School of Electronic Engineering
and Computer Science

Final Report

Programme of study:
BSc Computer Science

Project Title:
Automation of the Royal College of Radiologists' Job Description and Advisory Appointment Committee Processes

Supervisor:
Dr Fabrizio Smeraldi

Student Name:
Christian Juresh

Final Year
Undergraduate Project 2023/24

Queen Mary
University of London

Date: April 27, 2024

Disclaimer

This disclaimer serves as the primary document regarding the terms of use and distribution of this report and its supporting materials. The information herein supersedes any prior agreements.

I hereby declare that this report is my own original work, except where stated otherwise. No written content from this report or the supporting materials has been shared with any other individual prior to submission.

Redistribution of this report is encouraged for any purpose, under the condition that my full name is retained in the document.

The use and distribution of any source code not included in this report are prohibited, except by staff at Queen Mary University of London or the Royal College of Radiologists.

All statements beyond this point represent my own personal opinions and interpretations and should not be regarded as objective facts. They do not necessarily reflect the views of any organisation.

Acknowledgements

Everything we do is a collaborative effort of hundreds of people, from the people we come across in our personal lives to the developers who create the tools we rely on. I extend gratitude to them all.

However, there are a few people who have had such a significant impact on my life that they deserve personal recognition directly from me.

The reassurance of having a knowledgable and supportive supervisor was invaluable. Dr. Fabrizio Smeraldi provided me with detailed feedback, answered all of my queries, motivated me with his positive affirmations, and offered infinite guidance.

Without the work of all the staff at the Royal College of Radiologists, this project could not have been conceived. I am particularly grateful to Lucy Horder for bringing me on board the Professional Services team, and to Donna Barry for her training on Job Descriptions and Advisory Appointment Committees.

Most importantly, I thank my friends, family, and partner for their unwavering love and support throughout my entire degree.

Abstract

The Royal College of Radiologists (RCR) is a professional body responsible for advancing radiology and oncology throughout most of the United Kingdom. This includes oversight of high standards in the recruitment of Specialty and Associate Specialist (SAS) Radiologists, Consultant Radiologists, SAS Oncologists, and Consultant Oncologists across over 100 NHS Trusts, and some Crown Dependencies. This process includes mandatory statutory requirements.

Before a Trust advertises a position, they submit the Job Description (JD) to the RCR for review to ensure that the post is well-supported and adequately funded. The Advisory Appointment Committee (AAC), a panel of interviewers, evaluates candidates for these roles. The RCR contributes a representative to the panel to verify that the candidates' training aligns with the role's requirements and to identify any additional training needs.

The current system depends heavily on manual operations, requiring significant involvement from multiple teams. This project aims to develop an automated system that will reduce time commitments, and decrease the likelihood of errors.

Contents

List of Figures

List of Tables

Listings

Chapter 1: Introduction

1.1 Background

Ex Radiis Salutas
From Rays, Health

The Royal College of Radiologists (RCR) plays a pivotal role in ensuring high standards in the recruitment process within radiology and oncology. Of particular importance is the quality assurance provided by Job Description (JD) review and Advisory Appointment Committee (AAC) panels. The RCR states that its processes ensure "candidates have clarity and confidence in applications, recruiting organisations attract and retain the best possible appointee, and vacancies are filled efficiently, benefitting existing consultant staff." (The Royal College of Radiologists 2022)

The National Health Service (NHS) understands the complexity of this process, and states that "The administrative burden associated with job planning is considerable. Success depends on having systems in place and information available", and Trusts are highly recommended to invest in electronic job planning software (National Health Service 2017).

This is of particular importance, given the concerning shortfall in clinical radiology and oncology. With shortfalls of 17% in oncology and 29% in radiology, safe and effective care is limited. Moreover, these shortfalls are expected to worsen in the future. Doctors have had their pay cut, and 83% report some form of burnout (The Royal College of Radiologists 2023c). This has profound implications for patient care and service delivery. Limb (2022) found that shortages lead to longer wait times for diagnoses and treatments, potentially worsening patient outcomes, in fact every month of delayed cancer treatment increases patients' risk of death by 10%. By 2029, these professions will need to grow by 45 percent (National Health Service 2018).

Annual job plans, as defined by the RCR, are agreements between doctors and their employers "setting out the duties, responsibilities and objectives of the doctor" (The Royal College of Radiologists 2022). This process, which involves completing review forms in Microsoft Word and subsequent email communication, often leads to back-and-forth discussions between the NHS Trust, RCR, and Regional Speciality Advisor (RSA). Every detail, including every date an email is sent or received, is meticulously recorded in a large Excel spreadsheet. This process ensures that job plans are fair and realistic.

On the other hand, the AAC process is "a legally constituted interview panel established by an employing body when appointing consultants." (Royal College of Surgeons of England 2023). Representatives are mandated by the NHS to ensure that the panel constitutes a balanced Committee (National Health Service (Appointment of Consultants) 2005). This process, involving the handling of sensitive data such as doctors' emails and phone numbers, requires encryption and password protection. Trusts often require multiple lists to find an available representative, as these individuals, being consultant doctors themselves, typically have busy schedules. The RCR's involvement ensures that the selection process adheres to high professional standards and that candidates are evaluated fairly and competently.

Their process is deliberately extensive for good reason. Generic healthcare recruitment software does not address the specialised, nuanced steps that are required by the RCR. Many aspects are legal requirements by the NHS that must be followed by every non-foundation Trust and Royal College (Royal College of Surgeons of England 2023). Automation would allow for the streamlining of these operations, significantly reducing the administrative burden and likelihood of manual errors. Offering a more reliable and consistent approach to managing the recruitment process is a necessity to cope with the increasing demands of healthcare delivery, and to support the overburdened workforce.

1.2 Problem Statement

Despite its critical role in maintaining employment standards, the RCR's current system faces significant inefficiencies and limitations. Managed primarily through manual operations involving Excel, Outlook, and Word, the risk of errors is high, contributing to an already time-consuming and labour-intensive process. These stand-alone applications have limited integration capabilities, requiring manual data transfer, and making automation of repetitive tasks extremely difficult, unreliable, or downright impossible. Although Excel is powerful for data manipulation, it is ill-suited for tracking complex workflows. It lacks the ability to monitor process stages, complex data integrity mechanisms, concurrency and multi-user environments, and handling of large datasets. Consequently, the RCR is forced to create a new spreadsheet annually, increasing time wasted searching through multiple different files. The lack of data normalisation and continuity makes it difficult to track and analyse specialities, which is essential for reporting.

Data analysis is critical for improving efficiency. The RCR states that Advisory Appointment Committee (AAC) data is collected to "check that the appointee is qualified to train doctors for the future, track increasing or decreasing numbers of doctors, track increases or decreases in different types of posts, track where it may be difficult for NHS Trusts to attract new recruits" (The Royal College of Radiologists 2023a). Without these metrics it becomes difficult to provide guidance on how to allocate resources correctly, which is essential for resolving consultant shortages.

Delays and errors, inherent in manual systems, can cause Trusts to miss critical deadlines for filling vacancies and to lose out on high-quality candidates. This must be avoided at all costs, as the NHS Appointment of Consultants Regulations state that "Only in extreme circumstances should it be necessary to cancel an AAC." (National Health Service (Appointment of Consultants) 2005). Significant staff time, which could be better utilised in more critical roles, is dedicated to this tedious and manual process. This issue affects not only the RCR, but also Trusts' teams, Regional Speciality Advisors (RSAs) reviewing Job Plans, RCR representatives assessing panels, and the candidates who are pivotal in delivering lifesaving patient care.

1.3 Aim

This project aims to develop an automated system specifically tailored for the Royal College of Radiologists (RCR), to manage and accelerate their Job Plan and Advisory Appointment Committee processes.

Specific Objectives

Achieving this aim depends on a number of prerequisites. Integrating an SQL database will ensure that a more robust system supports an automated workflow, works with large datasets and complex data relationships, and ensures data integrity and security. This integration will give the RCR the ability to analyse the data for reporting with business analytics services such as PowerBI. Choosing a web-based platform application allows for easier access and collaboration across different users and departments. A stand-alone application would be extremely unpopular, as all users will have to install a separate application, which is also very likely to be blocked by IT systems. A website also allows users to access the site from any device. Web development frameworks such as SvelteKit and Django are selected not only because they streamline development tasks but, but also because it is much easier to automate complex business workflows with the tools they provide. Implementing data security and compliance with encryption software like OpenSSL, password protection, and secure data storage solutions will allow the project to comply with GDPR and NHS regulations. By creating a user-friendly, customised, and accessible interface, we can simplify the process for all stakeholders.

Chapter 2: Literature Review

2.1 Current Process

2.1.1 Job Descriptions

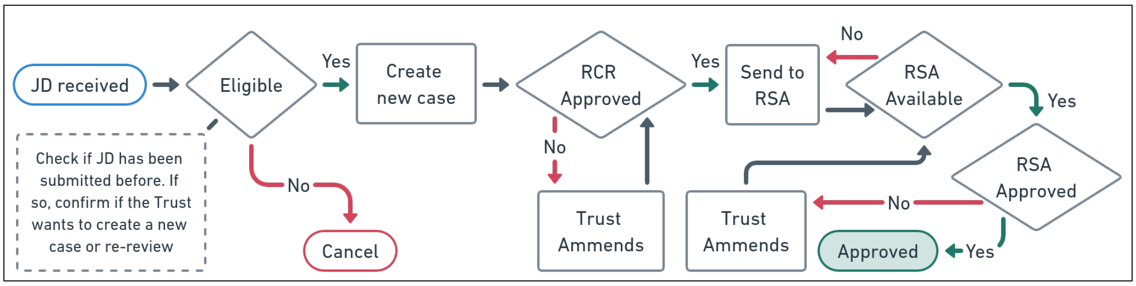


Figure 2.1: Job description process

The Job Description (JD) process in figure 2.1 reflects the meticulous approach followed by the Royal College of Radiologists (RCR) to ensure quality and precision. The process begins with the receipt of a job description, which is then subjected to preliminary checks. If the JD meets the initial criteria, it moves forward to a more detailed examination by a Regional Speciality Advisor (RSA). If the RSA is unavailable at any stage, a new RSA is promptly assigned. The diagram shows multiple checkpoints and potential loops that can occur if information is missing or incomplete, yet every effort must be made to complete this entire process within 2 weeks. The process concludes by informing the Trust that they are permitted to advertise their post and by sending them a form to complete to request a list of representatives.

2.1.2 Advisory Appointment Committees

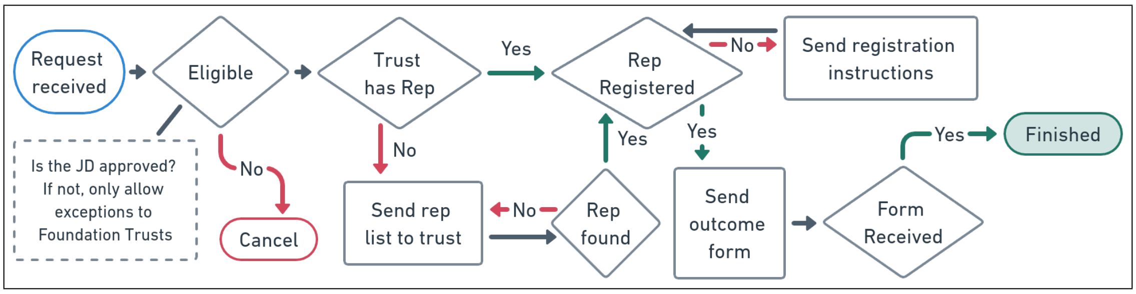


Figure 2.2: Advisory appointment committee process

The Advisory Appointment Committee (AAC) process in figure 2.2 is equally rigorous and focuses on giving Trusts the best chance of securing a representative. This process begins by checking whether the previous Job Description (JD) process was successful, only granting exceptions when requested by a Foundation Trust. Like the JD process, it often involves loops when sending multiple lists to the Trust. If the Trust opts to source their own representative, either upon request or after an unsuccessful search, the appropriate guidance and rules must be sent. All representatives must complete training and registration to attend panels. The process concludes when the representative returns an outcome form sent to them to complete after the panel is held, or if the Trust cancels the panel for any reason.

2.2 Related Works

2.2.1 Existing Storage Systems

With Excel, the primary limitation is the lack of ability to store speciality data. This is because job posts often have multiple primary specialities and secondary sub-specialities. If opting for a single column, data integrity becomes almost impossible as Excel does not inherently support complex data validation for multiple entries within a single cell. This makes it difficult to ensure that the data entered is consistent and makes analysing and querying this data challenging. Using multiple columns for each specialisation provides a more structured way to store this data. However, in this context it quickly becomes unwieldy – in our case we are left with 44 columns. Here the spreadsheet becomes cluttered and difficult to navigate.

The same issues apply to tracking the date of each step of the process, as due to loops in the workflow we often encounter multiple dates in each cell. With no automation whatsoever, these dates, as well as the workflow's status, must be entered manually every time.

The core issue with both approaches is the lack of normalisation. By splitting data into multiple related tables, and establishing relationships between them, we can reduce redundancy and improve data integrity. This is only realistic if we use a relational database management system like SQL.

2.2.2 Healthcare Recruitment Tools

Searching for software to assist with recruitment in the healthcare sector will bring up results such as ICIMS (2024) and The Access Group (2024). Unfortunately, these programs are typically directed towards the Trust rather than Royal Colleges. Their intended user base is employers looking to advertise and interview candidates, not for those involved in the rigorous job plan review and representative selection process required by the Royal College of Radiologists (RCR). These solutions typically require private meetings, so understanding their capabilities is difficult. For a set of requirements this specific, only personalised software is suitable.

2.2.3 Other Royal Colleges

Finding specific details on the software used by other Royal Colleges in the UK is not possible as this information is not typically disclosed publicly. Though the general process for Advisory Appointment Committees (AAC's) is somewhat standardised across various Royal Colleges due to the strict requirements set out by the NHS, there is no central piece of software. The advantages to an open-source solution may span beyond just the Royal College of Radiologists (RCR).

While the overall process is shared, there are still many differences between colleges. Each email and form would differ, and doctors in various departments have distinct needs. A software flexible enough to accommodate everyone is far beyond the scope of this project. For example, the Royal College of Physicians (RCP) requires that medical staffing departments complete the job description review form themselves, in contrast to the RCR filling it out for them. This may, however, be an indication that this project should accommodate any policy changes that the RCR may implement. The RCP also requires foundation Trusts to submit representative requests with a minimum of eight weeks' notice, a guideline that is optional but recommended in many other colleges, and offers an optional kitemark (Royal College of Physicians of London 2017). Some colleges require the trust to complete the review form for themselves and the reviewer. This is an example of an optimisation to the workflow that could be implemented into the system.

With differing requirements, it's no wonder that simply sharing software with each other is not a solution. A project that would suit every Royal College would be an expensive undertaking.

Chapter 3: Design

3.1 Visuals

The interface will utilize a dual-theme approach: a light mode with a subtle dotted background to enhance readability during daytime, while a dark mode will reduce eye strain for those working later hours. With more employees choosing flexible working options due to the shift to working from home, it is important for users to reduce the amount of blue light in the evening (The Chartered Institute of Personnel and Development 2023). The primary colour, purple, aligns with the Royal College of Radiologists' (RCR's) branding, creating a professional and cohesive aesthetic. Red will be used selectively for critical actions like logout and reject buttons, drawing immediate attention.

Clean, sans-serif fonts will be used for easy legibility, and consistent layout structures will be maintained, even across different screens, to support muscle memory and navigability. The layout will take some inspiration from the recent trend of Bento UI, where information is broken into separate cards of a single category. This design style is ideal for a project that requires the users to have access to so much data, because it allows for compartmentalising information effectively. 'isms' such as Neumorphism and Glassmorphism will be used sparingly. Whilst they add depth to the design, allowing users to differentiate between elements we want them to focus on most, they can also be distracting, and appear unprofessional when overused.

There are countless other design choices that go beyond this, the majority of which are applied subconsciously. My personal style is heavily influenced by *Grid systems in graphic design* by Müller-Brockmann (1981) and the more recent *Refactoring UI* by Wathan and Schoger (2018).

3.2 Frameworks

3.2.1 Django

Django is chosen for its comprehensive feature set that simplifies many web development tasks thanks to its "batteries-included" design. Django's ORM (Object-Relational Mapping) system will be used to manage database operations much more efficiently than through direct SQL, which is crucial given the complex data relationships in the RCR's processes. Its built-in security features as less essential as it will have limited internet access, however tools such as SQL Injection protection and Cross-Site Scripting (XSS) protection are still paramount.

To turn Django into an API, Django-Ninja will be implemented for its exceptional speed and ease of use. The use of Pydantic with Django-Ninja ensures strong type checking, significantly reducing the risk of bugs that are usually common in Python and JavaScript. Its integration with OpenAPIs automatic documentation allows us to test the API in SwaggerUI before writing any frontend code, and can even be used to transfer schemas from backend to frontend. This library also supports Django's asynchronous capabilities. However, it will not be used in this project as the frequency of requests is not high enough to warrant the additional complexity, though it leaves room for scalability if needed.

Django-Ninja-JWT, built on top of Django-Ninja-Extra, provides JWT (JSON Web Tokens) authentication, essential for secure API access in our application. This feature allows for stateless authentication, making it easier to manage sessions in a distributed environment where the RCR's data may need to be accessed securely across multiple locations. The JWTs will be stored securely using HTTP-only cookies, configured with strict properties to prevent unauthorized access. It also eliminates the need to handle CSRF tokens in Django, though SvelteKit already handles this by default.

Alternative API Framework

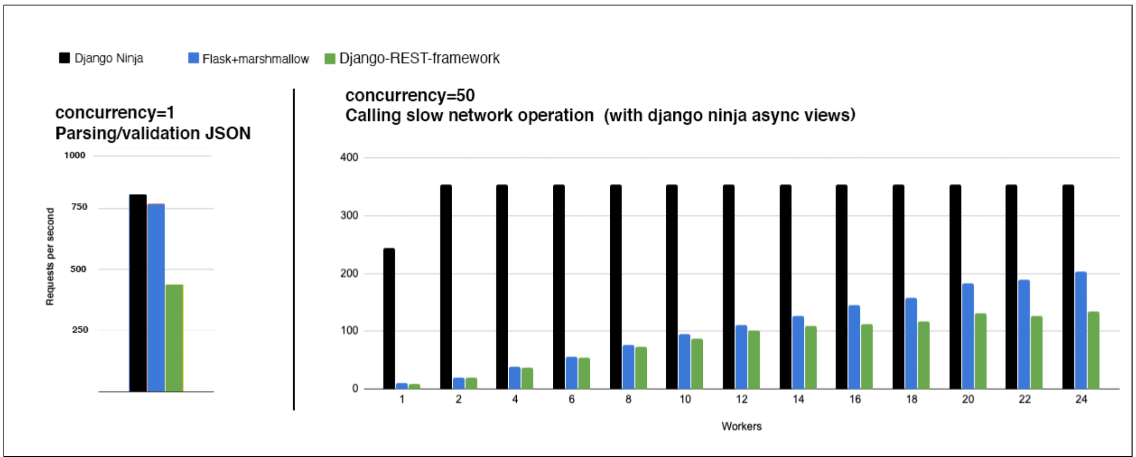


Figure 3.1: Various Django API framework speeds
(Vitaliy Kucheryaviy 2024)

Django Rest Framework (DRF) is the most popular choice for building APIs in Django. However, it lacks the type hints, Pydantic integration, simplicity, and automatic documentation of Django-Ninja. Figure 3.1 shows the enormous speed difference between the two, even when concurrency is not used.

PyTransitions

The Python 'transitions' library for state management of JDs. State machines are crucial for enforcing the rules and transitions of a workflow's various stages, and ensure that each job description progresses through its lifecycle in a controlled, traceable manner, preventing errors and ensuring consistency. It also gives users the ability to track the progress of each JD visually by generating state machine diagrams based on the current state in relation to others.

3.2.2 SvelteKit

You may have noticed that there is no dedicated Backend and Frontend section. This is because SvelteKit in itself is a full-stack framework. There are a number of reasons for this, most notably that the frontend, Svelte, and backend are extremely closely integrated and are essentially one entity. This allows us to have features such as a file based routing system, simplified state management, and much more. This is a key advantage of SvelteKit over other frameworks, as it allows for a more seamless development experience. The performance is also superb, as it compiles to vanilla JavaScript and removes the overhead of the virtual DOM - though the performance difference with a framework such as Vue3 is negligible. On low-end devices or poor network connections however, which can be common where organisations are underfunded, adaptive loading and progressive enhancement can make a huge difference. As discussed in 1.1, is extremely important that this system is as accessible as possible, especially to Trusts that need it most, as a large part of the National Health Service depends on this process.

The use of a 'second backend' allows a more modular design. By allowing Django to focus exclusively on serving and managing the database, SvelteKit's backend can focus on rendering the frontend, handling user interactions, and manipulating form data into a cleaner format. This leaves Svelte to handle the UI with data in a format easiest for HTML to display.

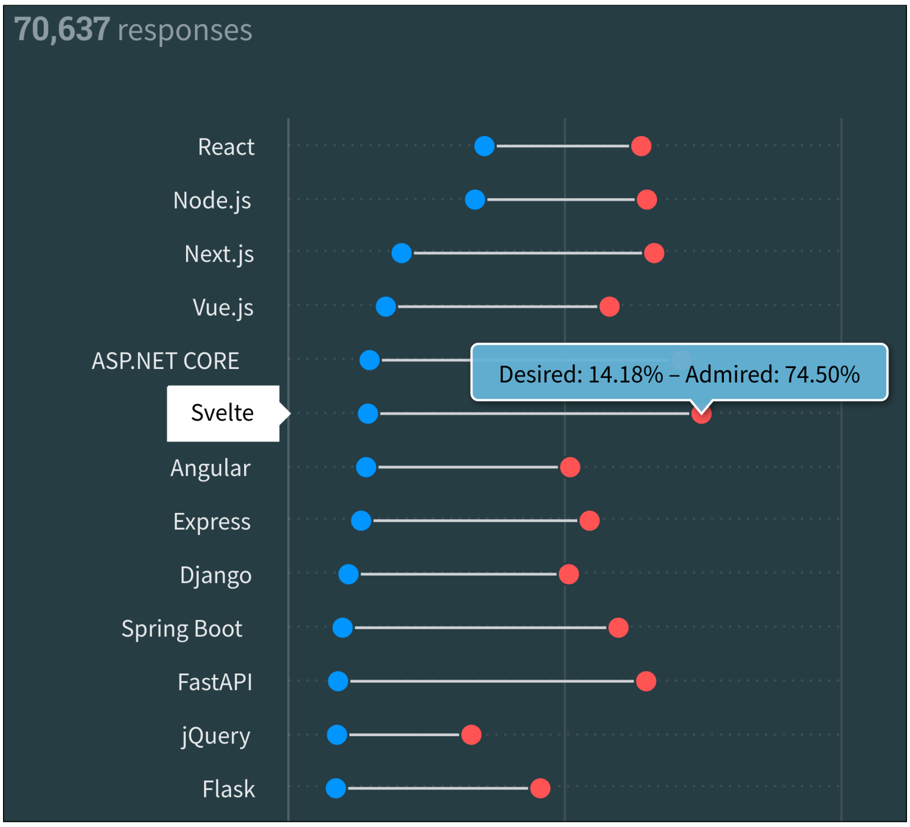


Figure 3.2: Svelte ranking
Consistently one of the most admired web frameworks, scoring second highest in 2023
(*Stack Overflow Developer Survey* 2023)

Libraries

TypeScript will allow strict typing, just like in the Django-Ninja Schema. As discussed in 3.2.1, OpenAPI-Typescript allows us to import this schema into SvelteKit, providing type safety and autocompletion without having to rewrite all of the types in a completely different language. This ensures that the frontend and backend development environment are always in sync, reducing the likelihood of errors and inconsistencies.

These types will be used in requests with Axios, which will be responsible for communicating with Django. It is easier to work with than native JavaScript's Fetch as it is more readable with less boilerplate, but more importantly, it has automatic JSON transformation which will be essential for handling the complicated forms that will be built. A custom API library will be built on top to abstract this code even more, which can turn over 10 lines of code into a single function call. These functions can be used anywhere in the program, and since the same data is likely to be used in multiple different pages, it also removes the need to repeat code. Lastly, by having a single source of truth for the API, not only is it easier to make changes to the API without having to change every single page that uses it, but it also reduces the changes of bugs due to typos to almost zero.

Postmark is an email delivery service which will be used for email verification and notifications. It is preferred over self-hosted emailing, as those are usually blocked as spam by all major email services, but still allows us to use our own custom domains through DKIM and Return-Path DNS records. It features a very easy to use API, and a very high deliverability rate, as all developers have to be approved by Postmark through a manually reviewed form.

shadcn-svelte is not so much a library, but rather a set of re-usable components build on top of a number of other libraries. It allows us to keep a consistent design across the entire application, and reduce time spent on building basic parts of the UI. This does however mean that the multitude of libraries it is built on top of have to be learnt and a deep understanding of the documentation is a requirement. These libraries will include; Svelte Headless Table for data tables. Formsnap, Superforms and Zod for form processing and validation. Bits UI (built on top of Melt UI) for the majority of the components. And TailwindCSS for styling and positioning. The form libraries are particularly important. While forms are one of the common features on the web, they are also one of the most complex - and this project is almost entirely based on large, complicated, and dynamic forms.

Luckily this setup will allow us to build well-structured and semantically correct forms, with client and server side validation. There are also Accessible Rich Internet Application (ARIA) attributes and proper labels for screen readers to make the website accessible to all users. Lastly they are generally easy to use and can be navigated exclusively with a keyboard.

3.3 Deployment

For the purposes of this project, the website will be deployed on a Linux server with Nginx, Gunicorn, and Node with its respective SvelteKit adaptor. Nginx will be used as the reverse proxy and HTTP server, along with Cerbot for automatic SSL certificates. The Django admin panel may also be exposed publicly to allow access for RCR Employees working from home. Gunicorn will be used with multiple workers to handle multiple requests at once, as opposed to using Django-Ninja's asynchronous features. Node.js for SSR will improve load times and SEO. Though the latter is not as important, assuming the website will be a link in an existing RCR webpage, it will still help users searching for the website avoid multiple navigations. This may reduce the number of phone calls and emails received attempting to find the correct RCR department.

3.4 File Structure

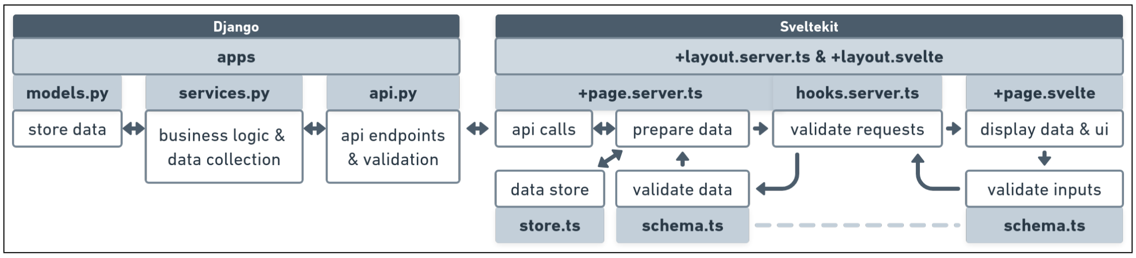


Figure 3.3: System file structure

The diagram in figure 3.3 presents the general communication between files, starting with `models.py` which defines and creates the database. `services.py` is used as an intermediary between the database and the API, and is responsible for handling the business logic. This includes complex algorithms to refine and filter data before moving it - which are usually called as functions in the api. `api.py` is then used to communicate with the frontend. The code here should be as simple and readable as possible, and rely entirely on functions defined in `services.py`. This is because the majority of the processing code can be made reusable, and a separate layer allows us to follow "Don't repeat yourself" (DRY) principles (Thomas and Hunt 2019). It also allows us to separate concerns (Reade 1989) by keeping the API code to a number of function calls with understandable names, some error handling, user validation, and returns.

These files will be repeated in six different apps to separate concerns, improve modularity, maintainability, and scalability. These will include users, roles, trusts, specialities, jds, and aacs. Communication between these apps will be handled through `services.py` and occasionally `api.py`. Thanks to Django-Ninjas routers, it will also give each of our API endpoints a url that begins with the app name, making it easier to understand. Most of the apps are self-explanatory, however, to be clear the roles app will be used to define the type of employee to make it easier to handle their permissions (this includes Reviewers, Representatives, and the RCR and Trust teams). The full structure of this database will be discussed in 4.1.

`+layout.server.ts` is a server-side script that acts as a data pre-processer for all child routes below it. It will fetch data that is shared across all pages, mostly for the navigation bar. `+layout.svelte` is the global layout of the application presented to the user. In our case, it will hold the navigation bar, define the structure of the page by providing a wrapper for other pages, and render a small logo in the corner as the footer. `+page.server.ts` is the server-side script that prepares the data which will be used on the corresponding page. It will fetch data from the API, potentially perform server-side processing, and then pass the data to the frontend to be rendered. `hooks.server.ts` is a server-side hook that will be solely responsible for validating requests over the API by checking whether the user is authenticated and authorised. `+page.svelte` is responsible for rendering the user interface. It will display data to the users and handle their interactions. Form data will often be passed back to `+page.server.ts`' Actions, where it will be validated server side, potentially processed, and then sent back to Django through an API call. `schema.ts` is a single file per page that defines the form schemas for that route. It is first sent through `+page.server.ts` to `+page.svelte`, where it is used to validate the form data live as the user makes inputs. It is used again in the backend to validate the data before it is processed, this is crucial because users could easily bypass the frontend validation.

3.5 Routes

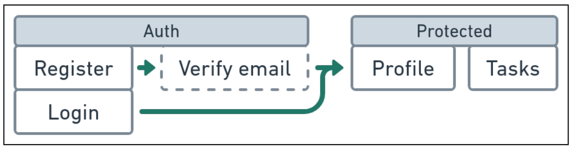


Figure 3.4: Navigation flow for all users

Figure 3.4 represents the general navigation we want all users to take. The grey cells represent the parent route, which will appear in the url, and the white cells may be a component, or child route. The reason for a shared auth route is that we can have one page have a login and register form without a page redirection. Verify email will actually be a child route with the sole purpose of verifying and redirecting the user.

The protected route is where all other pages will be. SvelteKits server hook will require all users to be authenticated through the Django API before they can access any pages - the most basic security measure. A common decision is to put authentication inside `+layout.server.ts`, as it encompasses all other routes after all. However, this is a massive security risk, as even unauthenticated uses can access the data fetched from the database. This is because the layout load function runs in parallel with the page's load function, so even if an unauthenticated user is redirected, the layout may still fetch data from the database. This isn't the only risk, for example if a user logs out in one tab, but keeps another tab open on a page that requires authentication, refreshing the second tab would bypass the authentication check because the layout's load function would not be aware that the user is no longer logged in.

All users will be initially redirected to the Profile page, which is to encourage keeping accurate records as they will see them every time the log-in. These records are important as they are automatically sent alongside forms that the user would otherwise have to fill out every time. They also have a Tasks button, which will automatically list all JDs and AACs that require action in one place.

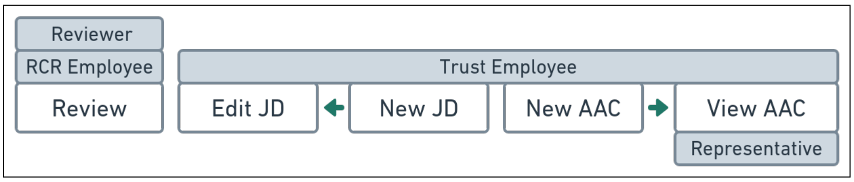


Figure 3.5: Navigation flow for specific users

Figure 3.5 represents the navigation flow for specific users. The white cells represent buttons in the nav-bar. The process technically starts at different stages for all users, however they all depend on a JD. A Trust Employee will start by clicking the New JD button in the nav-bar, and is redirected to that page. Once submitted, they are automatically redirected to the Edit JD page, which also has its own button. This page will display information about the JD, and provide a form for the Trust Employee to complete. This form will include a dynamic amount of questions, dependent on the consultant type of the JD, and answer fields for content such as page numbers and general comments. Navigation to the Edit JD page through a button will present a list of still editable JDs specific to that user which can be edited at any time.

Once saved, the next step is to be completed by the RCR Employee, who will have the same page as the Trust Employee, but with an added row for comments. Of-course, just like Edit JD, they are first presented with a list of JDs that require review. The RCR Employee will be able to save their changes, and submit or reject when they are ready. When submitting, they will be given a list of Reviewers they can send the JD to with relevant information, in order to help chose the best option. Reviewers will have the same page, but again with an added row for comments.

Once a JD is approved, the Trust Employee will be able to click the New AAC link to create a new panel. They will be able to select as many JDs as they like from a list, along with the date of the panel. They will then be redirected accordingly to View AAC. This page will have a list of valid Representatives along with contact details, in order of suitability. The Trust can choose one Representative, who will be given access to that AAC. Either the Trust or Representative can then download and upload an Outcome Form, which will be sent to the RCR Employee for data analysis. The reason this step is not a form is that interview outcomes are sometimes written by hand, or in person, where a Doctor may prefer not to use a computer.

3.6 Coding Style

The following style choices are personal preferences, with some influence from conventions.

PEP 8 (Guido van Rossum 2013) will be followed for all python code, more specifically, Black, and this style will be kept the same for variables passed from Django to SvelteKit. New variables and functions defined in SvelteKit will use camel-case in line with JavaScript conventions. A deviation from PEP 8 is that comments will be avoided as much as possible. This encourages code that is self-explanatory, such as descriptive variable and function names, and smaller easy to follow algorithms. It removes the risk of comments, which are subjective, being misinterpreted or outdated. There are exceptions, such as complex code that performs a task that is not immediately obvious.

Nested code in Python will be limited to a maximum of three layers of indentation. This practice helps reduce the cognitive load on the programmer. As aptly put by The kernel development community (2024):

> "If you need more than 3 levels of indentation, you're screwed anyway, and should fix your program."

In Django we can also avoid the fat models design since we have a service layer. This common idea violates the Single Responsibility Principle (SRP) and makes code hard to read as the models become huge (Martin 2003). Functions in models will be limited to essential methods such as `__str__` and `Meta`, `__init__`, and potentially `save`.

In SvelteKit, HTML will be separated into components to keep the number of lines in a file short, and allow component reusability. Linting will be used with ESLint to enforce a consistent style across all files, and to catch errors early.

Chapter 4: Implementation

4.1 Database

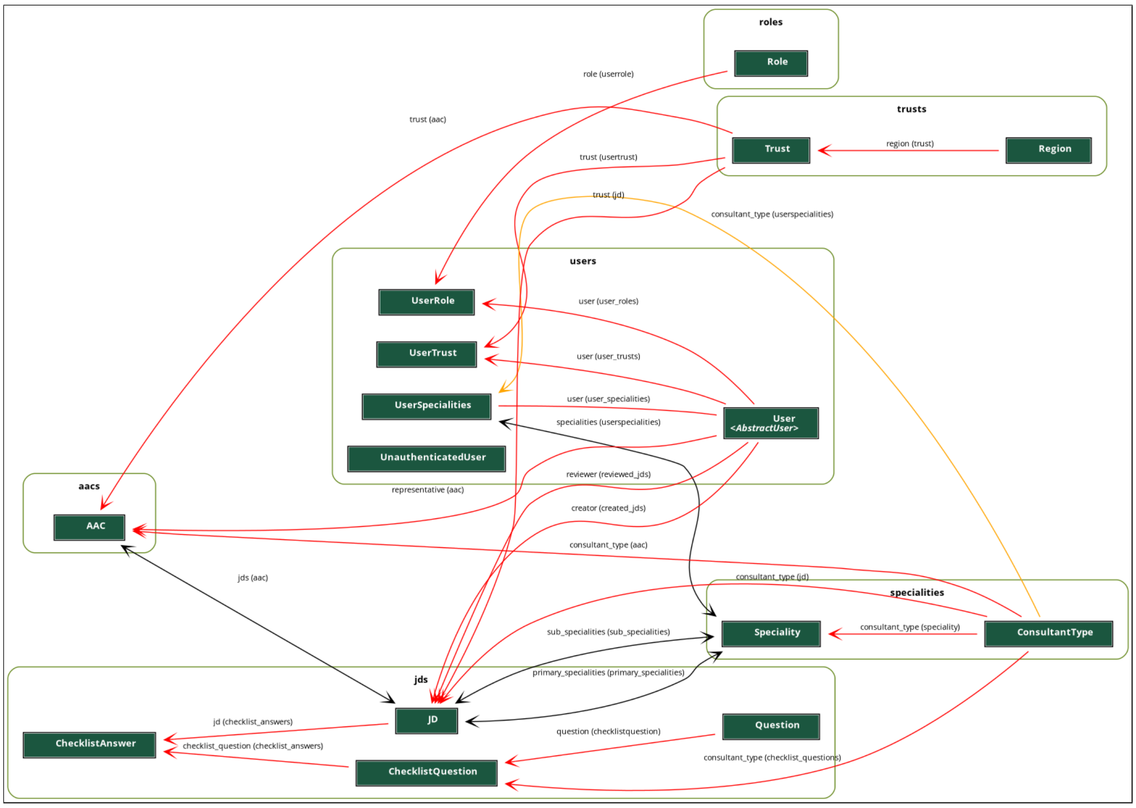


Figure 4.1: Database schema without fields
(backend/*/models.py)

Figure 4.1 represents the database schema. Due to the large number of models, fields have been omitted in this figure to maintain readability. See Appendix A for a full version. This diagram aptly presents just how intertwined this system is, as well as the sheet number of data taken into consideration during the process.

This relational database schema allows for handing of detailed user profiles, Job Descriptions (JDs), and Advisory Appointment Committees (AACs). It allows for a sophisticated access control system based on roles, Trust, and consultant types.

Normalisation helps in organizing data efficiently and reduces the redundancy by ensuring that each data item is stored only once. For instance, separating user details from roles and trust levels allows for easy updates and management without unnecessary duplication.

The clear definition of relationships between entities, such as one-to-many and many-to-many, is a manifestation of thoughtful consideration of how the models should relate to each other. It allows accurate, error-minimal and efficient querying of the database by using the minimum number of tables necessary. Foreign keys ensure that relationships between entities are consistent and that orphan records are avoided.

Separation of UserRole, UserTrust, and UserSpecialities is essential as users often have multiple roles and specialities. They may also be associated with different Trusts, which must be recorded to avoid conflicts of interest. A many-to-many relationship is not sufficient as each role and trust must be approved by an administrator first. We also need to ensure that users are not shown information from entities that they are approved for, but have not requested in the profile page.

Specialities and Trusts also have hierarchical relationships. Consultant Types are associated with differing specialities, so JDs should not be allowed to contain both radiology and oncology specialities. Trusts fall into Regions, which are used to decide which RSA should be prioritised to review the JD.

As always, fields are give intuitive names, and the correct data types and constraints are used for data integrity. God tables are avoided entirely as large numbers of rows cause performance issues and increase complexity.

Without such a detailed schema, at best the system would be completely static. Policies change often, new Trusts join or are renamed, and specialities are added or removed. Without an extremely flexible database, this website would be unusable within a matter of months.

4.2 API

4.2.1 Endpoints

```
1 @router.post("/register-authenticate", response=TokenOut)
2 def register_authenticate(request, token: TokenIn):
3     unauth_user = get_object_or_404(UnauthenticatedUser, token=token.token)
4     if user_exists(unauth_user.email):
5         raise HttpError(400, "An authenticated user with that email already
          ↪  exists")
6
7     user = register_user(unauth_user, token.token)
8     tokens = get_tokens_for_user(user)
9     return tokens
```

Listing 1: Registration endpoint example
(backend/users/api.py)

The code in Listing 1 shows the general structure and practises of all the API endpoints. You can view the full list of endpoints in Appendix B.1.

Line 1 defines the endpoint configuration as a POST request to users/register-authenticate, which returns a `TokenOut` type, as defined in `schema.py`. The Schema is essential for the OpenAPI documentation, shared types in SvelteKit, and error avoidance thanks to checked types.

Line 2 defines the function that will be called when the endpoint is hit. The function takes two arguments, `request` and `token`. The `request` argument is a Django request object which allows us to access various aspects of the incoming HTTP request. The `token` argument is a `TokenIn` type, just like in the response definition. This Pydantic model is used to validate and parse and validate the incoming data, and again, is used in the OpenAPI documentation.

Line 3 to 8 define the business logic, which error handling from Line 4 to 5. It begins by retrieving the `UnauthenticatedUser` object from the database using the `get_object_or_404` function. If this user already exists, then a 404 error is raised which can be handled by SvelteKit to show a user-friendly error message. As discussed in 3.4, the custom functions `user_exists`, `register_user`, and `get_tokens_for_user` are defined in the `services.py` file to abstract the business logic away from the API endpoints. Finally, line 9 returns the tokens to the request sender.

These patterns are used for all 27 endpoints, with a few occasional additions. For instance this PUT request to update a JD's checklist:

```
@router.put("{jd_id}/checklist/", auth=JWTAuth(), response=JDChecklistOut)
```

You can view the full documentation for this endpoint in Appendix B.2. This request has two additional parameters. `jd_id` is a path paramter used to identify the JD that the checklist is being updated for. The `auth` parameter specifies the authentication method that must be used. `JWTAuth` middleware checks if the request contains a valid JWT, if not, then the request is rejected before it can reach any part of the endpoint's function.

There are a number of advantages to using many small endpoints rather than a few large ones. Firstly, it provides modularity in the design, where each endpoint is responsible for a single task. Since our REST API is for a website where almost all data is kept separate from the frontend, and said data is sent back and forth very frequently, it is important that endpoints are each to understand, develop, and maintain. We can also reuse simpler endpoints across multiple pages. Attempting to create a single endpoint that can handle a huge number of requests would quickly become unmanageable.

4.2.2 Consumption

As discussed in 3.2.2, the API is consumed by the frontend using a custom-made API client. Figure 4.2 presents a more detailed view.

`api-utils.ts` defines the abstract functions and builds the required axios configuration dynamically based on the input parameters. Strong typing is especially important here as it ensures that errors are caught early. The use of promises ensures that errors are properly propagated and can be handled by the callers of these functions.

`api.ts` defines the actual API calls for each endpoint with asynchronous functions for non-blocking execution. openapi-typescript generates a `types.d.ts` as discussed in 3.2.2, and is combined with the response data to provide type safe data to SvelteKit.

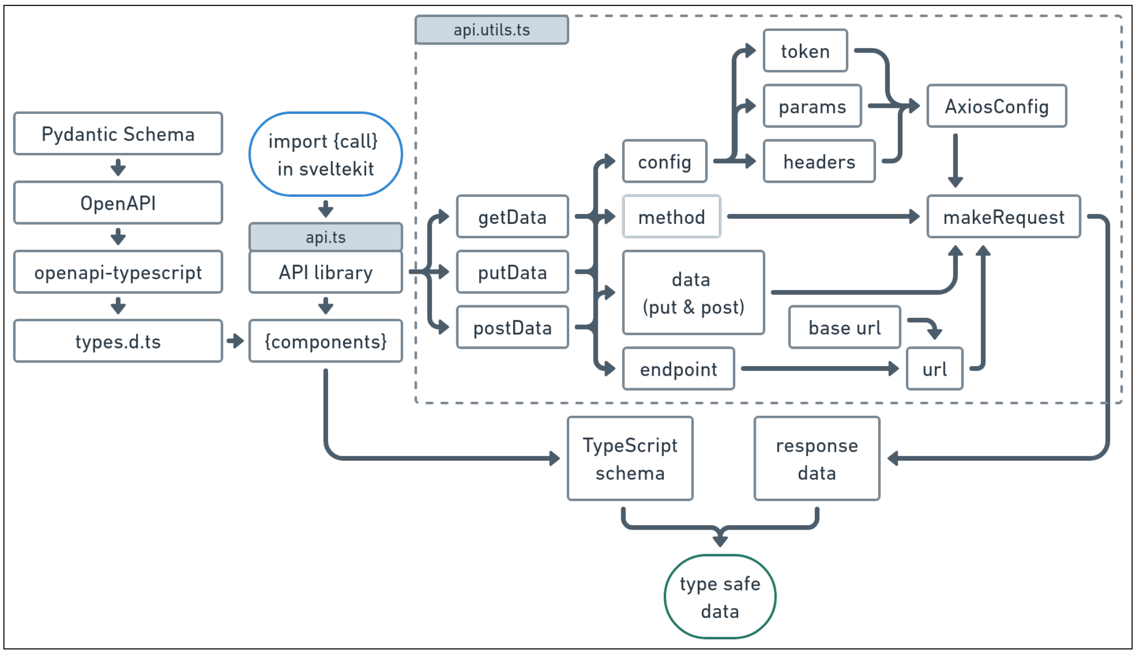


Figure 4.2: Custom API client
(frontend/src/lib/api*.ts)

4.3 Layout

Figure 4.3: Navigation bar
(frontend/src/routes/*.*/)

As discussed in 3.4, the layout is responsible for the header, body, and footer of all children. It is also responsible for importing the global CSS file `app.pcss`.

To present the correct buttons to the user in the nav-bar, a reactive store is used to keep track of the user's roles stored in the backend. If a user is not approved, but has requested a certain role, then they are still shown the button but in a greyed out format. This feedback makes the user aware of which roles they must select to gain access to specific parts of the system.

Included in the nav-bar is a Log Out button in red, chosen to stand out and remind users not to stay logged in on shared computers (though the JWT will automatically expire if no requests are made for a while). There is also a dark and light mode toggle button.

The body of the page is a `<slot>` enclosed in containers that make the page fully responsive to all devices, aspect ratios, and zoom levels. A logo is provided in the footer as a placeholder.

See Appendix C.1 for screenshots of the responsive layout on mobile devices, along with the dark mode feature.

4.4 Authentication

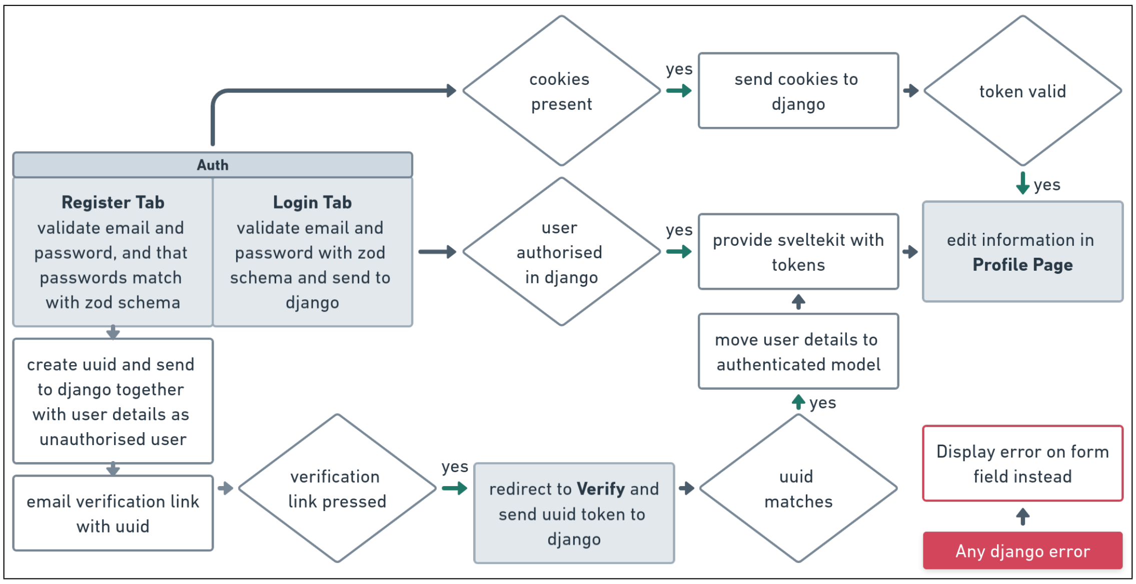


Figure 4.4: Authentication flow
(frontend/src/routes/auth/*) & (backend/users/*)

Figure 4.4 represents the authentication process. A high level overview has already been provided in 3.5, so this section will focus on the security steps taken. In both components, email and password is validated according to a predefined Zod schema which add a layer of data integrity and security, ensuring that the credentials meet the system's standards before being sent to Django. Screenshots of the login and registration components, along with their validation errors, can be found in Appendix C.2.

4.4.1 Registration

Email verification is a necessity to ensure that users do not create fraudulent accounts. Upon registration, the system generates a universally unique identifier (UUID) token and stores it in Django's database. It is a one-time, non-guessable token that securely links the email verification request to the user's pending account status in the database. This UUID is embedded into a link which is sent to the user's email address with Postmark. Once the link is clicked, the system verifies the token and activates the user's account.

For a more fluid user experience, this process returns a token automatically, so that the user does not have to log in manually. The token is stored in a Secure HTTP-only cookie, shielding access from client-side scripts. The HTTP-only restriction is an effective countermeasure against cross-site scripting (XSS) attacks, and the Secure flag, ensures that the cookie is only sent over HTTPS connections. A helpful addition is that the Secure flag is disabled in development mode.

Robust error handling is included to let users know why their login or registration attempts may have failed. Attempts to generate accounts with existing emails, or invalid login information is met with an immediate and clear error message. On the other hand, a successful registration attempt is met with a toast, as users require alternative positive feedback when there is no redirection.

4.4.2 Login

Users are automatically logged in as `hooks.server.ts` check whether cookies are present on every request. Otherwise the user is redirected to the auth route to log in.

4.5 Profile

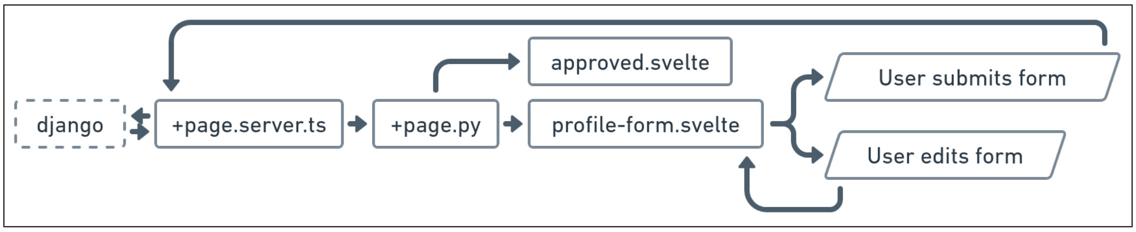


Figure 4.5: Profile page flow
(frontend/src/routes/profile/*) & (backend/*)

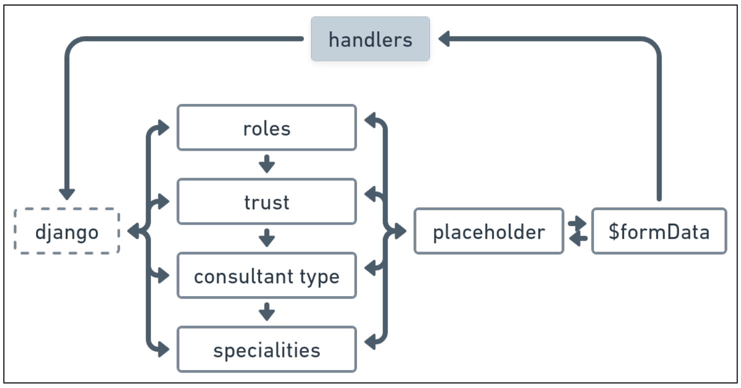


Figure 4.6: Profile form data flow
(frontend/src/routes/profile/profile-form.svelte)

Figure 4.5 presents the exact file structure and data flow of the profile page. `approved.svelte` displays the users currently approved roles and trusts, and `profile-form.svelte` displays a form for the user to edit. A higher level explanation of this setup can be found in 3.5.

Figure 4.6 presents the data flow in the form itself. Data is initially loaded from Django, and any existing data is used as the placeholder in the field. Otherwise the user is prompted to select an item. Fields are disabled in the order in the diagram, for example, a user cannot choose a Consultant Type unless they have chosen a Role that includes one. Specialities are far more complex. For example, if a user selects Radiology, they should only be shown Specialities related to Oncology. This field must be cleared any time there is a mismatch, and requires a surprising amount of code to handle all edge cases. And there are many such edge cases, as there are multiple sources of data to be taken into account, that is the data from the page load collected from Django, the current placeholder, and the user's current selection held in `$formData`. `profile-form.svelte` is an especially complex component as there are over 10 different placeholders that are chosen and presented dependent on a variety of factors. Screenshots of the profile page can be found in Appendix C.3.

4.6 Data Tables

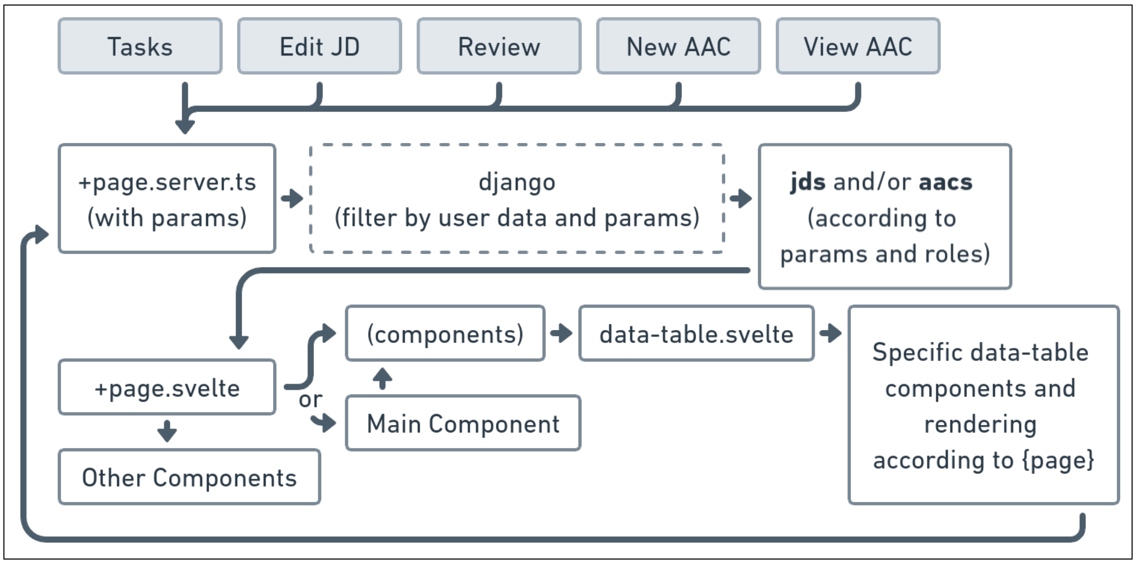


Figure 4.7: Data table flow
(frontend/src/routes/protected/*)

Figure 4.7 presents the flow of data to the data tables. You can find an example in C.4. These data tables are used to display various information dependent on the users roles, trusts, consultant type, and current page. This is not just limited to the data shown, but the columns are also dynamic - meaning that columns that aren't relevant to the page are automatically hidden. It is also sometimes embedded into forms to allow the user to select items. This is all possible because it is a single reusable component. The user can search and sort the data, as well as chose between pagination or increasing the number of items shown at once. A badge is placed next to the ID to show the user whether the entry is a JD, AAC, or Rep. Some pages have a button at the end of the table instead of a row-actions component. For example, the Edit JD page has an Edit > button at the end of each row to reduce the number of button presses for the user, and to make them aware of the action they are expected to take.

To make these tables so flexible, reactive variables have to be passed between through many layers. This data has to cleaned and placeholders have to be added to fit the format the table expects. Svelte's Logic blocks are used extensively to determine exactly how the table should be displayed.

4.7 Job Description Flow

4.7.1 State Machine

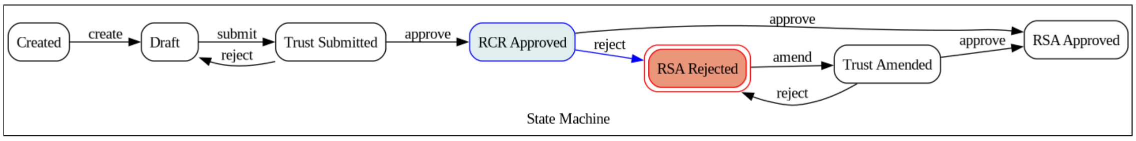


Figure 4.8: Job description state machine
(backend/jds/models.py)

4.7.2 Forms

Figure 4.8

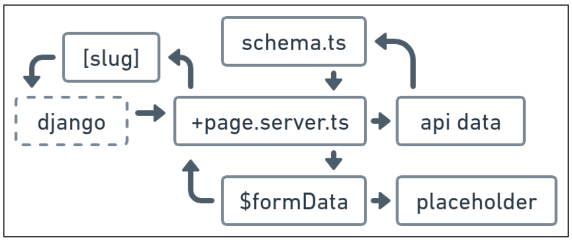


Figure 4.9: Job description questions flow
(frontend/src/routes/protected/*)

Figure 4.9

4.7.3 Permissions

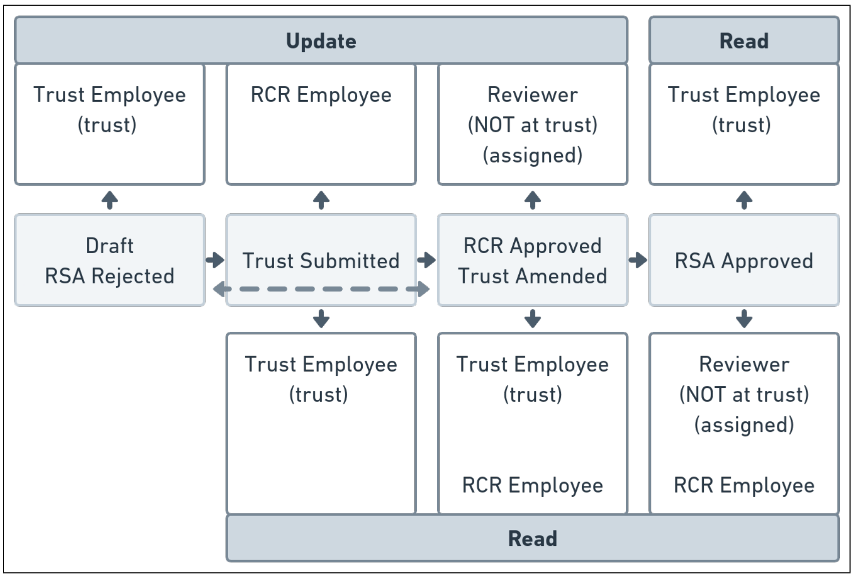


Figure 4.10: Job description permissions
(backend/jds/services.py)

Figure 4.10

4.8 Advisory Appointment Committees Flow

4.8.1 Form

4.9 Testing

django tests.py

UAT

going through the workflow

Chapter 5: Project Review

5.1 Reflections

This project was extremely time-consuming. Beyond the simple use of Django in a Web Technologies module, I had to learn every single framework I chose to use from scratch. While many of the libraries used are established or production ready, they are not as widely adopted or mature as some other mainstream options. I had to report and correct numerous errors in various documentations and create bug reports, one of which now has numerous participants.

On the other hand, this project provided an opportunity to impress the creator of shadcn-svelte, who invited me to write an article about my implementation of a dynamic form within a data table. This feature was completely undocumented, and I plan to complete a write-up after submission, as it has been requested by several individuals.

Altogether, it was a huge challenge, but I am proud of the result. I have a fully working application, more than just a prototype, and have achieved most of the goals I set. I am also confident that the project has no limitations; it has been structured in a way that allows any desired features to be added with minimal rewriting.

That being said, throughout the project, I have identified numerous features that could significantly improve the user experience.

5.1.1 Further Work

The project is fully functional and could theoretically be used in a production environment. However, the risk of bugs is high as User Acceptance Testing has not been extensive. Furthermore, while the application is functional and generally looks professional, it is likely that upon closer inspection, many aspects will require more polish. Additionally, most of the text used will need to be reviewed and overhauled by a separate communications team to ensure it is professional and consistent with branding and external information.

For the most part, the code is clean; however, during the later stages, it became less sensible to invest time in refactoring and optimisation. This was primarily due to time constraints, but also because the benefits of well-refactored code become less significant as the project nears completion. Nevertheless, since many useful additions can still be made, and the program may require maintenance by myself or other developers in the future, maintaining readable code is still beneficial.

This report concludes with Table 5.1, a list of additional features that I would like to incorporate into the application.

Implement automatic JWT token refresh instead of relying on an 8-hour timeout
Design the admin panel to be more user-friendly, as at the moment models are only registered
Add an opt-in for email notifications for when action is required
Add personalised metrics for users in the form of charts or data cards
Create Django tests.py
Protect routes based on role too
Consider adding an RCR Rejected state
Allow a JD bypass for eligible Trusts that does not require manual changes in the admin panel
Restrict data table actions based on role and add AAC versions
Add alerts for when a session expires
Use combo-boxes instead of long selects.
Predefine an order to lists based on how often that speciality is chosen
Add a grey mode
Add an AAC state machine and diagram
Use SvelteKit's pre-rendering feature
Refactor code and file base
Add a built-in PDF viewer to pages with a download link (PDF-LIB)
Implement a way for Trusts to upload an updated file without the RCR Employee having to update it themselves
Implement machine learning to try to automatically detect which content is present
Add more filtering options to the data table, such as a dropdown with predefined value
Add the ability to view the history of a JD for users
Add breadcrumbs to the UI
Implement a much better algorithm to automatically select Reviewers and Representatives
Check whether a job description has already been uploaded with identical specialities and warn the Trust
Improve the forms to include fields for certain rows where a single number can be input for easier data collection
Parse the data in the submitted outcome form and store it in a database for data analysis
Add more error pages for better coverage
Add reactivity to the Django backend so that users pages are updated when another person makes a change
Add better permission checking in Django to ensure any requests users might try to submit in forms outside what is presented in the UI are rejected

Table 5.1: Table of planned features

References

Guido van Rossum (Aug. 1, 2013). *PEP 8 – Style Guide for Python Code — peps.python.org*. Python Enhancement Proposals (PEPs). URL: https://peps.python.org/pep-0008/ (visited on 04/27/2024).

ICIMS (2024). *iCIMS — Recruiting Software Platform & #1 Applicant Tracking System*. iCIMS — The Leading Cloud Recruiting Software. URL: https://www.icims.com/ (visited on 04/25/2024).

Limb, Matthew (June 10, 2022). "Shortages of radiology and oncology staff putting cancer patients at risk, college warns". In: *BMJ*, o1430. ISSN: 1756-1833. DOI: 10.1136/bmj.o1430. URL: https://www.bmj.com/lookup/doi/10.1136/bmj.o1430 (visited on 11/26/2023).

Martin, Robert C. (2003). *Agile software development: principles, patterns, and practices*. Alan Apt series. Upper Saddle River, N.J: Prentice Hall. 529 pp. ISBN: 9780135974445.

Müller-Brockmann, Josef (1981). *Grid systems in graphic design: a visual communication manual for graphic designers, typographers, and three dimensional designers = Raster Systeme für die visuelle Gestaltung : ein Handbuch für Grafiker, Typografen, und Ausstellungsgestalter*. OCLC: 7172816. Niederteufen: Verlag Arthur Niggli ; [New York] : Hastings House Publishers. 176 pp. ISBN: 9783721201451.

National Health Service (2017). *Consultant job planning: a best practice guide*. URL: https://www.england.nhs.uk/wp-content/uploads/2022/05/consultant-job-planning-best-practice-guidance.pdf..

— (2018). *Strategic Framework for Cancer Workforce 05-07*. URL: https://www.hee.nhs.uk/sites/default/files/documents/Cancer-Workforce-Document_FINAL%20for%20web.pdf.

National Health Service (Appointment of Consultants) (2005). *National Health Service (Appointment of Consultants) Regulations Good Practice Guidance*. URL: https://www.rcseng.ac.uk/-/media/Files/RCS/Standards-and-research/Support-for-surgeons/NHS_AAC20GoodPractice_Guidance.pdf.

Reade, Chris (1989). *Elements of functional programming*. International computer science series. Wokingham, England ; Reading, Mass: Addison-Wesley. 600 pp. ISBN: 9780201129151.

Royal College of Physicians of London (2017). *RCP London Advisory Appointments Committees (AAC)*. URL: https://www.rcplondon.ac.uk/education-practice/advice/advisory-appointments-committees-aac#:~:text=.

Royal College of Surgeons of England (2023). *Advisory Appointment Committees (AACs)*. Royal College of Surgeons. URL: https://www.rcseng.ac.uk/standards-and-research/support-for-surgeons-and-services/aacs/ (visited on 11/26/2023).

Stack Overflow Developer Survey (2023). Stack Overflow. URL: https://survey.stackoverflow.co/2023/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2023 (visited on 04/27/2024).

The Access Group (2024). *The Access Group — Business software*. URL: https://www.theaccessgroup.com/en-gb/ (visited on 04/25/2024).

The Chartered Institute of Personnel and Development (2023). *Flexible and Hybrid Working Practices in 2023*. The Chartered Institute of Personnel and Development. URL: https://www.cipd.org/globalassets/media/knowledge/knowledge-hub/reports/2023-pdfs/2023-flexible-hybrid-working-practices-report-8392.pdf (visited on 04/27/2024).

The kernel development community (2024). *Linux kernel coding style — The Linux Kernel documentation*. URL: https://www.kernel.org/doc/html/v4.10/process/coding-style.html#:~:text=The%20answer%20to%20that%20is,Heed%20that%20warning. (visited on 04/26/2024).

The Royal College of Radiologists (2022). *Clinical radiology job planning guidance for consultant and SAS doctors 2022*. URL: https://www.rcr.ac.uk/media/gzxjd252/rcr-publications_clinical-radiology-job-planning-guidance-for-consultant-and-sas-doctors_october-2022.pdf.

— (2023a). *Advisory Appointment Committee (AAC) — The Royal College of Radiologists*. URL: https://www.rcr.ac.uk/our-services/management-service-delivery/advisory-appointment-committee-aac/ (visited on 11/26/2023).

— (2023b). *Advisory Appointment Committee FAQs*. URL: https://www.rcr.ac.uk/our-services/management-service-delivery/advisory-appointment-committee-aac/advisory-appointment-committee-faqs/ (visited on 11/26/2023).

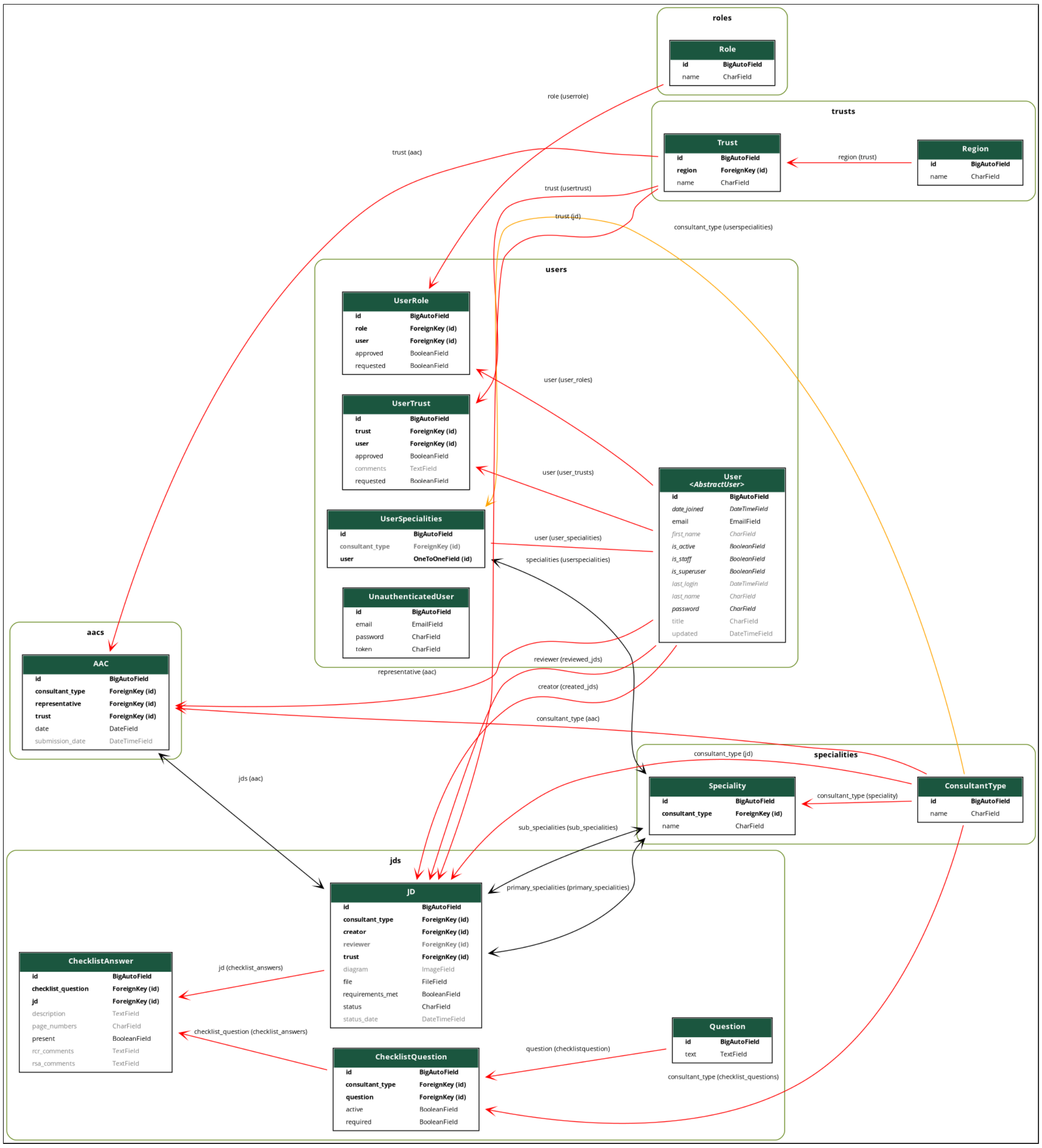
— (2023c). *RCR statement on industrial action — The Royal College of Radiologists*. URL: https://www.rcr.ac.uk/news-policy/latest-updates/rcr-statement-on-industrial-action/#:~:text=There%20is%20currently%20a%2017 (visited on 11/26/2023).

Thomas, David and Andrew Hunt (2019). *The pragmatic programmer, 20th anniversary edition: journey to mastery*. Second edition. Boston: Addison-Wesley. ISBN: 9780135957059.

Vitaliy Kucheryaviy (2024). *Django Ninja*. URL: https://django-ninja.dev/ (visited on 04/26/2024).

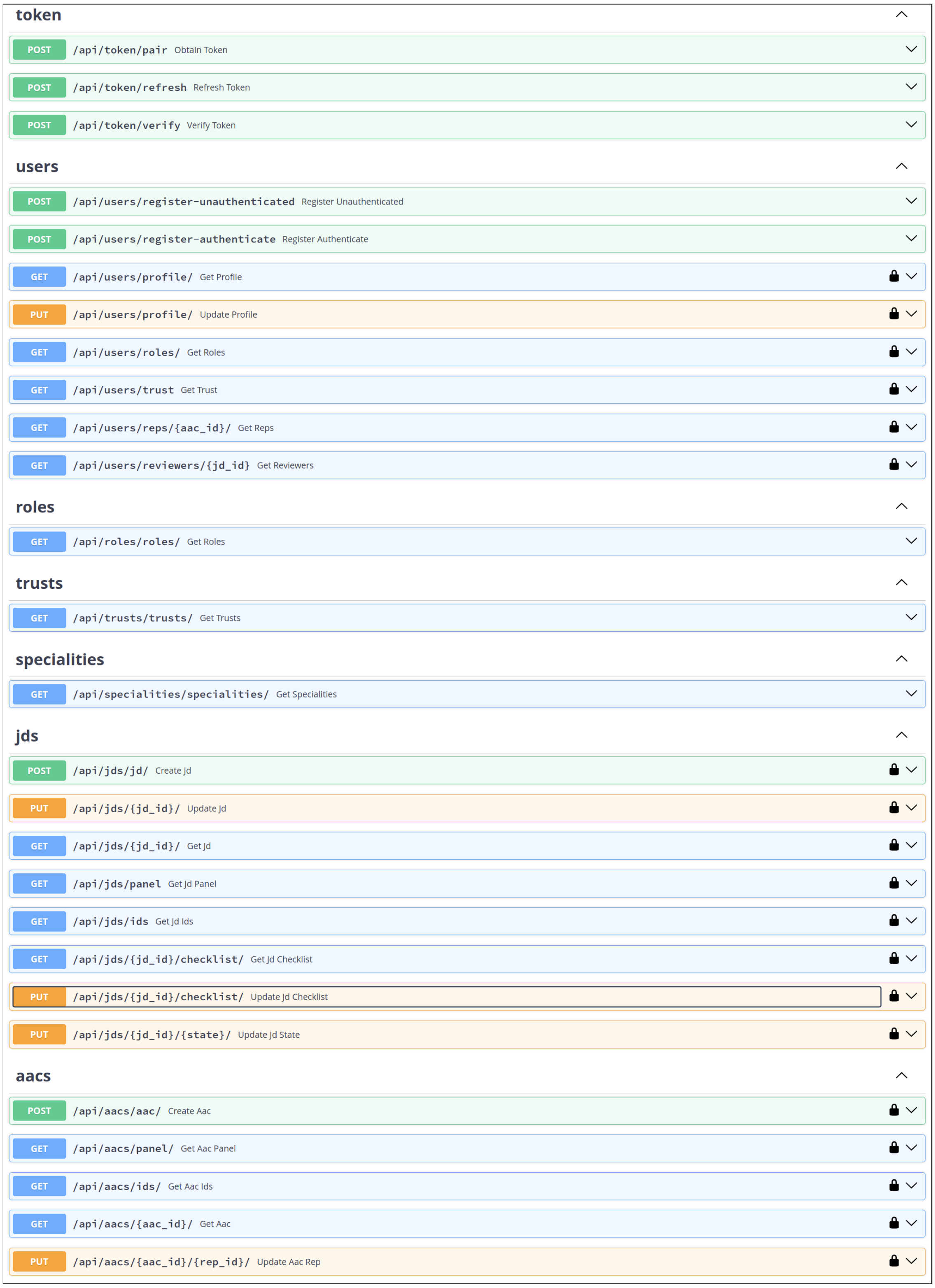
Wathan, Adam and Steve Schoger (2018). *Refactoring UI*. URL: `https://www.refactoringui.com/` (visited on 04/27/2024).

Appendix A: Database

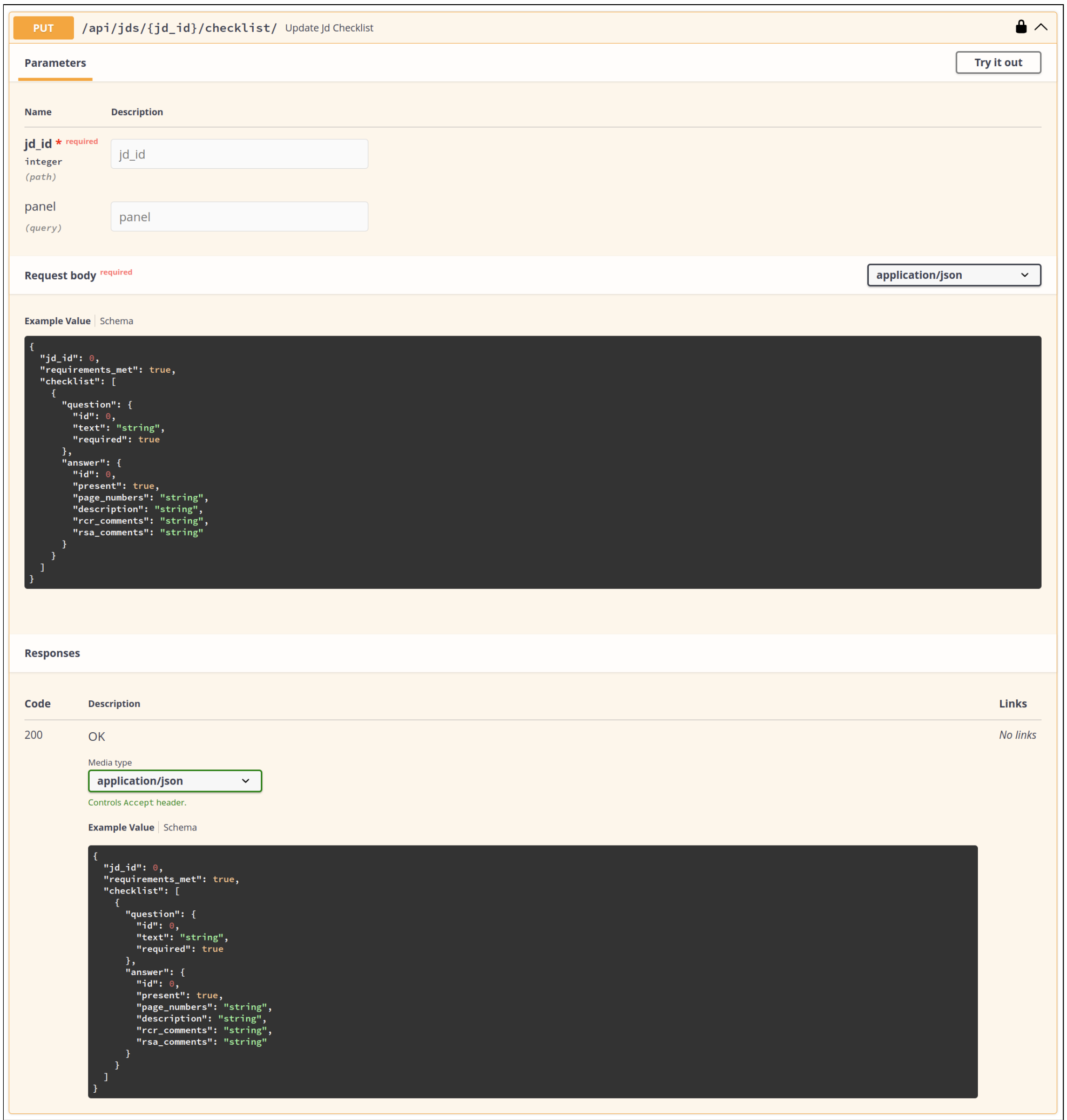


Appendix B: Endpoints

B.1 API Endpoints

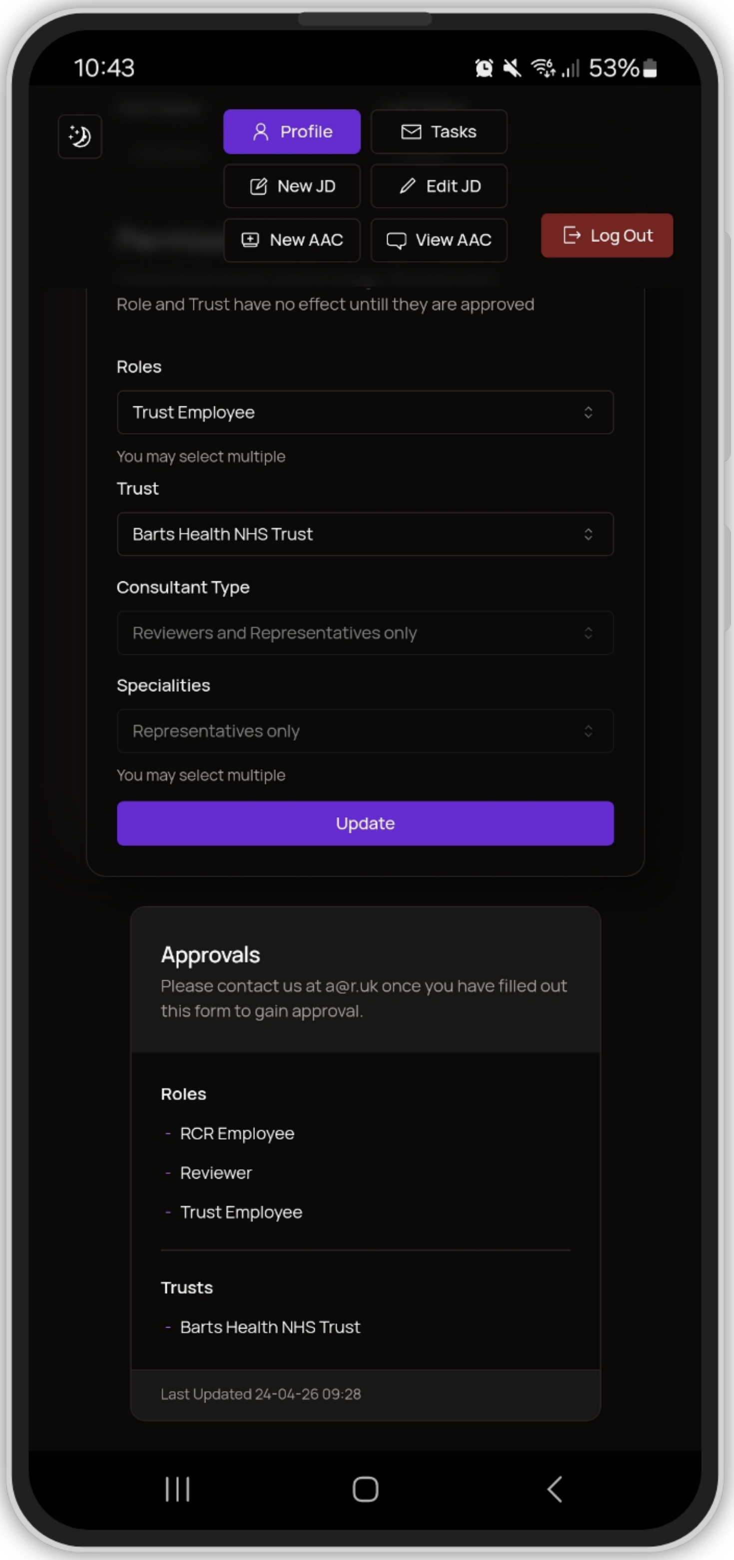


B.2 JD Checklist Endpoint



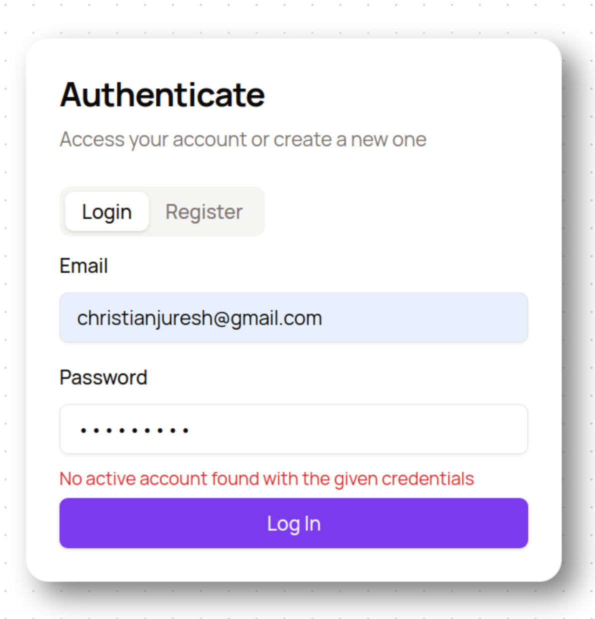
Appendix C: User Interface

C.1 Responsive & Dark Mode

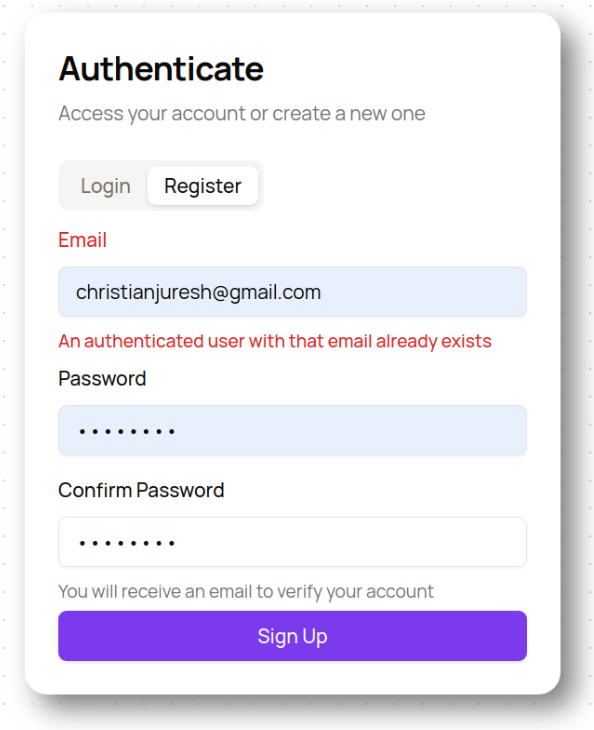


C.2 Auth

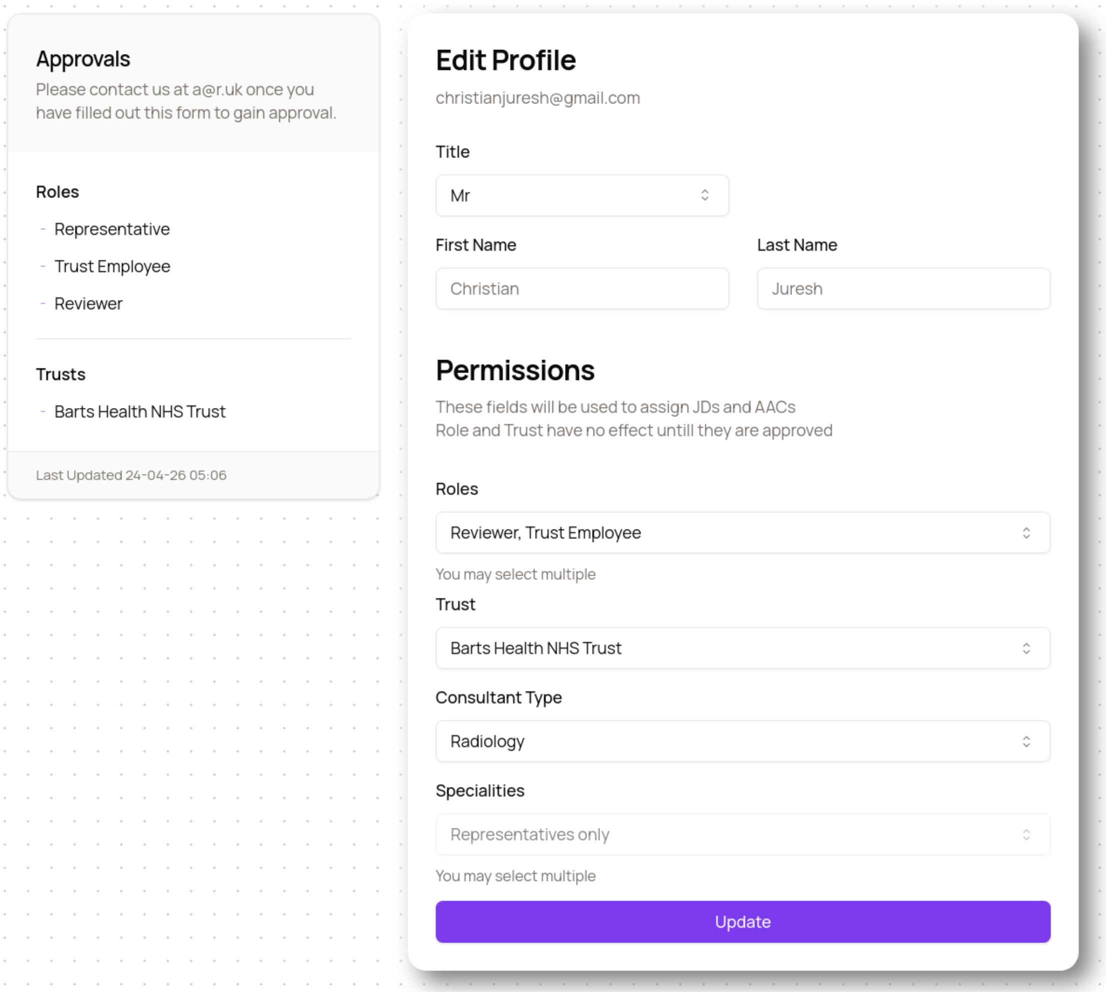
C.2.1 Login



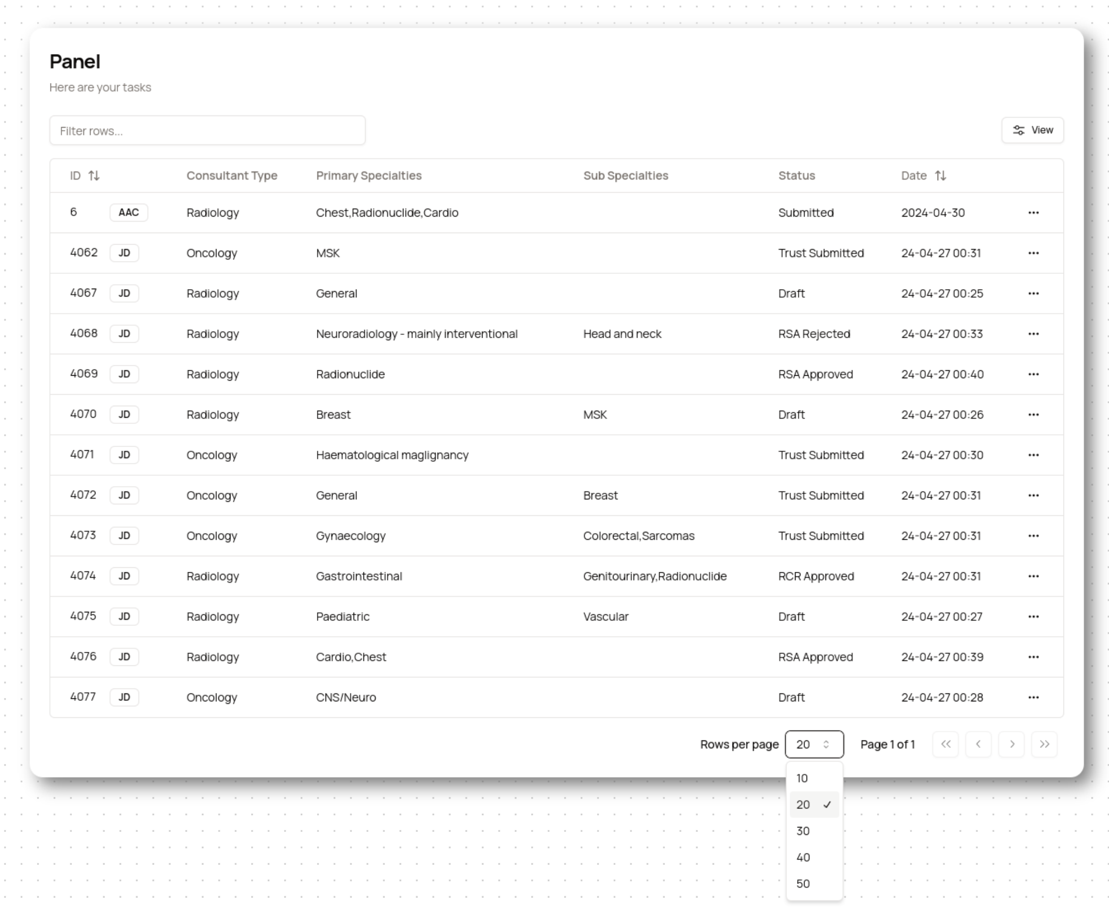
C.2.2 Register



C.3 Profile


Approvals
Please contact us at a@r.uk once you have filled out this form to gain approval.
Roles
Representative
Trust Employee
Reviewer
Trusts
Barts Health NHS Trust
Last Updated 24-04-26 05:06
Edit Profile
christianjuresh@gmail.com
Title
Mr
First Name
Christian
Last Name
Juresh
Permissions
These fields will be used to assign JDs and AACs
Role and Trust have no effect untill they are approved
Roles
Reviewer, Trust Employee
You may select multiple
Trust
Barts Health NHS Trust
Consultant Type
Radiology
Specialities
Representatives only
You may select multiple
Update

C.4 Panel



ID		Consultant Type	Primary Specialties	Sub Specialties	Status	Date	
6	AAC	Radiology	Chest,Radionuclide,Cardio		Submitted	2024-04-30	···
4062	JD	Oncology	MSK		Trust Submitted	24-04-27 00:31	···
4067	JD	Radiology	General		Draft	24-04-27 00:25	···
4068	JD	Radiology	Neuroradiology - mainly interventional	Head and neck	RSA Rejected	24-04-27 00:33	···
4069	JD	Radiology	Radionuclide		RSA Approved	24-04-27 00:40	···
4070	JD	Radiology	Breast	MSK	Draft	24-04-27 00:26	···
4071	JD	Oncology	Haematological maglignancy		Trust Submitted	24-04-27 00:30	···
4072	JD	Oncology	General	Breast	Trust Submitted	24-04-27 00:31	···
4073	JD	Oncology	Gynaecology	Colorectal,Sarcomas	Trust Submitted	24-04-27 00:31	···
4074	JD	Radiology	Gastrointestinal	Genitourinary,Radionuclide	RCR Approved	24-04-27 00:31	···
4075	JD	Radiology	Paediatric	Vascular	Draft	24-04-27 00:27	···
4076	JD	Radiology	Cardio,Chest		RSA Approved	24-04-27 00:39	···
4077	JD	Oncology	CNS/Neuro		Draft	24-04-27 00:28	···

Appendix D: Preliminary Planning

The final few figures are taken from the Interim Report and are included for the sake of completeness. They may provide insight into the project's developmental progress and initial analysis. The Risk Identification still contains important information. However, these figures are not referenced in this version of the report as some of the content is outdated or no longer relevant.

D.1 Project Plan

Week	Date	Task	Details
		Research the importance of this project and what needs to be done.	Completed in Chapter 1
		Research and document the current system	Completed in Section 2.1
		Research existing systems	Completed in Section 2.2
		Identify Risks	Completed in Chapter 3
		Create a Plan	Completed in Chapter 4
0	27 Nov 23	Interim Report Submission	Project development begins
3	18 Dec 23	Database Design	Develop a SQL database
7	08 Jan 24	Workflow Automation	Complete the workflow with Vue.js and Django
9	22 Jan 24	Security Measures	Integrate encryption, password protection, and ensure GDPR compliance
11	05 Feb 24	User Interface	Create a user friendly and presentable interface
13	19 Feb 24	System Testing	Ensure you can complete user requirements and workflows successfully
16	11 Mar 24	Documentation	Create instructions for users and developers
19	01 Apr 24	Final Report	Complete document
21	15 Apr 24	Presentation	Complete document and record video
23	29 Apr 24	Submit everything	Submit documents and implementation
6-17th May 24		Viva Presentation	

D.2 Risk Identification

Risk Description	Impact of Risk	Risk Likelihood	Risk Impact	Preventative Actions
System Implementation Failure	Delays in project completion. An unusable project.	Moderate	High	Implement and phased approach. Conduct thorough testing. Ensure deadlines are met.
Data Security Breach	Compromise of sensitive personal information. Legal consequences. Loss of trust.	Low	Very High	Employ strong encryption methods. Encourage security audits. Document the correct methods of data handling.
Poor User Adoption	Inefficient use of the new system. Failure to realise the system's benefits. Reverting to the previous system.	High	Moderate	Provide comprehensive documentation. Ensure user friendly interfaces.
Non-compliance with GDPR and NHS Regulations	Legal penalties. Reputational damage. Operational disruptions.	Moderate	Very High	Ensure the system is designed in compliance with regulations and standards. Research all laws adequately.
Technical Limitations and Bugs	System downtime. Frustration among users.	High	Moderate	Conduct extensive testing. Plan for scalability and updates.
Inadequate Scalability	Inability to cope with increasing user loads and data. System slowdowns of failures	Moderate	Moderate	Design the system with scalability in mind. Use a robust database.
Inadequate Flexibility	Difficulty in adapting to changes in recruitment processes, or simply the content of messages.	Moderate	High	Design the system with modular components. Ensure ease of updating and customisation.
Dependency on Specific Technologies	Challenges in updating or integrating with new technologies. May lead to obsolescence.	Low	Moderate	Use widely supported technologies.
Insufficient Training for Users	Ineffective use of the system. More errors and decreased productivity	Moderate	High	Develop documentation in a step-by-step format and use visual aid.

D.3 Initial Objectives and Research Questions

1.4 Objectives

1. Design and implement a robust database system.
2. Develop an automated workflow system.
3. Implement advanced data security measures.
4. Develop a user-friendly interface.
5. Ensure system scalability and flexibility.
6. Conduct thorough testing and quality assurance.
7. Provide documentation.

1.5 Research Questions

1. Which aspects of the Job Plan and Advisory Appointment Committee process can be automated?
2. How should the SQL database be structured to manage complex data relationships and allow data analysis?
3. What are the best practices for ensuring data security and GDPR compliance?
4. How can the user interface be optimised for all stakeholders?
5. Which metrics can we measure to evaluate the systems performance?

End