Please open this AI black-box!

Creating interfaces to make the inner workings of machine learning algorithms clearer

by

Technical Writing Group 11D

Silvia Mokráňová
Mihai Filimon
Mathieu Butenaerts
Kanish Dwivedi
Floris van Veen

Client: *Delft AI Bias Detectives*

Teaching Assistant: Ola Wolska
TU Coach: Luciano Cavalcante Siebert

17 June 2021

TU Delft

Preface

This project was done by 5 second-year computer science students studying at TU Delft. For the last quarter of the second year, students must sign up and work for a client to accomplish a task of their choosing. The students that make up this team are 5 students who chose the same client and project: to 'open this AI black box' - to create a web application that could allow users to better understand what is going on behind the scenes in Artificial Intelligence (AI) image identification. The clients of this project are the creators of the SECA method, and they asked for a clear interface for users to interact with their algorithm, and to understand what exactly is happening.

This report was primarily written for examiners at the TU Delft, who have a technological background and will be able to understand all the computer science related concepts that will be covered. Regardless, any specific concepts will be briefly introduced to ensure that the reader understands what is being said. Readers will be assumed to have at least a basic understanding of the background of machine learning algorithms and their difficulties or dangers, along with basic understanding on how websites communicate with a server to function.

Readers with an interest in what technologies are needed to create such an application should look to Chapter 4 (Product Design) for further information.

We would like to thank Agathe Balayn for helping us whenever we had questions regarding the algorithm or anything related to the project. We would also like to thank the rest of our clients: Natasa Rikalo, Christoph Lofi and Jie Yang, the creators of the SECA method for accepting us up on the project, along with helping us whenever we needed assistance. Along with the clients, we would like to thank for the support that we received via the TU Delft: Ola Wolska, for helping us ensure everything is on track and for answering our questions about university's expectations, and Luciano Cavalcante Siebert for helping us by giving feedback for various technical assignments, and Sjaak Baars for both teaching us how to properly write our reports, and for helping us improve our report through constant feedback.

We hope that our product is helpful to the clients, and that they find it as useful as we have. We thoroughly enjoyed making it and have learned a lot from this experience.

Delft 17 June 2021
Mihai Filimon, Silvia Mokranova, Mathieu Butenaerts, Kanish Dwivedi, Floris van Veen

Summary

SECA is a framework which was created by the Delft AI Bias Detectives, a team consisting of Agathe Balayn, Christoph Lofi, Jie Yang, and Natasa Rikalo, and which was created to metaphorically 'open the black box' behind artificial intelligence - to allow for easier understanding behind what is going on behind image classification software. The student team consisting of Mihai Filimon, Silvia Mokranova, Mathieu Butenaerts, Kanish Dwivedi and Floris van Veen were picked up by the clients - the Delft AI Bias Detectives - to create the final stages of this framework: to create an interface to show the information that is determined using the rest of the SECA framework. This should make the use of the SECA framework more accessible for the intended practitioners.

The aim of this report is to describe the development process in creating the interface, and to make the decisions the group made clearer.

The application should be displayed as a working set of interfaces, allowing a user to look through and clearly understand what classifications are being made, and where.

The first chapters of the report outline the research which has been done and the general analysis on the project itself.

After the Introduction, the second chapter of this report discusses the problem analysis that was conducted by the group at the start of the project. The problem statement and project goals are given. Research is conducted on other instances of similar products, and the appropriate methods which should be used to carry out the project. The actual visual design of the interfaces were already mostly complete because the clients created a desired layout, but there is much more to good design than the visual layout.

The third chapter discusses the requirements engineering the group conducted. This includes the Use-Case analysis to evaluate how exactly users would use the product, and an overview of the clients desired interfaces is given. The MoSCoW method is used to prioritise all the elicited requirements. Finally, additional requirements that were added by the client are also stated.

The fourth chapter describes the product designing phase of the project. The feasibility of the project is discussed and risk analysis is conducted. It is concluded that the project will be feasible in the given time limit. The products architectural design which is composed of the back-end, front-end and database are explained in detail, and thereby a solution to the problem is proposed. Finally, this chapter includes a discussion on the ethical implications around certain parts of the project.

The fifth chapter describes the development process. Scrum is chosen as the software development methodology and reasons for why this was chosen are provided. Communication along with the use of Gitlab is discussed. Finally, the chapter discusses how the group attempted to maintain code quality.

In the sixth chapter, the actual implementation of the product is discussed. All relevant information regarding the implementation of the final product is shown, including the issues that were faced during the implementation process, the decisions made, and a display of the final product's individual interfaces.

Finally, in the seventh chapter, a discussion is made of the final product. The requirements are evaluated along with the final product itself. Impact of challenges the group faced is also discussed, and the chapter ends with potential improvements and future recommendations.

In conclusion, the final product allowed the group to finish off all the must have requirements, and all but one should have requirement. As the product was made as maintainable as possible, the SECA Interface is readily available to be further improved with additional features.

Contents

1 Introduction

State-of-the-art computer vision methods are beginning to take up a significant part of people's daily lives. Nowadays, smart phones use image classification models for security, and autonomous driving use these models for identifying pedestrians. For these vision/image classification methods to be accurate and reliable, software developers employ neural networks which have been trained with large sets of data. Despite the developer writing the software themselves and having knowledge of how the neural network learn, these neural networks are still a "black box". The opaqueness of these models has become a major obstacle to verify their integrity along with making it harder to debug them [15]. Developers are yet to learn how exactly these complex networks with thousands of parameters, decide and classify an image. Knowledge of what features the model picks up on in the image that allows it to make the classification/decision is required. This is particularly the case in domains such as health and security where these models have been deployed [15]. Its entirely possible for the neural networks to develop biases during training; having the ability to open the "black-box" before deploying the network can prevent discrimination.

To tackle the issue of opening the "black-box", SECA has been developed by an interdisciplinary team between researchers from the EEMCS and Industrial Design faculties at TU Delft, the Delft AI Bias Detectives. SECA is a scalable framework that relies on algorithm explainability methods and human-in-the-loop approach for global interpretability of classification algorithms [3]; essentially producing interpretable and rich explanations about a model's behaviors [3]. Currently, SECA cannot be used in practice [15], as its methods cannot easily be used by practitioners. Having a proper user interface which allows for visualisations of the extracted explanations would make SECA user-friendly for practitioners.

The group's research aims to bring forth an user interface for SECA, denoted SECA Interface, that will support debugging and interpretation of deep learning-based computer models performing image classification. This report answers the question of how an interface could be implemented for SECA, with regards to both technical aspects, and design aspects. This will be done by first researching different software technologies and selecting the most suitable one to create a user-friendly and easy to use interface. This research will also provide insight into what the best design choices are for making such an interface. As the client has already provided the group with the SECA framework and its methods, the interface will need to be designed in such a manner that can easily integrate it.

The report is set out as follows. Chapter 2 discusses the problem analysis of the project; explaining the project problem, discussing conducted research, and deriving the project goals. Chapter 3 will go over the requirement engineering the group performed; discussing Use-Case analysis and giving a prioritized list of requirements using the MoSCoW method. Next, Chapter 4 will contain the study conducted at the beginning of the feasibility of the project, and a section on the risk analysis the group performed. Moreover, the chapter will explain the architectural design the group came up, and will discuss ethical implications. Chapter 5 will discuss: the chosen Software Development Methodology, how communication is managed, how the group used Gitlab, and finally how the group maintained code quality. Furthermore, Chapter 6 will go over the in-depth implementation of the interface. Lastly,

Chapter 7 will discuss the state of the final product and recommendations to improve the product if the client wishes to continue using it.

2

Problem Analysis

In this section, the background research that was conducted in order to understand and/or learn how to implement an interface is discussed. Subsection 2.1 specifies the problem statement, subsection 2.2 gives an overview of the prior research that was conducted by the group before the implementation began. Finally, subsection 2.3 establishes the goal of this project.

2.1. Problem Statement

With respect to a high level perspective, the client wants the group to create an interface that practitioners can easily use to debug and explore explanations of image classification neural networks. To achieve this, it is mandatory to integrate the current existing methods of the SECA framework they have developed into the interface, and allow it to be used based on practitioner inputs. The original description for the project can be found in Appendix B, from which the following parts of the problem were extracted. Once tackled, a solution that accomplishes the above stated overall problem will be produced:

1. Build a system that architecturally is split into two parts: back-end and front-end.
   The back-end component of the interface is responsible for data storage and computation. Specifically, the back-end component will answer to front-end requests and execute appropriate SECA methods to compute results.

   The front-end is the visual aspect of the interface, it is responsible for implementing interactive web-pages that match the visual designs the Delft AI Bias Detectives have created and more importantly communicate with the back-end to allow the practitioners to use the SECA methods.

   It is important to note that the intention will be that both the back-end and the front-end are implemented following good Software Engineering Methods; allowing for future developers to maintain the product and further enhance it with ease.

2. Integrating SECA method into the system.
   For the back-end to be capable of answering front-end requests, it will need to use existing parts of the SECA method. This means that parts of the SECA method will need to be extracted, and potentially patch code will need to be implement to connect these together.

3. Connecting the back-end and the front-end.
   Both the back-end and the front-end need to be connected in a fashion that fully supports the intended interactivity.

2.2. Prior Research

In order for the team to be able to create the best possible product, research had to be done. The team could not find any existing websites or other sources which had similar goals or outputs as SECA, and as such the research had to be split into different segments. The final product would essentially be a website that gives information on what is happening in the back end of an image classification algorithm,

and as such the team researched 2 main topics: website design and proper front- and back-end design principles. Website design covers the visual design principles in websites that may be applicable to the visual design of the interfaces, and proper design principles covers research on important principles that should be followed to ensure a successful and long lasting product.

2.2.1. Website design

While the clients have provided the team with a detailed outline of what they would like the interfaces to look like, the team still decided to make some brief research on examples of good website design. This would give everyone a better idea on proper visual design principles and allow for better visual implementations whenever they an addition that is not specified by the clients is made.

Brightspace

Brightspace is a website which is used to educate users, having functionality to create individual users, enrol in courses, and receive information that is present in that course in a clear and visually appealing way [4]. While Brightspace has nothing to do with machine learning, it is an educational website, and has a relatively similar layout to the client's specifications. The way that users move between pages is very intuitive and visually appealing - whenever a link is hovered over it changes colour and highlights itself. The interfaces all neatly slot into place and do not require horizontal scrolling to see anything. Navigation is always easy due to the navigation bar which is placed at the top of the screen. All these elements can be integrated into the final product if desired to ensure good visuals.

While Brightspace does have an elegant visual design and a diverse colour palette, there are some shortcomings which should be avoided. The largest shortcoming involves how the visual design changes as the page size changes, and how this can hide important elements of the page. While the page would ideally be viewed in a full screen, as the page decreases in size, more elements of the screen become hidden with no way to reveal them again. Different machines and layouts must, thus, be considered.

Squarespace - template page

While Brightspace is a website with a goal of educating its users, Squarespace is a website which is entirely focused around webpage design - it is dedicated to webpage templates [19]. This does mean that it has different goals than the proposed application, but it still does result in better ideas for visual design.

Ignoring the actual contents of the page, the template page does bear similarities to the proposed confusion matrix interface and query pages, in how they give filtration options, and below the actual contents are displayed. A confusion matrix would appear very differently from the template designs shown, but the layout would be similar in how the confusion matrix would stay in the center, and not pass beyond determined horizontal boundaries. This may not be entirely useful, as a confusion matrix could potentially be very large, and a readable matrix which the user would need to scroll horizontally to see everything is better than a matrix which has its size set to fit within the boundaries but is too small to actually read from.

There are few shortcomings to the visual design of Squarespace that should be avoided. The website is very well designed in general, but due to the large number of hours needed to implement the elegant visual effects present there, it is not feasible to replicate for the current project.

2.2.2. Proper application design principles

The final product will continue to be used and updated after the team has finished working with it. While the team themselves may not continue working on it, the clients may continue to develop it further and build upon the final product. This means that the team must work to ensure that the product is in a state which is sustainable, and clear for new developers.

Lecture material from TU Delft

Multiple proper design principles can be found in previous lectures provided by TU Delft. In particular, the lectures which were found to be relevant are lectures 2, 3, 5, and 6 of CSE2115 Software Engineer-

ing Methods, from November 2020. The second lecture revolves around 'requirements engineering', the third lecture covers the software architecture, and lectures 5 and 6 cover good design patterns that could be implemented in the final product.

Requirements engineering simply covers the appropriate methods one should use to properly evaluate the requirements of the project [23]. This is less required in this case, as the clients are also computer scientists and have delivered a clear set of requirements, but is still absolutely essential when making decisions on what to prioritize and for making a focused product.

Software architecture covers how components of a system are organized, assembled and how they communicate with each other [24]. From the set of architectures that are covered, the most appropriate architectures would be a model view controller, a layered architecture, or a client and server architecture. Each have their own benefits, but a client-server architecture appears to match the specifications given by the client.

Design patterns cover patterns which commonly occur in software, and can almost be considered 'recipes' to solve certain problems [22]. These design patterns can be helpful, but only the correct patterns should be used in the appropriate times. Examples of design patterns that could potentially be used in this project include the strategy pattern, the adapter pattern, and potentially a proxy pattern, as they all allow for increased flexibility in the back-end [22].

2.3. Project Goals

From the research, the group has learnt that there are no clear existing interfaces to learn from, and therefore the group needed to create the interface themselves following numerous Software Development design principles. In this subsection, the goals for the SECA Interface will be defined. As stated above, following design principles it was intended to make the software as maintainable as possible, because in the future the interface create during the Software Project will be further improved with additional features for the practitioners. The following project goals have been created with respect to the problem statement (Subsection 2.1) and the research the group members have conducted (Subsection 2.2):

1. Implement back-end with integrated SECA methods.
   The SECA Interface's back-end component needs to be completely implemented in order to provide all the required user functionalities on the front-end. All features requested by the Delft AI Bias Detective must to be taken into account when implementing the back-end, as these will influence which components from the provided SECA method needs to be extracted and integrated into the SECA Interface.
   Moreover, the back-end needs to be fully documented to easily allow future developers to scale it with increasing requirements.

2. Implement front-end imitating given designs.
   The SECA Interface's front-end component needs to be implemented in such a manner that it visually imitates the designs given by the Delft AI Bias Detective team, and needs to use the back-end to provide functionality.

3. Flexibility and maintainability.
   Flexibility refers to the SECA Interface being easy to use for the practitioners. Maintainability refers to the SECA Interface being easily expandable by future developers.

3

Requirement Engineering

This section of the report will focus on the requirements engineering process of the entire project. In Chapter 2 the report focused on problem analysis, the next step is to focus on the requirements. To extract requirements, the group had a meeting with the clients during the first week of the project. They provided the group with a list of requirements and their priority levels, as well as a high level description of different interfaces they would like to see in SECA Interface.

This chapter will first discuss the Use-Case Analysis, and then it will provide and overview of the requested interfaces. In the next subsection, all the requirements elicited from requirements engineering are stated. During the duration of the project, the client introduced some additional requirements changes, which are also discussed in that subsection.

3.1. Use-Case Analysis

After the first meeting with the client, where they proposed their created design schemes and which ended with a list of requirements, a Use-Case analysis was the second technique used to further refine the requirements of the SECA Interface system.

From the first meeting the group made clear that the stakeholders of the *SECA* Interface are Artificial Intelligence software developers and Image Classification Experts. Moreover, the first meeting also made clear as to what the interfaces looked like, however, one meeting was not enough to gain in-depth understanding of how the interfaces were intended to be used by the user/practitioner. The group required a more technical view of what the SECA Interface should look like from the perspective of the users themselves, as having such knowledge will allow the team to understand to gain more insight into the requirements. To do so, the team performed a Use-Case analysis, which can be found in Appendix C, along with a Use-Case Diagram (visual representation of how a user may use the application).

A Use-Case analysis involves ”designing a system from the user’s perspective, communicating system behavior in the user’s terms, and specifying all externally visible behaviors” [20]. This lets the group understand how the system will be used from the perspective of the user and what the system does in response to the user stimulus [20].

When performing this analysis the perspective of three user roles was considered - Basic users, Experts and Developers. It should be noted that ”Basic” users are no different from Experts and Developers. Both Experts and Developers are Basic users; the term ”Basic” is used to define the common characteristics between Experts and Developers. From the client presentation it was known these roles had distinct privileges in the SECA Interface, and had a distinct set of features available to them.

One of the main takeaways of the Use-Case analysis was the importance of authentication and se-

curity in the SECA Interface. The client had already made their intention clear that they required a login and registration system (as it can be seen in the following section), however during the analysis process the group understood additional research needed to be done in order to define how to ensure the application's security. In fact, security will be further discussed in Chapter 4.

3.2. Overview of client's request and desired interfaces

As mentioned, the client tasked the group with making an application in which their algorithm's results can be used by developers. The application is expected to have various interfaces and functionalities, as well as a consistent design. The overall application should be a web application with tabs that contain the described interfaces. This section describes the various interfaces that the client requested during the first meeting. It only contain a brief description, the more concrete requirements are described in sections 3.3.1 to 3.3.5.

Welcome page
The client would like the user to be first welcomed with a simple page, that includes the logo of the project and a simple welcome message. From this page, a user should be able to choose either log in or registration, which transfers them to the correct page accordingly.

About interface
The interface that a user should land on after logging in is an *About Interface* interface, which should contain a brief description of everything that can be done on the website, as well as direct links to the described pages.

Description of SECA
Next there is an interface which actually describes what the SECA method is and how it works, so that a user is informed about what will happen to their data, if they choose to upload it and use the website.

Confusion Matrix
The *Confusion Matrix* interface is the first dynamic interface that the group was asked to implement. It should contain a matrix with data extracted from the algorithm, with clickable cells. The cells should lead to two different follow up pages, based on which cell it is. These follow up pages should show typicality scores and some overview of the selected data, which is all extracted from the integrated SECA framework methods in the back-end.

Explore
Another thing a user should be able to view is an overview of common concepts extracted by the SECA framework. The *Explore* tab should display these extracted rules along with their typicality scores and should contains various filters.

Query
In the *Query* tab the user should be able to select various concepts present in the image and view a confusion matrix or typicality scores based on that. The SECA framework has support for this filtering and the interface needs to display the results. The user should also be able to query specific images based on the same filters.

Wiki
With a more complicates issue, a developer might need help from an expert with ensuring their algorithm is giving the correct categorization, as well as seeing if the concepts and rules retrieved from the rule mining make sense. For this, the *Wiki* page should be used, where an *expert* user can input information about specific classes, as well as a brief description, expected concepts and an example image.

Info
The *Info* tab should contain an interactive tutorial on how the interface can be used. This tab should help the user with navigating the interface if they are using it for the first time or if they are unsure how to use something.

User Profile
The *User Profile* section was not of high priority for the client, so they did not provide a detailed description of it. However, in the version of the interface the group worked on, it should simply contain basic information about the logged in user and about their uploaded projects.

3.3. All Requirements acquired from Requirement Engineering

In this section, the entire list of extracted requirements are stated. The first set are the Non-Functional requirements, which are requirements that specify criteria that can be used to judge the operation of a system [14]. The second set are the functional requirements, which contrast the non-functional requirements as these specify criteria that define specific behaviour [10].

The group partitioned the functional requirements into tiers of priority using the MosCOW methodology [11]. This results in the first tier of requirements, the *must have*. These are requirements the group must achieve before the deadline of the project. The second tier are the *should have*; requirements that the group should be able to achieve if there are no severe consequences encountered. Thirdly, there are the *could have*, these requirements are optional and are only to be prioritized if there is any left over time for the group.

3.3.1. Non-Functional Requirements

1. The group is asked to setup a database
   (a) They will design a database suitable to the given data
   (b) They will populate the database with provided data
2. The group is asked to setup the web application with a system for logging in
3. The group is asked to setup a user interface based on the design provided by the client

3.3.2. Requirements the project must have

1. The system must have a login and registration feature for users
2. The system must allow a user to populate the database
3. The web application must have the following static interfaces
   (a) “About interface” page - this page contains plain text with explanations, provided by client
   (b) “About SECA” page - this page contains plain text with explanation of the SECA method, provided by client
   (c) “Wiki” page - this page contains static information about data
4. The system must allow a user to view a confusion matrix based on their data in an interactive way
   (a) The user must be able to select a cell on the confusion matrix to see more information about it
   (b) The user can view typicality scores for the concepts of the selected cell
5. Users can view the different attributes per class that SECA has selected via the Explore tab
   (a) Users can select if they want to see information about all images or just correct/incorrect predictions
6. Users can query different concept (or combination of multiple concepts) via the Query tab
   (a) Users can select if they want to view a confusion matrix or typicality scores overview
   (b) The users can select if they want all images, only correct or only incorrect predictions for the calculations
   (c) The users can select the classes they want included in the calculations

3.3.3. Requirements the project should have

1. On the Query interface users should be able to query images
    (a) Users can use the same filters as for the rest of the query page
    (b) Users can view images based on their actual class and their predicted class
2. Users should be able to take notes while browsing the different interfaces
    (a) The system should have at least a simple plain text sidebar for note taking
    (b) The sidebar for note taking should be available throughout the interface
3. User should be able to view extra information in the form of a pop-up by clicking concepts, rules and images
4. The system should have a tutorial interface that shows how to use the application, referred to as the Info interface

3.3.4. Requirements that the project could have if time allows

1. If possible, there could be two types of users:
    (a) Developer type
        i. Primary user, makes use of all aforementioned features
    (b) Expert type
        i. Can answer questions about data in their field of expertise
2. The group could expand the aforementioned sidebar with more features
    (a) They could add the option to take two types of notes
        i. Notes on rules
        ii. Free notes
    (b) They could add the ability to link generated rules to each other
    (c) They could add the ability to take notes about the links of rules
    (d) They could add the option to interact with an Expert user inside the notes interface
3. Developer users and Expert users could have the ability to comment, ask questions and answer questions in the *Wiki* interface

3.3.5. Requirements that certainly will not be completed

1. The user will not be able to import their own images via the *Wiki* interface
2. The option for multiple users to use the same session and interact with the same data
3. The user will not be able to pick specific annotations and get more details from them

3.3.6. Additional requirements proposed by the client

While the group was working on the project, the client had interviews with potential users of the application and has come up with some additional requirements. The client and the group had a meeting discussing these requirements and the group then discussed which of the new requirements they are able to implement in the remaining time. The group then informed the client about which requirements they are able to tackle and these are listed bellow. The client placed the priority of these above the priority of the *could have* requirements, giving them same priority as the *should have* requirements.

1. Integration of the simplest Dashboard possible
    (a) The dashboard should contain a smaller version of the confusion matrix and simplified query page
    (b) The dashboard should be the first interface a user sees after login

2. Integration of the F1 Score
    (a) Integrate the F1 Score with the other metrics shown
3. Integration with the concept list (ranking and filtering)
    (a) Display coverage within images and typicality from rule mining and statistical calculation

4

Product Design

Now that the requirements have been properly handled and evaluated, the team needed to start thinking about how to make the actual product. This section will cover a feasibility study, discussion on ethical concerns, and a brief overview of the frameworks and design decisions that will go into the creation of the product.

4.1. Feasibility of the project

After properly evaluating the requirements given by the client, a small investigation needed to be conducted to see if the project can feasibly be done in the time available, with the available frameworks, and with the experience of the team.

4.1.1. Technical feasibility

In regards to technology required, the project is nothing special. The clients have already completed their algorithm, which is by far the most technically complicated and advanced segment of the overall SECA framework. The interfaces are very technically simple, only requiring simple *Python* and *JavaScript* to meet all the requirements.

The team has decided to use *Django* to run the back-end of the project, and *React.js* to create and style the front end user interfaces. Django is a web framework that allows for fast development and clean design, called 'The web framework for perfectionists with deadlines' [7]. React is a simple JavaScript library for building user interfaces, which allows developers to implement these interfaces much more easily than they otherwise would [16]. With these two frameworks, the team believed that the project is definitely technically feasible. Along with this, Ant Design, the second most popular user interface library of React, will also be used in the front end [2]. This made it easier to ensure that the team is able to create a good, workable product in the time available.

4.1.2. Operational feasibility

The SECA algorithm was made to 'open the black box' of machine learning, to make sure that developers and regular users would be able to better understand what is causing certain decisions to be made in image identification machine learning algorithms. Users would be able to input their data, and be able to clearly understand what exactly is happening in what parts of the algorithm. The suggested application is the last step in this process, where users will be able to properly understand what is happening where. Assuming the team is able to deliver on the requirements in an adequate fashion, the resulting product is definitely operationally feasible.

4.1.3. Schedule feasibility

The deadline for the final product is the end of the university quarter. One quarter spans 10 weeks, but the final product is expected to be in a suitable state to hand in to the clients in the middle of week 9. The requirements which were given by the client were sorted in terms of priority, in order to determine if the most important requirements could be done in a reasonable time frame. Each member of the

team was expected to work for approximately 40 hours each week. It was expected that this would be enough time to at least cover all of the *"must have"* requirements, and to deliver at least a workable product.

In summary, the team has determined that the project is both technically and operationally feasible, and that it would also be feasible to complete a workable product within the available time.

4.2. Risk analysis

All projects involve some risks. The team is a group of second-year computer science students, and as such are relatively inexperienced with many risks that may arise in projects like this one. This section will analyze the main risks that may be encountered, and some preventive measures that can be taken against them.

4.2.1. Risks related to the dependency on SECA

The project task involves using pre-existing code that was delivered by the clients. While the code was assumed to be fully functioning and without bugs, it was entirely possible that the code which is delivered still has some undetected problems. Along with this, the team was not expected to be able to completely understand the code right away as it is quite complex, and as this task was falling outside the scope of the project. The team had to still make use of several of the methods in the delivered code, and as such they had to have a good enough understanding of the code and how to execute it to actually be able to make use of the correct methods. It could have turned out that the code is too complex for the members in the team. To minimize the effects of these potential risks, or to prevent them entirely, communication had to occur regularly with the clients. Communicating with the clients can reduce the effects of any problems with the algorithm that could occur, and it can also help the team to better understand any segments of code which is particularly confusing.

4.2.2. Risks due to the team's lack of general project experience

The team consists of five second-year bachelor students. None of the members of the team have much developmental experience outside of the university. This comes with the disadvantage of not having a wide range of different experiences, which would lead to a more diverse set of options or opinions on what to do if a problem is encountered. In order to offset this, each member had to perform more thorough research to solve any problems that occur, and not be afraid to ask for help from any of their teammates, the client, or the TU Delft teaching assistants. This lack of experience may also lead to some inter-group issues as each member may not properly know how to resolve issues or how to communicate properly. Again it was emphasised that communication is the best method to solve these issues - either with other group members or with the teaching assistant.

Possible security risks

Creating the best possible product for the client includes ensuring that all the information stored within the application is properly secure. The team are not experienced security experts, and as such there will likely be some security risks, especially if they implement the security themselves. As such, the default Django security was enabled and configured. This is a simple security implementation, but is easily set up and easy to understand. This allows for future developers to easily replace or upgrade the security system if required, while still providing good basic security to the application.

COVID-19 related risks

During the duration of the project, the effects of Covid-19 have been very prominent. The team was not able to meet every day like they would otherwise be able to, and they had to be careful to not get infected by the virus. Standard anti-virus precautions had to be taken: practise social distancing, attempt to keep the amount of people met to a minimum, wear a face mask in public, etc. If a group member were to become infected, they had to isolate themselves and warn any other members of the team that they may have come into personal contact with within a two week time frame.

In accordance with Covid-19 guidelines, the team and clients decided not to meet in person, and should try to keep physical meetings to a minimum. Instead, online meetings were used along with other online

methods of communication. This drastic change in communication can lead to several problems, such as decreased team cohesion and group overview. It can be more difficult to solve problems that any members may encounter, as they must either arrange voice calls with other members, or they must communicate over messages, which are less clear and take more time than simply communicating in person. These issues can be alleviated by planning regular meetings with other group members, and attempting to be available as often as possible to answer questions other members may have. Voice calls should be used often to solve difficult problems, and communication in general must be heavily emphasised to ensure that the project continues smoothly regardless of the dangers and problems introduced by the pandemic.

4.3. Product architectural design

Before starting on the implementation of the application, a design had to be made first of al. This section first introduces the overall product architecture, which is followed up by detailed design explanations for each of the components introduced in the overall architecture.

4.3.1. Overall Product Architecture

Chosen Software Architecture

The server-client architecture was chosen as the primary architecture for the project. This was done as it is easy to understand and allows for relatively simple expansion if any additional features were to be added.

Backend

The back-end part will be a *Django Application*, making up the server which will manage the calculations and send all required information to the user's device. It will act as a REST API for the client, waiting for requests and returning the appropriate responses assuming the user is authenticated to do so. To accomplish this, *Django* security is enabled to ensure users only have access to data they should have themselves and to ensure user privacy within the application.

Database

The database is the general storage for the information that would be vital for the application to work efficiently. *SQLite* is used as the database for the application. *SQLite* is a memory based database system, allowing for easy expansion if the system ever needs more space [1]. This database management system was chosen because it is the default database system that is used by *Django*, and as such *Django* has several in built support systems in place to ensure simple, seamless use.

Frontend

The front-end will be created using *React.js*, an open source front end library used to create user interfaces and UI components [16]. The use of React.js ensured that the front end will be both visually appealing, and functional - being able to make requests to and receive relevant data from the back-end and display it appropriately.

4.3.2. Details on Individual Components

Backend

The backend portion of the code is responsible for hosting a server and managing the connection to the database along with all processes related to it (performing creation, update, delete, and read queries). Because the team is using the Django web-framework, the backend code is also split up into Django Applications. A Django Application "describes a Python package that provides some set of features" [9], simply put: it allows the programmer to split up the entire code into sections based on conceptuality. Not only does this make programming the back-end easier for the developer it also provides additional side-benefits:

- Code is already refactored to some extent.
  Increasing future scalability and making debugging easier
- Increases code cohesion and reduces code coupling.
  Code Cohesion refers to "how the elements of a module belong together" [13], which means

esentially the higher code cohesion, the better. High code cohesion is advantageous as similar code resides together, making debugging easier, and making the code intuitive to the programmer.

Code Coupling, on the other hand, refers to how much a single module/package depends on another. High code coupling is harmful because changes in an individual module or package may break code in the other who depends on it.

Using the Django web framework and its Applications allowed the group to write their code in this separated manner where similar features reside in the same Django Application, resulting in an increase to code cohesion and a reduction of code coupling.

- Allows for reusability and improvement.
  Django Applications are reusable in projects [9]. This means that the client has the possibility of using a single Django Application that the group implemented and if desired also adjust it.

With the above in mind, the back-end should consist of three separate applications: later named SECAAlgo, UserSECA, and Wiki-SECA. The following are the purposes of each of them:

UserSECA:
The main purpose is to implement all the required API endpoints related to Users (as an entity). This included providing features of authentication, logging in, logging out, and registration.

SECAAlgo:
The main purpose of the SECAAlgo application was to implement all the required API endpoints that required the use of the SECA pipeline code the client provided us. For example, the Matrix tab (Interface 3 according to the client's presentation of requirements) requires the back-end to run analysis tools that the client gave the group to extract relevant information that is to be displayed to the client.

Wiki-SECA:
The purpose of the Wiki-SECA application is to have a connection with the Wiki page on the frontend. This application has a direct connection to the database and retrieves or saves information requested by the frontend. It can be used to read, create, delete and update rows in the wiki database.

In the implementation section, the report will further discuss each of these subsections in further detail; specifically how these sections allowed the group to meet the must-have requirements and how the group actually went about implementing them so that the main purpose described above was met.

Database
The database stores all the relevant information that allows the back-end to query it and then apply computation on it, finally returning the output to the front-end. The database schema can be seen in figure 4.1. It was designed based on the data that the client delivered.

Front-End
The front-end part of the product is responsible for communicating with the back-end and displaying that information to the user. Implementing the front-end required thorough research as well before the group could get started. This is because there exist many web front-end libraries for the interface. In the end, the group decided to use React.js to implement the front-end.

React.js allows the group to easily create views for the user, and dynamically update them with content. In fact, one additional advantage of using React. js is that there exist many open source React UI libraries. The group used *Ant Design*, the most used React UI library, because it provides many components, such as dynamic tables, forms for user inputs, or navigation bars, that create a professional look and are easy to modify according to the developers' needs.

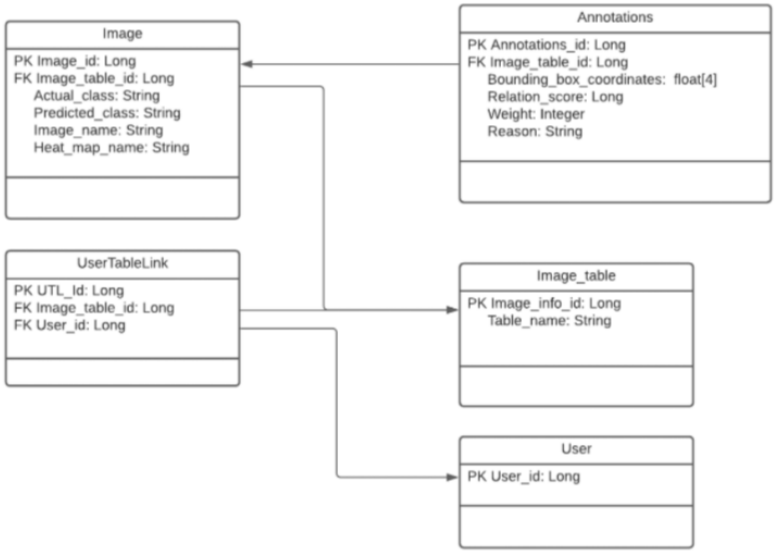


Figure 4.1: The Database schema - The tables represent the different entities and each of the elements inside the tables represent their attributes

As seen in figure 4.2, the Django applications make use of the SECA algorithm code that the client has given, and naturally interact with the database to retrieve image data. That data is used as input for the analysis tools in order to calculate the scores - typicality for the request. The front-end code is responsible to initiate this communication, and in the next section, this is discussed in detail.

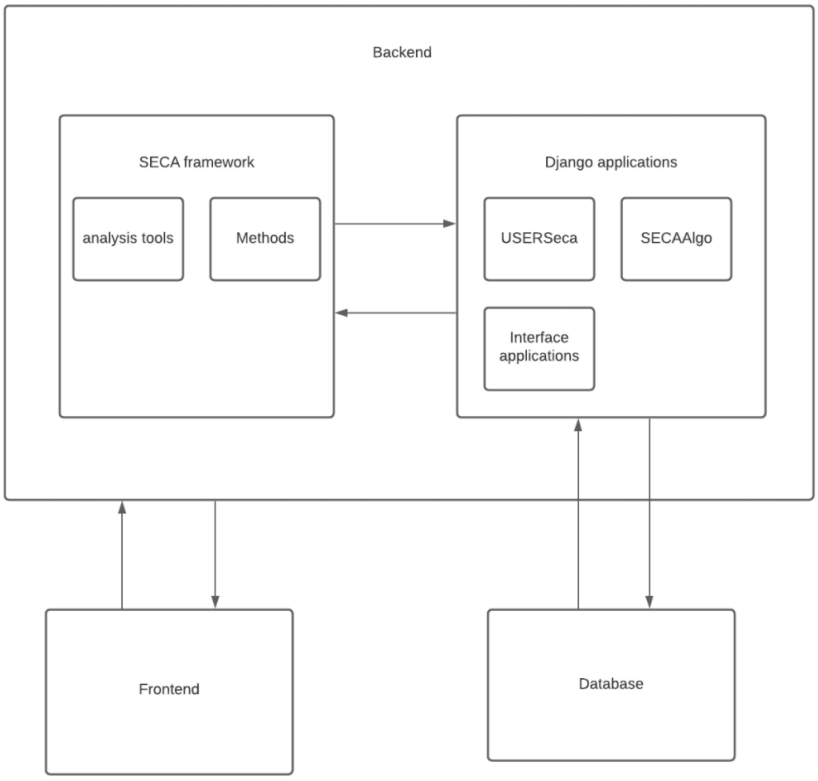


Figure 4.2: Communication diagram. This is a diagram displaying the communication between the front- and backend, and between the Django applications and the database

4.4. Ethical implications

In today's age, it is nearly impossible to go a day without facing technology. It can be found in the search algorithms that can find the best solution to one's problem or in the most trivial situations, such as the algorithm that keeps a fridge at the right temperature. Because of this big role of technology in modern life, it is only logical that engineers must adhere to regulations and ethical responsibilities while developing their project, as they might have a large influence on society in an unfair or unethical way. As software engineers it is not only expected of the team to deliver an application that checks off every requirement, but it is also the group's responsibility that the product does not violate any ethical values. The two main ethical implications which will be covered are the group's responsibility in keeping the privacy of the application's users intact, and on the dangers of bias and discrimination in machine learning algorithms.

4.4.1. Privacy

The Cambridge Dictionary defines privacy as the right to withhold or share personal information with whomever they want [8]. Every person has the right to have their privacy which cannot be violated arbitrarily, as stated in the Universal Declaration of Human Rights, article 12 [8].

Yet a lot of technology uses data that is considered private, for example, addresses and contact information of employees. In these cases, the people have given consent to the developers to use the data only for the intentional goal of the project, which will have to be clearly stated beforehand. It is the developer's responsibility to secure this data in order to respect the right of privacy of the people in question. The product the team is developing will be used to examine classification algorithms and to calculate statistics based on the outcome of the classifications. For this task it is expected from the team to save data that will be used during this process in the database. This might contain classified information, like medical records, making it not only the responsibility of the developer of the classification algorithm, but also their responsibility as the developers of the product. Retrieving the data from anonymous individuals and constructing a secure database by possibly encrypting the data would guarantee the safety of the privacy, as the data would be unable to be linked back to the person to whom it belongs.

As such, the application must have an authentication system to validate the identity of the user before they can make use of the program. This is done to ensure that restricted data that is being used should be secured and only visible to those who are authenticated to view it. Once the user is authenticated, they can add a new classification algorithm along with one or more batches of data. The application will store all the information to the database along with predictions calculated based on the data batches. Then, the user will only have access to the interfaces of the algorithms and data batches which they have the permission to see.

4.4.2. Bias and Discrimination

It said there is bias against an individual or group if they are treated in a different manner compared to others because of one's personal opinion influencing their judgement and treatment towards this individual or group [8]. Since the team will be working with artificial intelligence and classification algorithms, there will be a risk of discrimination based on biases in the training data or implemented by a biased developer. This would also raise some ethical issues in case the algorithm is being used on a day-to-day basis. If used in real applications, bias in machine learning algorithms can have effects which are only present in the training data but not in the real world, potentially influencing the lives of those wrongly affected [5].

Even though the team has no control of what algorithm or data is being used by other users, they will be responsible for any possible ethical concerns applicable to the input of the analytical tool. The application will be able to analyze the classification procedure and, in case there are unexpected results, it will show in the statistics on what they were based, possibly exposing discrimination if it was the cause. If this occurs, the group will still be responsible for any faulty assessment of the algorithm. In case there is a bias present in the analyzed algorithm and the program is used to validate the quality, the bias must be visible in the presented statistics constructed by the application. In case this is not clearly derivable, the developer would be oblivious of any ethical problems concerning their algorithm,

which would make the group partly responsible, since the final responsibility lays on the shoulder of said developer.

With a disclaimer in the terms and conditions section, which the user would have to actively agree to, stating that the group members as the developers would not be reliable for any false assessments of the quality of the classification and any ethical issues following these assessments, the team would, juridically speaking, be absolved from responsibility. However, that would construct the notion that the quality of the application is rather poor and unprofessional. Instead, the team should be sure that the product will find the faults in the algorithms so that we, as the developers, can assure validity and accept any responsibility for wrong statistics.

If the developer of a classification algorithm refuses to accept the biases that are presented to them and proceeds to publish it anyway, the group would have done everything in their power to inform the developer, implicating that the group cannot be responsible for any further ethical problems following the publication of the algorithm in question.

5

Development Process

This section provides a high level overview of how the development process took place. Subsection 5.1 discusses the Software Development methodology the group decided to use. Subsection 5.2 discusses how communication was handled. Subsection 5.3 discusses how the group used *Gitlab* for version control throughout the project. Subsection 5.4 discusses how the group maintained code quality throughout the project, and finally Subsection 5.5 indicates how the group came up with the definition of *done*.

5.1. Software Development Methodology

Scrum

The Scrum software development methodology was chosen for this project, for which there are several reasons. Firstly, due to the short time-span of the project, it required constant communication between the developers, the client and the Teaching Assistant to maintain a global status of it. Often, plans and/or requirements could change, and therefore, it was decided that a less rigid development methodology should be chosen [21]. For this, an Agile design was required to better fit the potentially 'constantly changing' style [6].

Secondly, the Scrum development cycle encapsulates short, repeatable processes - sprints - in which the team works on implementing relatively small features. The development cycle is short, allowing for small, consistent measures of progress [21]. This also means that it is extremely flexible, and is open to change if required. At the beginning of each sprint a meeting was held to decide which issues need to be addressed for the next sprint. At the end of each sprint, another meeting was held to reflect on the sprint, and find points of improvement. Due to this frequent re-evaluating nature, Scrum has the advantage of providing group members the chance to continuously improve on their development method.

Finally, the group needed to give regular updates to the TA assigned to the group, and to the client, which very clearly lends itself to the Scrum development style [6]. It was decided to set the duration of each sprint to one week, so that each week both the TA and the clients can be updated with the work that has been done during the last sprint, and with the plan for the following cycle. The feedback that is given during these meetings is directly applied to the plan and can result in improvement quickly and easily. With other methodologies such as *Waterfall*, most of the project must be planned out from the start, making iterative feedback like this less useful and more difficult to apply [21].

Testing

Because the team used the Scrum development methodology, testing was done in the same cyclic fashion as code. Although the intention was to follow Test Driven Methodology - writing tests as often as possible before code - this idea was quickly dropped, and tests were written towards the end of the project, as explained in detail in Chapter 6.

5.2. Communication

Communication is a vital part of any project when working with multiple people, and as such problems relating to communication can have a very large impact. COVID-19 causes more issues in regards to communication between the group and client, as for the most part everyone should try to stay apart as much as possible. This means that efficient use of online communication is vital to the success of the group.

Communication within the Team
Due to the effects of the Coronavirus, the team was unable to meet in person and work together physically. Naturally, this hinders performance due to distractions in a work-from-home environment. However, the group nonetheless managed high influx of communication via a WhatsApp group and weekly Scrum meetings.

Communication with the Teaching Assistant and Coach
Similarly, due to the Coronavirus, the group was unable to meet in person. Instead, for communication the team used Mattermost, Jitsi, or Zoom.

Communication with the Client
Weekly meetings were held with the client to discuss progress and ask questions if need be. These meetings took place via Microsoft Teams.

5.3. Gitlab for Version Control

GitLab is a platform made to provide developers with the opportunity to improve collaboration and management of their projects. It allows the developer to share their code with their peers and supervisors and to receive feedback by both the platform and other participants. The entire project existed on this platform, making it easy for the group to have an overview of the management, development and general organisation of the project.

Each individual of the group could make an issue concerning their current task. The issue would store information about the task, the deadline, the importance towards the project, the expected time it would take and the people in charge of it. With the features and the many uses of the issue, the developers could efficiently create a branch, originating from the development branch, where they could submit their code. This ensured an efficient organization and work flow.

A branch can be considered as a line through the development of a certain feature of the project. It originates from the development branch which is the main life line of the application. Here resided the most up to date code. The developer would construct the functionalities for the task in question on the respected branch. GitLab initiated a pipeline for every branch where the build and tests of the branch were evaluated for both the back-end and the front-end code in order to maintain consistency and to ensure that no faulty features are introduced in the development branch. Once this pipeline passed, the developer could create a merge request with the goal to insert the branch in to the main development branch. Before this merge took place, the other developers were encouraged to review the recently added functionalities and their integration in to the project. If the merge request was reviewed and approved by at least two peers, the merge took place and the issue was completed.
It was the intention to use scrum, an efficient development method that is used to produce an application in an organized manner. The project would be divided in weekly coding sprints, during which developers would work intensively on a single task. At the end of the sprint, the additions made would be reviewed by the client and supervisors, which allows the developers to promptly receive feedback, which they would take into account in the following sprint.

Using the functionalities from GitLab and with the philosophy of Scrum in mind, the development group build the application in an efficient way with the correct requirements in place within the agreed upon time frame.

5.4. Maintaining Code Quality

Code quality is an extremely important part of ensuring that code will be usable for long periods of time, and ensuring that other developers will be able to actually understand and make changes to the code. It is very easy to neglect this when creating new functionality, and if left alone for too long it can become significantly difficult to actually improve at that point. Developers who created the methods could forget what exactly happened in the code, and not be able to run it properly themselves.

To ensure that code quality remained high, proper documentation was left and constantly updated, and comments were added throughout all methods. Before a commit was made, the developers made sure to add proper documentation to methods that were newly created, and any methods which were changed had their documentation updated. This would include stating the input types for each variable, what they would do, and what each method would return. This made it much easier to quickly understand what each method would do, and made it easier to continue ensuring that the code quality remained high.

The most difficult instance in ensuring that the code quality was high was the first time that the team dedicated to doing this. As relatively little documentation or notation was made before the decision was properly made, those improving documentation had to look through each method, understand what the parameters were, what the method did, and what it returned. This took a surprisingly high amount of time that exceeded the expectations, but was worth it to ensure that going forward the quality would be easily maintained constantly.

Along with clear documentation, functions and classes were named appropriately, so a developer can have an idea of what a function is for even before reading the documentation.

5.5. Weekly iterations and definition of done

New functionality is been added in increments during the weekly sprints. Initially it was hoped that all additions would be merged to master at least weekly and in an at least partially functioning state to prevent too many merge conflicts. However it was possible for new additions to take more than one week to fully implement. After discussion with the TA, the group preferred to also accept having a functioning iteration of the product a few more days after the end of that week. For a new functionality to be added, it had to be worked on until it was considered ‘done’.

As such, a definition of ’done’ had to be made. The initial perception for a new function to be considered ‘done’ was to be properly functioning and have at least 100% method coverage. It should ideally have had 100% branch coverage, but for especially large methods at least 50% branch coverage was also considered sufficient. Nonetheless, in order to present each weekly addition to the clients, the group sometimes skipped on this definition of done, and instead showed their progress without sufficient testing. The testing would need to come after the product was displayed, even though it should have been done before, in a stable state.

6

Product Implementation

6.1. Implementing the back-end

As stated in subsection 4.3.2 the back-end was split up into three main Django Applications. Here the report will now discuss details on these. The following subsections will discuss each application in detail; describing what features it brings forth for the user (and therein for the Front-End), and also how the group actually attempted to implemented it.

6.1.1. The 'UserSECA' Application

Features it provides with respect to Requirements

With respect to the must have requirements stated in Subsection 3.3.2, the SECAAlgo Application allows the following requirement to be met:

- *Must have requirement 1 - The system must have a login and registration feature for users*
  The UserSECA application accomplishes this by extending the existing User authentication system in Django. As Django already has an existing User authentication system that is secure and heavily tested (along with very useful password safety features), writing a new and original authentication system is tedious and unsafe for the client. However, the team couldn't directly use Django's implementation either because the needed User entity is more verbose. Therefore, the implementation uses Django's implementation as a base and adds to it. Moreover, the group implemented authentication by using Token Authentication (generating a unique token for each user). This Token Generation is handled by the Django-Rest-Framework, for reasons similar to above; the existing framework is tested and has widespread use.

Implementation

This was the first Django Application that the group implemented, and therefore it posed many challenges.

Firstly, the group had to decide how they were going to set up the authentication, which system they were going to use. There exist many authentication means, for example: Token Authentication, Session Authentication, Remote User Authentication, OAuth, JWT Tokens, and many more. The team first had to do research in each of these and determine which specific one would be implementable in the limited time the group has, and of-course suit the architecture. In the end, the group decided to use Token Authentication. Firstly because Django-Rest-Framework has inbuilt support for it, and secondly on the front-end it means that for each request made to the back-end it only needs to append this token, rather than re-authenticating the user for each request made and adding overhead.

Secondly, the group had to get familiar with many new concepts such as Serializers, Viewsets, and Models; specifically how each of these were connected to each other. The team learnt these concepts in detail by watching online tutorials, reading official documentation, and finally through trial and error after implementing it themselves.

In the end, the many hours spent properly learning the new frameworks and their concepts paid off. The current authentication system is stable and working as intended, and is written smartly with concise code due to the proper use of the available frameworks.

6.1.2. SECAAlgo Application

Features it provides with respect to Requirements

With respect to the must have requirements stated in Section 5.3, the SECAAlgo Application allows these requirements to be met:

- *Must have requirement 4 - The system must allow a user to view a confusion matrix based on their data in an interactive way*
  SECAlgo Application exposes endpoints that query the database to return data that allows the Front-End to create the confusion matrix itself. Moreover, there also exists endpoints that have integrated the analysis tools the client provided return data about a specific cell of the confusion matrix, specifically the typicality scores and the concepts (4b)
- *Must have requirement 5 - Users can view the different attributes per class that SECA has selected via the Explore tab*
  SECAlgo Application exposes endpoints that run the analysis tools but now for a given specific set of image classification classes. This in turn allows users to retrieve information (typicality scores and concepts) about all images or just correct/incorrect predictions (5a)
- *Must have requirement 6 - Users can query different concept (or combination of two concepts) via the Query tab*
  SECAlgo Application exposes endpoints that after running the integrated SECA analysis tools for a given specific set of image classification classes (6b) then apply additional filtering (6c). This filtering is based on the users input given on the Front-End. The user has the flexibility to exclude/include specific classes and concepts that are returned.

Implementation of the SECAAlgo application

This application was, in hindsight, the most challenging portion of the back-end the group worked on. This was because this was the most complex part of the back-end (as it dealt with integrating the given SECA methods), and required immense communication within the group and with the client to properly understand what needed to be done and how.

Firstly, the team had to get a deep understanding of the SECA algorithm and the code the client provided. This was crucial as the group would need to use it to display data to the user. The client had given the group their master code repository which also contained data and a file that ran the SECA algorithm. During week 2-3 the group started looking at the code and attempted to understand it. The team initially tried to port the entire SECA algorithm into the back-end by making use of virtual environments. However, this was not successful, and after discussing this with the client the group learnt this was not the intended use. The client told the group that they only needed the final two steps of the entire SECA algorithm pipeline for the interface. Towards the end of week 3 the team was able to run the final two steps of the SECA algorithm, and they showed this to the client. The client then made aware that the group was using the data that they found in the master repository, and that they needed to actually use the data they had given to the group separately, but in-order to use that data, they would need to correct its formatting (so that it could be inputted into their code).

Throughout week 4 the group faced many challenges related to converting the given input into the correct format and running the last two steps of the SECA algorithm (the analysis tools). Writing the conversion patch code did not have complications and worked smoothly. The hard part was being able to run the analysis tools to extract information about the data, specifically for the analysis tool called rule mining. When the group first ran it, it took over 3 hours to run, and ended with a memory overflow error, as it produced results worth 4 GB. Clearly the team was doing something wrong. Both the group and the client worked together and figured out that this was occurring because of the three main reasons: the data sample size was too little, the data was sparse, and the parameters the team were giving to rule mining resulting in it finding too many results (the group needed to find the sweet spot for the parameters that gave just the right amount). Fortunately, with trial and error the group finally managed

to find the correct parameters. Despite the tedious nature of trying to get the clients code integrated in the back-end the group are very proud of their efforts. The group had effective communication with the client, and the client also appreciated the group finding caveats in their code; the team opened the client's code as a black box.

In week 5 after integrating the SECA algorithm into the back-end, the team needed to make proper use of its outputs. After thorough discussions with the client (via both emails and meetings) the group made two discoveries. First the group learned how to use the output of the rule mining analysis tool to return desired information to the user, and more importantly they learned that they needed the ability to run the rule mining analysis tool based on user input for different parameters (and therefore execute the analysis tools for each unique user request). In other words, the group needed to refactor the current client code integration into another separate dynamic function; which would operate based on user inputs.

Throughout weeks 6 and 7 the group continuously developed this specific function that integrates the clients code, and runs on various different user inputs, behaving differently for each type and returning different outputs. Finishing this piece of complex code allowed them to finish the must have requirements stated earlier.

As developers, the group members are very proud of the learning curve they faced, and how they tackled it. The group took the time to truly gain a deep understanding of the clients code, which therein allowed them to efficiently integrate it into the group's back-end code. Of-course if wanted, the group could have taken the easy way out, and made many functions, one for each different user input type required. This would have led to code duplication, and made the code more confusing for the client and future developers. Indeed, the problems the group encountered caused certain aspects of the product to be completed slightly later than usual, and the problems with communication resulted in some parts of the product being slightly mismatched with others, and further work had to be done to properly match each other. This disorganisation happened due to the lack of effective communication between the group at times during the project. For instance, when someone in the group needed help with understanding the SECA methods, it was much more difficult to effectively help that person as members found it more difficult to properly communicate online, rather than in person. When working together in person, it is possible to point out certain segments of code that could be problematic, and it is much easier to communicate instantly rather than via messaging. The group quickly adapted and some users began to communicate more via voice chat and using programs that allowed users to share their screens - for example using Discord. When this was done, the group was able to deal with integrating the SECA methods into the back-end mush faster.

6.1.3. The Wiki SECA Application

Features it provides with respect to Requirements

With respect to the must have requirements stated in Section 5.3 and 5.5, the Wiki-SECA Application allows these requirements to be met:

- *Must have requirement 3.c) - "Wiki" page. this page contains static information about data*
- *Could have requirement 3 - Developer users and Expert users could have the ability to comment, ask questions and answer questions in the Wiki interface*

Implementation Process

The application first started with just a retrieval endpoint, where a request with a problem ID gets sent from the frontend and the Wiki-SECA application returns the corresponding data. The next step was to allow for editing of the database from the frontend, so the create and update endpoint was added. This endpoint either updates an existing database row or adds a new one. Finally an option to delete rows was added as well, which simply removes the requested row from the database.

The application was implemented with no issues, as it was a fairly simple and straightforward task. The only challenge was storing and sending images from backend to frontend. After research, it was decided that the images would be stored in a folder on the backend and their names would be stored in

the database. The images' base 64 encoding then gets sent in a JSON format to the frontend, where it is decoded and displayed. A similar thing was also done for importing images from frontend to backend.

6.2. Implementation of the Front-End

In the following subsections, the report will depict each of the main components of the front-end, along with implementation process details for each one. The report starts with introducing the static interfaces the group developed, and then it will individually discuss the dynamic ones.

6.2.1. Static Interfaces - About

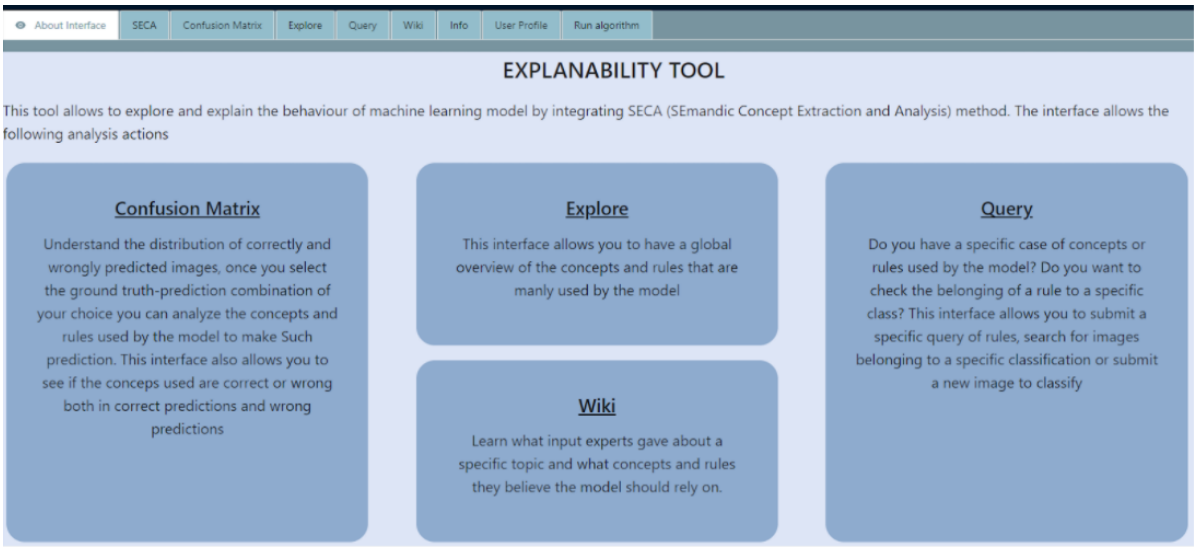


Figure 6.1: Static about interface - the main page that a user will see when first opening the application

For a comparison between the client specified and final implementation of the about interfaces, refer to Appendix F

The group was tasked with creating a couple algorithm-independent, static interfaces. The first two interfaces to be implemented were the About interfaces, one with an explanation of the application and one with an overview of the algorithm used and the possible metrics it can show. The client provided the group with a design for these pages and it was the group's job to implement them accordingly. This was in itself not a difficult task, however this was the first time the group has used the frameworks that they decided to use for the project, so these tasks took longer than expected, as there was a learning stage at the beginning. However, in the end the two interfaces met the client's expectations.

6.2.2. Static Interfaces - Wiki

For a comparison between the client specified and final implementation of the wiki interfaces, refer to Appendix G

The Wiki interface can be seen in Figure 6.2. The interface is not fully static, however it started with a static version and then options to upload own data and ask questions were added. The Wiki interface contains three subinterfaces, as shown in Figure 6.2. The first interface is an overview of the inputted data in a table format. That was also the first part the group tackled, as it is simply a static view. This view was later connected to the database and now it is fully loaded from the backend and database.

The second subinterface is the Edit page, where an expert can edit the given information. The developer can input their questions in this interface for the expert and the expert can leave a response or

edit the page based on the questions.

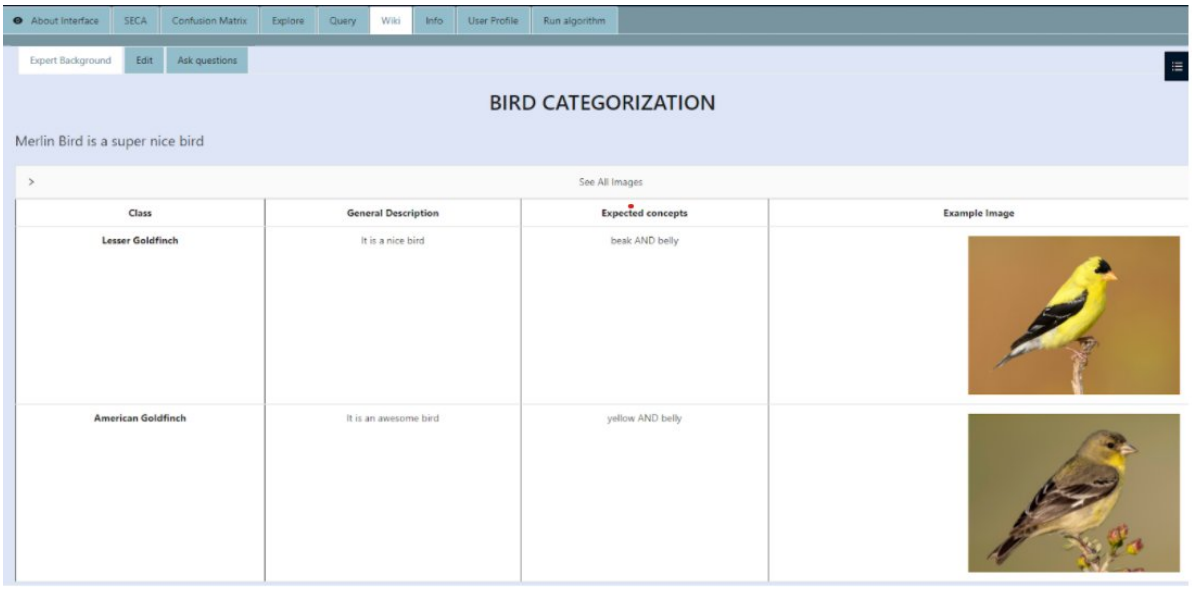


Figure 6.2: The wiki interface - the page that will be filled with expert information for others to read and learn from

The subinterface allows a user to add, remove or edit rows of the table in real time. The most difficult part of this was actually editing the frontend to update as a user clicks something. The database also gets updated immediately after a user saves their changes. The third subinterface is an interface for developers to ask questions and experts to answer them. The interface looks different based on whether the logged in user is an expert or a developer. Specifically, developers can view questions they have asked and the added answers. Developers also see an input field to ask questions. An expert user can also view the questions and they can also see an input field for each question, giving them the ability to answer it. The experts do not have the ability to ask questions. Both user types are able to delete any row from the questions interface.

6.2.3. Matrix View

For a comparison between the client specified and final implementation of the Matrix interfaces, refer to Appendix E

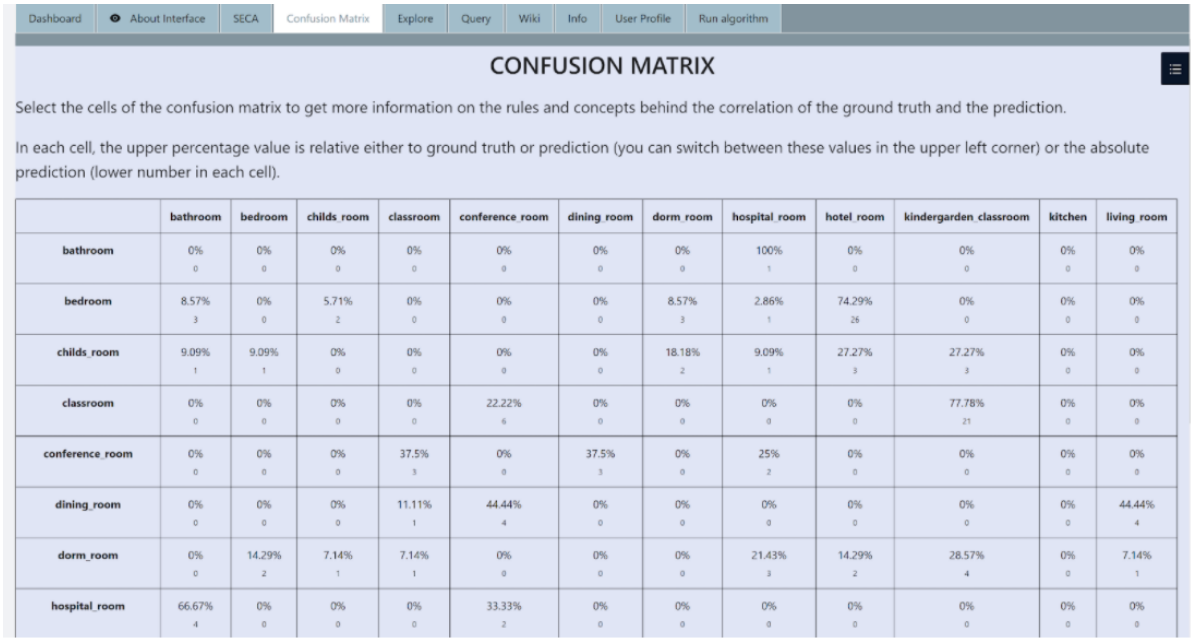
The group has spent a considerable amount of time designing the percentage bars. The width of the bars depends on percentages that were requested to and received from the backend.

For each cell in the confusion matrix, a set of images are presented that are classified in one of four categories. One of the obstacles was how to send the images over the network from the backend to the frontend. The solution was to encode the data of the image with base64 so it can be sent as a string after which it would be decoded at the frontend so the image can be properly displayed.

A relevant challenge was regarding exception handling. When moving from the matrix interface to the one displaying statistics, errors were blocking the process. This required more time and more developers trying to finalise this functionality. In fact, the issue behind the problem was that certain classes had no images or scores associated with them after running the SECA method.

6.2.4. Explore View

For a comparison between the client specified and final implementation of the explore interfaces, refer to Appendix H



	bathroom	bedroom	childs_room	classroom	conference_room	dining_room	dorm_room	hospital_room	hotel_room	kindergarden_classroom	kitchen	living_room
bathroom	0% 0	0% 0	0% 0	0% 0	0% 0	0% 0	0% 0	100% 1	0% 0	0% 0	0% 0	0% 0
bedroom	8.57% 3	0% 0	5.71% 2	0% 0	0% 0	0% 0	8.57% 3	2.86% 1	74.29% 26	0% 0	0% 0	0% 0
childs_room	9.09% 1	9.09% 1	0% 0	0% 0	0% 0	0% 0	18.18% 2	9.09% 1	27.27% 3	27.27% 3	0% 0	0% 0
classroom	0% 0	0% 0	0% 0	0% 0	22.22% 6	0% 0	0% 0	0% 0	0% 0	77.78% 21	0% 0	0% 0
conference_room	0% 0	0% 0	0% 0	37.5% 3	0% 0	37.5% 3	0% 0	25% 2	0% 0	0% 0	0% 0	0% 0
dining_room	0% 0	0% 0	0% 0	11.11% 1	44.44% 4	0% 0	0% 0	0% 0	0% 0	0% 0	0% 0	44.44% 4
dorm_room	0% 0	14.29% 2	7.14% 1	7.14% 1	0% 0	0% 0	0% 0	21.43% 3	14.29% 2	28.57% 4	0% 0	7.14% 1
hospital_room	66.67% 4	0% 0	0% 0	0% 0	33.33% 2	0% 0	0% 0	0% 0	0% 0	0% 0	0% 0	0% 0

Figure 6.3: The confusion matrix - allowing users to easily see what predictions the algorithm made for each image, and what the image really is

For the class specific explanations the backend calculates the rules for specific classes and presents them with a typicality score, the percentage of concept-associated images among images with the predicted class or the percentage of correctly classified images within rule-associated images.

One of the functionalities that the team wanted to implement was that the table would change dynamically when changing the filter values without needing to push an extra button or to reload the page. The team achieved this by the UseState and UseEffect methods provided by the React framework. With UseEffect the web application could run a method that constructed the table whenever one of the parameters changed value.

6.2.5. Query View

For a comparison between the client specified and final implementation of the query interfaces, refer to Appendix J

The query interface contains two subinterfaces. The first one is shown in Figure 6.5, and it allows a user to select rules based on which they want to filter the classes of their project. The different filters allow a user to pick concepts present in images, the actual class of the image and whether they want to view correct predictions or incorrect ones. They can also of course choose to view all predictions, correct and incorrect. They can choose to view the data as a confusion matrix or as a list of typicality scores, which look similar to the view of the confusion matrix interface and subinterfaces.

The second subinterface of the query interface is a tab where a user can query for specific images. A user can select similar filters as on the previous interface and additionally they can also query based on the predicted class, which allows them to see why specific images were predicted wrong or what these images have in common.

One of the issues with implementing this page was gathering all the filters in a logical way and then correctly sending them in a backend query, in order to always get a correct result from any arbitrary input. Once that was done, the results of the query had to be displayed in a nice way, which was something that the group tackled well.

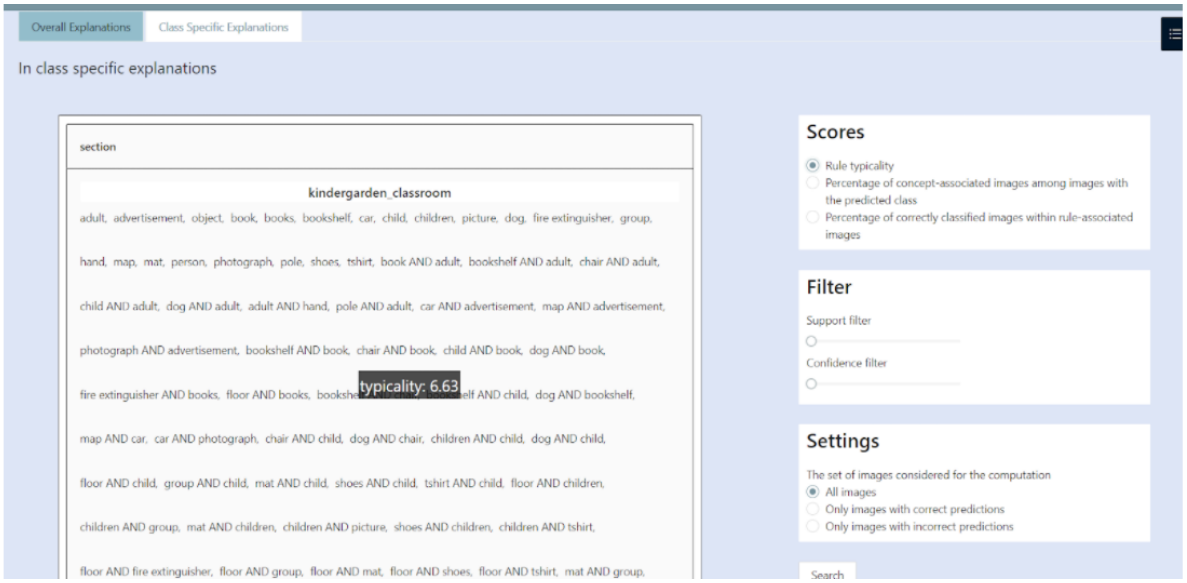


Figure 6.4: The explore interface - An interface which allows the user to look through and find annotations for all instances in the session.

Figure 6.5: The query interface - the interface which allows users to query for specific concepts and will return a filtered version of the confusion matrix, showing only those concepts, rules, not those rules, classes, etc

6.2.6. User Profile View

The user profile view allows the user to log out, and see some basic information about themselves. Implementing this part of the front-end didn't raise any challenges as it was very basic and only required a few interactions with the back-end.

Connecting Front-End and Back-End:

The front-end communication with the back-end is simple and intuitive. After the user triggers some action (such as a button click) the front-end code makes a HTTP request to one of the known exposed endpoints. Once the request reaches the endpoint, the back-end proceeds to compute the desired results (with means as explained in the Back-End section) and finally returns it. This is then displayed to the user with means as explained in the Front-End section. In terms of implementation, this connection didn't take long and hardly any hardships were encountered.

Testing - Verifying the implementation

Testing the code is an important part of implementation. Testing has the obvious advantage of allowing the group to make certain that the implemented code is indeed behaving as intended. However, there

are other additional advantages. Firstly, working in a group inevitably leads to clashes and changes in code, having tests allows the group to detect if the changes have altered the behaviour of the written code. This makes the implementation process as a group a lot more smoother, and completely negates the possibility of losing work done by others or modifying their indented behaviour. Secondly, writing tests made the debugging process smoother. Failing tests allowed the group to catch bugs early and solve them. Finally, tests serve as a validation criterion for the client. Having a complete test suite gives the client confidence on the product, and also provides them with example use cases of the product (as the tests mimic the application as if a user was using it).

Writing tests themselves was a tedious process. Firstly, tests weren't written for the Front-End code of the product. This is because these tests would ensure that the Front-End is connected with the backend and is displaying the intended content. However, this can easily be verified visually rather than writing complicated simulation test cases. The Back-End was tested as extensively as possible (there are exceptions; stated in Appendix D). The group has tested the product with both Unit tests and Integration tests.

Unit tests refer to testing individual modules (like singular functions or pieces of code), whilst treating them excluded from the rest of the code [17]. These tests allowed the group to test all of the small pieces of functions/methods that the API endpoints used to retrieve data for the client. In other words, the Unit tests validated the returned content of the back-end code, and verified if the output matched the intended behaviour.

Integration tests refer to testing multiple components of the application to verify if they work together as intended [17]. These tests are in essence mocking the behaviour of a client interacting with the front-end and triggering the requests to the back-end. The team wrote these tests using Django's test suite, and therefore, the group members were able to verify that if a client makes a request to the back-end, then it will be listened to and there will be an appropriate response to it.

7

Product discussion

With the final product completed, an evaluation of how the final product turned out can be done, along with what can be done to improve it in the future. This can be done by evaluating the requirements, and on the group's thoughts on the quality of certain elements. This chapter will cover an evaluation on how well the group completed the requirements, an evaluation of the final front end result, an evaluation of how the back end turned out, the impacts of all issues which were faced during the implementation stage, and recommendations to the clients on how they could carry on and improve the final product.

7.1. Evaluation on requirements done

Table 7.1: Table evaluating how complete the requirements have been completed. Y means the requirement is completely done, while X means the requirement has not been done. The numbers correspond to the requirements in chapter 3.3. Unfortunately a description of the requirements themselves could not be added as that caused the table to be far too large and was unable to fit in the page of this report.

Must have		Should have		Could have	
1	Y	1.a	Y	1	Y
2	Y	1.b	Y	2.a	X
3.a	Y	2	Y	2.b	X
3.b	Y	2.a	Y	2.c	X
3.c	Y	2.b	Y	2.d	X
4	Y	3	X	3	X
4.a	Y	4	Y		
4.b	Y				
5	Y				
5.a	Y				
6	Y				
6.a	Y				
6.b	Y				
6.c	Y				

Prioritising certain requirements over others is very important when deciding which tasks should be done. In the final product, the completed requirements can be seen in table 7.1. The 'must have' requirements have all been met, which means that the team has met the minimum requirements to consider the project successful - at least from a requirements point of view. The product meets all the minimum requirements, and nearly all of the 'Should have' requirements, but it does not meet many of the 'Could have' requirements.

Every single one of the must have requirements have been met, and in a convincing fashion. All of the requirements have been tested and function properly. The team is confident that the client will be satisfied with the minimum product which has been delivered. The team has also expanded slightly

upon the requirements, such as by applying basic security to the application on top of the login and registration requirement, and expanding on the querying requirement, allowing users to query for more than was initially specified.

5 out of 6 of the 'Should have' requirements have been met. This is a satisfactory result for the team, but they would have preferred if the final should have requirement would have been finished. Nonetheless, the client is also content with the final product.

Only one of the 'Could have' requirements were met, but even then it was very simply done. The requirement specifies for two types of users, which is done, but currently there is no real distinction between them. This is simply present so future developers can more easily implement the different functionality between the user types. Overall, the team would have preferred if they were able to do more in terms of the 'Could have' requirements, but are not too distressed over it, as there were higher priority tasks which were done instead.

7.2. Evaluation of the front end

Early on in development, the client provided models of how they envisioned each interface to look. The models gave an overview on what each interface would do, how it would look, and what should be in each interface. These models gave a good overview of how the interfaces are structured and what is expected to be present in each one, making it easy to implement the design of the webpages. Overall, the team is quite pleased with how the general design of the front end ended up, and think that it matches up with the models quite well and is properly functional.

7.3. Evaluation of the back end

When creating the back end, there were focuses on quality of integration and code longevity. In the backend, average users will not see what exactly is going on, the specific design is less important, but a clear clean repository is incredibly important for future maintenance. Proper integration of the SECA backend is incredibly important, both with how new code will work with it, and in making it clear what does what exactly.

7.3.1. SECA integration

Overall, the integration of the SECA backend went smoothly. In order to more easily allow for the creators of the SECA algorithm to identify work they have done, the parts of the algorithm that they have done is separated from the rest of the repository. It is left unchanged and in its own folder, allowing for them to properly change anything that they wish, without being unsure if the team has made any changes to it. While the team found it difficult to understand what was happening in the algorithm initially, good communication with the client led to a much better understanding of the specifics of the algorithm.

The communication with the client led to several robust methods which interacted with their algorithm. A large streamlined method was created which allowed for the 'rule mining pipeline' to be executed, a method which simplifies the entire rule mining process. Developers are now able to easily execute rule mining with several additional parameters to easily execute the algorithm. While there is more to the SECA algorithm, this pipeline encompasses most of the important information that is required for the front end. Integrating the algorithm into the repository was done quite effectively and efficiently, making use of only what was required and executing it in a simplified manner that allows future developers to easily alter all the parameters. To further aid the pipeline, a query and filtration method was integrated to allow the front end to communicate with this method. All the new methods were thoroughly tested and made to allow for expansion and future use, along with ensuring that they returned information in a clear format.

Not every user will have everything required to run the SECA algorithm already downloaded, and as such there have been preparations to allow users to set up a virtual environment with all the dependencies installed. If more of the SECA algorithm would be integrated, the dependencies required could easily be updated and users could very easily update their virtual environment.

7.3.2. Ensuring longevity of the back end

While delivering a workable and successful product is important, it is also extremely important to allow the final product to be expanded upon. Codebases are often extremely difficult to understand for newer developers. If the clients want to continue work and base it off of this project, they would need to understand what the team has done and to be able to make use of methods that have been implemented. In order to make it as easy as possible for developers who have not seen the code to integrate themselves into the project, the file layout and the code itself must be as clear as possible. In order to do this, proper documentation and layouts must be made.

The layout of the repository was made specifically to be simple to understand. Instead of keeping everything in large vaguely named folders, the directories are placed within each other to allow for simpler navigation if the project continues to grow. Each of the Django applications are in separate folders - one for specialised users, one for the rule mining, and one for information regarding the wiki page. Along with this, the files which were imported from the SECA algorithm are in its own directory. Along with directories, classes and files are made to be a certain maximum size. Good design requires a clear structure and layout, and it should be simple to find methods/classes [12]. As such, even if methods communicate with each other, they are often set in different files to ensure that everything is clear, and to prevent clutter. An example is the rule mining pipeline, initially when the files were created the rule mining pipeline and query method were put together. To keep good clarity, the pipeline and queries were separated into their own files.

Repository layout is not the only important requirement for project longevity, proper documentation is just as important. Documentation makes it much easier for new developers to work with old methods, instead of trying to understand everything by reading the code itself, they would be able to read what the inputs should be, what the outputs are, and what the method itself does. This is especially important in large methods, where it becomes near impossible to understand what happens without any previous knowledge, without proper documentation. The team has made sure to add comments and documentation throughout all the new methods that have been created, and hope that any future developers will be able to easily understand what is happening in it.

7.4. Impacts of issues faced on the final product

Despite being able to communicate via voice chat for very difficult problems, the majority of general communication through the group was done through WhatsApp messaging. This meant that at times, there was a lack of overview of who was doing what tasks, and certain things were not being done. This issue lessened as time went on, but still persisted throughout the entire project. When a member realised that something that should have been done was not being worked on, someone would quickly go work on it. Usually these were relatively small issues, but with better communication they could have been avoided altogether. COVID likely played a part in this, as if all members of the group were all together at once problems like this would be caught more quickly, and would likely be addressed immediately. The cause of these issues changed often, but could be systematically removed if the group had more experience and discipline when working with the issues in Gitlab. Apart from this, there were no real problems with communicating with the client.

Overall, the issues regarding communication likely had more impact on the product than any issues regarding implementation. While the implementation problems caused certain aspects of the product to be completed slightly later than usual (as mentioned subsections of chapter 6), the problems with communication resulted in some parts of the product being slightly mismatched with others, and further work had to be done to properly match each other. This cost significantly more time than implementation problems, and resulted in working sections of the code being changed to a potentially less robust state due to the speed in which they had to be altered. As time went on, these issues lessened and work was able to proceed more smoothly, but sometimes members were left waiting for other members to need to change some parts of their code to properly fit what was needed from them. All the issues were addressed reasonably well, and the end result still meets all ‘must have’ requirements, but with better organization the final product could have been further improved.

7.5. Potential improvements and future recommendations

While the team is happy with the product, there are still several improvements which could be made.

Aesthetics
In the front end, improvements could always be done to the aesthetics of the interfaces. While they are overall quite good, certain aspects can be improved to better match the initial specifications - for example by aligning the rules displayed in the confusion matrices to the concepts. This can be seen in Appendix E, where the concept *'computer AND product'* would appear on the left and the right would only have rules including *'computer AND product'*, instead of whatever rule just happens to be at the same level. Along with matching initial specifications, the aesthetics of a product can heavily influence user satisfaction, so even a small improvement can result in a significant improvement in user satisfaction [18].

Updating existing sets of predictions
In the back end, improvements could be made through expansions of the existing functionality. Currently users are only able to add entire sets of predictions to the database at any given moment, instead of adding to previous sets. Changes could be made to properly allow for incremental additions to the database, and to give users more control over sets of images. This can also be done for the users allowed to access those prediction sets, as currently there are no methods which allow users to change who is able to see those existing sets.

Updating security
If the application becomes more widely used, or if it contains more personal data for users, improving security should be considered. The security of the application is quite basic, only being Django basic security, and as such malicious users with knowledge of the Django framework and its security's shortcomings could potentially cause safety breaches.

Implement the uncompleted requirements
The rest of the 'Should have' and 'Could have' requirements could be added to the product. The team attempted to meet as many requirements as possible in the time that was available, but unfortunately were not able to complete all the optional requirements that were given. A very simple way to expand upon the application would be to complete the requirements which were not completed.

Allow for public or private prediction sets
Currently, the only way for a user to be able to see another user's set of predictions, is for another user to add them to the list of users who can have access to it on the creation of the set. Instead, a 'marketplace' could be added to allow experts to look through all the prediction sets that have been set to being public, and make comments or annotations on those predictions. This will make it significantly easier to get additional information, similarly to how Stack Overflow deals with questions.

A way to see the bounding boxes from each annotation on the images
Bounding box coordinates are saved in the database for each annotation, but are currently unused. It would be interesting to be able to view individual images, and to be able to toggle the bounding boxes and where the annotations were placed. This would allow for better analysis on what exactly is causing certain images to be predicted in what ways, as users would be able to more easily see where the annotations are placed, and if there are any mistakes with this placement.

8

Conclusion

The aim of this report was to describe the development process of the SECA Interface. This Interface is a layer of abstraction over the SECA method; it integrates the SECA method, and provides an easy to use user interface for it. Thereby facilitates the means for practitioners having easy access to explanations on image classification neural networks. In other words, allowing software developers and domain specific experts to collaborate together and learn why these neural network models were making the decisions they made; essentially, opening this ”black-box”. Having insight of the biases of which these neural network models make their decisions allows developers to catch biases earlier on to prevent future discrimination.

Research was done prior to the implementation of the SECA Interface, and it was concluded that the group would need to follow several application design principles for the success of the project. With this in mind the project goals in Subsection 2.3 were established.

The group made use of the MoSCoW method to prioritise the set of requirements that were attained from performing requirement engineering and analysis. The Use-Case analysis was crucial to gaining deeper insight into how the SECA Interface would be used from the perspective of the different user roles; the group especially understood the importance of security and authentication. At the end of the project, in accordance to the MoSCoW prioritisation all of the must-have and should-have requirements were implemented, as well as most of the could-have requirements. The client had made a change in the requirements in Week 6 of the project, and these were accounted for.

At the end of the project the SECA Interface had the required operational back-end and front-end components. The back-end was implemented using Django, and had integrated the required SECA method components into a dynamic user responsive pipeline. This was perhaps the most challenging technical aspect the group faced, as not only did the group need to spend time understanding the SECA method, but also then integrate its code and implement patch code to support that integration. Many communication issues were faced as-well, which contributed to the large portion of time spent in the back-end, however, these were tackled by the group and solved smoothly. As per the request of the client the back-end was made as maintainable as possible: with complete documentation, and support for flexibility in the code. Thereby, the first project goal was met.

The SECA Interface front-end was implemented using React.js, and had implemented all the required views matching the clients given designs. Moreover, the front-end made proper use of the back-end, and exposed the full intended functionality of the SECA methods. Similar to the back-end, the front-end was also fully documented and was implemented in a maintainable fashion to support any future development the client has in mind. The front-end provided a tutorial that explained how to use the interface, and also provided explanations at individual pages to help the user. Therefore, the second and third project goal was also met, resulting in all goals being completed.

In the end, the team believes that the delivered product will be able to be used in the future by the client, the team hopes that the future users will find it useful.

Bibliography

[1] G. Allen and M. Owens. *The Definitive Guide to SQLite*. 2010.

[2] *Ant Design - A design system for enterprise-level products*. 2020 - 2021. URL: https://ant.design/.

[3] A. Balayn, C. Lofi, and J. Yang. *What do You Mean? Interpreting Image Classification with Crowdsourced Concept Extraction and Analysis*. Delft University of Technology, 2021. DOI: 10.1145/3442381.3450069.

[4] *Brightspace*. URL: https://brightspace.tudelft.nl/d2l/home.

[5] A. Caliskan and R. Steed. "A Set of Distinct Facial Traits Learned by Machines Is Not Predictive of Appearance Bias in the Wild". In: *AI Ethics* (2021). DOI: 10.1007/s43681-020-00035-y.

[6] T. Dingsøyr, S. Nerur, and V. G. Balijepally. "A decade of agile methodologies: Towards explaining agile software development". In: *Journal of Systems and Software* 85.6 (2012), pp. 1213–1221. DOI: 10.1016/j.jss.2012.02.033.

[7] *Django - The web framework for perfectionists with deadlines*. 2005 - 2021. URL: https://www.djangoproject.com/.

[8] S. Flaxman and B. Goodman. "European Union Regulations on Algorithmic Decision-Making and a "Right to Explanation"". In: *AI Magazine* 38.3 (2017). DOI: 10.1609/aimag.v38i3.2741.

[9] Django Software Foundation. *Applications*. 2005 - 2021. URL: https://docs.djangoproject.com/en/3.2/ref/applications/.

[10] *Functional requirement*. 2021. URL: https://en.wikipedia.org/wiki/Functional_requirement.

[11] S. Hatton. *Choosing the "Right" Prioritisation Method*. The University of Western Australia, 2008. DOI: 10.1109/ASWEC.2008.4483241.

[12] *Introduction to Repository Design Pattern*. 2021. URL: https://cubettech.com/resources/blog/introduction-to-repository-design-pattern/.

[13] I. Montiel. *Low Coupling, High Cohesion*. 2018. URL: https://medium.com/clarityhub/low-coupling-high-cohesion-3610e35ac4a6.

[14] *Nonfunctional Requirements*. 2021. URL: https://www.scaledagileframework.com/nonfunctional-requirements/.

[15] *Please open this AI black-box! - User interfaces for supporting debugging and interpreting deep learning-based computer vision models*. 2021. URL: https://projectforum.tudelft.nl/course_editions/35/projects/1059.

[16] *React - A javascript library for building interfaces*. 2013 - 2021. URL: https://reactjs.org/.

[17] A. Shaw. *Getting Started With Testing In Python*. 2021. URL: https://realpython.com/python-testing/#writing-integration-tests..

[18] A. Shukla, N. Sharma, and Swami S. "Website characteristics, user characteristics and purchase intention: Mediating role of website satisfaction". In: *International Journal of Internet Marketing and Advertising* 6.2 (2010), pp. 289–306. DOI: 10.1504/IJIMA.2010.032479.

[19] *Squarespace*. 2021. URL: https://www.squarespace.com/templates.

[20] *Use-Case Analysis*. 2020. URL: https://en.wikipedia.org/wiki/Use-case_analysis.

[21] W. Van Casteren. *The Waterfall Model and the Agile Methodologies : A comparison by project characteristics*. 2017. DOI: 10.13140/RG.2.2.10021.50403.

[22] E. Vargas, S. van den Oever, and A. Panichella. *TU Delft, Design Patterns*. 2021.

[23] E. Vargas, S. van den Oever, and A. Panichella. *TU Delft, Requirements Engineering*. 2021.

[24] E. Vargas, S. van den Oever, and A. Panichella. *TU Delft, Software Architecture*. 2021.

Appendix A: Division of labour

Table 1: Table displaying the division of work within the group. This simply displays who worked on what, and is not necessarily representative of the work done in the project as the size of each task may be different.

Task	**Primary contributors**
Gitlab Pipeline setup	Floris van Veen
Setup authentication + security	Kanish Dwivedi
Database design	Mihai Filimon, Mathieu Butenaerts
Add tabs interface	Silvia Mokranova
Setup default front end task bar	Silvia Mokranova, Mihai Filimon, Mathieu Butenaerts
Set up virtual environment	Floris van Veen, Kanish Dwivedi
Connect SECA framework to repository	Floris van Veen, Kanish Dwivedi
Set up repository	Mihai Filimon
Create static interfaces (About SECA, WIki page)	Silvia Mokranova
Setup 'ask expert' interface	Silvia Mokranova
Automate rule mining from the SECA algorithms	Kanish Dwivedi, Floris van Veen
Allow for back end querying + filtration of results	Floris van Veen
Create confusion matrix interface	Mihai Filimon
Create explore interface	Mathieu Butenaerts, Kanish Dwivedi
Backend testing	Kanish Dwivedi, Floris van Veen
Create front end querying interface	Silvia Mokranova
Create dashboard interface	Mihai Filimon
Allow users to submit data from front end	Mathieu Butenaerts, Floris van Veen

Appendix B: Original problem statement from project forum

The following is the original project problem statement that was found on Project Forum [15]. This was used in Problem Analysis to extract the main problems and goals for the project.

Please open this AI black-box! - User interfaces for supporting debugging and interpreting deep learning-based computer vision models

Background about the project
State-of-the-art computer vision methods employ neural network models (deep learning), generally operating as ”black-boxes”. The opaqueness of these models has become a major obstacle for debugging, tuning, deploying, and using them in practice. This is particularly the case in critical domains such as health, security, and justice, where the ability to understand, or, at least, to interpret their behavior is increasingly demanded.

Particularly, the models could be biased in some dangerous ways. For instance, an autonomous driving device might have incorrectly learned to detect white faces to identify pedestrians. And if darker-skin pedestrians cross the road, the device might not detect them, leading to an accident... If we could have opened the ”black-box” before deploying this autonomous driving device, we could have identified this bias and prevented the accident.

The European Union General Data Protection Regulation (GDPR) also states that ”[the data subject should have] the right ... to obtain an explanation of the decision reached”. Hence, computer vision practitioners urgently need help to open these black-boxes before being able to use them in practice!

Our proposition
We recently proposed a new method (termed SECA) that aims at opening the black-box. Our method relies on algorithmic explainability methods and also involves humans-in-the-loop in order to collect interpretable and rich explanations about a model’s behaviors.

However, our method can’t yet be used in practice... While the building blocks of the method are already implemented, they are not yet connected automatically, and would require manual effort by the practitioners to do so. Besides, practitioners would require proper interfaces to explore easily the explanations.

In this project, we propose you to design and implement a fully-integrated system on top of our method in order to finally help practitioners open the black-box.

Nature of the work
On the one hand, this would mean that you would build a system that directly supports our method. For instance, you would need to connect the different components that are used by our method.

On the other hand, we are currently designing what the user interfaces could be for the practitioners. You would then design and implement the back-end and front-end for these interfaces.

Finally, you would need to connect these two parts in order to fully support the interactivity. You would have a lot of freedom in choosing the technologies you prefer to use.

About us

We are an interdisciplinary team working in collaboration between researchers from the EEMCS and Industrial Design faculties. We are developing a set of methods and interactive user interfaces to support different practitioners in understanding what their computer vision model is doing. At the end of the project, we should be able to discover new dangerous biases in cutting-edge deep learning models!

Appendix C: Use-Case Analysis and Use-Case Diagram

.1. Use cases for basic users

- User wants to register themselves as a new user
- User wants to sign in to authenticate themselves
- User wants to view the training data
  - User can view images and see what the results of the training data result in
- User wants to look at the current classification results
  - Viewing a confusion matrix on a project
    - User clicks on a cell to view the mistakes that are made
  - Individual image predictions
  - What parts of each image is classified as what
- User wants to learn about the algorithm
- User wants to learn about SECA
- User wants to navigate the page for something specific
- User looks at the wiki page
- User wants to query classification formulae
- User wants to make notes on pages of interest

.2. Use cases for the ’expert’ user type

- all use cases that apply to regular users also apply to experts
- Expert wants to contribute to the training data
  - Expert can view images input by developers and fill in what objects are in the images, and where in the image
- Expert wants to see what a new image will be classified as
- Expert wants to add new information to the wiki page on their area of expertise
- Expert wants to fix mistakes given in previously inserted training data

.3. Use cases for the ’developer’ user type

- All use cases that apply to regular users also apply to developers.
- Developer wants to alter existing training data
- Developer wants to introduce new classifications to the machine
- Developer wants to alter non automatically information on the wiki or general page

- Developer wants to expand the training data
  - Developer can insert new training data and ask Experts to help in both classification and in verifying if the results are correct
- Developer wants to see what a new image will be classified as

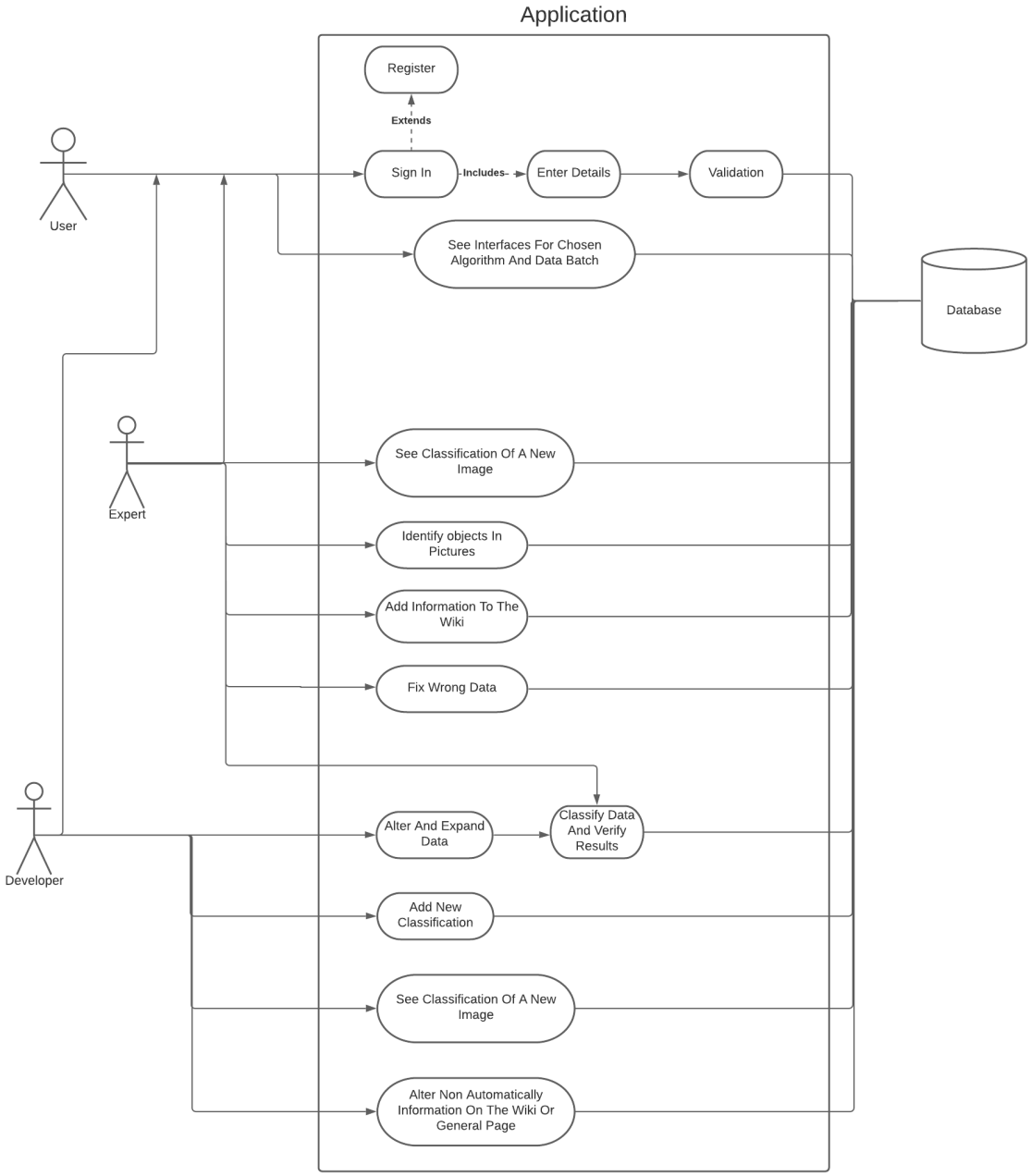


Figure 1: Use case analysis figure

Appendix D:Test code coverage

Our product has not tested the following:

- SECA code given by the client
- Security methods given by Django itself

Our product has tested the following:

- Methods in SECAAlgo
- Models in SECAAlgo
- Methods in UserSECA
- Models in UserSECA
- Methods in Wiki-SECA

Currently the tests achieve an 84 percent line coverage among the methods and views in the above segments

Appendix E: Confusion matrix interface comparison

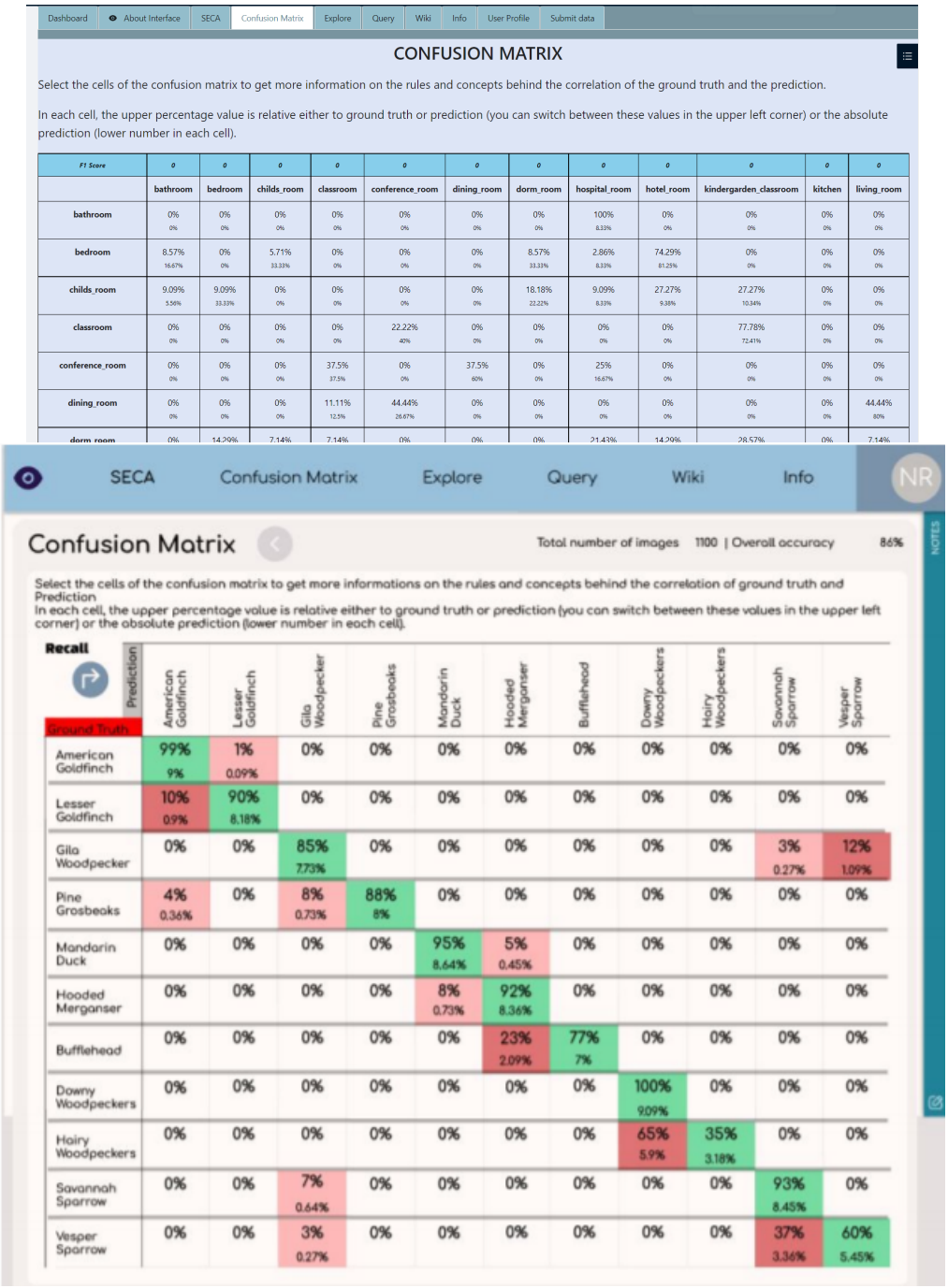


Figure 2: Confusion matrix interface. The image shown above is the final product, below is the client specification

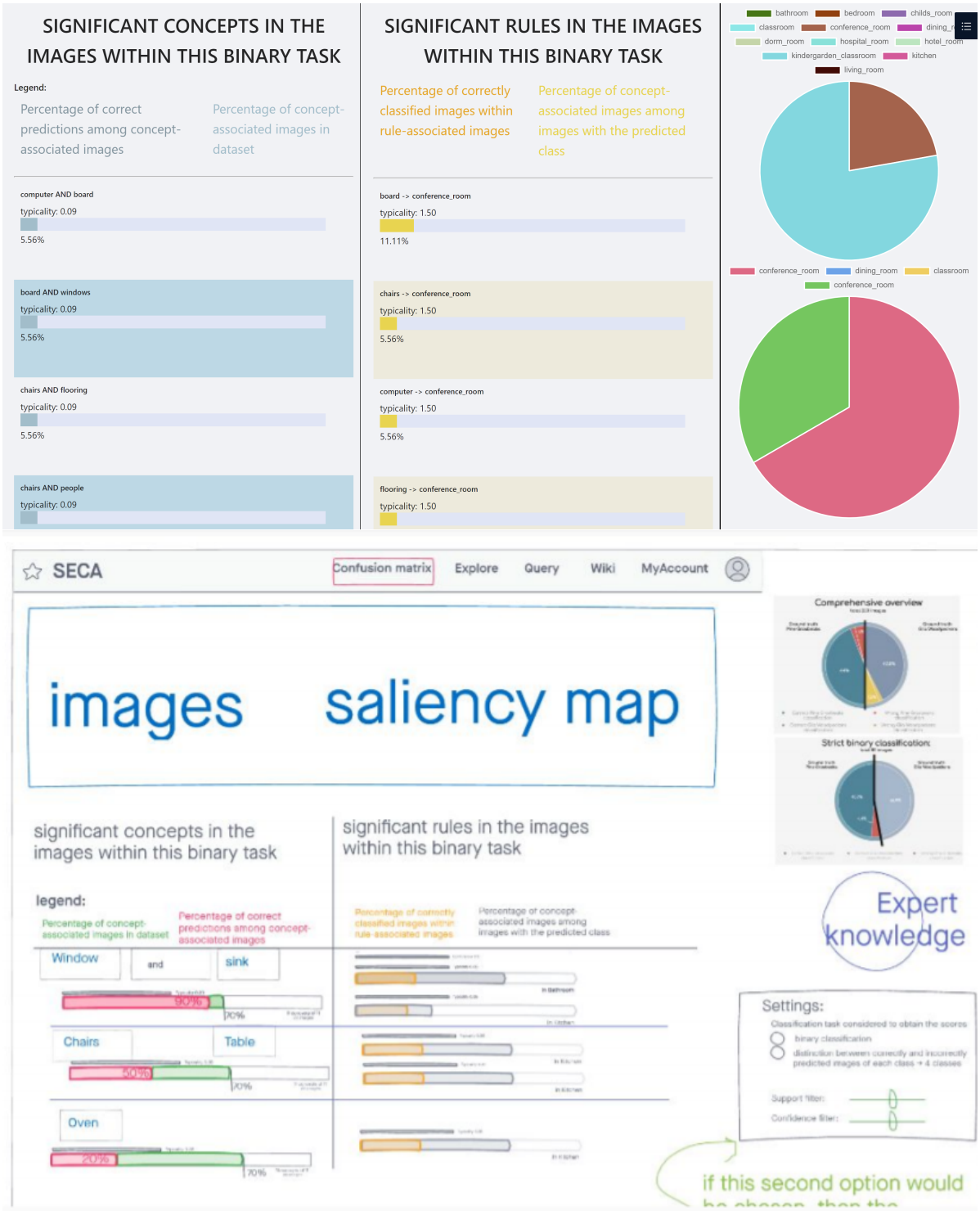


Figure 3: Confusion matrix classification page. The image shown above is the final product, below is the client specification

Appendix F: Static interfaces - About SECA comparison

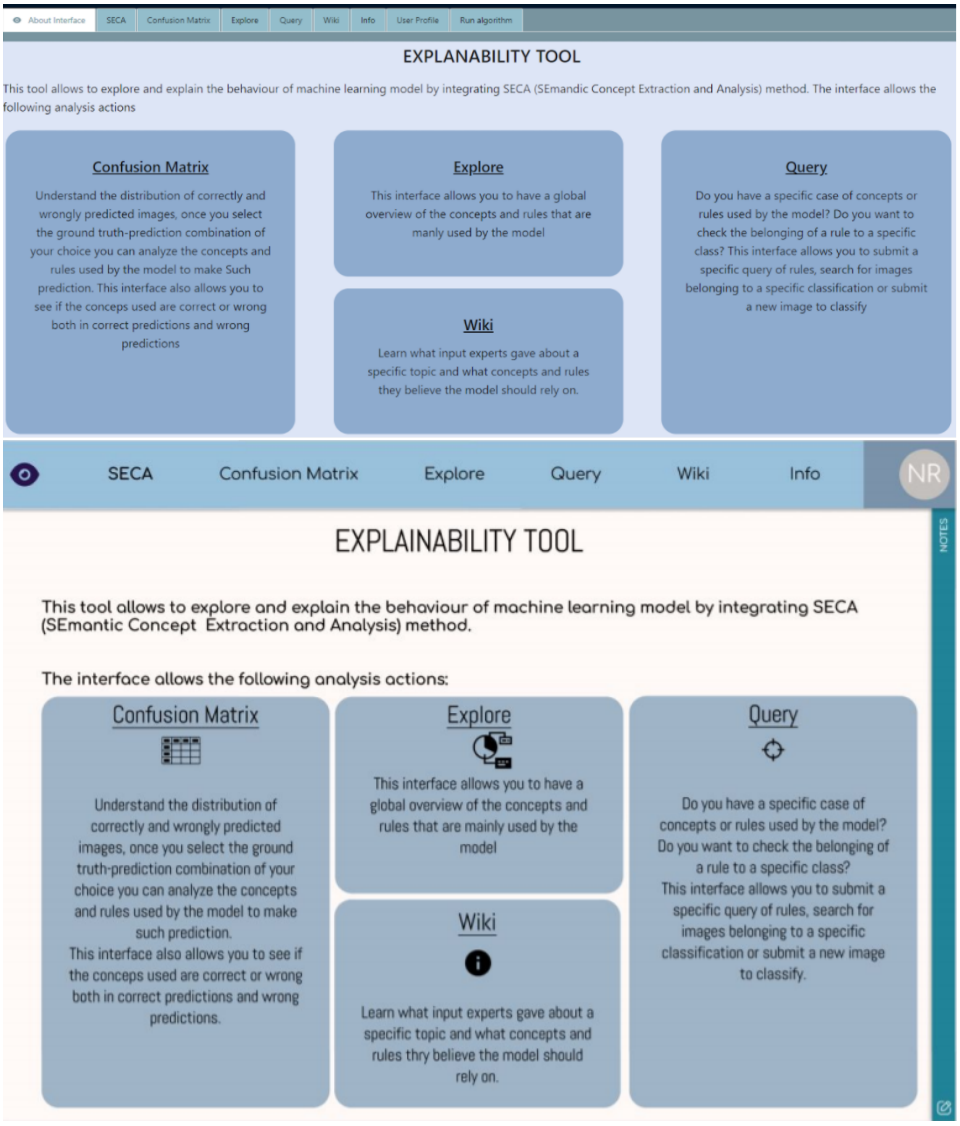


Figure 4: SECA main interface comparison. The final product is shown above and the client specification below

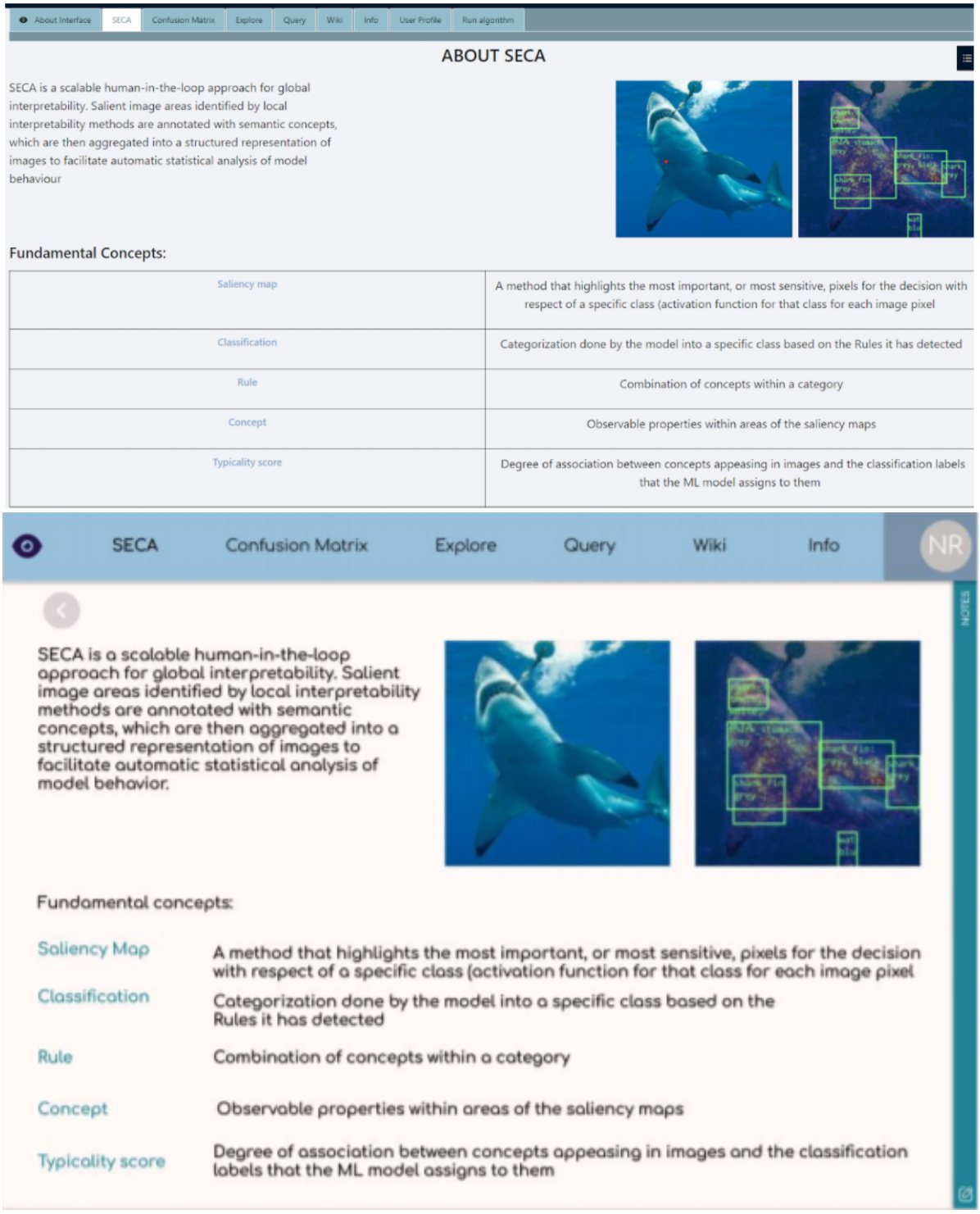


Figure 5: About SECA interface comparison. The final product is shown above and the client specification below

Appendix G: Static interfaces - Wiki pages comparison

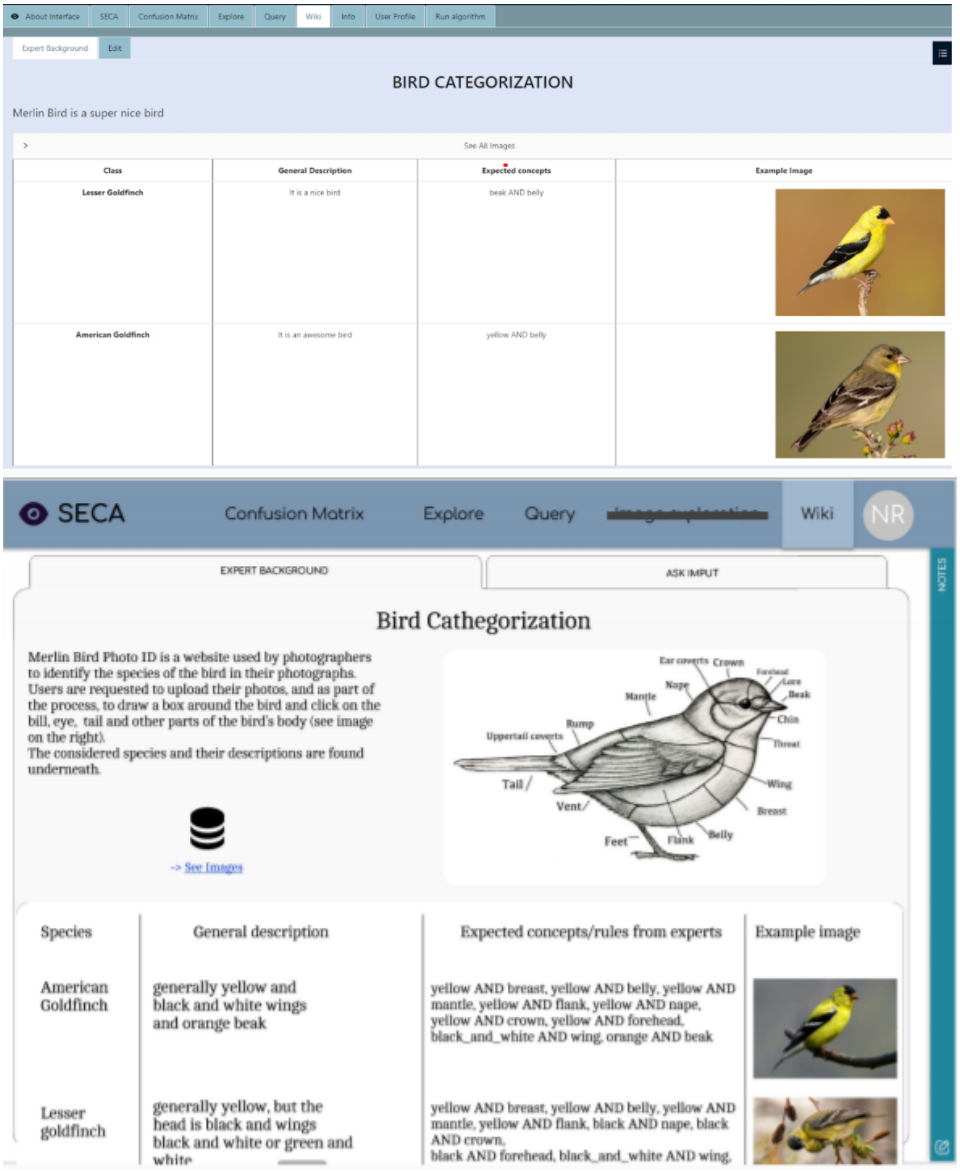


Figure 6: Wiki page 1 - expert background information view comparison. The final product is shown above, and the client specification is below

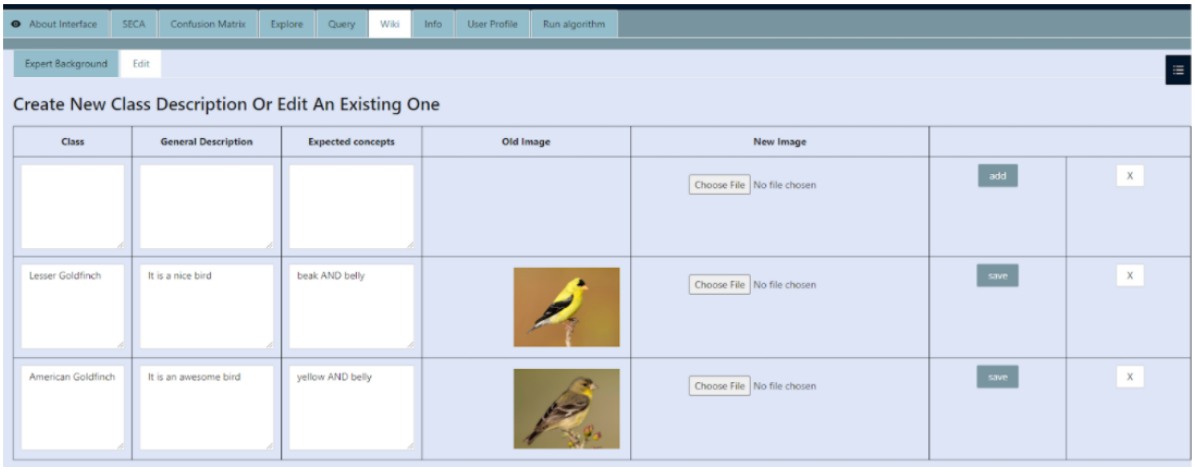


Figure 7: Wiki page 2 - edit information view. There was no client specification for this interface

Appendix H: Explore view comparison

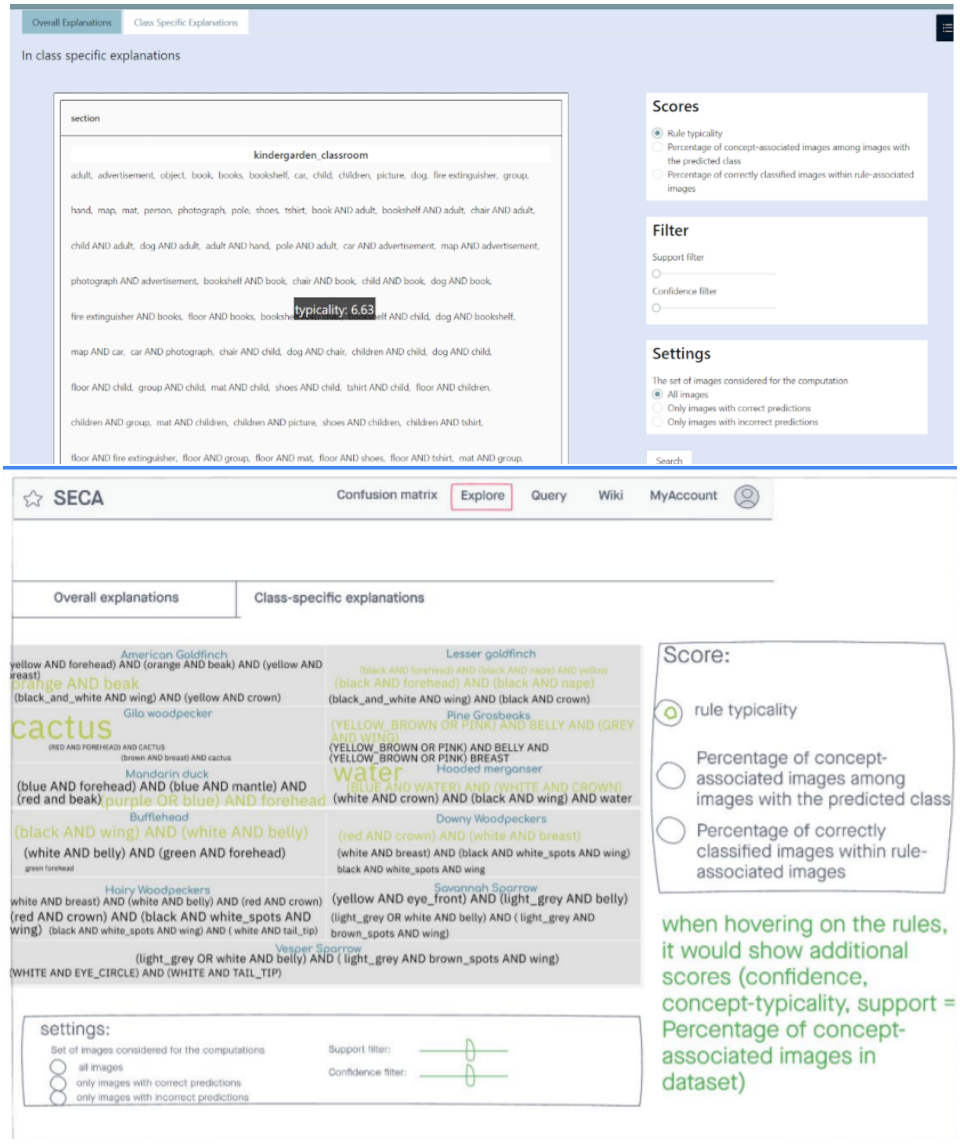


Figure 8: Explore view. Final product shown above, and the client specification is below

Appendix I: Heatmap view

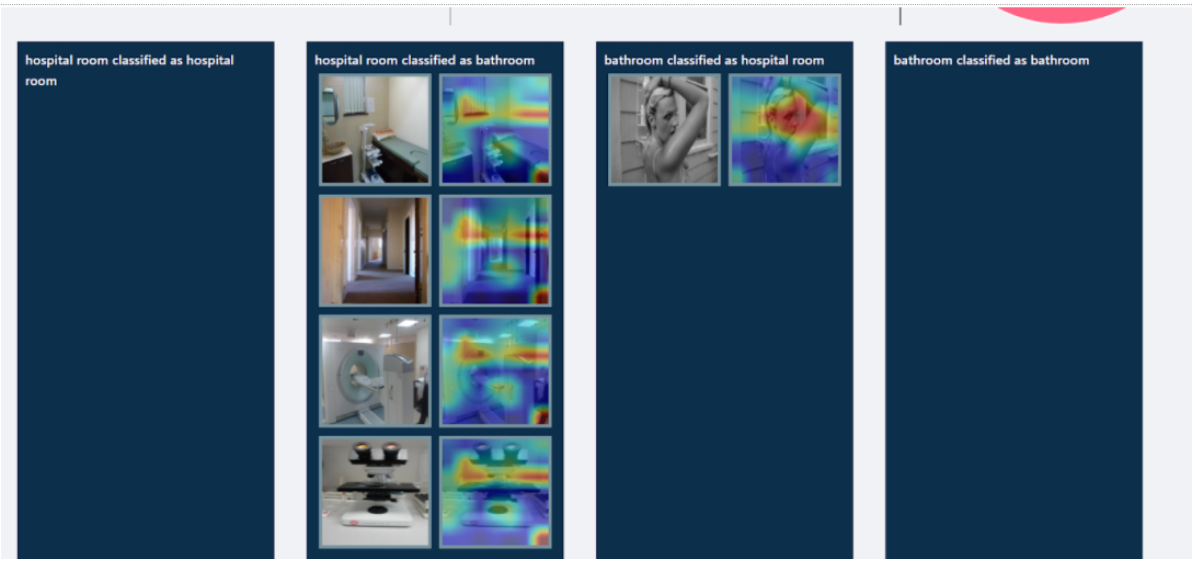


Figure 9: Heatmap display. There was no client specification for this interface

Appendix J: Query view comparison

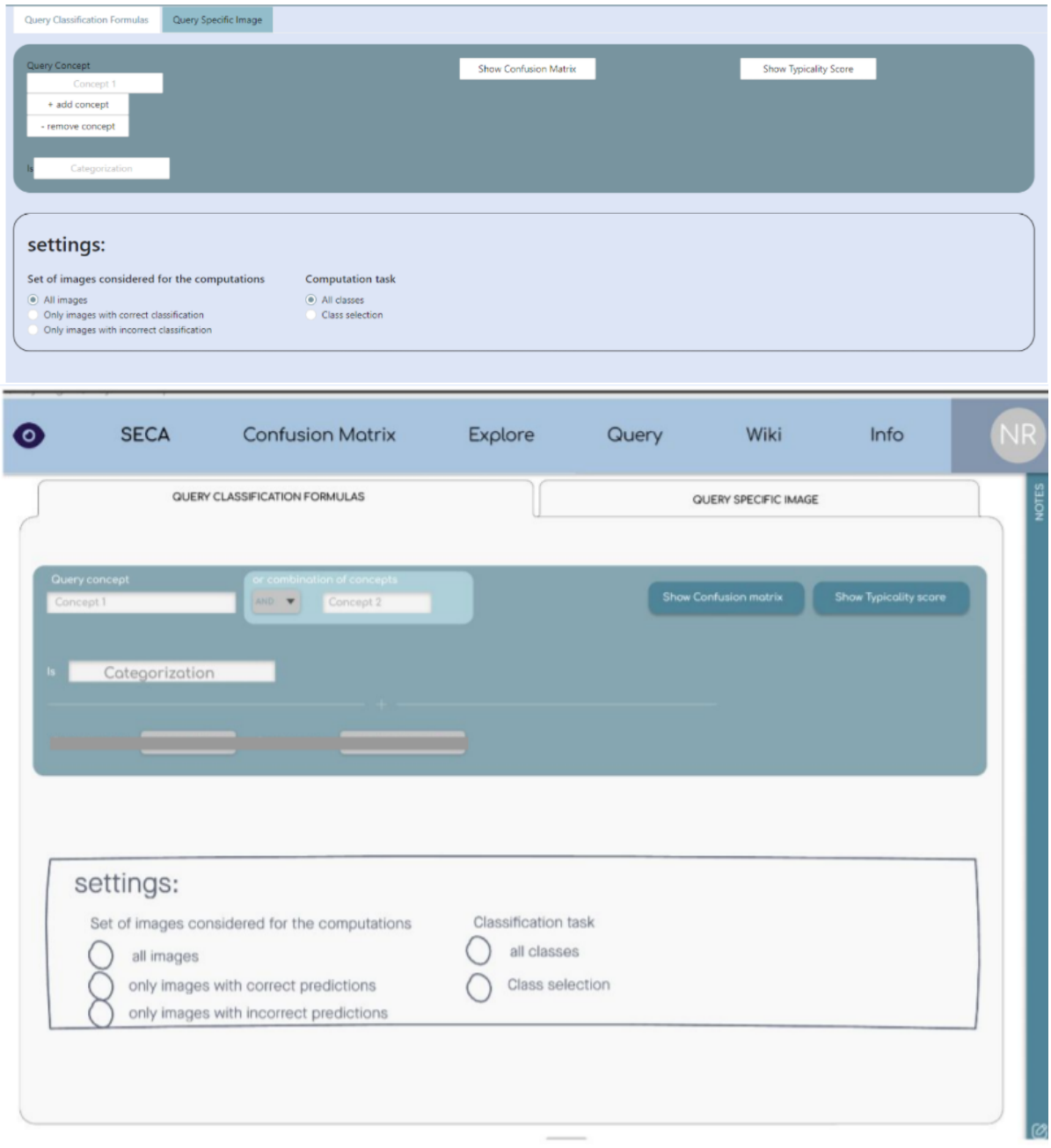


Figure 10: Query general view. Final product shown above, client specification shown below

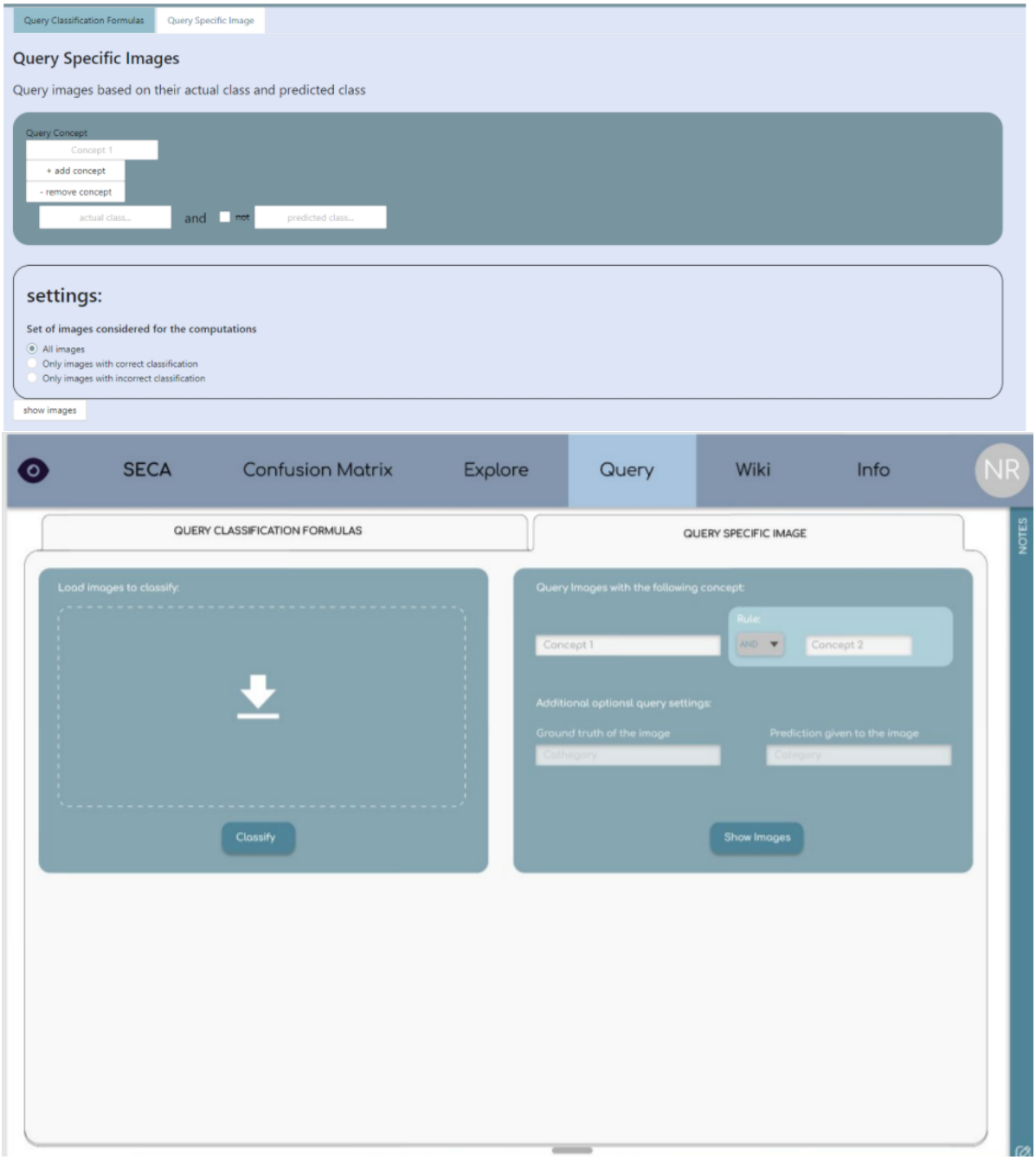


Figure 11: Query specific image view. Final product shown above, client specification shown below

Appendix K: Dashboard view

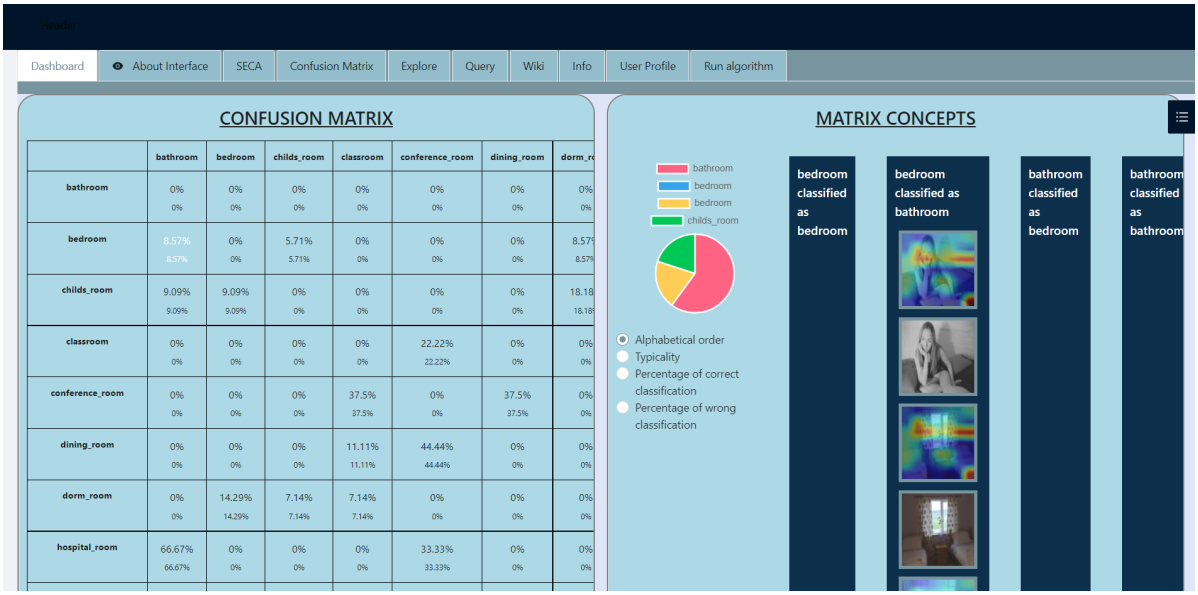


Figure 12: Dashboard view

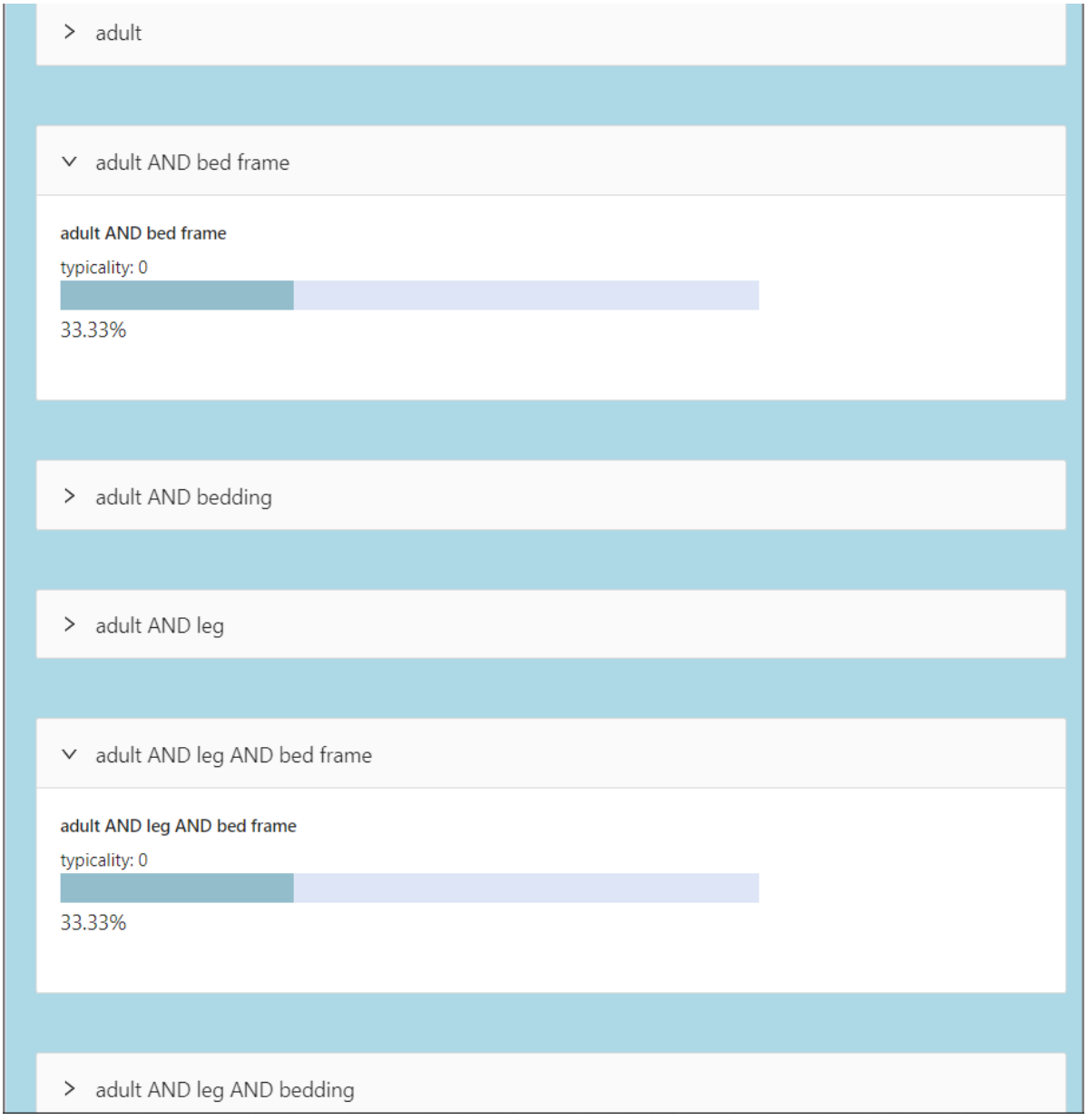


Figure 13: Dashboard rules display

Appendix L: Data submission view

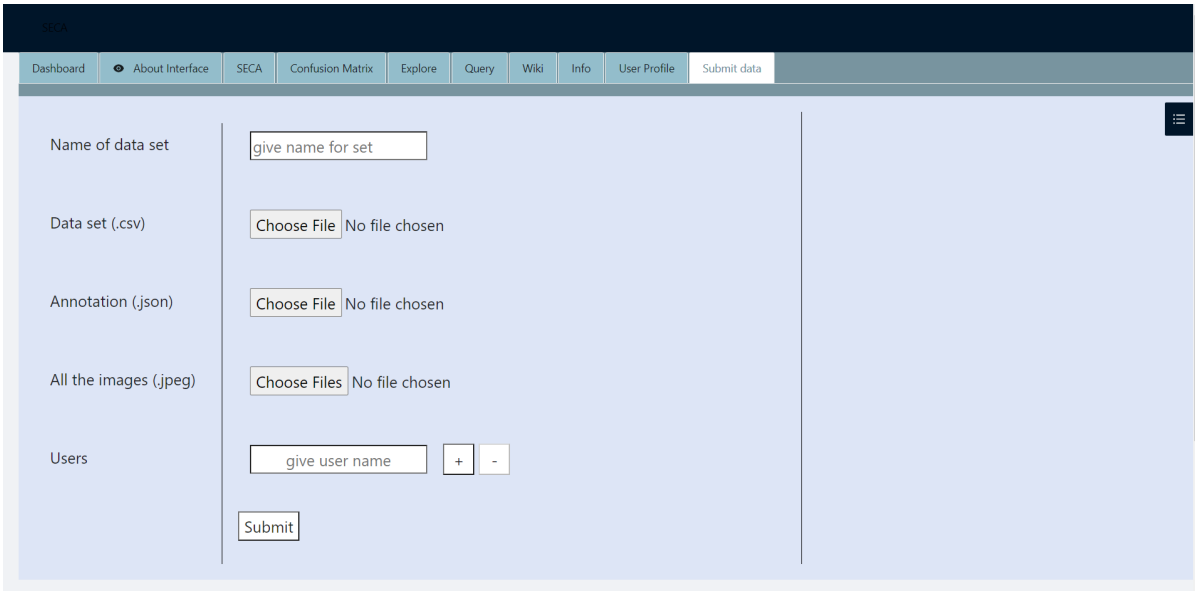


Figure 14: Data submission view