Ubuntu Server for its lightweight nature, stability, and compatibility with the technologies I plan to use (such as Python, Flask, and machine learning libraries).

Vagrant allows me to easily configure, deploy, and manage virtual machines using a simple Vagrantfile. In this file, I have defined key configurations such as memory allocation, CPU usage, and network settings to ensure that the VM performs optimally for my development needs. Specifically, I allocated 2GB of RAM and 1 CPU core to the VM, which provides a balanced environment for working on smaller tasks while ensuring that resources are available for the deployment of machine learning models and other tools.

The Vagrantfile also contains specifications for the virtual machine provider (in this case, VirtualBox) and its settings. While the default network setup created some challenges initially, I have adjusted the configuration to allow for a more streamlined connection between the VM and the host system. This configuration is important as it ensures compatibility with external services and provides a foundation for future integrations with Kubernetes or container orchestration tools. Although the networking setup will need further refinement when deploying to more complex systems, this simplified environment will support the scope of the first few phases of the project.

As I work on the pipeline and the app, I'll use this VM to install all necessary software and libraries. The Vagrant setup provides flexibility, allowing me to easily replicate this environment on different systems or share it, ensuring a consistent development experience. For now, the environment remains lean, focusing only on the essentials for building and testing the pipeline and app. However, I anticipate scaling the setup or transitioning to other infrastructure as I move into the deployment phase, particularly when utilizing Kubernetes or H2O to manage the model deployment.

In the future, the virtual machine's configuration may need to be adjusted to accommodate higher resource demands, especially when moving to production. If required, I can update the Vagrant file or migrate to cloud infrastructure, but for now, the VM provides a solid, replicable foundation for the development work ahead.

7.2. Pipeline Technology Selection

To determine the best framework for my machine learning pipeline, I compared Scikit-learn (Scikit-learn, n.d.) and H2O (H2O.ai, 2025) based on key criteria. Each has distinct strengths, making it essential to evaluate them against the project's needs. ([Attachment A](#))

One of the first aspects I considered was own experience, as familiarity with a framework can significantly impact development speed. Since I have worked with Scikit-learn before, it felt like a more intuitive choice. H2O, while powerful, comes with a steeper learning curve, which could slow down initial development. When evaluating scalability and distributed computing, H2O clearly stood out. It is designed to handle large datasets efficiently and has built-in support for distributed computing, making it a strong candidate for data-intensive applications. Scikit-learn, while widely used, is better suited for smaller-scale tasks and requires additional tools like Joblib (Joblib, 2021) for parallel processing.

Model training speed was another important factor, as faster model training enables quicker iteration and optimization. H2O demonstrated better performance in this regard, particularly when dealing with large datasets, whereas Scikit-learn can become slower without optimization techniques.

Both frameworks performed well in terms of integration and flexibility, allowing for seamless incorporation into machine learning workflows. They support a variety of use cases and can be adapted to different pipeline structures, making them both viable options in this area.

Considering these factors, H2O emerged as the better choice for this application due to its scalability, distributed computing capabilities, and efficient training. Whereas Scikit-learn requires additional tools to reach the same level of performance as H2O. (Scikit-learn, n.d.)

7.3. Deployment Technology

To determine the best framework for my machine learning pipeline, I compared Kubernetes and H2O based on several key criteria. Both have unique strengths, making it important to assess how they align with the project's needs. ([Attachment C](#))

The first criterion I considered was own experience. In this case, I have more experience with Kubernetes, which initially gave it an advantage. However, since H2O offers a more user-friendly setup for machine learning workflows, I rated Kubernetes lower in this category, as its complexity could slow down the initial development phase.

When evaluating scalability, Kubernetes had the edge due to its robust ability to handle large-scale distributed systems. It supports a wide variety of cloud-native solutions, making it ideal for scaling applications across multiple nodes. On the other hand, H2O, while capable of scaling efficiently, is more focused on machine learning tasks and has slightly lower scalability compared to Kubernetes, especially in complex distributed environments.

Complexity was another critical factor. H2O shines here due to its streamlined setup and ease of use, which reduces the time required for deployment and management. Kubernetes, while highly flexible, can be complicated to configure and manage, especially for those who are less experienced with container orchestration and distributed systems.

In terms of performance, H2O was slightly favoured. It is optimized for machine learning tasks, and its performance on large datasets is very efficient. Kubernetes, though capable of handling heavy loads, is more general-purpose and might require additional optimizations to reach the level of performance that H2O provides out of the box.

When evaluating integration with the application that will interact with the machine learning model hosted on the cluster, H2O excels in this area. It is well-suited for integrating with machine learning models, allowing seamless deployment and interaction between the application and the model through REST APIs and its native support for serving models. Kubernetes, while capable of hosting applications and models through containerized deployments, requires more setup and management, especially when integrating the application with the machine learning model. Kubernetes offers more flexibility in terms of infrastructure, but H2O provides a more specialized and straightforward setup for serving models directly to applications, making it more efficient for this use case.

Finally, flexibility was a close comparison. Both frameworks offer a high degree of flexibility, but H2O's focus on machine learning allows it to provide a more specialized environment for ML tasks, whereas Kubernetes excels in broader use cases, including microservices and containerized applications.

After weighing these factors, H2O emerged as the better option for this particular machine learning pipeline. While Kubernetes offers excellent scalability and flexibility for distributed applications, H2O's optimized machine learning performance, ease of use, efficient model serving, and straightforward integration with the application make it the preferred choice. The decision ultimately favours H2O for its strong support of machine

learning workflows and model serving, whereas Kubernetes, while powerful, requires more complex setup and additional tools to match H2O's specialized performance in this domain.

7.4. Backend technology

To decide on which framework I will use, I made use of a weighted ranking matrix ([Attachment B](#)). The backend will be written in Python, so only Python frameworks are considered. I compared four major frameworks based on several criteria. The frameworks in question are Flask (Flask, n.d.), Django (Django, n.d.), Streamlit (Streamlit, n.d.), and H2O Wave (H2O.ai, n.d.). These all have their strengths and weaknesses, so it's important to focus on the most relevant features for my project.

The first thing I looked at was own experience, which is crucial when working under a time constraint. Since I have worked with Streamlit for past school projects, I gave this the highest score. I am already familiar with its simple setup and how quickly it allows for building interactive UIs, which makes it appealing for rapid prototyping. However, I have also used Flask before in a personal project, which gave me some exposure to the workings of this framework. That combined with my experience with other backend technologies like Java, .NET, and PHP makes Flask a solid contender. Flask's simplicity and flexibility, along with my familiarity with backend concepts, will make it easier for me to work with. Django, on the other hand, I have never used or been exposed to, and since I am working with limited time and need to avoid a steep learning curve, I would not prefer this framework for this project. H2O Wave, on the other hand, has a less steep learning curve. But since it is completely new to me, I also give it a very low score.

Scalability is another critical factor, as the application needs to be production-ready and able to support multiple users. Here, Django excels, as it is designed with larger, complex applications in mind, offering built-in features for handling large-scale projects, authentication, and user management. Flask, while scalable to some extent, is typically used for smaller applications, which suits my current needs for this project. Streamlit and H2O Wave, however, are primarily built for quick prototyping and is not designed to scale for production-ready applications. Its limitations in handling high loads or user concurrency make it a less viable option for a production system. (Toxigon, 2025)

Another thing to take into consideration is performance when choosing between Flask, Django, and Streamlit. Flask is lightweight, which can be a benefit in terms of speed for small applications, as it does not add unnecessary overhead. It is also well-suited for API integrations, which is important for the machine learning pipeline in this project. Django's performance is solid but comes with a heavier structure due to its "batteries-included" approach, which might add some overhead for the specific needs of this project. Streamlit, although optimized for displaying data visualizations and simple UIs, is not ideal when considering performance for complex backend processes or large-scale deployments. H2O Wave offers very good performance, especially when it comes to model interactions. (Restack, 2024)

Security is another key criterion to consider for any web application. Django comes with many built-in security features, including protection against common threats such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF). It is a robust choice for securing an application out of the box. Flask does not come with built-in security features like Django but allows for flexibility in choosing the security measures you implement. You will need to manually integrate features like authentication, authorization, and protection against common vulnerabilities. Streamlit and H2O Wave, being focused on rapid prototyping and simplicity, do not offer much in terms of security out of the box and would require extra work to secure the application properly, especially for more sensitive data handling. (Geeks for geeks, 2024)

Another important factor is integration, as the backend requires integration with APIs that connect to a machine-learning model. Flask is highly flexible and integrates well with various Python libraries, including scikit-learn, TensorFlow, and others. It also offers simplicity in building REST APIs, which is crucial for connecting the backend with the trained model. Django, with its more extensive features and structure, would likely require more effort to integrate. Streamlit, on the other hand, can quickly integrate machine learning models and is well-suited for visualizing model predictions and outputs, but it's not as well-suited for handling complex backend workflows and API integrations in a scalable way. Compared to H2O Wave, which is specifically designed for model interactions. (Restack, 2024)

Flexibility was an important factor, as the project might need to be adapted and changed over time. Flask's flexibility shines here, as it provides a minimalistic framework that lets me design the app exactly as needed without enforcing any strict project structure. This is particularly useful when you have specific requirements and limited time. Django, with its opinionated structure, is more rigid, which could be a benefit in larger applications but could slow down development in smaller, more customized applications. Streamlit and H2O Wave offer great flexibility for building UIs quickly, but it is somewhat limited when it comes to backend features and customization beyond data display. (Geeks for geeks, 2024)

When adding up the scores based on these criteria, Flask comes out as the best option for my project. It provides the necessary flexibility, scalability (to a reasonable extent), and ease of integration with machine learning models, which makes it ideal for the limited scope and time I have. Django, Streamlit and H2O Wave each have their strengths, but for the specific requirements of this project, Flask strikes the best balance between performance, scalability, and development speed.

7.5. Frontend Technology

For the frontend, I have chosen to use Bootstrap (Bootstrap, n.d.), a widely adopted CSS framework known for its simplicity and efficiency. Bootstrap offers a robust set of pre-designed components and responsive grid layouts, allowing for the rapid development of modern, user-friendly, and intuitive UIs. Its built-in mobile-first approach ensures that the application will be responsive across a wide range of devices without requiring extensive custom CSS. Additionally, Bootstrap integrates well with other libraries and frameworks, making it flexible and adaptable to the specific needs of my project. Given its ease of use and ability to significantly speed up the UI development process, Bootstrap is the ideal choice for this application, allowing me to focus more on functionality while ensuring a clean, professional design.

8. Developement Plan

For this project I have made a first draft of a detailed planning. This includes all sprints I will perform, and all task included in those sprints. It is important to note that this planning is not final, as the deployment part of the project is not well defined yet.

8.1. Sprint 1 24/02 – 14/03 Setup & Research

Define project scope & requirements: During this task I will gather all requirements and make sure to have a proper definition of them. I will also define the scope of the project using MOSCOW analysis.

Research Technologies: I will research all suggested technologies that I will use, but also alternatives and technologies in which I am free to choose.

Setup Environment: I will make sure my entire development environment is setup correctly, that way the realisation fazes will go much smoother.

Write Project Plan: During this sprint I will also complete my project plan document. This is a crucial document that I can use as a guideline throughout the internship.

Make Planning: I will also make a well thought out planning, giving me a better overview of how I spend my time.

Make Low Fidelity Prototype: This acts as a guideline when realising the application, ensuring that I don't miss any important features.

Make Use Case Diagram: This will act as a guideline when implementing user roles in the application.

At the end of this sprint, I should have a finished concept plan which includes: a prototype, use case diagram, planning, risk analysis, and a clear definition of the project. I will also ask feedback on this and adjust where needed.

8.2. Sprint 2 17/03 – 28/03 Building End-To-End Prototype

Data Ingestion Dummy Data: I will write the code of the pipeline to handle data ingestion of dummy data.

Data Preprocessing Dummy Data: Creating the part of the pipeline responsible for data preparation with dummy data.

Model Training Dummy Data: I will train a model on the dummy data integrated into the pipeline with dummy data.

Model Evaluation Dummy Data: I will automatically evaluate the model that was trained on dummy data, giving some key metrics as output.

Basic Backend Implementation: I will implement a basic version of the backend of the application, this way I can use the trained model to make predictions on the app.

Basic Frontend Implementation: This is needed to interact with the model in the backend. This will cover close to none styling but is based on functionality. Ensuring model interaction and result retrieval from the model on the application.

API Pipeline: An API that will enable me to send requests to the model for inference.

API Application: An API that will enable the application to communicate with the model API, enabling model inference and retrieval of key statistics.

Authentication Auth0: To make sure the application is safe and can only be accessed by approved users. I implement this now to get more experience with Auth0, this way I make sure the production application will be as secure as possible.

Input Parameters Page: This page will use a form to get the parameters from the user to the model for inference.

Models Page: This page will show key metrics of the selected model.

Dashboard: This page will be used to get more insights into a single prediction made by the user.

Admin Page: This page will be used by admins to manage access control and models.

At the end of this sprint, I should have a basic version of the application working, which can then be presented to my mentor. The reason for making this first simple draft is that changes are easily made without losing too much time. This also gives my mentor a very good overview of my understanding of the requirements of the project.

8.3. Sprint 3 31/03 – 11/04 Application Development

API Development: Building a connection between model and application for inference.

Model Integration: Making sure the correct parameters can get passed to the model.

SHAP Integration: The model can return SHAP features from a prediction.

Deployment: Deploy the app so it is accessible for clinicians.

Document Backend: Documenting all my work of building the backend.

Make Coherent Styling: Designing the application so it's easy to navigate and use. Sticking with a minimal design to optimise user friendliness.

Implementing Styling: Using the design on the frontend of the application.

Implement Authentication: Restricting access to the application to those who need it.

Document Frontend and User Manual: Documenting all the work done on the frontend. And writing a user guide for new users.

At the end of this sprint, I will have a fully working application which I will request to be tested by my mentor to ensure user friendliness. The app should be secure and fully integrated with the model.

8.4. Sprint 4 21/04 – 28/04 Data Ingestion & Preprocessing

Analyse Real Data: I will start by analysing the real data used for the model training, I will not analyse this data in depth to be clear, but I will analyse the features that need to be considered for a later stage – preprocessing.

Implementing Data Ingestion: Requires me to understand what the data format is so it can be properly ingested into the pipeline ready for preprocessing.

Define/Document Data Ingestion: Documenting all my findings, keeping in mind the confidential nature of my findings.

Implement Data Preprocessing: Implementing all previously researched preprocessing steps into the pipeline to make the data ready for training.

Save Model Features: Needed for inference when the model is deployed, so the correct fields are used as parameters.

Define/Document Preprocessing: Documenting all my findings, keeping in mind the confidential nature of my findings.

At the end of this sprint, I should have a good working first half of the pipeline that will be able to handle all data. I will also request adequate feedback on this, as the data I am working with is very sensitive and delicate.

8.5. Sprint 5 29/04 – 09/05 Model Training & Evaluating

Choose Model Based on NULL Values: Important to select different models when there are missing values.

Train Models in Sequence: Training the models in sequence to experiment.

Train Models Parallel: Ensuring maximum efficiency, all models can be trained at the same time.

Document Model Training: Documenting all my findings, keeping in mind the confidential nature of my findings.

Evaluate Models on Predefined Criteria: To define a model's performance.

Make System to Choose the Best Model: Using the evaluation criteria to select the best model.

Save Model: Saving the trained model to a .pkl file.

Document Model Evaluation: Documenting all my findings, keeping in mind the confidential nature of my findings.

Include SHAP analysis: To make the models choices explainable.

Finishing this sprint, I should have a good working pipeline that is able to handle all the steps without failure and deliver a model that is ready for deployment. I will request feedback about the entire pipeline, but will put emphasis on my model selection system, as this can have a big impact.

8.6. Sprint 6 12/05 – 23/05 Model Deployment

Research H2O: To determine the best fitting framework.

8.7. Sprint 7 26/05 – 06/06 Testing & Finetuning

Test System End-to-End: Ensuring everything works correctly and no bugs remain.

Fix Bugs: Fixing all remaining bug found during testing.

Improve Documentation: Revising documentation where necessary for better readability.

Finishing this final sprint, I will be ready for my end presentation. I will have refined everything and ensure a fully working pipeline and application end-to-end, which will be tested by me and my mentor.

9. Testing & Validation

To validate and test my solution, I will foresee regular demo meetings with my internship mentor. This way I can ensure that my solution is aligned with the expectations. For more detailed testing, I will be making use of Swagger. This allows me to test my APIs to see if data is transferred correctly between containers. Allowing me to spot any issues and bugs in due time.

10. Security Considerations

Working with sensitive clinical data requires extra security precautions. This has impacted some of my choices as described above, but also some design choices. Security is one of the reasons for why I have opted for Auth0. This allows me to omit storing users' data into the local database which can be a significant security issue. Auth0 handles all user authentication and authorization, streamlining the development process and ensuing up to date security. With this approach I can make use of a JWT token which I can add metadata to, allowing for easy route protection on both the back and frontend.

11. Challenges & Solutions

The biggest challenge of this assignment will be GDPR and HIPAA compliance, as it is something I have no formal experience with. I am eager to learn, so that is why I will request a review from my internship mentor of everything I make that has to do with data before it gets deployed or shared. This way I can ensure that I keep data confidential, and that I maximise my experience with sensitive data.

Another challenging factor of my assignment will be deployment. I have little experience with the deployment of machine learning models and applications, so I expect a steep learning curve. That is why I have foreseen 2 weeks for deployment alone in my planning as well as 2 weeks buffer at the end of the internship for if this proves more challenging than expected.

12. Risks

There are several risks associated with my internship assignment. One of these risks is access to the production model. When I don't have access to the production model, I cannot integrate it in the application. In case this happens, I will make a fully working proof of concept utilising dummy models and data. This ensures that a fully working version is available, which can easily integrate the production model.

Another associated risk is security. My application will eventually be deployed on a live server, so fortification against common web vulnerabilities is crucial when working with sensitive data. Since I have no formal experience with making web applications secure, there is a risk of some vulnerabilities sneaking in unnoticed. To combat this, I will do research and will ask my internship mentor to review my code before deploying.

Lastly, time management is a risk. My internship assignment encompasses many aspects: development, machine learning, architecture, and infrastructure. Areas like architecture and infrastructure where I don't have much experience in, will require additional research and trial and error to get working correctly. To mitigate this, I will follow extra online courses to accelerate my knowledge, resulting in quicker results.

13. Reference List

- Bootstrap. (n.d.). *Introduction*. Retrieved from Bootstrap: <https://getbootstrap.com/docs/4.5/getting-started/introduction/>
- Django. (n.d.). *Django Documentation*. Retrieved from Django project: <https://docs.djangoproject.com/en/5.2/>
- Flask. (n.d.). *Flask documentation*. Retrieved from Flask: <https://flask.palletsprojects.com/en/stable/>
- Geeks for geeks. (2024, September 25). *Flask vs Django – Which Framework Should You Choose in 2024*. Retrieved from Geeks for geeks: <https://www.geeksforgeeks.org/flask-vs-django/>
- H2O.ai. (2025). *The World's Best Deep Research*. Retrieved from h2o: <https://h2o.ai/>
- H2O.ai. (n.d.). *H2O Wave Documentation*. Retrieved from H2O.ai: https://docs.h2o.ai/h2o_wave/index.html
- Joblib. (2021). *Joblib: running Python functions as pipeline jobs*. Retrieved from Joblib: <https://joblib.readthedocs.io/en/stable/>
- Lundberg, S. (2018). *An introduction to explainable AI with Shapley values*. Retrieved from Shap: https://shap.readthedocs.io/en/latest/example_notebooks/overviews/An%20introduction%20to%20explainable%20AI%20with%20Shapley%20values.html
- Restack. (2024, November). *Streamlit vs Flask vs Django comparison*. Retrieved from Restack: <https://www.restack.io/docs/streamlit-knowledge-streamlit-vs-flask-vs-django>
- Scikit-learn. (n.d.). *Pipeline*. Retrieved from Scikit-learn: <https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html>
- Streamlit. (n.d.). *Streamlit Documentation*. Retrieved from Streamlit: <https://docs.streamlit.io/>
- Toxigon. (2025, January 19). *Django vs Flask: Comparing Python Web Frameworks in 2025*. Retrieved from Toxigon: <https://toxigon.com/django-vs-flask-a-comparison-of-python-web-frameworks>

ATTACHMENTS

ATTACHMENT A

Criteria	Weight	Scikit-learn	H2O
Own Experience	0,20	7	3
Scalability (large data sets)	0,20	5	9
Distributed computing build in	0,20	5	10
Model training speed	0,15	6	8
Integration	0,15	9	9
Flexibility	0,10	9	9
Total	1,00	6.55	7.85

ATTACHMENT B

Criteria	Weight	Flask	Django	Streamlit	H2O Wave
Own Experience	0,20	5	0	6	2
Scalability	0,20	7	9	4	4
Performance	0,15	7	7	5	8
Security	0,20	7	9	5	2
Integration	0,15	7	7	9	10
Flexibility	0,10	9	7	5	4
Total	1,00	6.8	6.6	5.6	4.7

ATTACHMENT C

Criteria	Weight	Kubernetes	H2O
Own Experience	0,10	5	3
Scalability	0,20	9	7
Complexity	0,10	6	8
Performance	0,20	8	9
Integration	0,30	9	9
Flexibility	0,10	8	8
Total	1,00	7	7.8

ATTACHMENT D

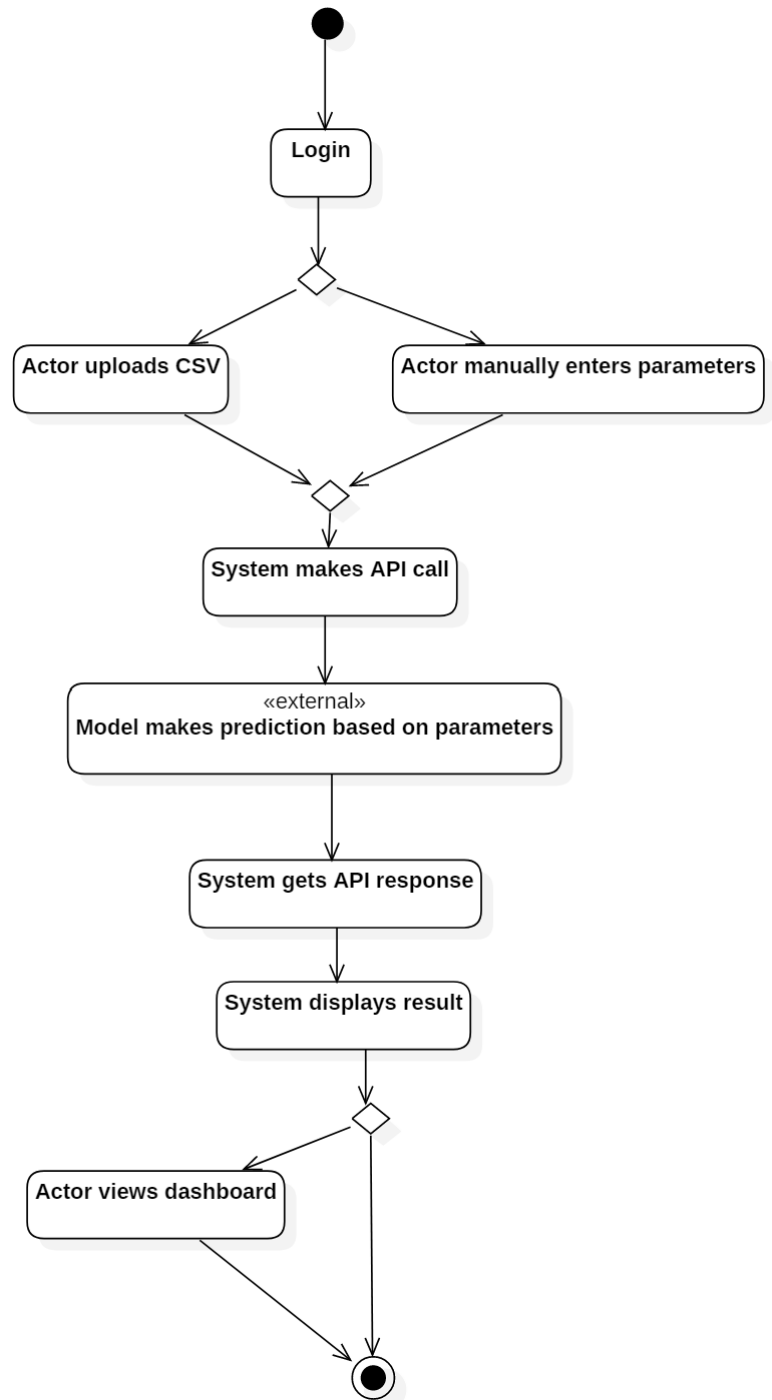


Figure 6. Application flow for making predictions.

ATTACHMENT E

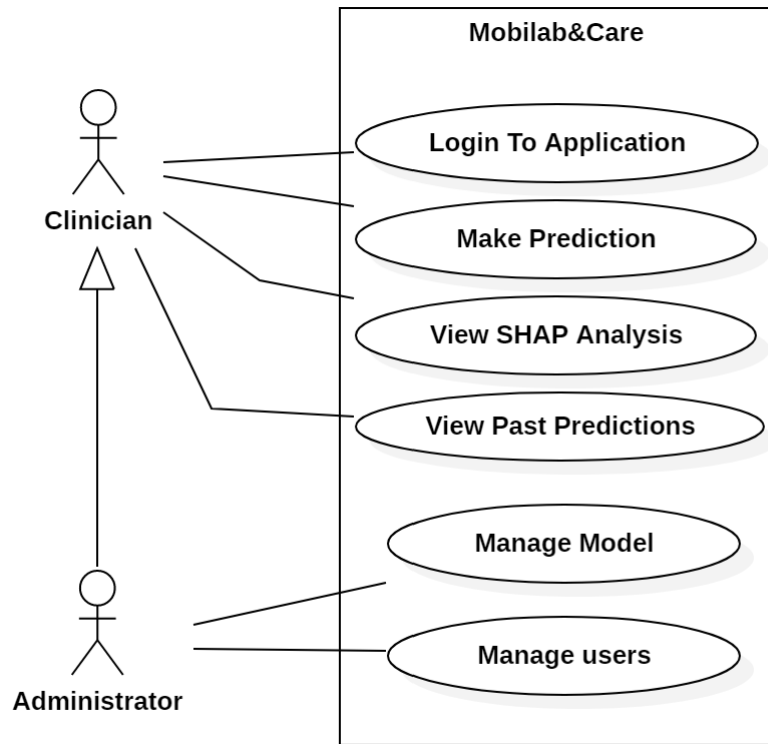


Figure 7. Use case diagram of the application.