



UNIVERSITY OF TWENTE.

Faculty of Electrical Engineering,
Mathematics & Computer Science

Report Companion: Data driven reporting in radiology

Group 3

Andrei Popovici (s2349779)
Omid Fattahi Mehr (s2412659)
Leonardo Pasquarelli (s2293358)
Daan Assies (s2131943)
Pepijn Mesie (s2130815)

Design report
April 2022

Supervisor:

dr. F.A. Bukhsh
University of Twente

Client Contact Person

Jeroen Geerdink
Zorg Groep Twente

Abstract

The medical world is becoming more and more data-driven. Radiologists are using speech-to-text reporting, and this results in very unstructured data and reports which are becoming outdated very quickly. In order to make this more structured, a standardized way of reporting using a web-app to create and fill out forms will be put in place. Forms are the best way to structure reports as there is no room to deviate from the standard.

Contents

Abstract	ii
List of acronyms	vi
1 Introduction	1
1.1 Existing system	1
1.2 Proposed System	1
1.3 Stakeholders	2
1.4 Design process	2
2 Requirements	3
2.1 Must Have	3
2.2 Should have	4
2.3 Could have	4
2.4 Won't have	4
2.5 New requirements	4
2.6 Use cases	5
2.6.1 Maintainer	6
2.6.2 Resident	6
2.6.3 Radiologist	6
2.7 Limitations	7
3 Overall Design	8
3.1 Database design	8
3.2 Front-end	10
3.3 Frameworks	11
3.3.1 Back-end	11
3.3.2 Front-end	11
3.3.3 Testing	12
3.3.4 Deployment	12
3.4 Advantages over the state of the art	13

4	Detailed Design	16
4.1	Zero footprint	16
4.2	Visual design choices	16
4.3	Front-End Implementation	17
4.3.1	Explanation of React Concepts	17
4.3.2	Overview page	17
4.3.3	Form Input	19
4.3.4	Authentication	24
4.3.5	Managing forms	24
4.3.6	Creating forms	25
4.4	Back-end	28
5	Test plan and results	31
5.1	Back-end testing	31
5.2	Front-end testing	32
5.2.1	Creating a form	32
5.2.2	Filling in a form	32
5.2.3	Modifying a form	32
6	Risk Analysis	34
6.1	Spoofing Identity	35
6.2	Tampering with data	36
6.3	Repudiation	37
6.4	Information disclosure	37
6.5	Denial of Service	37
6.6	Elevation of privilege	38
6.7	Conclusion	39
7	Evaluation	40
7.1	Reflection on Planning	40
7.2	Reflection on Meetings	42
7.3	Distribution of Tasks	43
7.4	Changes to the initial system	43
7.5	Final Result	44
8	Future Work	46
8.1	Extension of Report Companion	46
8.1.1	User Experience	46
8.1.2	Data Collection	46
8.1.3	Functionality	46

8.2 New Version of Report Companion	47
9 Summary	48
Bibliography	49
A UML Diagrams	1
B Prototypes	3
B.1 LoFi-Prototypes	3
B.2 HiFi-Prototype	7
C Colour Schemes	8
C.1 Coluor schemes of final product	8
C.2 Colour schemes for prototyping	9
D Application Program Interface (API) requests	14
D.1 Example Form JSON	14
E Testing Report	17
E.1 Back-end Unit testing	17
F Setup Instructions	18
F.1 Development Setup	18
F.1.1 Front end	18
F.1.2 Backend end	18
F.2 Deployment using docker-compose	19

List of acronyms

API	Application Program Interface
CIA-Triad	Confidentiality, Integrity & Availability Triad
GDPR	General Data Protection Regulation
Hi-Fi	High Fidelity
JWT	JSON Web Token
Lo-Fi	Low Fidelity
MUI	Material UI
MVP	Minimum Viable Product
PBKDF2	Password-Based Key Derivation Function 2
ZGT	Zorg Groep Twente

Introduction

Zorg Groep Twente (ZGT) has a radiology department which is in need of modernization. As mentioned by the journal *Radiology Today* [1], the workflow of radiologists is changing to become more digital, which means that in consequence the tools need to be changed and reconsidered. This project shows an instance of what such a change in workflow could look like.

1.1 Existing system

The current way for radiologists to report their analyses is speaking out their analysis of a radiology image through a microphone. This input is directly processed and converted into text, after which the text is saved into text files and attached to a patient's health record. The main advantage of this form of reporting is that it takes very little time. However, this form of reporting does not only carry advantages. The biggest disadvantage is that radiologists have no way of standardizing this reporting. Every radiologist freely speaks into a microphone, and speech is very unstructured. Because of this, saved data is unstructured, unlabelled and not standardized. As there is no standardized reporting scheme, it is hard for beginning radiologists or residents to get used to the system. Furthermore, unstructured data is very hard to use for any kind of analysis, which is something the hospital is investing in for the ability to create smarter and more efficient processes. [2].

1.2 Proposed System

In order to perform data analysis on the images and make reporting easier, ZGT is looking for an application where forms can be created which provide all the steps required for a full image analysis, aiding radiologists by taking them through the analysis step by step. The application aims to get close to the efficiency of a speech-to-text application, with the added benefit of standardizing and structuring reports.

1.3 Stakeholders

Multiple stakeholders can be identified who will be affected by the product. They are the following:

ZGT ZGT and some of its employees will make use of, and host, the system. The ZGT will also be the main entry point of communication throughout the design and development of the product.

Radiologists Radiologists can add new templates to the system, and they will also be able to maintain the system where needed.

Residents The residents at the hospital are the main target audience for this product. To make it successful, the product has to be intuitive and require interactions to be as efficient as possible.

Others There are other people who can benefit from the product, this includes radiologists and residents from all over the world. These will not be the main stakeholders, but because this product will be publicly available on the web, they will still be able to make use of the product.

1.4 Design process

In the first interview with ZGT, requirements were gathered, then research was done on existing products and ideas were discussed. After the requirements were formulated, the group split up into groups working on the project proposal, initial prototypes and creating the framework that would be used for the actual product.

The project has seen multiple development iterations, consisting of 2 rounds of prototyping (see Appendix B) and the creation of the actual product. The initial prototypes were built to be as broad as possible, this was achieved by making multiple simple prototypes. The second prototype iteration was a more complete prototype focused on a single approach to confirm the plan.

Throughout the design process, meetings were held with the customer and future end-users to ensure that any wishes for the design of the product would be met.

Requirements

A document with a small introduction to the project was given. This document states the goal of the end-product, but not any specific requirements. In order to capture requirements, there was an interview with two radiologists and the innovative manager of ZGT. From this interview, some Low Fidelity (Lo-Fi) prototypes were made and presented to the radiologists a week after the first interview to fully capture the intent of the system and how the web app should take shape. From these two interviews, the following requirements became apparent. They have been categorized according to the MoSCoW principle.

2.1 Must Have

The system must have

- a web form where users can fill in forms.
- a structure that can enable calculations based on user input.
- a web-page for managing (creating, editing, removing) the available forms
- the capability to produce a standardized output report from the inputs in a form that can be processed by radiologists
- the ability to skip over questions in a form based on previous answers (decreasing time spent on filling in the form)
- a structure that is both intuitive and fast when filling out a form
- an index page where you navigate to the correct web form using a search function

2.2 Should have

The system should

- have a dropdown menu attached to the search function to aid the searching
- be dockeriseable
- be usable by residents of any level
- be zero-footprint

2.3 Could have

The system could

- have the possibility for storing user input data in a database
- have the capability to upload images related to the form
- be scalable to wider uses (possibly through API)

2.4 Won't have

The system won't have

- the usage of Open RTC
- theme tweaks depending on the location of the hospital

2.5 New requirements

Two meetings were had, which will be talked about in more detail in Section 7.2, in which the radiologists were presented with the High Fidelity (Hi-Fi) prototype and the Minimum Viable Product (MVP), respectively. After each of those two meetings, ZGT came up with new requirements, listed below.

Must have The system must have

- a way to link text to scores based on a defined formula
- a button to automatically fill in (parts of) a form with default answers set in the form creation.

Should have The system should have

- the possibility to pre-define text to be shown in the report in place of the default for each answer field
- highlighted text in the report if an answer is not the default answer to that question

Could have The system could have

- a button to add a page to the browsers' bookmark at the top of the webpage
- a button to add forms to a user's favourites, that would then be shown on the main overview page
- the ability to add URL-links questions, which could link to extra information/explanation on a question
- the ability to link different translations of forms with each other, giving the user an easy option to translate

Won't have The system won't have

- default questions that always occur in every form
- user input stored in a database

2.6 Use cases

For the system, use cases were identified as well. Some of these use cases have also been covered in the system requirements, but these use cases give a better idea of what interactions each different kind of user will have with the system. The users have been broken down into three different categories:

- Maintainer, who will update the system and add forms
- Resident, who wants to learn how to report findings
- Radiologist, who wants to generate a report as fast as possible, next to the use cases of the residents

2.6.1 Maintainer

As a maintainer, I want to

- Log in to get extra access to the system
- Create forms
- When creating forms, I want to:
 - Add questions of different types
 - Give default answers to questions
 - Give formulas needed
- Remove forms

2.6.2 Resident

As a resident, I want to

- Fill in forms
- Change language between Dutch and English
- Search for reports using report name or keywords
- See my previous answers when filling in the forms
- Go back to previous questions
- See which questions are unanswered
- Receive a score needed for e.g. advice
- Generate a report based on my answers
- See the so-far generated report during filling in a form

2.6.3 Radiologist

As a radiologist, I want to

- Fill in the form while using as little time as possible looking at the computer screen.
- Skip questions when findings are normal.

2.7 Limitations

The system will be deployed onto the internet. This means that the application will be used by anyone that has access to the website. In order to comply with the General Data Protection Regulation (GDPR), **no** user data that can lead back to any one person is stored. This data includes answers filled in the form, the generated report and on the person filling in the form. Any integration with hospital software will **not** be done in this project. This means that the system will be zero-footprint. This is further extended on in Section 4.1.

Overall Design

This chapter will discuss the overall design choices made for the system. The separate components and their functionalities are described in this chapter. More of the details and why these design choices were made will be covered in Chapter 4.

3.1 Database design

The database consists of several tables, and three sections. This chapter will focus on the section that includes the tables that store the forms. A visual representation of the database design can be found in Appendix A.

Form

This table stores anything needed for the top-level form. When the form is a sub-form, the creator, tags and title fields will not be used. This table consists of the following fields:

- ID: Used to identify the form and is also used in the URL to get to the form.
- creator: Saves who has made the form, which can be manually set by the creator.
- is_sub_form: Used to check whether this form is at top-level or whether it is a sub-form which is used when a certain answer is given.
- tags: Used to identify the form and are used to search the form on the homepage.
- title: The title of the form. It is used to search the form on the homepage and is also shown when filling in the form.

FormSection

This table is used to identify form sections. A section is usually a body part which needs examining.

- ID: The identifier of the section, needed as primary key.

- form_ID: Foreign key from Form. Used as a connection between form and section
- name: Used to display on the page where you fill in the form.

FormQuestion

This table is used for saving each individual question. It is part of a section, and stores the following fields:

- ID: The identifier of the question, needed as primary key.
- section_ID: Foreign key from FormSection. Used as a connection between section and question.
- question: Character field that contains the actual question.
- question_link: Contains a hyperlink for the question. This can be used if a question can be confusing and the hyperlink offers clarification
- question_type: Contains an integer that signifies the question type. The different question types are further expanded on in Section 4.3.3.

QuestionPossibleAnswers

This table contains the answers to a question with type radio buttons or checkboxes. These are displayed below the questions.

- ID: The identifier of the answer, needed as primary key.
- question_ID: Foreign key of FormQuestion. Used as a connection between question and answer.
- form_ID: Foreign key of Form. When this answer is chosen, and it has a sub-form, then this field will be equal to the ID of that sub-form.
- answer_score: this field is an integer and can be used for formula calculation.
- answer_text: The actual answer.
- default: Boolean, if True, then this answer will be selected when a radiologist says that the section is *normal*.
- report_text: Can be set when creating a form in order to generate predefined text into the report.

Formula

This table stores formulas used in forms. The results of these formulas are used by doctors to make decisions.

- ID: The identifier of the formula, needed as primary key.
- form_ID: Foreign key of Form. Used as a connection between form and formula.
- formula: The formula. An example format can be seen in Appendix D.
- name: Formula name, used for display.

FormulaEdges

A formula and its result does need an output. Rather than outputting the total score, you can set an output text based on that total score. This table is used to store the scores needed for certain text.

- ID: The identifier of the formula edges, needed as primary key.
- formula.ID: Foreign key of Formula. Used as a connection between formula and formula labels.
- score_above: This is an integer and is used as the edge above which the text of the next field is output.
- text: The text that is output for this certain edge.

3.2 Front-end

The front-end of Report Companion consists of the following pages:

Home page The home page serves as the starting point of the application. A search bar is included to find forms, since this would typically be the first action that a user performs when intending to fill in forms. Selecting such a form then redirects the user to a page where they can fill in that form.

Fill in form As the name suggests, it is possible to fill in a form on this page. This page consists of three compartments - two small ones on the sides and the main body in the centre. The large column in the centre is the form, through which the user can fill in their answer. The column on the left shows an overview of the questions and sections, similar to a table of contents. The column on the right shows the report, and it allows the user to generate and copy their answers.

Log in page Before allowing users to manage or create forms, it is important that they are authorized. The login page contains a credentials based log in form. Once logged in, access to the pages below is granted.

Manage forms The *Manage forms* page consists of a table displaying all existing forms and details thereof, and it allows authorized users to delete or edit existing forms and to create new forms.

Create Forms Creating Forms is done through another form page, in which authorized users can create sections and questions. For questions of type radio button or check box, it is possible to assign a score to each answer option, or to link sub questions that appear if the answer option was selected. The scores will be taken account,

when creating formulas, which is also done in this page. To facilitate navigation, a hierarchical overview of all sections, questions and sub questions is displayed on the left of the form.

Edit Forms The editing form page has the same structure as the *Create Forms* page. When accessing the *Editing Form* page, the selected form will be loaded into the page as if the user had just filled in the exact same form.

A more detailed depiction of the design can be found in Chapter 4.

3.3 Frameworks

3.3.1 Back-end

The back-end is implemented using Django, a powerful python framework for developing back-end services. Django will handle the accessing and storing of information through API endpoints and will also manage the database.

The choice for this framework was made because a couple group members had prior experience with Django, furthermore this framework has many security features built in and a lot of libraries that aid the final product in being more secure like for example a library that adds a simple and secure JSON Web Token (JWT) implementation. This saved a lot of time, which was spent on front-end improvements. More information on the back-end can be found in Section 4.4

3.3.2 Front-end

For the development of the front-end, The choice was made to use React.js. React.js is a popular front-end framework developed by Meta. The choice for React.js, was made because half of the team, as well as one of the radiologists who will use the system, had some experience working with React.js.

To allow for a complete product to be made, packages have been used with React.js and front end development in general. These packages help simplify and abstract the development, while also ensure the security of the application.

For developing components, the choice was made to use the Material UI (MUI) React component library. This library helps save time creating components such as radio-boxes and check-boxes, and as it would provide basic styling functionality that can

easily be build upon. the choice for MUI specifically was made due to its large popularity and because some team members having used it in the past.

To improve stability and scalability, the amount of re-renders within the web page had to be minimized. This was achieved through the React Hook Form library. This library simplifies the handling of form data, making sure that only the fields that should change on any given input are being updated. Using React Hook Form reduced the time a re-render takes by 5 times in the form input, going from 60ms per re-render to 12.

Lastly, to create a smoother product, the React Router package was used, which provides a way to switch between parts of the websites, without causing the website to reload. This library was ideal for the product, as it would further reduce the radiologists' time spent looking for forms.

3.3.3 Testing

For the back-end, Django's built-in testing functionality is used. In there, API-endpoints and internal functions are tested, this is extensively described in Section 5.1. Further testing on the API is done through Postman, this application can manually send requests to a web-page. This emulates the actions the front-end executes.

The front-end tests are written using Selenium IDE. Selenium IDE is used to emulate a user executing certain actions like creating, editing and deleting forms. The front-end testing is further expanded on in Section 5.2

3.3.4 Deployment

The client requested that the product should be dockeriseable. This has been taken into account when choosing frameworks. All frameworks are easily dockeriseable, and a docker-compose file was included in the product to allow for easy deployment by the ZGT. These are the services that are containerised:

- Front-end: this container builds a production build for the front-end and uses nginx as a web server
- Back-end: there are three containers launched: one for migrating the database, one for applying the migrations and one for running the actual back-end
- Database: upon ZGT's request, MySQL is used.

The containers are supplied with a `.env` file, which specifies important environment variables, such as the MySQL username and password.

3.4 Advantages over the state of the art

A question that has been asked many times during this project is: "Why is this not Google Forms". At first glance, this project may seem to have many similarities with Google Forms, but after using the application for a while, one can see that this application has been designed to specifically fit the purpose of filling in forms about Radiology Images. This resulted in a significantly different product compared to the likes of Google Forms that has its own place in the market.

User Experience

The client made it very clear to us, that a great user experience and a particularly fast to use product would be the most important part of the project. Therefore, a lot of time was spent on this area of the product, improving the user experience and incorporating the feedback of ZGT into it.

Three-column layout in Form Input

The page in which a user fills in a form is laid out in three columns. The middle column is the traditional body: this is where the questions are displayed and where the user can fill in his answers. The column on the left shows an overview of the form. They can see all questions and sections and by clicking on one of the questions or sections, the screen moves to the location. If questions have been answered, they are highlighted. That way, users can scroll through the form and leave out questions unanswered for later. On the right column, a report is generated. This contains all the sections and questions, as well as all the answers. By clicking a button that is located below the report, the user can copy the report directly into the clipboard.

Fast to use

Small implementation details, such as jumping to a question by clicking it in the overview, or clicking the normal button which fills in questions, greatly help to reduce the speed of filling in a form. When comparing the product to Google Forms, it was found that for the use case of filling in radiographical images, using Report Companion results in a substantial decrease in usage time.

Privacy

The product does not track any data at all, whereas for example, Google Forms has the control over the data that is filled in. This is a deliberate design choice that was made. This way, there will be no future issues concerning installation of the program on the

internal network of ZGT if they are inclined to do so, and GDPR. Several benefits can be named for storing data. Some of the data that can be stored, and their benefits are:

- Storing the resulting forms. This can be used to see which forms are filled in most frequently or for future products, where machine learning models can use the data stored by Report Companion.
- Storing individual answers. This can, for example, be used for frequency analysis on which conditions are most commonly found.

Functionality

Through the span of this project, the product accumulated a high amount of features, which are not present in any of the state-of-the-art systems. These features can be found below.

Sub questions

When answering a question (referred to as a parent question), a sub question related to the parent question can appear on the same page. This primarily happens with radio buttons.

Formulas

It is possible to create formulas, which are composed of the questions from the form. Depending on the value of the formula, a diagnosis can then be made with more ease.

Scores

With radio buttons and check boxes, a score can be assigned to each question. The scores are then reused in the formulas.

Default question values

When filling in forms, for a lot of patients, the radiologists won't find any special findings. Therefore, a default button, which fills in the questions with default values, is displayed in each section.

Exporting answers (client-side)

In Google Forms, questions are submitted and stored in a server. In Report Companion, the focus lies in providing all the results of the form directly on the client side, and to allow the user to copy the data without them needing to interact with the server-side.

This gives the users the ability to put the output information in their own computer systems, requiring no integration with already existing data handling systems.

Extensions and Modifications

While this is a smaller feature of the product, it could become relevant in the future. It is possible to tweak the product, while it is not possible to tweak Google Forms.

Deployment

The ZGT can deploy the product on their own. If desired, the product can be deployed in such a way, that it cannot be accessed outside their network. Furthermore, the ZGT could redistribute their product to other hospitals. This could be useful, if hospitals wanted to adapt forms or the colour schemes used in the product.

Detailed Design

4.1 Zero footprint

The product is zero-footprint. This means that there is no install or download required to use the system from the client's end. This is a specific requirement from ZGT because they want the system to be accessible to as many radiologists as possible. When the system is zero-footprint, there are no concerns with installation. This is done through a docker deployment, in which the system will run and radiologists across the globe have access to the system.

4.2 Visual design choices

Dark Mode

For a more pleasant user experience, a user can switch the appearance of the product to dark mode. This is especially relevant to the Radiologists, since they do their work in a very dark room. Adding the dark mode removes a bright light source in the room where the radiologists work, helping them in their process by improving visibility of the images. The dark mode is implemented with the MUI Framework, which provides themes, that can be used to change the design of every component, without having to manually adjust the styling of every button. MUI provides a default dark mode, which has been altered slightly in order to adjust to the preferences of the Radiologists. The dark mode can simply be switched in the navigation bar with a simple click on a button. The design mode preference is stored locally for every user, such that the Radiologists can work in Dark Mode without having to switch when the page is reloaded.

Colour schemes

A number of colour schemes were suggested to ZGT that can be used when using the product. From these suggestions, ZGT concluded that they have no preference for a

colour scheme, which meant that the colours of Report Companion were in the developers' hands. The suggested colour schemes can be found in Appendix C.2. Having these colour schemes helps users by improving readability depending on the environment of the user. After trying out different alternative colour schemes, a decision was made to implement the colour scheme that can be found in Appendix C.1. The light mode uses `ghost white` for the header, which is slightly different from the white background, `black` for the text and `cadmium orange` as the overarching theme colour. The dark mode uses the colour `gainsboro` for the text, as per request of the radiologists, the dark mode should not use bright white since Report Companion could be used in dark rooms. Furthermore, `eerie black` is used for the background and `cadmium orange` as overarching theme colour.

4.3 Front-End Implementation

4.3.1 Explanation of React Concepts

Components Components split parts of the website into reusable building blocks. They typically contain some state variables, functions, properties, and they return a DOM representation that is displayed in the browser.

Props Props - shorthand for *properties* - can be seen as function arguments for a component. They are assigned from a parent component, when creating a component. Changing props causes a re-render.

State States are similar to props, with the difference that they are instantiated and maintained within a component.

Hooks Hooks are functions that depend on the state of a component. An example of this is the `useState` hook, which creates a state variable. State variables belong to a single component, and they cause the component to re-render.

For a more detailed explanation of React concepts, refer to React Docs.

4.3.2 Overview page

Navigation bar

The navigation bar is persistently displayed in each page, and it provides buttons to navigate to the home page, the login page and to the manage forms page if the user is logged in. There is also a switch for the dark and light themes. Clicking a button

will run the `useNavigate()` hook from *React-Router-DOM*. The page will be changed without forcing the web-page to reload. Through the variable `loggedIn`, it is checked whether the user is logged in. If the user is logged in, a button for managing the forms is displayed. The navigation bars can be seen in Section 4.1.

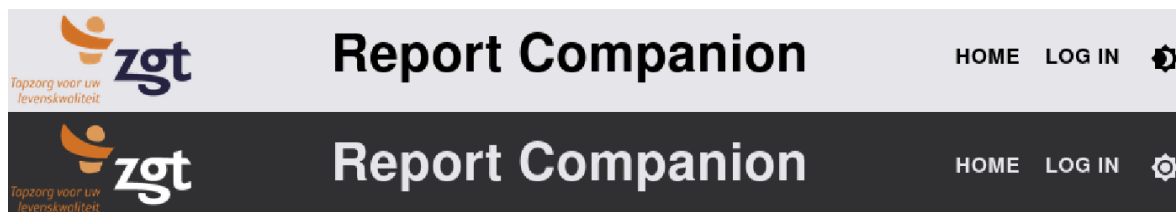


Figure 4.1: Navigation bars in light and dark mode

Search Bar

The search bar first retrieves all form names and tags from the back-end and saves them to the state variable `formList`. The forms are then loaded into the MUI `<Autocomplete />` component. The user can either select a form with the arrow keys, with the mouse, or they can type in a form name and the results will be filtered. After the selection of the form, the user will get redirected to the filling in page of that form.

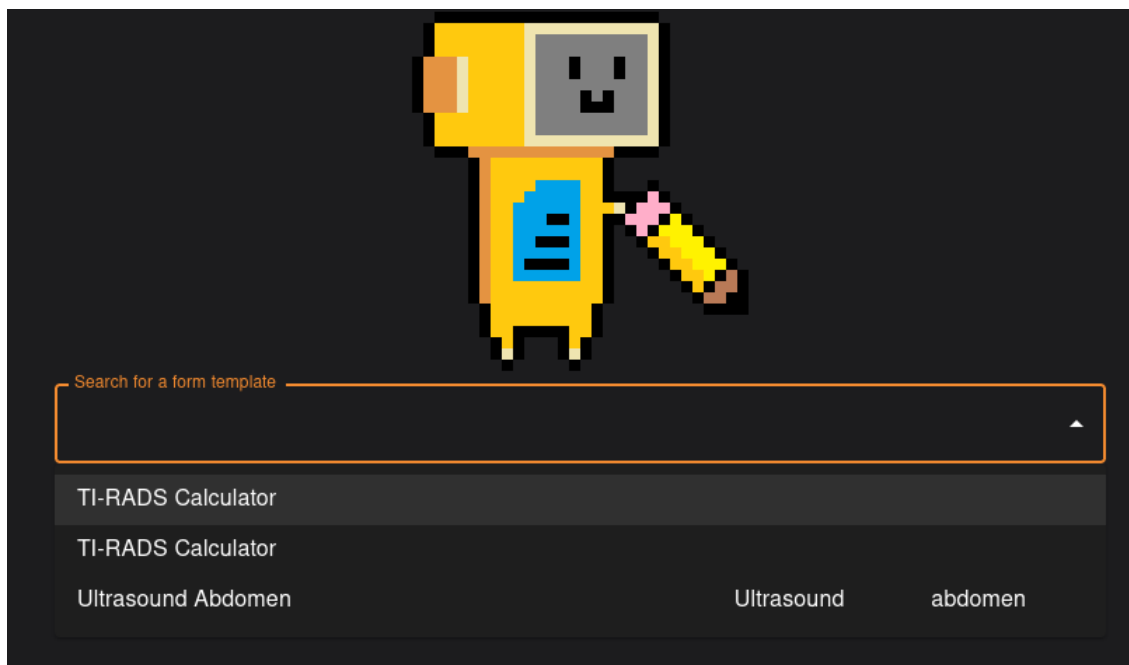


Figure 4.2: Search bar in dark mode

4.3.3 Form Input

Overview

The Form-Input page is the main feature of the product. On this page, the user will fill in forms related to the analysis they are carrying out. The page consists of 3 main components, namely:

- Overview, which is a table of contents for the form, showing which questions have been answered.
- Form, which is the actual form that the users interact with
- Report, which displays the report as generated by the user input, this includes results of calculations of any formula

Structure

First, when accessing the page, a `GET` request is made to the back-end to get the form that has to be displayed. While this request is created, a few data-structures are set up to allow for the components to interact with each other. In the beginning, some functions are called that allow components to add to and read from an object. This is achieved through the `React-Form-Hook` library to allow for updates to happen smoothly. Afterwards, a `DataContext Provider` is created to allow children components to access the functions to interact with the `React-Form-Hook` functions. While the form data is loading, the word "Loading" is shown as to not confuse the user and signal that the website is still working. Once the form data has finished loading, the main components will be shown.

Form generation

The entire form is created by splitting the form into small components and generating a React component for each of those form components. The generation of the form is done in the same order as the JSON object received from the back-end structure. An example of the JSON object can be found in Appendix D. The `<Form />` component will create `<Section />` components for each section. The `<Section />` component will create a `<Question />` component for each question, which finally generates the type of the question via the `generateComponent(question)` method.

Question types

The different question types can be found in Table 4.1. Each question gets the question ID and the question name as input props (properties).

Table 4.1: Data types of the questions

Number	Type	Component
0	Text	<code><TextInput /></code>
1	Number	<code><TextInput /></code>
2	Radio Buttons	<code><RadioButtonInput /></code>
3	Checkboxes	<code><CheckboxInput /></code>
4	Booleans	<code><RadioButtonInput /></code>
5	Text area	<code><TextInput /></code>

The question types can be split up into 3 main types.

- Open inputs, which are used for the `TextField`, `Number` and `TextArea`.
- Radio Inputs, which are used for the `Booleans` and `Radio buttons`.
- Checkbox inputs, which are only used for `Checkbox inputs`.

Open Inputs

Questions using the `<TextInput />` component have a `answer` prop (property), which acts as a placeholder for the input, that is blank. Lastly, there is an `inputType` prop, which can be filled in with *text* for text fields, *bigText* for text areas or *number* for number inputs.

Whenever the value of the field changes, the `onChange` method will add the answer to the object containing all the user input to each question in the form.

Radio Inputs

Questions using the `<RadioButtonInput />` component have an `answers` prop, which is set to the answers that can be selected in the form.

The `answers` object will iterate over all the answers and create the radio buttons.

Whenever the input is changed, the ID of the answer is added to the object with all the answers from the form. Furthermore, to improve the layout of the website, the radio buttons are replaced by a dropdown menu, if a multiple-choice question contains more than 5 possible answers. The dropdown menu has the same functionalities as the radio buttons, but is visually more compact. The user does not decide what is displayed, as this is done completely automatically.

Checkbox Inputs

Checkbox questions are generated in `<CheckboxInput />` and it uses props for the question and its answers. To handle inputs, a list is used to keep track of the currently

selected answers. Updates, resulting from a change of the list, will be made to the form inputs object that is used for the entire form.

Subforms

All `<RadioButtonInput />` and `<CheckboxInput />` questions create a `<GenerateSubForms />` component. The props for this component include the `answers` object of the question as well as a list named `visible`, containing the selected options.

Every time an input of a question changes, the sub-form component will iterate through all answers. For each answer, it will check if the answer ID is inside the state variable `visible`. If this is true, then it means that an answer containing a sub-form was selected. In this case, the corresponding sub questions will be displayed, which is otherwise not happening.

Figure 4.3: A question with its sub-form on the right, and without it on the left.

Normal button

The normal button is located inside the `<Section />` component. Once clicked, the function `setDefaultAnswers(section)` will be called. The current version supports normal values for radio button and for check box questions. It iterates through all questions inside the section, and it clicks the default answers. Which answers belong to the default answers is determined by the creator when creating a form.

Overview

The overview is generated by iterating over all the questions, using a `<TreeView>` component for the styling. It uses both the form and the user input for its implementation. While iterating over the questions, it checks whether the question has an answer in the user input object, which indicates whether a question has been filled in or not. This determines the colour of the text in the overview. Questions that have not been answered show up in grey (light and dark mode), while questions that are answered show up in

Pancreas NORMAL

Normal echotexture of the pancreas

☒ True ☐ False

Focal lesions

☐ True ☒ False

dilation of the pancreatic duct

☐ True ☒ False

Figure 4.4: A question, filled out by its normal button

black (light mode) and white (dark mode). The visual difference is used to indicate the user that a question has not been answered yet. This can help them prevent possible oversights in their report. While iterating over the questions, the component will check for sub-forms. Whenever a sub-form is present in an answer field, the component will check if this answer is selected (meaning that the sub-form is visible to the user) and if so, generate the overview for the sub-form as well. The updating of the the overview happens in real time whenever the user selects a field that opens a sub-form.

Pancreas

Normal echotexture of the pancreas

Focal lesions

size of the focal lesions

dilation of the pancreatic duct

Liver

Normal echotexture of the liver

Sharp/Uneven liver margins

Length of the liver (mid-clavicular in cm)

Figure 4.5: Overview of a Form with three questions filled in

Report

The report summarises the user inputs into a standardised form, giving the users the option to copy the result. Initially, the choice was made to display each answered question as **Question Name: Answer**. However, after this feature was implemented, the radiologists requested the option to display custom text whenever an answer was given.

When generating the report, the component iterates over all the questions. It checks whether a question is answered and if so, checks whether the "report" field is present in the form data. If this is true, the component will return that text, otherwise the component will render the original layout for the answer given. This gives the administrators the option to add custom text for a question in the report, helping the final report appear more natural for others to read.

Layout and Scrolling

Using a grid system, the page is split into 3 parts, containing the overview, form and report. Each component can be individually scrolled. This means that, if the form is long, the user will be able to see the overview and report, even if the user is scrolled all the way to the bottom of the form.

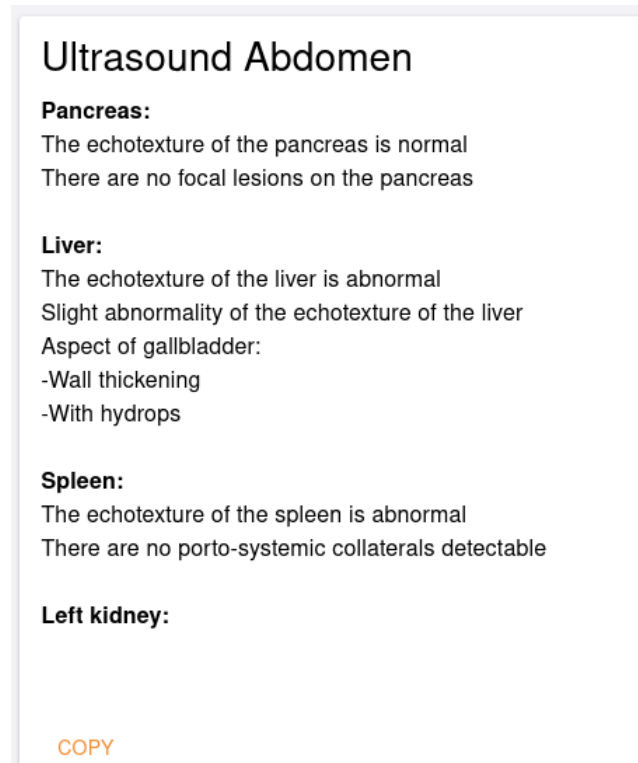
Formula Calculation

During form creation, formulas can be added to a form, these formulas can use regular numbers, mathematical expressions and variables in the form of questions or answers. Each answer can be assigned a numerical value in the form creation phase, after which it can be used to make calculations based on user input. The formulas are stored as a string, where the variables are stored as the question name, surrounded by @ characters. During the user input process in the forms, a component keeps track of all the answers given. Whenever an answer changes, the component checks whether that answer was used in a formula. If this is the case, the formula replaces the name of the formula with the value of the answer given. If a variable has no value assigned to it, 0 is used.

For radio-buttons and true-false-questions, the question is used for the formula, using the value of 1 of the answers. This approach would not work for check-boxes. Therefore, an answer is selected with the score being either the value given to the answer by the creator of the form, or 0 as default. This required a different implementation. Appending -> followed by the answer name to the variable name, allows different answers to be added individually to a formula. On parsing, the variable name is split on -> and a check is made whether the answer is selected. Afterwards, the original variable name is replaced with either the answer value or 0.

When the formula is parsed, it needs to be evaluated. There are multiple options to do this, each with their own risks. For this product, the decision has been made to use the built-in `eval()` function. This does however come with risks. The entire process happens on the client-side, meaning that there are no major risks involved with using

the function. However, code can still be executed from within the function. To prevent this from happening, before the evaluation is executed, the component will check that only certain allowed characters are used. If any letters are present in the formula, the component will not execute the `eval()` but render an error message instead.



Ultrasound Abdomen

Pancreas:
The echotexture of the pancreas is normal
There are no focal lesions on the pancreas

Liver:
The echotexture of the liver is abnormal
Slight abnormality of the echotexture of the liver
Aspect of gallbladder:
-Wall thickening
-With hydrops

Spleen:
The echotexture of the spleen is abnormal
There are no porto-systemic collaterals detectable

Left kidney:

COPY

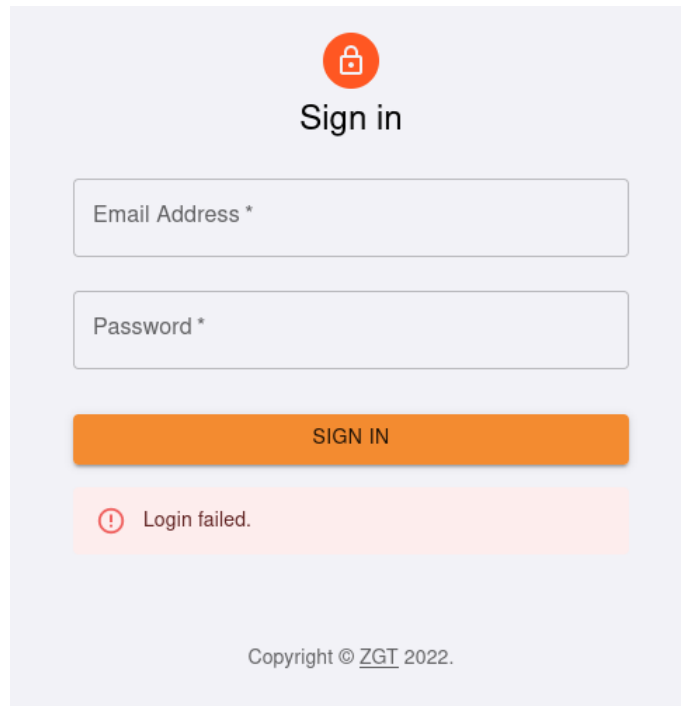
Figure 4.6: Report of a Form

4.3.4 Authentication

Authentication is handled through a login page, using username and password authentication. After successfully logging in, a JWT refresh token, as well as a timestamp, is stored inside the `localStorage`. Redirection to the main page is handled through the `React-Router-DOM`.

4.3.5 Managing forms

The logic for managing forms is inside the `<ManageForms />` component. A button for manage forms is rendered on the homepage in the navigation bar, which redirects to `/form/add`. A table containing all forms and related action buttons is rendered from `<FormTable />`. The design and implementation of the form creation page will be thoroughly discussed in 4.3.6.



The image shows a login form on a light gray background. At the top center is a red circular icon with a white padlock. Below it is the text 'Sign in'. There are two white input fields with gray borders. The first is labeled 'Email Address *' and the second is labeled 'Password *'. Below these fields is a wide orange button with the text 'SIGN IN' in white. Under the button is a red rectangular box containing a white exclamation mark icon and the text 'Login failed.'. At the bottom center, in small gray text, is 'Copyright © ZGT 2022.'

Figure 4.7: Log in page

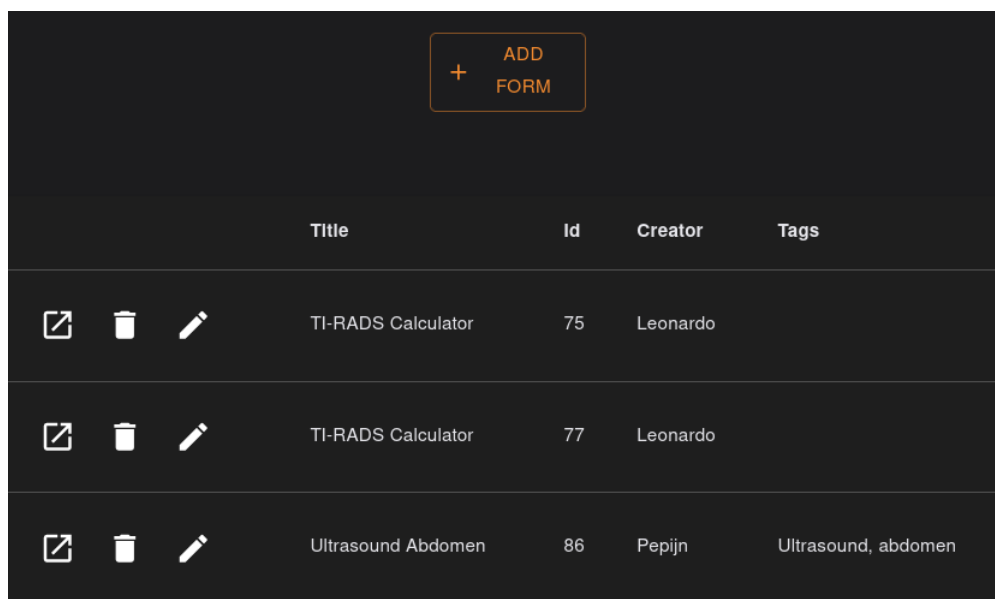
The table will fetch the forms from the back-end and afterwards store them in the state variable `formList`. Clicking the view button and the edit button redirects the user to `/form/:id` and `/form/edit/:id`, respectively. Clicking on the delete button will set a state variable called `id` with the form ID, as well as set the state variable `open` to `true`. This will trigger a modal to pop up. After that, a confirmation text and a button to delete the form are shown.

Editing forms

The functionality of editing the form is similar to the creation of the form. The only difference is that the editing page fetches all the data from the API and displays it on the page in the form of inputs. For more information about the Form creation, refer to Section 4.3.6.

4.3.6 Creating forms

For the implementation of the form creation, the library *React Hook Form* was used. With the library, it is possible to create dynamic forms where fields can be added and removed. There are section input fields which, similar to the database structure, encapsulate all the questions by their category. When a section is created, the section name and ID are pushed to the sections array, which is used to store information about












	Title	Id	Creator	Tags
  	TI-RADS Calculator	75	Leonardo	
  	TI-RADS Calculator	77	Leonardo	
  	Ultrasound Abdomen	86	Pepijn	Ultrasound, abdomen

Figure 4.8: Managing forms page

the section in the database. The ID of the section is generated automatically by the UUID library, but the name is given by the creator of the form. There is an input field for the name of the section, which is used to split the questions into their respective sections. After creating the section, there is a possibility to create questions of different types inside the section. There is an array in place which holds the questions of each section, and when a question is created they are appended to this array. There are 5 types of questions that can be created:

1. Short text field
2. Long text field
3. Radio Button group
4. Boolean Radio Button group
5. Checkbox group

Every question has the field to add a link to a URL. This link can be useful for some complex symptoms which might be uncommon, and the link can then be used to provide extra information.

Text field questions

The first two types of questions have a field for the question name and for the link.

Radio Button group questions

The Radio Button questions contain the same two inputs discussed above, as well as a list of multiple choice answers that are added dynamically by pressing the *Add option* button. Each answer then has a couple options.

- The answer name itself.
- The report input. This is a custom text the creator can set when the question is answered with this answer. If left blank, the standard system text output will be used.
- The score, which is a number input. This can be used in formula calculations.
- Default option. Per radio button group question, you can set one default answer which is selected when the *normal* button is pressed in form filling in. This normal button is mentioned in Section 4.3.3
- Adding sub-question. This adds a question which stems from the answer as parent. You can use any type of question as sub-question.

This hierarchical structure makes it easy to keep track of what question belongs where. In order for the user not to get lost in this hierarchical structure, there is an overview of the so-far created questions on the right.

True/False Questions

The only difference between True/False questions and radio buttons is that there is no possibility to add different answers to the group. The names of the answers in a Boolean(True/False) radio group are predefined as True and False.

Checkbox Questions

The checkbox group questions have almost the same functionality as the radio group field, but there are some differences. It is not possible to add a sub question from the answer, and there can be more default values for a group if nothing is selected on the form filling page.

Formulas

There is the possibility to add some formulas to the form, and create the formula from the fields that were added earlier. For this, a new array is created that holds each formula object with its name. When typing a answer or question name into the formula field, a drop-down menu with matching variable names is expanded. Once a

user selects a variable, the variable is added, surrounded by @. It is also possible to add some "edges" for the result of the formula, which contain a number value and a text field. If a form has a formula with edges, the report will show the text attached to the highest numbered edge that has a number below the score calculated by the formula. When creating the edges, the number and text values are pushed to the edges array of the formula objects.

Tags

Tags can be added to forms by typing them in an input field, where multiple *chips* can be entered. These are stored in the form data object. The tags are used for searching for a form on the main page.

Submitting

When the form is submitted, the form is converted to the structure that is accepted by the server. After that, the form is sent to the back-end. If everything goes well, a success modal is shown, that lets the user know that the form was submitted successfully. In the case that the submission is not successful, an error modal pops up instead and displays the error to the user.

4.4 Back-end

The back-end of Report Companion is built using Django, which is a Python framework and the back-end system was designed using the Model-View-Controller pattern. The back-end handles the model and the controller, whereas the view is handled by the front-end. Within each of the components, security features are integrated to ensure that the product follows the Confidentiality, Integrity & Availability Triad (CIA-Triad). This is further expanded on in Chapter 6

Model

The model defines the database structure, and is implemented inside `app/models.py`. Django has a built-in ORM (Object-Relational-Mapper), which helps us define the database classes and the queries through python code. The database contains classes for forms, formulas, sections, questions and answers. Further explanation on the changes made to the model during the development are discussed in Section 7.4

Next to the models created for the product, Django has a built-in admin dashboard which allows us to create users. ZGT can give the privilege to create, edit and delete forms to those users.

Figure 4.10: Decoded Header Data

```
"alg": "HS256", "typ": "JWT"
```

Figure 4.11: Decoded Payload Data

```
"sub": "1234567890", "name": "John Doe", "iat":  
1516239022
```

Figure 4.12: Signature

```
HMACSHA256(base64UrlEncode(header) + "." +  
base64UrlEncode(payload), 256-bit-secret)
```

Authentication

Authentication takes place in the Django `admins` model. Using Django's built-in dashboard, users can be created and modified. Created users have, by default, the right to create, modify and delete forms. The following endpoints, defined in `urls.py` serve authentication and authorisation:

- GET `admin/`: accessing the admin dashboard
- POST `api/token/`: creates a JWT token based on the login credentials and sends back an access and a refresh token.
- GET `api/token/refresh/`: expected refresh token supplied in request body and returns a new access token.

SQL Injection

The Django ORM automatically sanitises any user input, and it ensures that SQL Injection is prevented.

Test plan and results

5.1 Back-end testing

The back-end is tested through the built-in testing library in Django. It enables for easy user testing using a temporary database file, this ensure no faulty data ends up in the production database.

Before the tests, a standard set of actions are taken. A form is added to the database, and 2 clients are set up. Clients are another feature built into Django, these can be used to simulate a user making requests to the API endpoints. The first client is one without any authentication, whereas the second client is authenticated and should be able to create, edit and delete forms.

The first few tests are general tests on the general functions. They check to see if the added form is correctly represented in the database and can be requested as expected. These tests include for example requesting the title, sections, questions, answers and sub-forms.

Other tests are used to make sure the API endpoints work as expected. The form creation/editing and deleting are tested by both the authenticated and an unauthenticated user. The tests make sure that the unauthenticated user gets a **401 - Unauthorised** error status while the authenticated user gets a **200 - OK** status. These tests also make sure that only the actions requested by the authorised user are executed.

Finally, the client also tests the **GET** request that makes sure that the request that returns all forms works as intended.

To make sure that the entire back-end is tested correctly, a coverage report has been generated for the tests. For this, the library **Coverage.py** was used. The coverage report can be found in Appendix E.

5.2 Front-end testing

Testing the front-end is done using the Selenium IDE, which allows recording actions and play them back. A test suite with three cases was created, in which the following actions are performed and tested:

1. Creating a form with five questions, scores and a formula
2. Fill in the form
3. Modify the form

To run the tests, it is important that an administrator logged into Report Companion with administrator credentials beforehand, since the first and last test need privileged rights to be executed. All tests executed successfully, and they cover a large portion of Report Companion's functionality.

5.2.1 Creating a form

The form creation test creates a replica of TI-RADS. The form contains five questions, with radio buttons and check boxes. To each answer, a score is assigned, which then are reused in a formula. Formula edges are assigned to the formula. If the test succeeds, the form will be created and Selenium will navigate to the form managing page.

5.2.2 Filling in a form

Since a form was created in the previous test, it can be filled in now. This test case starts from the home page and types the name of the form into the search bar. After opening it, it asserts that the correct form was opened, and afterwards it performs two runs, in which it fills out the form.

In the first run, the questions are answered by clicking the checkbox and radio button answers. After each action, it is verified that the answer was no selected. Every time that formula is expected to change, there is an assertion that the result of the formula reflects the changes.

After reloading the page, the second test run will be executed. This time, the form is filled in by clicking the normal button. After clicking the normal button, it is verified, that the correct default options were selected.

5.2.3 Modifying a form

Finally, it needs to be tested if a form can be modified correctly. The test starts from the managing forms page and visits the form that was just created by clicking

on a button in the manage forms page. It is verified that the newly created form was accessed, after that, if the correct form was selected, the test runner will navigate back to the manage forms page and edit the form by clicking on the edit button. In the page of the form editing, a section with a conclusion as a large text-field is added. The update is submitted, and the runner will go back to the form managing page and access the form that was created, where it will verify that the conclusion was added to the form. Finally, the runner goes back to the manage forms page, it deletes the form and it verifies that it was deleted successfully.

Risk Analysis

In the following, a risk and threat analysis of the entire system will be conducted using the threat modelling framework STRIDE, introduced by Microsoft [3]. The STRIDE framework has a great reputation in the security field, which is why it was chosen for this analysis. It captures the majority of the security threats that Report Companion faces.

Table 6.1: List of the Threats of the STRIDE framework [3]

	Threat	Violated Property	Threat Definition
S	Spoofing Identity	Authentication	Pretending to be something or someone other than yourself
T	Tampering with data	Integrity	Modifying something on disk, network, memory, or elsewhere
R	Repudiation	Non-repudiation	Claiming that you didn't do something or were not responsible; can be honest or false
I	Information disclosure	Confidentiality	Providing information to someone not authorized to access it
D	Denial of service	Availability	Exhausting resources needed to provide service
E	Elevation of privilege	Authorization	Allowing someone to do something they are not authorized to do

6.1 Spoofing Identity

The goal of spoofing identity is to find a vulnerability in the authentication process. There are two places, in which one can authenticate: the login page of Report Companion and the admin dashboard of the back-end.

The first instance of Report Companion where this could happen is the login page. An attacker would have to log in as an existing user on the system, in order to spoof their identity. The simplest way to do this is by brute forcing a username and password combination. Although it is simple, it would take a lot of time. Foremost, an attacker would have to brute force both the username and the password, and this would massively increase the time to brute force the password. Even if an attacker managed to get the name of an email address, it would still be hard to brute force the password. A strong password ensures that there will be no reasonable way for anyone to brute-force someone's password. If the end-users don't have a strong password, an attacker might have success using a dictionary attack, making it easy to find that password.

Although the administrators of the ZGT are responsible for creating strong passwords, Further security measures to improve security in this area could be to implement a brute-force prevention tool like Google's reCAPTCHA v3 [4]. Which can prevent and halt suspicious login attempts.

Next, one could spoof their identity by stealing someone else's JWT token, for instance by performing a man in the middle attack. This is easily prevented by operating on HTTPS, which ensures the validity and encryption of the data sent. The task of getting an TLS certificate for Report Companion is deferred to ZGT.

Another option is to falsely authenticate in the admin dashboard. Accessing the admin dashboard is only possible through the back-end, which is isolated from the public internet. Therefore, the only people that could try to log into the admin dashboard are employees at the ZGT. It is expected that to be able to access the back-end, an administrator needs to be logged into the system. This adds another layer of protection before someone could access the back-end. If the administrators choose strong passwords for the back-end accounts, it would be even harder to access the system. Even if an employee somehow got access to the administrator's environment and got logged in, it would be highly suspicious to spend much time to brute force passwords.

In conclusion, it is unlikely that an attacker will be able to spoof their identity in Report Companion.

6.2 Tampering with data

The goal of tampering with data is to modify something anywhere in the application, hence breaching the integrity of the system.

In any case, changing data is only possible through the form managing page, and this feature is only accessible to authorized users. Privilege escalation and the violation of authorization will be further discussed in Section 6.6.

Below, the risks that emerge from tampering with data will be analysed. Data tampering primarily targets the forms in the system, and it could include the malicious modification, deletion or creation of forms.

Violating the integrity of the system would be disastrous. One could modify the formula edges, and thus the outcome of a description of a radiographic image would be wrong. An attacker could also remove sections and questions, thus violating the validity of the forms. Furthermore, an attacker could try to inject code, through SQL Injection in the backend or XSS on the client side. This works in the following way:

The attacker needs a way to submit malicious code to the backend, which gets stored there. Suppose a hacker successfully managed to escalate their privilege and bypass the validation on the form creation or editing page. The data will then go through the controller of the backend, and then it will be stored in the backend. Thanks to Django's built-in ORM, any input is automatically sanitised, and it makes it impossible to perform an SQL injection. The database is therefore protected from malicious user input.

On the other hand, an attacker could submit user input that would be executed in the client side, when a user wants to fill in forms. This is called Cross-site scripting (XSS). Parts of the website that try to evaluate or digest data from the backend pose a vulnerability in this case. This happens when parsing and evaluating the formula string in the form filling in. The front-end runs a check for detecting malicious user input, which prevents XSS.

Tampering with data is unrealistic tough, as there are many security layers that need to be bypassed in order to succeed. Unfortunately, tampering with data is possible if an attacker elevates their privilege. However, there seems to be no way to prevent that.

6.3 Repudiation

Repudiation takes into account that actions cannot be repudiated. In other words, a user should not be able to carry out an action and claim that they did not carry out that action.

One way to ensure repudiation is to log the access and the usage of the website. Django has a built-in logging module, which the back-end is using. This way, access and audit logs are created and stored on the file system. The ZGT has the option to remove the logging option and instead to use a different logging option.

6.4 Information disclosure

The goal of information disclosure is to leak data that should not be available to the public, breaking the confidentiality of the system.

Most of the data stored in the database is intended to be publicly available. The database with the forms does not contain any sensitive or private data. The data that should not be publicly available is the login credentials of the users who are allowed to create a form. Hence, information disclosure could come in the form of a data breach of the list of usernames and their passwords. The most common way for a data breach to happen is through SQL Injection. The Django ORM sanitises any input and therefore SQL Injection is impossible.

Someone with access to the admin dashboard with malicious intents could leak the data. Even though this is very unlikely to happen, password encryption was added to the system. By default, Django does not store passwords in plain text, but instead uses a secure cryptographic hashing function. The default hash function is Password-Based Key Derivation Function 2 (PBKDF2) using SHA-256 [5]. PBKDF2 runs 260000 iterations which slows down a decryption attempt. On top of that, every encryption takes in a salt, which adds security and prevents attackers from cracking multiple passwords at the same time.

6.5 Denial of Service

The goal of a Denial of Service is to overload the system, thus making it unavailable. The risk in the system is to repeatedly call one of the API routes and to overload the back-end with excessive requests. The API endpoints are publicly available, and they can be accessed by anyone on the internet. Therefore, the system could be a target for

DoS attacks. It is not possible to add DoS protection in the application itself. Instead, the preferred way to prevent DoS attacks, is to add a layer of protection in between the backend and the front end, which could be a proxy protection.

Since the deployment of Report Companion is done by ZGT, there is no way in which the threat can be circumvented within this project. For this reason, it is urged that ZGT to add a layer of protection such as a DoS proxy.

6.6 Elevation of privilege

The main goal of privilege elevation is to exploit a vulnerability in the authorization process.

Report Companion is designed by the principle of the least privilege, which states that any user should be granted the least amount of privileges and rights to operate, in order to limit the inappropriate use of privilege, accidents and other security risks [6]. Normal users only have the ability to fill in forms, whereas radiologists can only modify and delete forms.

The only way to escalate the user privilege is for a person without an account to obtain the ability to create forms, edit forms and delete forms.

The authorization process is developed using JWT. The JWT tokens are encrypted using HS256, which combines HMAC together with the secure cryptographic hash function SHA256. Even though a secure hashing function was selected, brute forcing the access token is still possible, as tools have been developed, to crack JWTs fast. To prevent brute forcing the access token, it is extremely important that the administrators choose a safe password. This responsibility is left to the administrators.

Another way to get access to the JWT is by performing a man in the middle attack, where someone would get the token by intercepting the request. In order to get a secure connection to the server, it is important that Report Companion is operating on HTTPS. Since the ZGT handles the deployment of Report Companion, the administrators of the ZGT are responsible for this. Since all of their pages are running on HTTPS, they are expected to get a valid TLS certificate for Report Companion as well.

There are other ways to perform privilege escalation by spoofing one's identity. The protection against those methods are listed in Section 6.1).

In conclusion, it is improbable that an attacker manages to escalate their privileges.

6.7 Conclusion

The final version of Report Companion provides sufficient security protections against **tampering with data**, **Repudiation**, **Information Disclosure** and **Elevation of privilege**. Some responsibility is left for **Spoofing Identity**, since the administrators need to ensure that a secure password is chosen. Protection against **Denial of Service** is fully left to ZGT, since it is hard to influence this outside of the deployment phase.

Evaluation

7.1 Reflection on Planning

Before starting with the development of the product, a plan was created that included all the necessary steps to complete the project in time. This plan can be found in Figure 7.1. The plan was executed as is for the first two weeks, however development of the actual product started earlier than expected. Both front- and back-end development started in week 3 and substantial progress was made in the first week, such that an MVP was already finished in week 6.

In the original plan, testing was supposed to start in week 4, but since the focus was set on having an MVP as soon as possible, testing was started later in week 6. Although testing was started later than expected, the product is still thoroughly tested through both unit and usability tests, as mentioned in 5.2 and 5.1.

Unfortunately, it was not possible to have explicit user testing sessions to allow for further feedback on the user experience of the application. Despite of that, a lot of feedback on features and interactions with the application were gathered. This feedback was gathered because of the nature of the meetings where the progress that was made was showed to the product owner and the radiologists, who will be using the product. This helped ensure a smooth user experience specifically drafted for the workflow of radiologists describing radiographical images.

The initial back-end structure was finished ahead of schedule. However, due to requirements changes, minor oversights and shifts of focus, the back-end kept seeing changes and updates in its structure up until around the time the front-end was finished.

The basic functionality of the form creation was working faster than was planned for. The development, however, still continued up until week 9 when testing and general

Week	Tasks	Deliverables
0	Form project group Discuss and choose a project	
1	Contact project contact Find and contact a project supervisor Gather project requirements	
2	Create first Lo-Fi prototypes and gather feedback Have the first meeting with Project supervisor Form a project proposal	Initial design prototype
3	Determine and start on back-end structure Perform final touches and submit the project proposal Peer feedback session Design Class and Use Case Diagrams	Project proposal
4	Finish and test back-end Structure Start on Hi-Fi prototype for front-end	Second design prototype
5	Gather feedback on front-end prototype Start on front-end development Peer feedback session	
6	Continue front-end development	
7	Start on project poster Peer feedback session	
8	Finalize project poster Start final testing of the product	Project poster
9	Finalize product Peer feedback session	Final product
10	Finalize report Poster showcase session	Project report

Table 1: Overview of the tasks and deliverables for each week

Figure 7.1: Original plan for the project

bugs were taken care of. While the development of the front-end went mostly smooth, performance issues were encountered throughout the later stages, when testing the system with more extensive forms. These performance issues were caused by components re-rendering unnecessarily often. This meant that when a form got very big, the system would take longer to load components. Solving these issues required some major alterations to existing code, and cost a lot of time. However, the planning left space for this, and it caused no major issues.

7.2 Reflection on Meetings

The weekly meetings with the supervisor were helpful in order to get feedback on the progress that was made on the project at that point in time, as well as advice on the planning of the project. In these meetings, the communication with the product owner in face-to-face meetings was discussed. This proved helpful, especially when boundaries of what was possible to implement in the amount of time allotted to the project needed to be set.

During the initial meetings with ZGT, the focus was put on gathering requirements and setting a scope of the project. This was achieved by creating and explaining multiple prototypes, both Hi-Fi and Lo-Fi and gathering feedback on them. Images of the prototype can be found in B. These prototypes were used to show what was possible, and gave new insights to the wishes of the client, e.g. the overview component as used in the final product. While the product owner had some idea as to how the project would look, to confirm this and discuss alternatives, various Lo-Fi prototypes were made. These prototypes were used to make design decisions by taking a very broad range of options, and then narrowing it down using Hi-Fi prototypes for confirmation. As a result of presenting the Lo-Fi prototypes to the product owner and radiologists, they decided that a search bar with auto-completion would be most suited for the homepage, and the form filling displayed on a single page would suit best. During the later meetings, demos of the updated product were shown to the client. During these meetings, feedback was gathered on the new functionalities, the UX and the design. Usually, this brought new requirements along the way based on the practical insights by the radiologists. In that sense, what was experienced during the meetings was that they were mostly to gather new information rather than feedback on the system that was implemented up until then, since the radiologists were mostly always enthusiastic of what was done up to then.

There was one meeting that was particularly difficult due to the large amount of feed-

back and feature requests made. Although at first the group had difficulties managing and prioritizing the large amount of changes, after meeting with the supervisor, a plan was made to manage and handle the different requirements. First, the new requirements were ranked using MoSCoW, after which the priorities were validated and discussed in the next meeting. This was a boost in confidence in the ability to finish the project, and set a clear line for the client to know what would be implemented and what would not. The meetings held with the ZGT provided the experience necessary to deal and work with the requirements of a stakeholder. With the newly acquired skills, such as discussing and understanding requirements, stating what is and what is not possible and presenting new changes, following meetings will be effective and efficient. Dealing with such challenges can now be done in a better way.

7.3 Distribution of Tasks

- Andrei: Form Creation, Form Validation, Form Editing, Code documentation
- Pepijn: Form Filling-in, Back-end, Back-end unit testing, e-mail contact, Code documentation
- Leonardo: Form Filling-in, Managing Forms, Homepage, Front-end testing, Code documentation
- Daan: Styling, Presentations, Themes, Poster, e-mail contact, Quality Assurance
- Omid: Styling, Homepage, Themes

Everybody has been working on creating prototypes, writing the project proposal, the reflection report and the final report, and internally peer-reviewing each other.

7.4 Changes to the initial system

Database

The design of the database model has gone through several changes throughout the project. During the initial product design brainstorm, a basic design was made A.1, which has undergone minor changes, based on further requirements received later during the project. In the final database design A.2, the table **FormulaEdges** is added. This table holds information on what text needs to be displayed based on the result of a formula. This gives the form creator the option to directly add scores and recommendations like for example used in this TI-RADS calculator [7] into the report based on the user's input.

Furthermore, a table has been added in between the form and the questions, to separate questions into specific sections. This can be seen in the final version of the database schema in Appendix A.2. This extra table allows the option to add more structure to forms, improving the user experience.

The biggest and most obvious change is the removal of the input tables. Initially, the option to store user information was left open and taken into account for the database design. Though as the project went along, the decision to not include the option to store user information was made due to privacy, security and time constraints.

The final database design also contains tables that are required for the authorization and authentication of the users. These were generated by Django. These handle the session data and login credentials of the users.

Form creation options

The form creation page has undergone significant changes since the proposal was written. Boundaries were added for the formula's result, and the overview of the form was also implemented for the form creation. Besides this, there was also added a new report field for the multiple-choice question, which makes the result more readable for the end-user. There was also added a new field to each question that references the topic on the internet.

7.5 Final Result

In general, all major features that were needed in order to create a well functioning product have been implemented. All necessary features and a handful of optional features that were requested by ZGT were included in the final product. This includes the following:

- Creating and managing forms
- Filling in forms
- Reducing the number of interactions when filling in a form
- Changing the report text
- Linking questions to a URL
- Formula calculations

- Bookmarking forms in the browser
- Dark Mode
- Dockerisation

Due to time constraints, some features that were requested during the development stage did not make their way into the final end product. These features were the following:

- Multiple language support: This feature entails creating a special tag for English and Dutch forms, and to filter forms by those two languages.
- Enriched text: the text in the report could be styled or highlighted, if the answer differs from the default answers.
- Adding forms to a favourite list: this feature is about saving forms in a list, so that they can be accessed quickly. This feature turned out to be unimportant, since it is possible to bookmark forms in the browser.
- Answers stored in a database: The client said that it could be nice to store the answers in the database. However, they also specified that this is unimportant, and that this will be a part of a different version of Report Companion. See Section 8 for the future work.

All in all, we as a group are satisfied with the resulting product. Both the design and the functionality turned out to be what was expected when planning out the project. Overall, everything that was planned for and was regarded as important has been implemented. ZGT have also been happy with the design of the product both for the light and dark mode.

Future Work

8.1 Extension of Report Companion

8.1.1 User Experience

The user experience can be improved in many ways. Unfortunately, due to the tight schedule for this project, user testing was not performed. Therefore, testing the application and getting insights on the user experience would be the next step in order to improve the system. Not only it would give valuable insights, the product would also be used by a wider audience like for example other radiologists working in the ZGT or other hospitals. With more feedback, more potential improvements and problems could be revealed.

8.1.2 Data Collection

The application has been designed to be easily usable within another component, such that it can be used to extend other hospital software in order to better integrate to the workflow of the radiologists. The application does not store privacy-sensitive data, such as patient records. For further improvement of the app, machine learning could be utilized in order to implement new features, such as speech recognition. However, by introducing new features that require data collection, new security risks might occur that need to be solved. Not only security but also privacy issues regarding the data of the patients could pose a challenge.

8.1.3 Functionality

A list of functionality features can be found in table 8.1.

8.2 New Version of Report Companion

Some features could not be implemented due to time constraints. Therefore, if further development is undertaken, those features could be used to further improve the application. The speech to text recognition can be implemented within the system to make it easier for the Radiologists to enter data into the forms and to quickly edit information. The approach of using speech recognition has been attempted previously, but this did not have satisfactory results. Therefore, trying to implement it again with the new app can be a massive improvement.

Table 8.1: List of features for an extension of Report Companion

Feature	Description
Clear Button	A button, with which the Radiologists can clear all data that has been entered into the current form.
Multiple Themes	Implement the option to have multiple themes that can be customized by the current user and stored to be able use it later
Statistics	Store and show statistics that display the frequency of certain answers; this can be extended to show other metrics as well, but it requires data collection (See above)
Enriched Text	Text inputs with the option to change size, color or font styles
Multiple languages	Connecting forms of the same nature but of different language to each other and giving users the possibility to change language of the entire system

Summary

To summarize, with this product, Radiologists are able to structure and standardize their workflow. The product helps reduce errors and oversights, while laying a foundation for future (machine learning) innovations in the medical field. By using secure platforms and implementing further security measures, the risks of this product are minimized, without getting in the way of the optimized user experience. By taking the current radiologist workflow and adapting the new system around that, ensures that the product satisfies the customer.

The development process, although not without hick-ups, produced a product that matches the requirements of the client, and should see use in the near future.

Bibliography

- [1] D. Yaeger, “Radiology needs better workflow tools,” *Radiology Today*, vol. 16, no. 6, p. 12, 2015.
- [2] G. A. T. Peter A. Marcovici, “Journal club: Structured radiology reports are more complete and more effective than unstructured reports,” *American Journal of Roentgenology*, vol. 203, no. 6, pp. 1265–1271, 2014.
- [3] N. Shevchenko, T. A. Chick, P. O’Riordan, T. P. Scanlon, and C. Woody, “Threat modeling: a summary of available methods,” Carnegie Mellon University Software Engineering Institute Pittsburgh United . . . , Tech. Rep., 2018.
- [4] “recaptcha,” <https://developers.google.com/recaptcha/>, accessed: 2022-04-21.
- [5] “Password management in django,” <https://docs.djangoproject.com/en/4.0/topics/auth/passwords/>, accessed: 2022-04-11.
- [6] J. H. Saltzer and M. D. Schroeder, “The protection of information in computer systems,” *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278–1308, 1975.
- [7] “Ti-rads calculator – calculates ti-rads score,” <https://tiradscalculator.com/>, accessed: 2022-04-11.

Appendix A

UML Diagrams

Class Diagram

The initial database that was designed for the storing of forms and their questions can be found below.

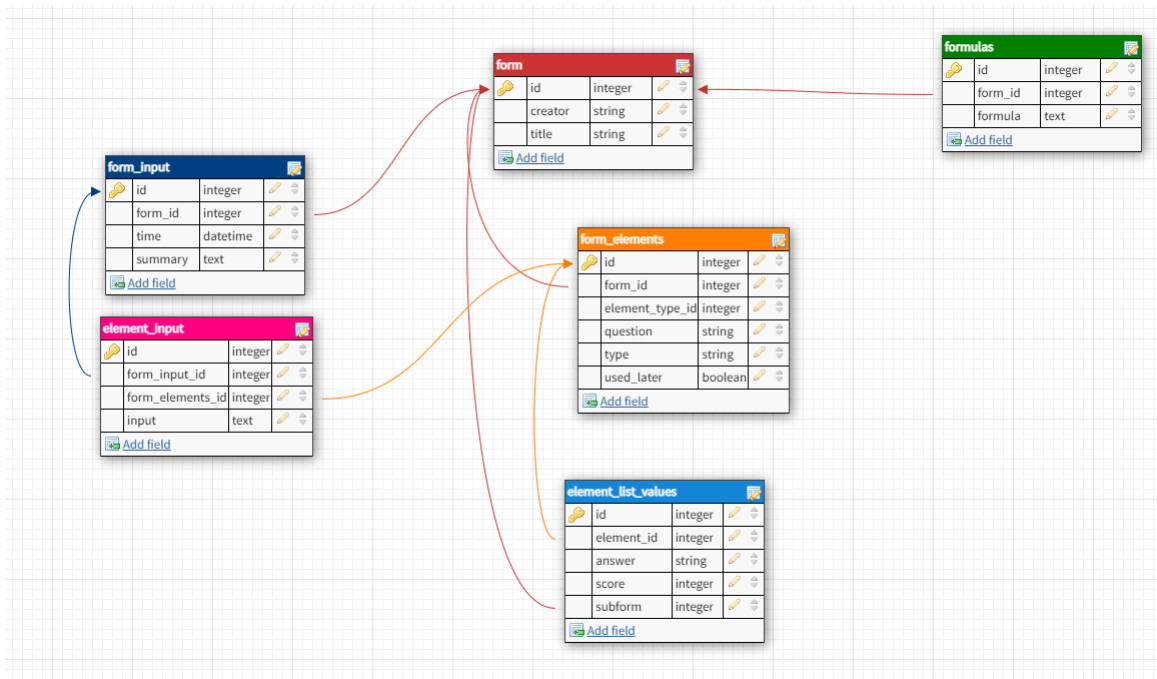


Figure A.1: Initial database schema

The final database design as used in the final product.

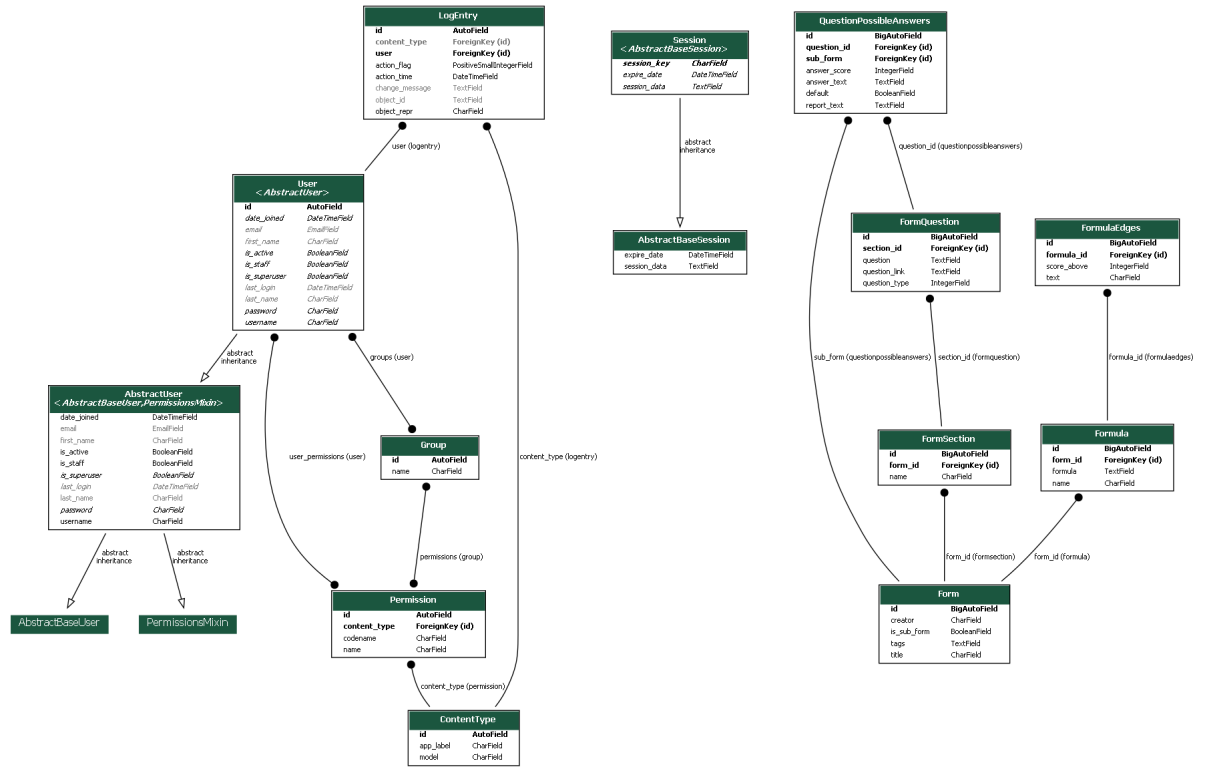


Figure A.2: Final database schema

Appendix B

Prototypes

B.1 LoFi-Prototypes

Categories	
AB Abdominal Radiology	CTA Heart Fractional Reserve
BR Breast Imaging	CT Angiography
CR Cardiac Radiology	CT and MR Liver
CH Chest Radiology	CT Kidney
CT Computed Tomography	CT Knee
ER Emergency Radiology	CT Chest
FL Fluoroscopy	CT Wrist/Hand
GI Gastrointestinal Radiology	
GU Genitourinary Radiology	
GU Genitourinary Radiology	
HN Head and Neck	
IR Interventional Radiology	

Figure B.1: Dropdown menu to find the correct form

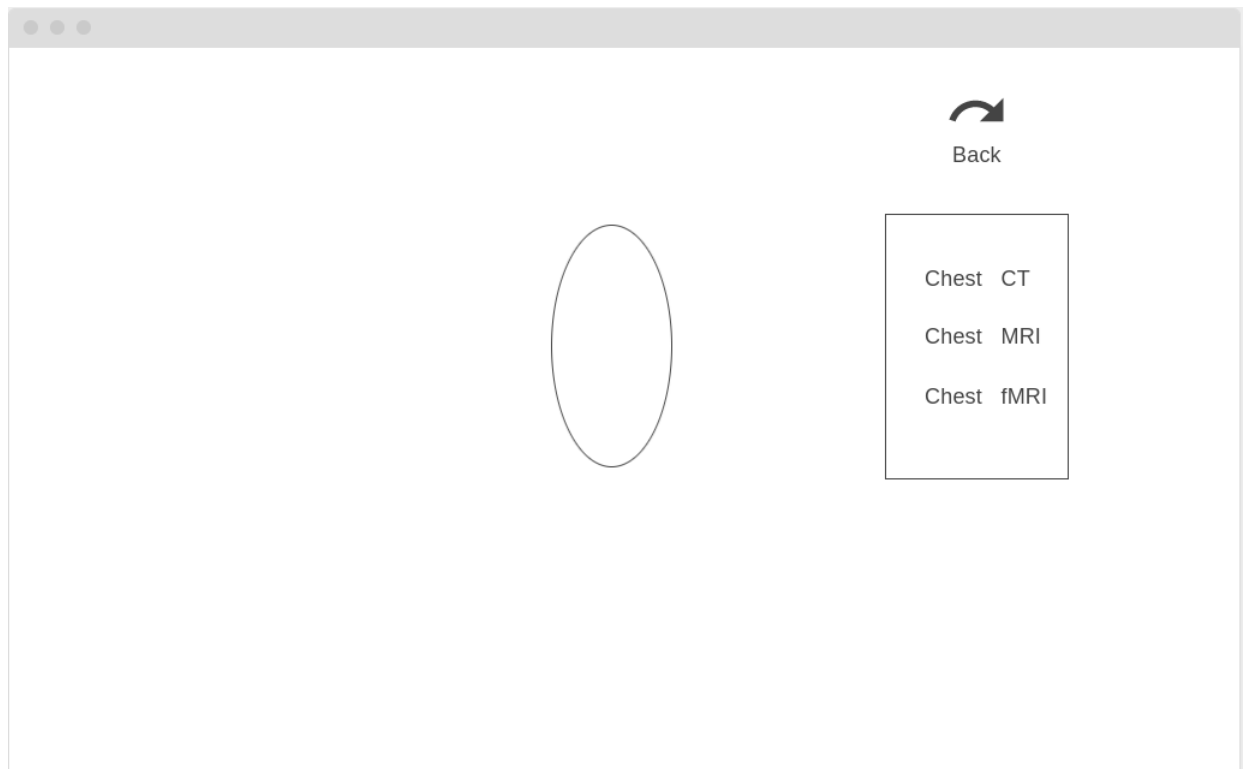


Figure B.2: Homepage showing a silhouette with clickable bodyparts, upon which forms are opened

CT report

Question 1 (Open New Questions):

Yes

No

Question 2:

CT report - COVID-19 Follow Up Questions

Question 1:

Question 2:

Finish

Question 3:

Figure B.3: Form with popup modal for filling in questions

Lorem ipsum dolor sit amet, consectetur adipiscing elit?

Sed tincidunt libero eu orci mattis

sit amet facilisis leo bibendum

Proin sodales leo id

elementum vitae odio vel

Sed quis lacinia leo

Sed urna ipsum

imperdiet at diam eu

Go Back

Next

Figure B.4: Form with questions using up a single page

B.2 HiFi-Prototype

Website

Template title	Information	Information
CTA Heart Fractional Flow Reserve	Text	Text
Cardiac MRI: Right Heart Failure	Text	Text
CT KHK (CT coronary heart disease)	Text	Text

Figure B.5: Searchbar for querying the forms

Overview

General

- ☒ Clinical information
- ☐ Indication
- ☒ Procedure
- ☒ Protocol
- ☒ Comparison exam
- ☒ Study quality

Pancreas

- ☒ Echotexture
- ☒ Focal lesions
- ☒ Dilation of duct

Liver

- ☒ Echotexture
- ☒ Steatosis
- ☒ Liver margins
- ☒ Length
- ☒ Focal lesions
- ☒ Dilation biliary tract
- ☒ Diameter bile duct
- ☒ Choledocholithiasis
- ☒ Aspect gall bladder
- ☒ Cholelithiasis

Ultrasound Abdomen

General

Clinical information
Please type here...

Indication
Please type here...

Procedure
Ultrasound Abdomen

Protocol
Ultrasound of the Abdomen standard protocol

Comparison exam
☐ Yes
☐ No

Study quality
☐ Good
☐ Acceptable

Website

Overview report

General:
 Clinical information: text
 Indication: text text text
 Procedure: Ultrasound ab
 Protocol: Ultrasound of th
 Comparison exam: availa
 Study quality: average

Findings:
Pancreas:
 Normal echotexture of th
 Focal lesions: Yes, 5mm
 Dilation of the pancreatic

Liver:
 Normal echotexture of th
 Liver margins: Sharp
 Lenath of liver (mid-clavi

Figure B.6: Filling in a form

Colour Schemes

C.1 Coluor schemes of final product



Figure C.1: Colour scheme light mode



Figure C.2: Colour scheme dark mode

C.2 Colour schemes for prototyping

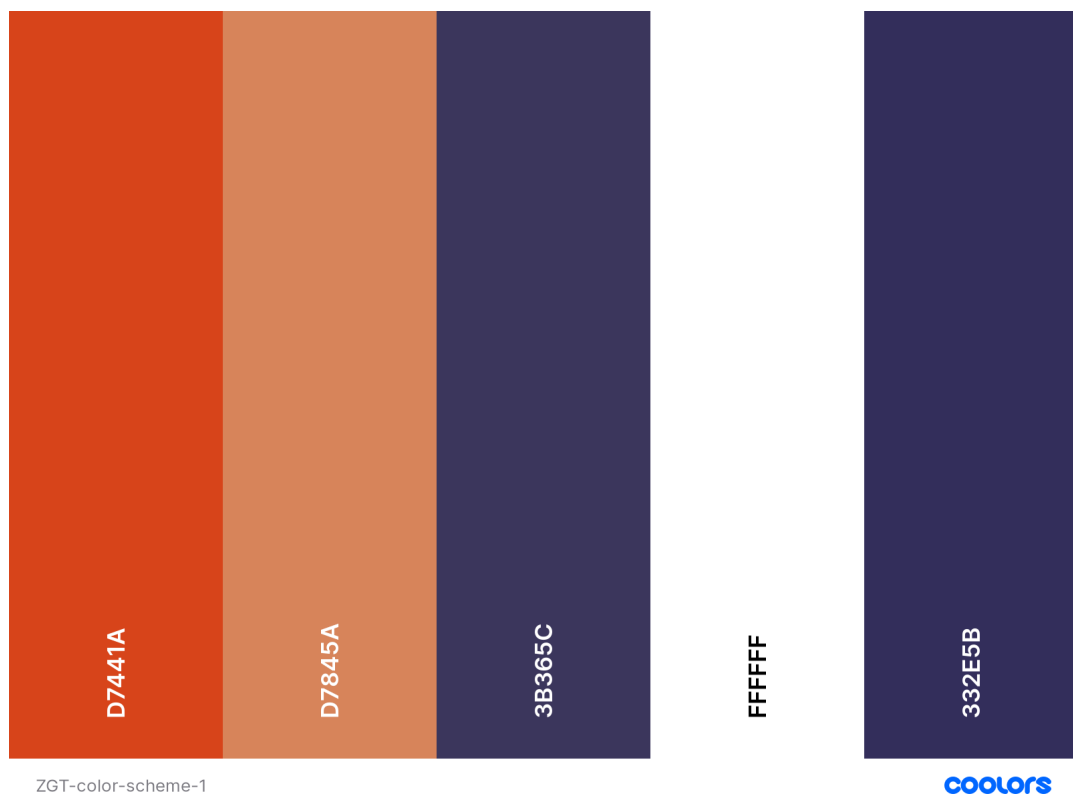


Figure C.3: Colour scheme 1

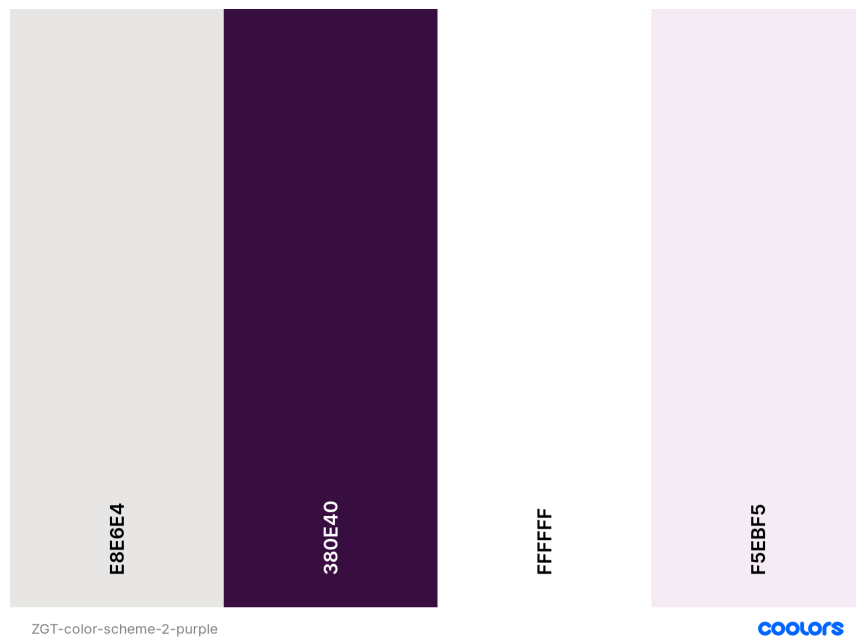


Figure C.4: Colour scheme 2

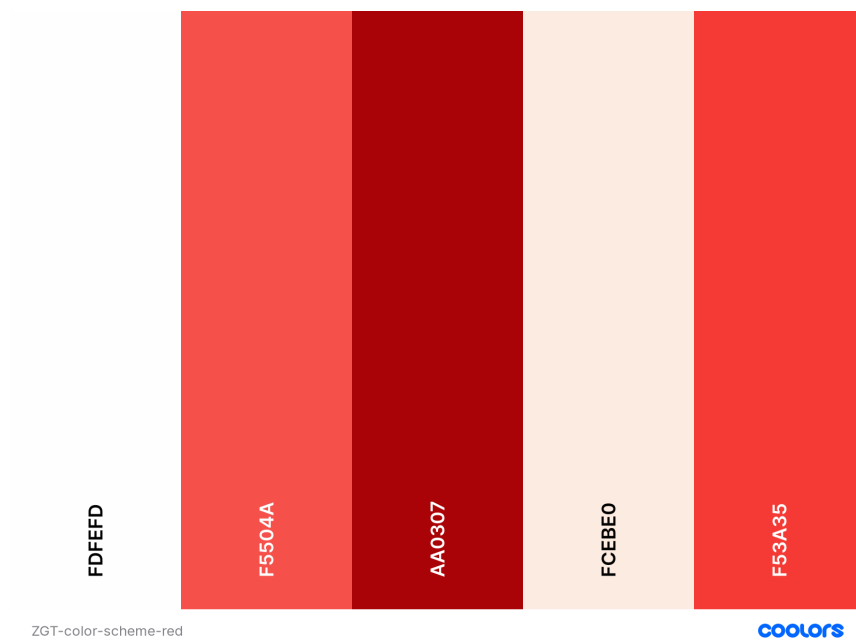


Figure C.5: Colour scheme 3

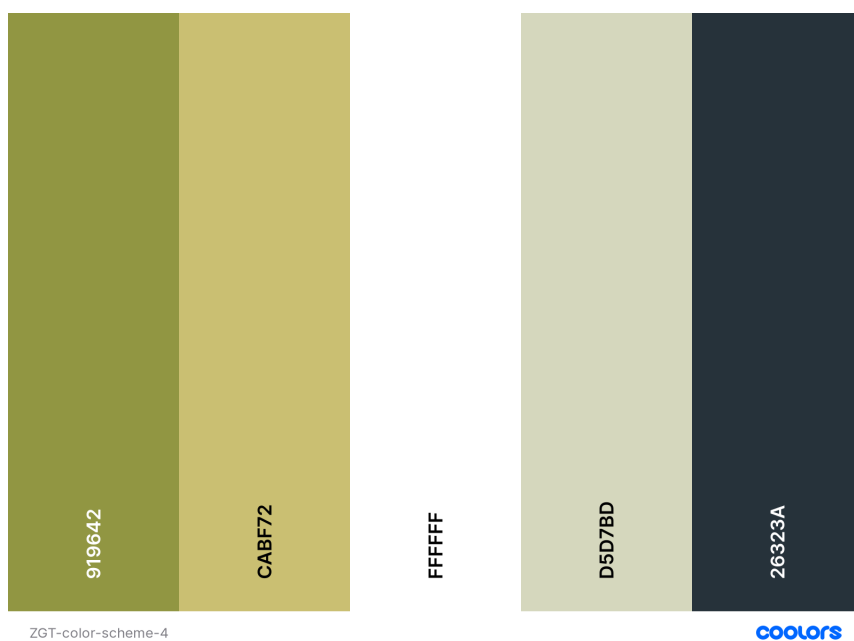


Figure C.6: Colour scheme 4

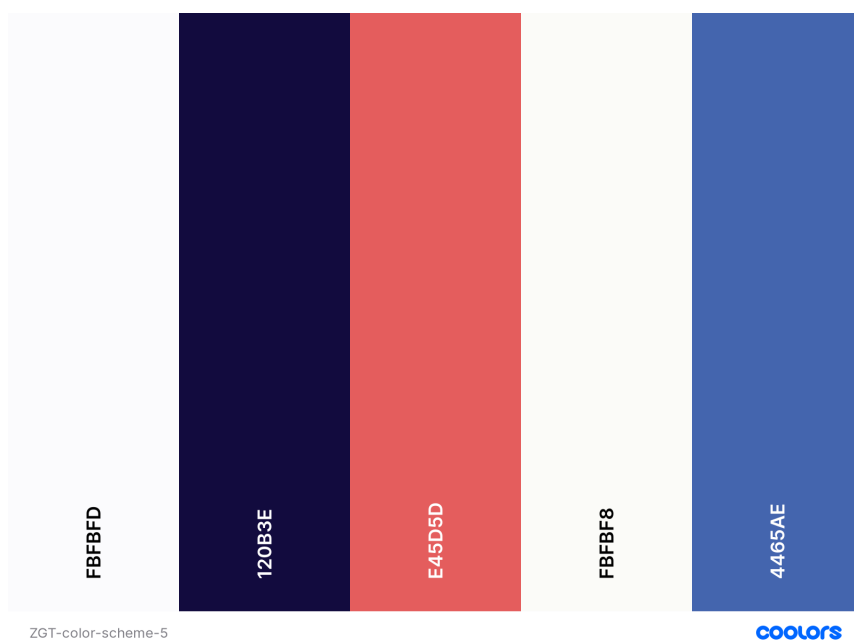


Figure C.7: Colour scheme 5

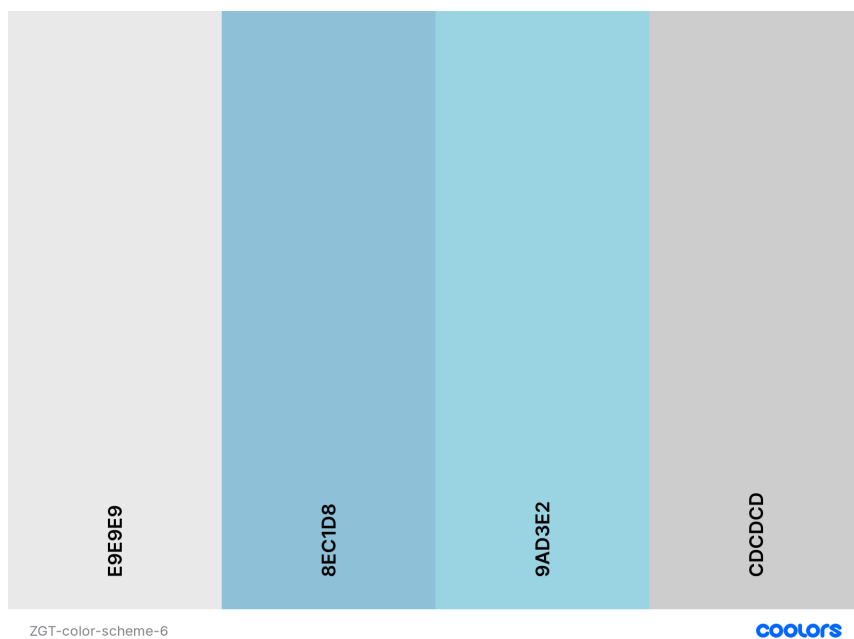


Figure C.8: Colour scheme 6

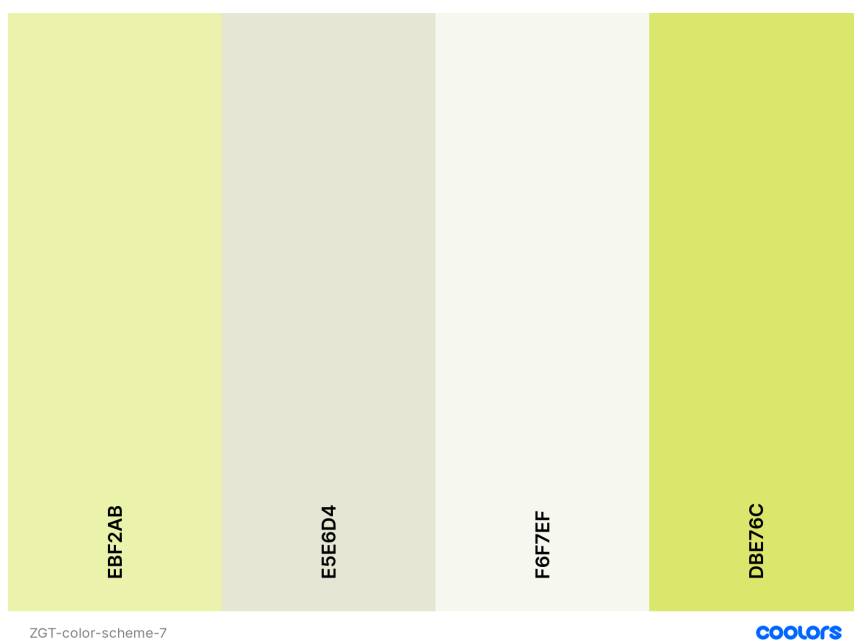


Figure C.9: Colour scheme 7

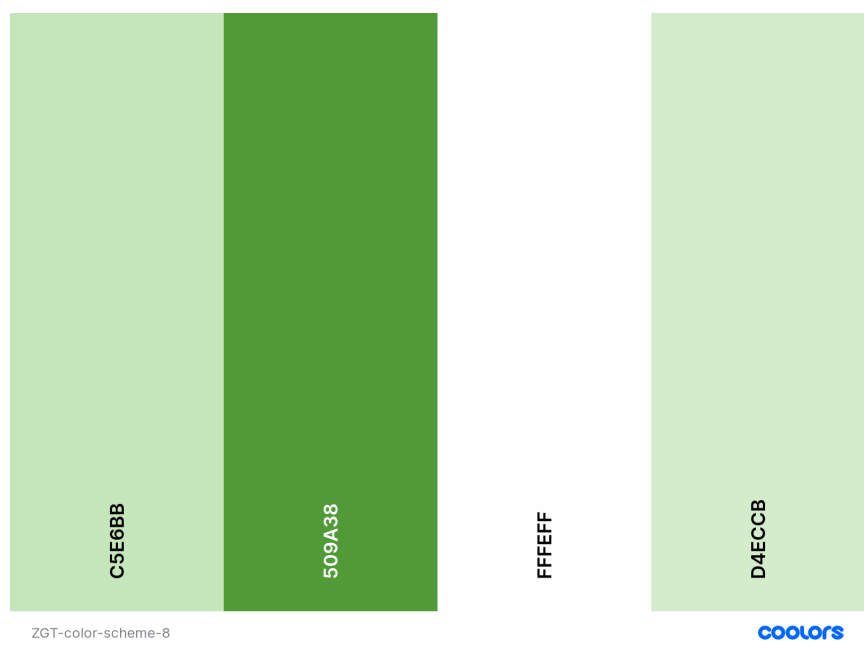


Figure C.10: Colour scheme 8

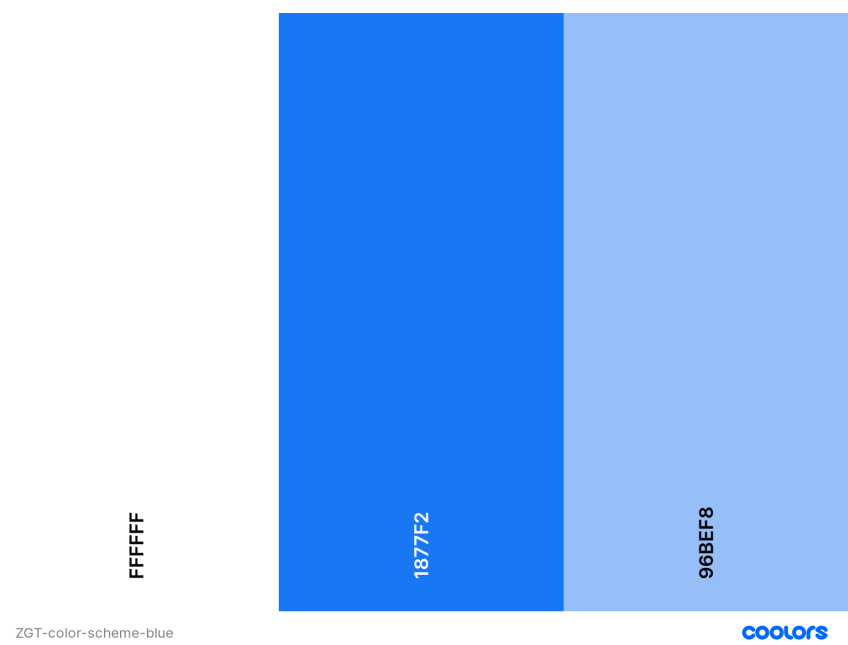


Figure C.11: Colour scheme 9

Appendix D

API requests

D.1 Example Form JSON

```
{
  "tags": "[]",
  "id": 75,
  "form_title": "TI-RADS Calculator",
  "creator": "Leonardo",
  "formulas": [
    {
      "id": 22,
      "name": "Total Points",
      "formula":
"@Composition@ + @Echogenicity@+ @Shape@+@Margin@+
@Echogenic Foci->None or large comet-tail artifacts@+
@Echogenic Foci->Macrocalcifications@
+@Echogenic Foci->Peripheral (rim) calcifications@
+@Echogenic Foci->Punctate echogenic foci@",
      "edges": [[0, "TR1"], [2, "TR2"], [3, "TR3"], [4, "TR4"], [7, "TR5"]]
    }
  ],
  "sections": [
    {
      "name": "Thyroid",
      "id": 110,
      "questions": [
        {
          "id": 232,
          "question": "Composition",
          "type": 2,
          "link": "https://en.wikipedia.org/wiki/Thyroid",
          "answers": [
            {
              "id": 354,
              "question": "Cystic or almost completely cystic",
              "score": 0,
              "default": True,
              "sub_form": {
                "tags": "[]",
                "id": 105,
                "form_title": "Ultrasound Abdomen",
                "creator": "Leonardo",
                "formulas": [],
                "sections": [
                  {
                    "name": "Pancreas",
                    "id": 153,
                    "questions": [
                      {
                        "id": 349,
                        "question": "Abnormality of the echotexture of the pancreas",

```

```

        "type": 2,
        "answers": [
          {
            "id": 600,
            "question": "Hypo-echogenic",
            "score": 0,
            "report": "the echotexture of the pancreas is Hypo-echogenic"
          },
          {
            "id": 601,
            "question": "Hyper-echogenic",
            "score": 0,
            "report": "the echotexture of the pancreas is Hyper-echogenic"
          }
        ]
      },
    ],
  },
},
{
  "id": 355,
  "question": "Spongiform",
  "score": 0
},
{
  "id": 356,
  "question": "Mixed cystic and solid",
  "score": 1
},
{
  "id": 357,
  "question": "Solid or almost completely solid",
  "score": 2
}
],
},
{
  "id": 233,
  "question": "Echogenicity",
  "type": 2,
  "link": "https://www.medicalnewstoday.com/articles/325298",
  "answers": [
    {
      "id": 358,
      "question": "Anechoic",
      "score": 0,
      "default": True
    },
    {
      "id": 359,
      "question": "Hyperechoic or isoechoic",
      "score": 1
    },
    {
      "id": 360,
      "question": "Hypoechoic",
      "score": 2
    },
    {
      "id": 361,
      "question": "Very hypoechoic ",
      "score": 3
    }
  ]
},
},
{
  "id": 234,
  "question": "Shape",
  "type": 2,
  "answers": [
    {
      "id": 362,
      "question": "Wider-than-tall",

```

```

    "score": 0,
    "default": True
},
{
    "id": 363,
    "question": "Taller-than-wide",
    "score": 3
}
],
{
    "id": 235,
    "question": "Margin",
    "type": 2,
    "link": "https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4968280/",
    "answers": [
        {
            "id": 364,
            "question": "Smooth",
            "score": 0,
            "default": True
        },
        {
            "id": 365,
            "question": "III-defined",
            "score": 0
        },
        {
            "id": 366,
            "question": "Lobulated or irregular",
            "score": 2
        },
        {
            "id": 367,
            "question": "Extra-thyroidal extension",
            "score": 3
        }
    ]
},
{
    "id": 236,
    "question": "Echogenic Foci",
    "link": "https://www.verywellfamily.com/what-is-echogenic-focus-4584330",
    "type": 3,
    "answers": [
        {
            "id": 368,
            "question": "None or large comet-tail artifacts",
            "score": 0
        },
        {
            "id": 369,
            "question": "Macrocalcifications",
            "score": 1
        },
        {
            "id": 370,
            "question": "Peripheral (rim) calcifications",
            "score": 2
        },
        {
            "id": 371,
            "question": "Punctate echogenic foci",
            "score": 3
        }
    ]
}
]
}
}
]
```

Appendix E

Testing Report

E.1 Back-end Unit testing

Coverage report: 97%



<i>Module</i>	<i>statements</i>	<i>missing</i>	<i>excluded</i>	<i>coverage</i>
RadForms/__init__.py	0	0	0	100%
RadForms/settings.py	22	0	0	100%
RadForms/urls.py	4	0	0	100%
RadForms_App/__init__.py	0	0	0	100%
RadForms_App/admin.py	1	0	0	100%
RadForms_App/apps.py	4	0	0	100%
RadForms_App/migrations/0001_initial.py	6	0	0	100%
RadForms_App/migrations/__init__.py	0	0	0	100%
RadForms_App/models.py	153	5	0	97%
RadForms_App/tests.py	92	2	0	98%
RadForms_App/urls.py	3	0	0	100%
RadForms_App/views.py	43	2	0	95%
manage.py	12	2	0	83%
Total	340	11	0	97%

coverage.py v6.3.2, created at 2022-04-14 11:48 +0200

Figure E.1: Coverage Report

Setup Instructions

F.1 Development Setup

F.1.1 Front end

1. Open the folder with the front-end source code in a terminal
2. Run `npm install` to install the required dependencies
3. Run `npm start` to start the application
4. Visit `localhost:3000` to access the running website

F.1.2 Backend end

To run the backend, run the following:

```
docker-compose up db web
```

Otherwise, you can do the following:

1. Open the folder with the back-end source code in a terminal
2. Create a virtual environment by running `virtualenv venv`
3. Activate the virtual environment by running `source venv/bin/activate`
4. Run `pip3 install -r requirements.txt` to install the required dependencies
5. Launch a MySQL server at port 3066 with the same credentials as in `.env` This can be done manually, or with `docker-compose up db`, ran from the parent directory
6. Load the environment variables from `.env` into your environment
7. Run `python manage.py runserver` to start the backend at `localhost:8000`

F.2 Deployment using docker-compose

To start both the Backend and the Frontend, you need to run the following command in a terminal:

```
docker-compose build && docker-compose up
```

Next, you need to create a superuser inside the backend.

1. Run `docker ps` to see the running docker containers
2. Copy the ID of the container with the name `report_companion_web_run` into your clipboard
3. Run `docker exec -it <container_id> sh`
4. Change to the backend directory by running `cd ZGT_backend`
5. Run `python manage.py createsuperuser`
6. Enter a **secure** username and password combination and store the credentials safely
7. Press CTRL+D to exit the container environment

To stop the docker containers, run

```
docker-compose stop
```