Doctor’s surgery management system (Doc2Home)

Thursday 23rd March 2023

Anthony Nkyi
Candidate number: XXXX
Centre name: X
Centre number: XXXXX
Qualification code: 7517

Table of Contents

ANALYSIS

Introduction

A GP (general practice) is a primary care service offered by general practitioners, doctors who treat acute and chronic illness, and are not specialised to deal with one area of the body. They are usually the first point of contact for patients with undiagnosed health concerns.

In the UK, a GP's responsibilities include:

- Responding to patient health issues by performing diagnoses, investigations, treatment, and referrals where appropriate.
- Co-operating with other medical professionals.
- Organising preventative medical programs for individuals.
- Conducting administrative work such as signing prescriptions, death certificates, fitness for work statements, reports, and letters.

A GP partner – a doctor who plays an active role in running the practice – assumes these responsibilities, alongside overseeing general business functions, which may include managing contracts, financial matters, and staff recruitment among other tasks, [1] and ensures that quality standards of care are met. This is all very difficult work, and juggling these responsibilities can be incredibly difficult and inefficient without the use of computer software in the modern age.

Solving the problem

With my NEA project, I aim to produce an application that allows staff to store all the information necessary to builds a detailed profile on each patient of a practice. This application will then use the patient's profile to aid medical professionals in administering diagnoses, treatments, and other patient-related decisions; then communicate certain information to the patient in an accessible format. It will allow doctors to schedule appointments based on patient requests. The intended end-users will be the GP doctors and partners, and registered patients. Do note that the terms 'practice' and 'surgery' will be used interchangeably across this report to refer to a GP surgery.

Primary research

To further understand the requirements of building a successful practice management system, I discussed my project with Mrs Debbie Woolmer at my school. Mrs Woolmer's previous experience working with the NHS systems such as SystmOne highlighted her as an ideal referee for my application.

Our first discussion highlighted the importance of:

- Building a system with no bias: Mrs Woolmer frequently dealt with information regarding patients from a diverse range of contexts. Based on these contexts some of the information may have seemed surprising. It is thus important to leave any presumptions out of the decision-making processes and the system algorithms.

- Communication: in Mrs Woolmer's case, most communication is performed over the phone or through written reports. As these methods have many limitations, my system should aim to take on this key responsibility to ease access and make the healthcare experience more user-focused. Ideally appointments should be available for patients as soon as possible, but lengthy phone waiting times
- Security: staff-patient confidentiality must always be maintained with the utmost respect. This means that notes should be inaccessible to staff uninvolved in a patient's care or not of a certain status. They could also be encrypted to prevent unauthorised access.

Real-life examples

FIGURE: Screenshot of the home page of the London Road Surgery website. [2]

NHS GP surgery websites use the same template as shown in the image above. While not all elements of the page are useful to analyse for my project, it should be noted that the website design could very much be improved. It is simple to navigate but inefficiently built. For example, the "Reception and Enquiries" and "Administration Office" rooms link you to pages that are almost identical, and therefore should be merged.

At the back end, GP surgeries currently use the integrated computer system SystmOne to store health records for every single patient, which can be accessed by primary care providers involved in a certain patients. Patients are also able to use this service to order prescriptions and book appointments. [3] To emulate this, my program will use a database including a table for key patient personal information, and many linked relations to store relevant information that can be accessed by authorised users.

Data requirements

Key information such as identity, home address, health conditions, prescriptions, allergies and reaction to medicines, test results, care history and lifestyle information is all stored by the NHS in a patient's record. [4] Each end-user must therefore have a unique login and access level, where patients can view their own records, and staff can only view records of the patients they engage in the care of. Medical professionals should also be able to store further notes on patients after appointments, to communicate with the patient and to make co-ordinated decisions on the patient's healthcare.

This information must all be kept confidential unless absolutely necessary to share or it is in the public interest to do so, and patients have the right to object to the sharing of their confidential information, even if done anonymously. [5] This will inform design considerations, especially for the modelling of algorithms and access levels for records.

In a process that took almost two weeks, I was able to gain access to my personal medical records for use in the research for this project. The wait was prolonged by multiple security verifications. While the NHS's methods are difficult to emulate in the context of my project, it showed that retrieving your records is evidently difficult due to the sheer amount of information they store on you. Once again, not all information is necessary, or possible, to be recorded in the scope of my project, but I believe that with efficient organisation the information could be condensed to a screen with an easy-to-navigate user interface.

My personal record contained information on:

- Full name, date of birth and address
- Gender
- NHS Number
- Active health problems
- Significant and minor past health problems
- Appointment/consultation history, and notes
- Repeat and past doses
- Allergies
- Vaccination and immunisation history
- Referrals
- Test results
- Height and weight measurements
- A&E visits, diagnoses, and outcomes
- Blood type

Diseases

Diseases can be communicable or non-communicable, where communicable diseases are caused by pathogens and can be spread between organisms.

There are 5 types of pathogen:

- Viruses (e.g., flu, COVID-19)
- Bacteria (e.g., meningitis, gonorrhoea)

- Fungi (e.g., ringworm, athlete's foot
- Protozoa (e.g., malaria, amoebic dysentery)
- Worm (e.g., tapeworms)

Meanwhile non-communicable (chronic) diseases can be caused by pathogens, and fit into one of these categories: [6]

- Cardiovascular diseases
- Diabetes
- Cancers
- Chronic respiratory disease
- Injuries
- Mental health conditions (not covered by the scope of this project)

All diseases known to have affected a patient should be stored, for better understanding and to record additional notes on them. Furthermore, new diseases are discovered frequently and thus there should be an option to add these, in the interests of keeping informed records on current medical research.

Scheduling considerations

An average part-time GP partner works 39.3 hours a week. Over 25% are working more than 45 hours a week. The average full-time partner works 56.3 hours per week, with 12% working over 70 hours, [7] while the average doctor is only contracted to work 40 hours. This is way above the weekly standard of 37.5 hours for those in other professions. The scheduling algorithm may have to accommodate this fact to provide partners with adequate time off and a reasonable workload. The usual length of a GP appointment is 15 minutes.

Algorithm building considerations

My primary research highlighted the importance of realising that serious illness can be developed by people of any age, sex, or ethnicity. While these attributes may influence correlation to risk factor, you must ensure that these are accounted for with as little bias as possible.

App framework considerations

The nature of the app means that it must be built with a database to store all the information necessary to run a surgery, including but not limited to that mentioned in the data requirements section. Therefore, a database structure should be built and accessible via certain methods through the website. Similarly, the app needs to be available on a local network for multiple staff to log in and collaborate with each other. This means a combination that the Flask and SQL-Alchemy libraries are best equipped for building this app.

Using these libraries in a virtual environment, I built a prototype web app for logging into and creating a doctor account. This showed that it is possible to connect to the database from the app, for reading or writing.

Jinja2 and JavaScript will also be used for some processing, client-side. This will be used for functions such as IF statements to display the correct page for different user types, and for UI changes such as autocomplete submissions, among others.

For resetting passwords, users will need email verification to access the page. For this, I will require my application to have access to an auxiliary email which can then distribute the links on request. For a Flask web application, the Flask-Mail module is the most streamlined to integrate. To increase security, each reset link contains a JSON web token (pyjwt).

Database considerations

The app's database must contain multiple tables based on the information required to access and maintain updated and consistent records. This will include tables on:

- Patient personal information: this will need to store NHS Numbers, names, emails, dates of birth, sex, heights, weights, blood types, addresses, and password hashes.
- Doctor information: emails, access levels, NI Numbers, forenames, surnames, gender, dates of birth, addresses, and password hashes.
- Appointments: the doctor, the patient, the test date and time, the room, whether it was attended, feedback for patients viewing, and notes for medical staff.
- Doctor hours spent: the doctor, the date, the clock-in time, and the clock-out time.
- Medicines: the name, and the recommended dosage.
- Prescriptions: the medicine, the patient, the doctor who first prescribed it, the start date, the dosage, and the date it was last repeated.
- Known and potential allergens: allergen names.
- Medicines containing these known and potential allergens: medicines and their ingredients that are also allergens.
- Each patient's allergies: the patient, the allergen, and the severity.
- Known diseases: names and whether they are inheritable.
- Carriers of inheritable diseases: the patient and the disease they carry.
- Addresses: house number, street address and postcode.

Patient account creation

This program makes 2 assumptions:

1. Every patient creates an account upon joining the GP and thus information is only stored on users with accounts.
2. Every user is legitimate, i.e. not performing identity theft to view another individual's private and confidential information.

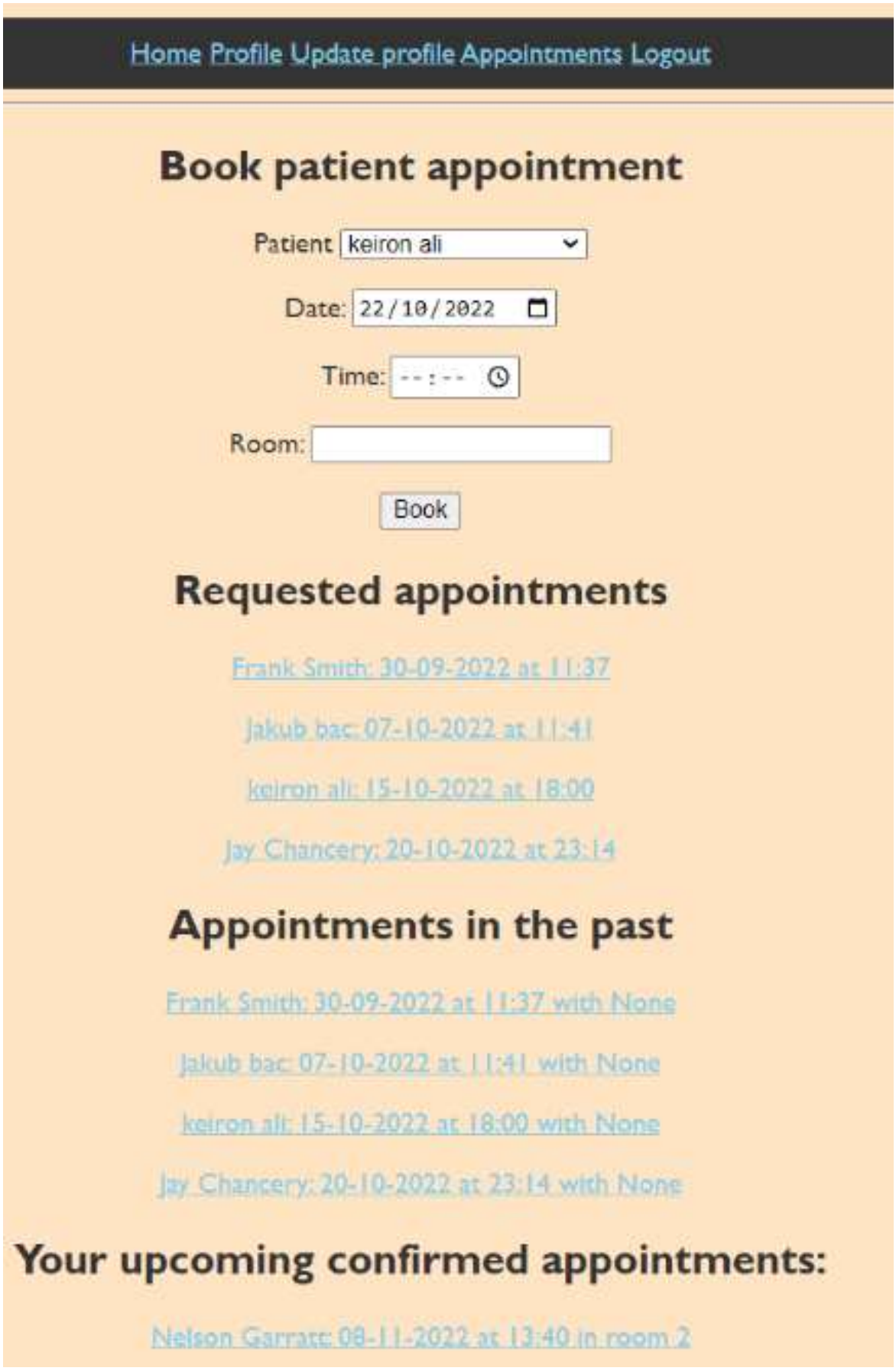
Note that in real life, accounts usually require in-person GP approval before two step verification. My program bypasses this step as per the above assumptions.

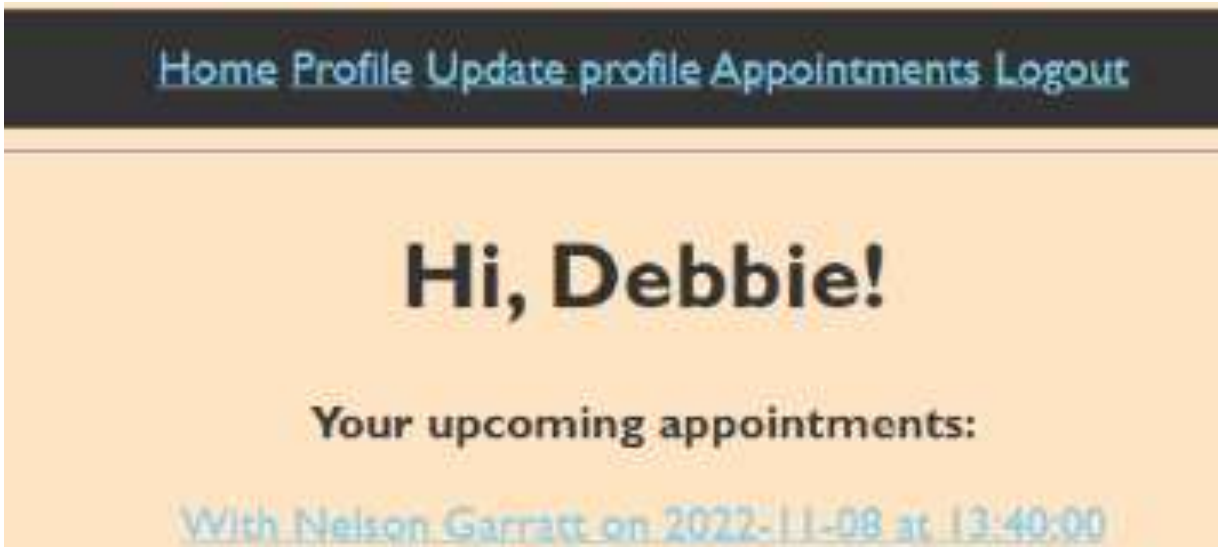
If an NHS number is valid, it maps to stored records already inside the database. This means that a patient only needs an NHS number, email, and password to create an account. Details such as name, date of birth, sex, blood type, height and weight are already known by the GP. However, while name can be inputted, and date of birth must be submitted by the patient as a form of validation, in this instance these other details must be randomly generated via a suitable distribution. This is to aid with medical procedure and advice provision.

- As of December 2018, to the nearest whole number, UK blood type distribution is:
    - O positive: 35%
    - O negative: 13%
    - A positive: 30%
    - A negative: 8%
    - B positive: 8%
    - B negative: 2%
    - AB positive: 2%
    - AB negative: 1% [8]
- UK adult male height is normally distributed with a mean of 175.3cm and a standard deviation of 7.42cm. UK adult female height is normally distributed with a mean of 161.9cm and standard deviation of 7.11cm. [9]
- Using UK growth charts for children, a height may be generated based on age using median and centiles to inform a model. [10] [11] [12]
- Weight cannot necessarily be modelled using a suitable distribution so it must be inputted by the user.

Prototyping

Following this analysis, I built a basic prototype for my practice management application, with systems to login and create profiles for both patients and staff. Both doctors and patients may request or book appointments for themselves.


Home Profile Update profile Appointments Logout
Book patient appointment
Patient keiron ali
Date: 22/10/2022
Time: --:--
Room:
Book
Requested appointments
Frank Smith: 30-09-2022 at 11:37
Jakub bac: 07-10-2022 at 11:41
keiron ali: 15-10-2022 at 18:00
Jay Chancery: 20-10-2022 at 23:14
Appointments in the past
Frank Smith: 30-09-2022 at 11:37 with None
Jakub bac: 07-10-2022 at 11:41 with None
keiron ali: 15-10-2022 at 18:00 with None
Jay Chancery: 20-10-2022 at 23:14 with None
Your upcoming confirmed appointments:
Nelson Garrate: 08-11-2022 at 13:40 in room 2



FIGURES: The screenshots show Mrs Woolmer's journey through the application, where she (1) logs in after registering her account, (2) visits the appointments page and creates an appointment for a patient at the surgery, and (3) is redirected to the home screen after confirming the appointment, which is displayed.

Program objectives

1 To use the GP application, a user must be able to log in.

1.1 A patient can create an account if they have not already done so.

1.1.1 The patient can input their NHS number and other details into the registration page.

1.1.2 The program must check the NHS number used to create an account is valid, and then if it is already in use.

1.1.3 If the NHS number or the email is already in use by a doctor or patient, an error message is shown, and the patient is asked to re-enter their information.

1.1.4 If all the details entered are valid, a height is automatically generated based on the age and gender inputted, via suitable subroutine(s).

1.1.5 A blood type is randomly selected based on the current UK proportions.

1.1.6 The account is added to the patient database, including a password hash and address id, and they are redirected to the login page.

1.2 An account may be created for a doctor if they have not already done so.

1.2.1 The account creation must have the following details inputted: email; access level; NI Number; date of birth; street address; postcode; password (twice).

1.2.2 If email or NI Number is already in use by a doctor or patient, an error message is shown, and the patient is asked to re-enter their information.

1.2.3 If all the details entered are valid, the address entered is searched for in the address database.

1.2.3.1 If found in the database, that id is added to the doctor's record.

1.2.3.2 If the address is not in the database, it is written to the database as a new address.

1.2.4 The account is added to the doctor database, including a password hash and address id, and they are redirected to the login page.

1.3 When a user's inputted registration details are valid, the address entered is searched for in the address database.

1.3.1 If found in the database, that address id is added to the doctor's record.

1.3.2 If the address is not in the database, it is written to the database as a new address.

1.4 A user can log in at the login page, which displays email, password, remember me and password reset fields.

2 If the user selects the password reset option and enters a valid email, they are sent an email with a unique link to reset their password.

2.1 The reset link expires within 10 minutes to ensure account integrity.

2.2 At the reset password page, the user is prompted to enter their new password twice.

2.2.1 If the new password is the same as the old password, the user is prompted to re-enter the details.

2.2.2 If all the fields are valid, the new password's hash is added to the database.

3 The user is displayed a home page upon log-in success.
3.1 If the user is not logged in, the home asks them to log in or register to view their information, which redirects them to the relevant page.
3.2 If the user has any appointments upcoming in the future, they are displayed on the dashboard with a link to each individual appointment.
3.3 If the doctor has any appointments in the past which they have not submitted notes for, they are displayed on the dashboard with a link to each individual appointment.
3.4 If the patient is logged in, it displays a dashboard with their name, date of birth, and NHS Number.
3.5 If the patient has any current prescriptions, they are displayed on the dashboard with a link to each individual prescription.

4 The logged in user can create an appointment.
4.1 The patient selects an appropriate date and time, which is added to the database for approval.
4.2 Once the appointment is created, doctors can approve the appointment by assigning a room.
4.3 The appointment is checked for clashes before being approved.
4.3.1 If the doctor or room involved has already been assigned an appointment that begins within 15 minutes of the start of said requested appointment, said appointment must be re-assigned to another doctor, moved to a different room, or date-time.
4.4 The patient should have the option to cancel the appointment.
4.5 After the appointment has been fulfilled, a doctor may add notes to the appointment for medical professional viewing, and feedback for the patient to follow up on.
4.5.1 The patient feedback must be able to be viewed by the patient, and all the medical professionals involved in the patient's care, or doctors of access level 1.
4.6 Doctors can record test results that are linked to appointments by date and NHS Number.
4.7 A test may be added after all relevant information regarding a fulfilled appointment has been saved in the database.
4.8 The type of test each individual record is assigned will be selected from a list generated from the previous records.
4.9 Doctors may add notes to a test, that can be viewed by both the patient and other doctors.

5 Registered doctors and staff will be able to search for medicines by name.
5.1 If the doctor has the access level 1, they will be able to add and modify medicines.
5.1.1 To add a medicine, the doctor inputs the name of the medicine.
5.1.2 If the medicine does not exist, they are prompted to add the further details: recommended dose and any allergens in the medicine.
5.1.3 If the medicine does exist, they redirected to a page displaying its details (name, allergens and patients prescribed it) before submitting.

6 Registered doctors with access level 1 or 2 may add allergens to the database.
6.1 Registered doctors can link allergens to a patient's profile.
6.1.1 Each patient's allergy is given a severity rating from 1 (life-threatening) to 5 (minor).

6.2 If a patient is allergic to a certain allergen, they cannot be prescribed a medicine containing the allergen.
6.3 Any doctor may view a patient's verified allergies.
6.4 If a new allergen is added to a medicine, the prescription is automatically ended if the recipient is allergic to it.

7 Registered doctors with access level 1 will be able to add diseases and illnesses to the system.
7.1.1 The disease must be assigned a unique name.
7.2 The disease must be assigned a status to show if it is inheritable.

8 Doctors of any access level will be able to link inheritable diseases to a patient's profile.
8.1 Any doctor will be able to view the traits a patient carries.

9 Doctors can link previous illnesses to a patient's profile, so there is a comprehensive account of the patient's medical history.
9.1 The form will take the disease, date of diagnosis and date of recovery, alongside any notes the doctor wants to make.
9.2 Referrals may also be recorded detailing the occasions where a patient had a medical intervention outside the GP, taking the disease, date, referral type and any notes.

10 Users can view prescriptions.
10.1 Doctors may create a prescription by choosing a patient they have recently had an appointment with, a verified medicine, the disease it is for and a dosage.
10.1.1 If the patient is already prescribed on the medicine, the old prescription is found and ended on today's date. The new prescription is then created with the information submitted and saved to the database.
10.1.2 If the patient is not currently prescribed on the medicine, the medicine is checked for allergies.
10.2 Doctors of any access level can view prescriptions given to any patient via a search function.
10.3 Patients may view their own prescriptions – the medicine and dose to be administered.

11 Hours spent at the surgery are recorded so doctors do not spend more than a certain amount of time there over the course of a week.
11.1.1 If a doctor is access level 1, they may not work more than 48 hours in a 7-day week or 11 hours in a day.
11.1.2 If a doctor is access level 2, they may not work more than 43 hours in a 7-day week or 10 hours in a day.
11.1.3 If a doctor is access level 3, they may not work more than 38 hours in a 7-day week or 9 hours in a day.
11.2 The hours spent are recorded by the total duration of appointments, plus intervals between appointments that are less than 1 hour long.

DOCUMENTED DESIGN

Overview

The home page will greet the user with a message prompting the user to login or register, either as a doctor or as a patient. After inputting the relevant personal details and gaining access to their password-protected account, they will be able to access user type specific features. This includes viewing your own medical record and profile (with the opportunity for minor edits), appointments and prescriptions from a patient's perspective. For a doctor, this extends to adding and modifying medicines, prescriptions, test results, diseases and referrals (which include vaccination), depending on their pre-defined access level.

Note: over the following sections, all id numbers including NI Numbers and NHS Numbers are randomly generated, and do not reflect real links between accounts and named personnel.

Database system

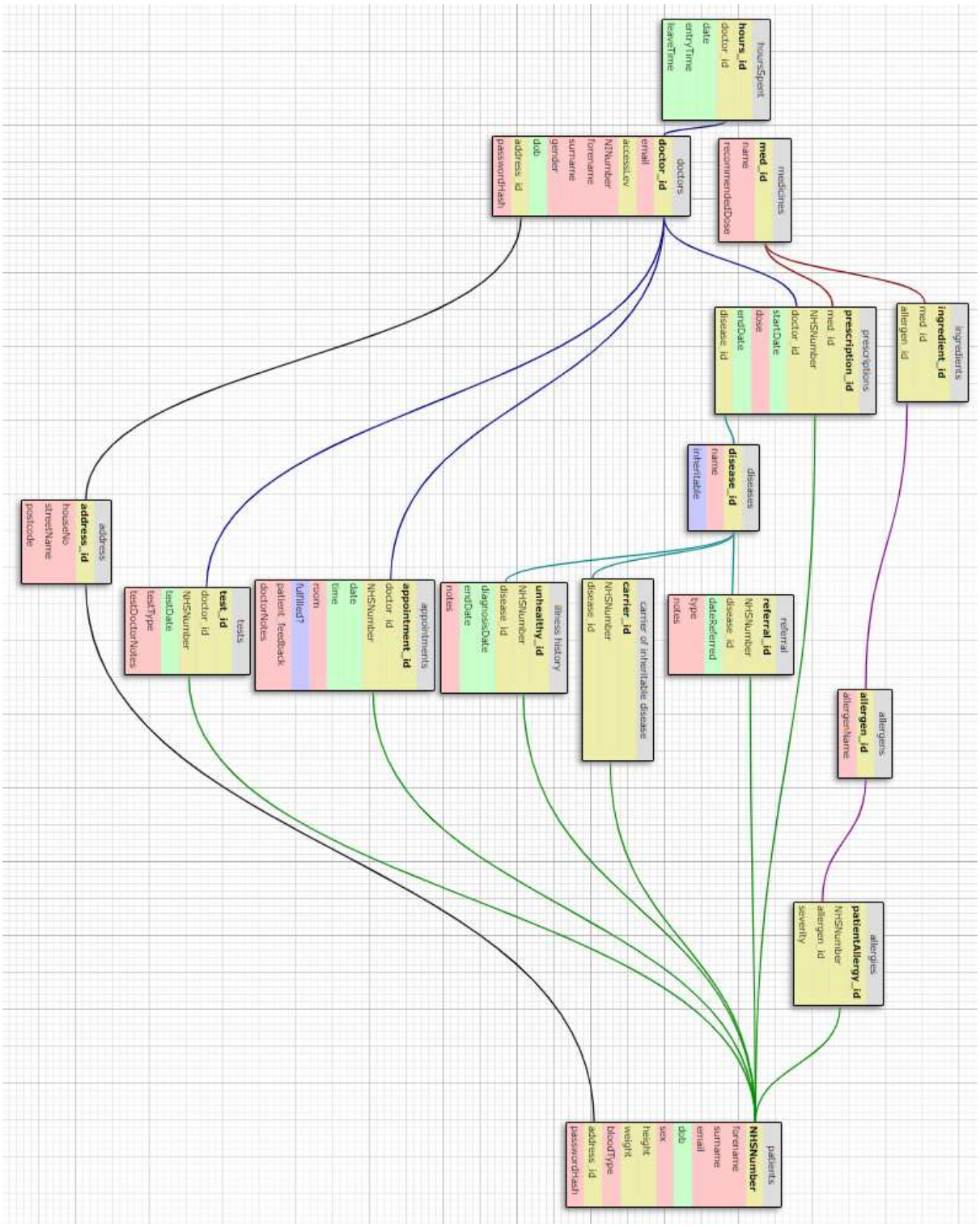


FIGURE: Entity relationship diagram displaying how the relations are linked by their keys, and each relation's columns.

The most suitable method for my web-based application with a back-end database was to use Flask-SQLAlchemy for the construction, and most communication. I built the relations using classes inheriting from these models, and the Flask migration feature, and attributes corresponding exactly to a column in its table. Furthermore, my decision to build the database in third normal form allows for straightforward communication and querying.

Additional database notes

Below are additional notes explaining the structure of select relations inside this database. Primary keys are in **bold**, while foreign keys are in *italics*.

Doctors table

- The primary key of each doctor user account will be a user ID ('doctor_id') generated independent of the NI Number, for security reasons.
- Contains the fields:
    - **Doctor id: integer, primary key**
    - **Email:** string, max length 50 characters
    - **Access level:** integer, range 1-3
    - **NI Number:** string, length 9 characters
    - **Forename:** string, max length 30 characters
    - **Surname:** string, max length 50 characters
    - **Gender:** character, options M,F,O
    - **Date of birth:** date
    - ***Address id:*** *integer, foreign key*
    - **Password (hashed):** string, length 128 characters

Patients table

- The primary key 'NHSNumber' is inputted by the user: an individual's NHS Number is a completely unique ID. This also allows for streamlined searching functions on the doctor's side.
- Patient height is randomly generated by a subroutine modelling a normal distribution based on estimated values for the mean and standard deviations of height, per age.
- Patient blood types is randomly generated based on actual proportions of the UK population.
- Every record contains the columns included in the doctor table, except for doctor id (primary key is NHS Number), access level and gender (replaced with sex).
- It also includes **height** (integer range), **weight** (float), and **blood type** (string).

Address table

- When a user (patient/doctor) account is created, their address is searched for in the address table. If not present, it is added. The id of the user's address is then written to the user's record and linked as a foreign key.
- Contains the fields:
    - **Address id: integer, primary key**
    - **House number:** string, max length 5 characters
    - **Street name:** string, max length 36 characters
    - **Postcode:** string, max length 7 characters

Hours-spent table

- The doctor's hours are automatically added to the table, recorded by the algorithm detailed by the accumulation of time spent subroutine (covered later).

Contains the fields:

- **Hours id: integer, primary key**
- ***Doctor id:*** *integer, foreign key*
- **Day:** date
- **Entry time:** time
- **Leaving time:** time

Prescriptions table

- Every prescription has an individual primary key. When changes are made to an individual prescription, a new record is created for it.
- Every prescription is always linked to a disease. This is for two reasons:
    - It allows for analysis of treatments in relation to each disease.
    - There is no need to be prescribed anything by a GP if you are not ill!
- Contains the fields:
    - **Prescription id: integer, primary key**
    - ***Medicine id:*** *integer, foreign key*
    - ***NHS Number:*** *integer, foreign key*
    - ***Doctor id:*** *integer, foreign key*
    - **Start date:** date
    - **Dose:** string, max length 2048 characters
    - **Final repeat date:** date
    - ***Disease id:*** *integer, foreign key*

Allergens table

- The allergens table is created to catalogue every substance known to cause an allergic reaction to a patient at the surgery. Thanks to the normalisation of the database, this allows the 'allergies' table (patients are linked to allergens they have a reaction to) and

the ‘ingredients’ table (medicines available for prescription are linked to allergens they contain) to be separated.

- Contains the fields:
    - **Allergen id: integer, primary key**
    - **Allergen name:** string, max length 128 characters

Diseases table

- Like the allergens table, normalisation allows documented diseases to be linked to multiple tables, such as the ‘carriers’ table, where patients are linked to inherited diseases they carry.
- Contains the fields:
    - **Disease id: integer, primary key**
    - **Disease name:** string, max length 128 characters
    - **Inheritability:** Boolean

Referral table

- Each referral has a unique primary key – however, it is not composite because there would be too many foreign keys required to create a completely different one for each record.
- Every record is also linked to a disease foreign key to assess hospitalisations for different illnesses.
- Contains the fields:
    - **Referral id: integer, primary key**
    - ***NHS Number:*** *integer, foreign key*
    - ***Disease id:*** *integer, foreign key*
    - **Diagnosis type:** string, max length 64 characters
    - **Diagnosis date:** date
    - **Referral notes:** string, max length 2048 characters

Appointments table

- Each appointment has a ‘fulfilled’ status which allows the surgery to evaluate attendance.
- Contains the fields:
    - **Appointment id: integer, primary key**
    - ***Doctor id:*** *integer, foreign key*
    - ***NHS Number:*** *integer, foreign key*
    - **Room:** string, max length 64 characters
    - **Fulfilled:** Boolean
    - **Patient feedback:** string, max length 1024 characters
    - **Doctor notes:** string, max length 2048 characters
    - **Appointment date:** date

- **Appointment time:** time

Tests table

- Contains the fields:
  - **Test id: integer, primary key**
  - ***Doctor id:*** *integer, foreign key*
  - ***NHS Number:*** *integer, foreign key*
  - **Test date:** date
  - **Test type:** type
  - **Doctor test notes:** string, max length 2048 characters

Medicine table

- Contains the fields:
  - **Medicine id: integer, primary key**
  - **Medicine name:** string, max length 64 characters
  - **Recommended dose:** string, max length 128 characters

Ingredients table

- Contains the fields:
  - **Ingredient id: integer, primary key**
  - ***Medicine id:*** *integer, foreign key*
  - ***Allergen id:*** *integer, foreign key*

Allergies table

- Contains the fields:
  - **Allergy id: integer, primary key**
  - ***NHS Number:*** *integer, foreign key*
  - ***Allergen id:*** *integer, foreign key*
  - **Severity:** integer, range 1-5

Inheritable disease carrier table

- Contains the fields:
  - **Carrier id: integer, primary key**
  - ***NHS Number:*** *integer, foreign key*
  - ***Disease id:*** *integer, foreign key*

Illness history table

- Contains the fields:
    - **Unhealthy id: integer, primary key**
    - ***NHS Number:*** *integer, foreign key*
    - ***Disease id:*** *integer, foreign key*
    - **Diagnosis date:** date
    - **End date:** date
    - **Notes:** string, max length 2048 characters

Website map

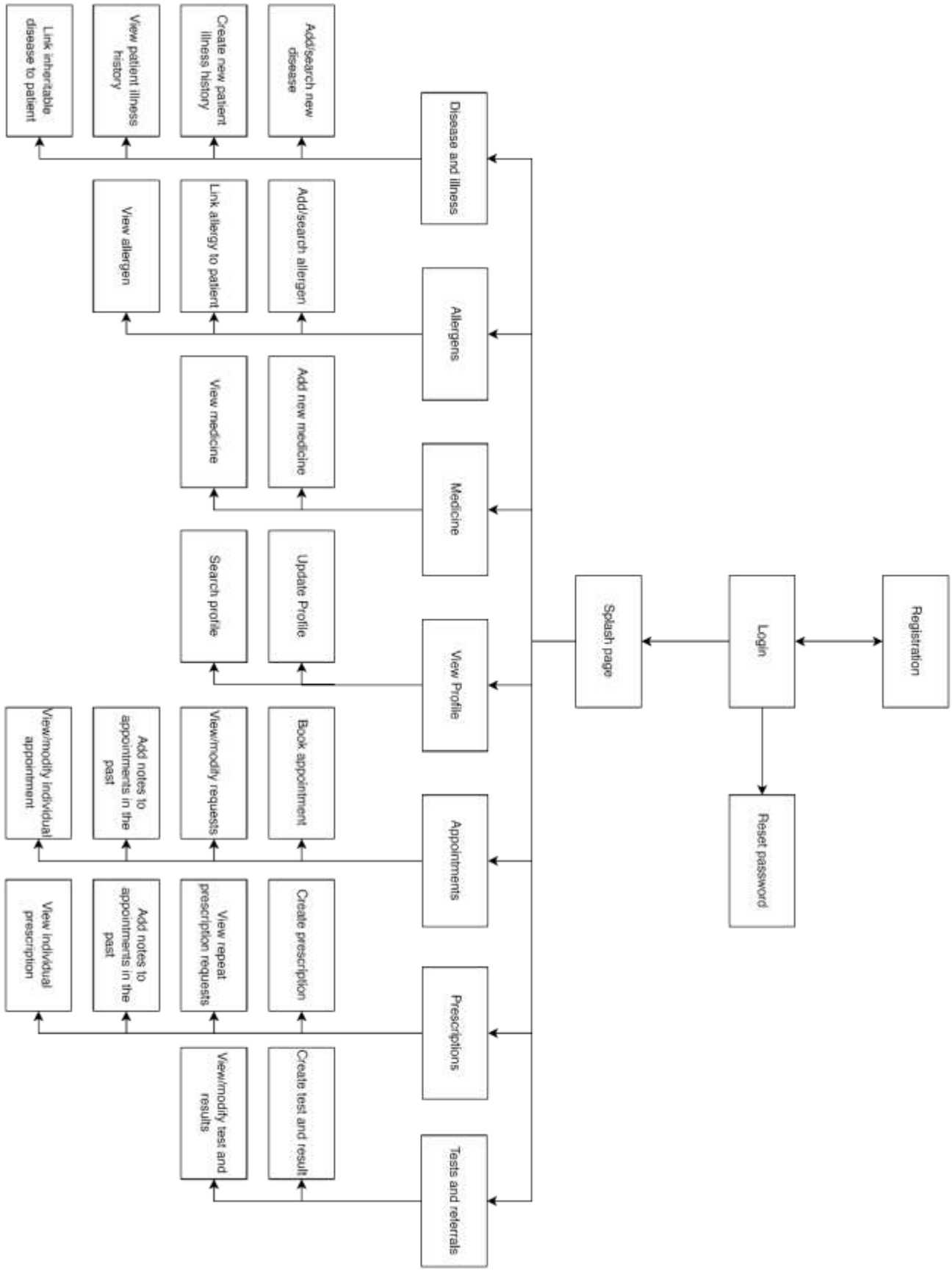


FIGURE: Website map for the Doc2Home web application.

My web application is built with intended end-users being medical staff and patients. This means the website interface should be accessible to all demographics of user – simple, clean, and efficient.

The next sections will detail the sections displayed on the application sitemap, including the relevant techniques to demonstrate the designs of processes.

File structure and hierarchy

The program is run as a virtual environment, meaning the files are decentralized and the program can be modularised. (Note: in the following key, pure file names are expressed with “” quotes around them while folders are not).

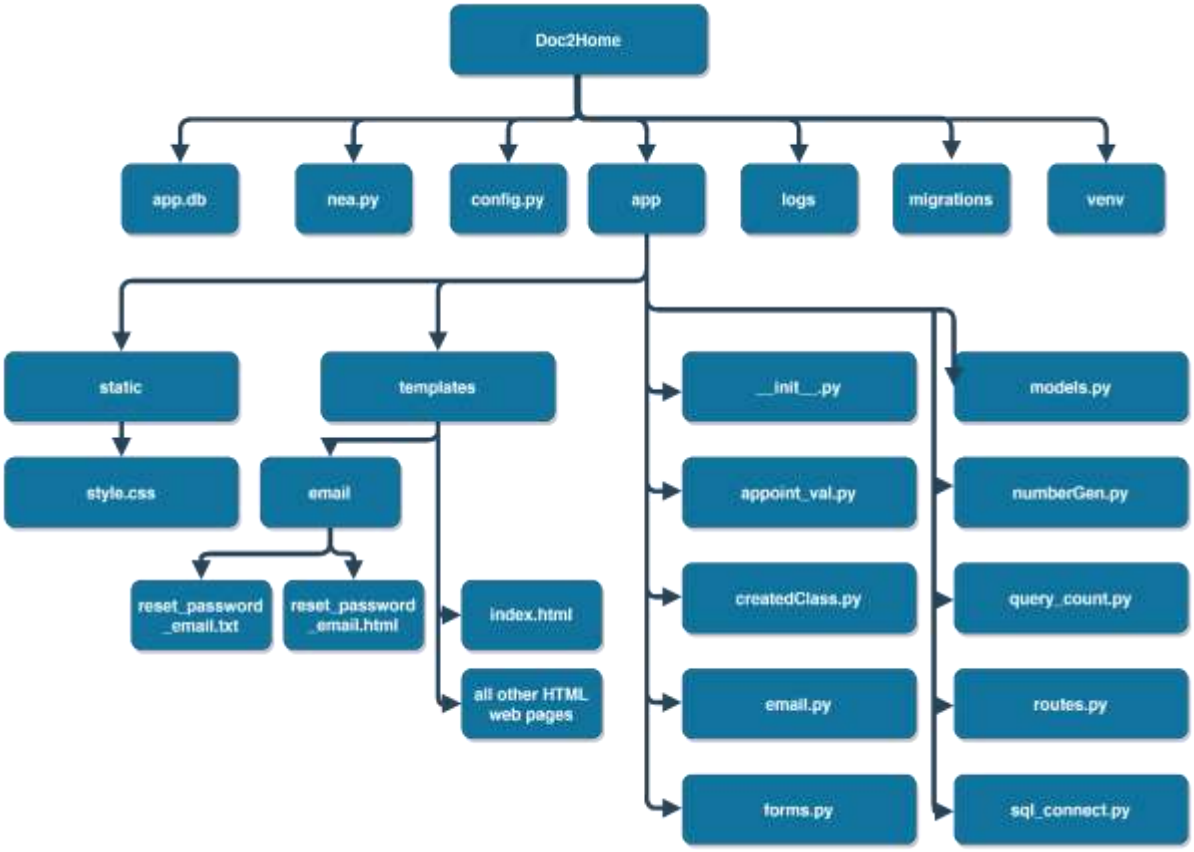


FIGURE: Folder hierarchy for the Doc2Home application.

Top level

- **“app.db”:** The database used to store information from the Flask app’s running.
- **“nea.py”:** Imports the Flask app and the database for use when the virtual environment is called to run the Flask application.
- **“config.py”:** Configuration file containing key information for the running of the app, including the secret keys for web forms, the location of the database, mail server ports, and mail account details for the sending of emails.
- **app:** Contains the main modules of the program, including the constructor, Flask routes, web forms, database models, and other modules required for the running of the app.
- **logs:** Stores a log file that documents errors when the app is NOT running in debug mode.
- **migrations:** Stores details of all the changes made to the database through the virtual environment’s Flask-Migrate actions.

/app folder

- **“__init__.py”:** Constructs the instance of the Flask app, imports configurations, routes, the database class, the migration class and its model classes, the login handler class, and the Mail class. It is also used as a directory by the Flask app when required to load resources.
- **“routes.py”:** Runs the functions required when a URL of the Flask app is visited. Imports from the templates folder to display web pages.
- **“appoint_val.py”:** Stores the modules that accumulate time spent at the practice on any given day, and calculates whether a doctor will be working overtime.
- **“createdClass.py”:** Stores the Queue and hidden values classes.
- **“email.py”:** Runs the functions required for email handling, necessary for the resetting of passwords.
- **“forms.py”:** Stores the Flask web form classes that are imported in the webpages.
- **“models.py”:** Stores the database model classes linked to each table – these are used to query and modify the database.
- **“numberGen.py”:** Contains the functions used to check if the user is a patient or doctor; check the age of the user; generate a random height based off a normal distribution model; generate random NHS and National Insurance Numbers by following real world criteria; and select a blood type for the user based on real world proportions.
- **“query_count.py”:** Imports from createdClass.py to generate a queue based on the results of a SQLAlchemy query, and another function counts the length of the query result.
- **“sql_connect.py”:** Contains the class object that connects the program to the database for complex SQL “SELECT” queries.
- **static:** contains CSS for the web application.
- **templates:** contains all the web templates and pages used by routes.

Defensive design techniques

Authentication

For every user operation except registration, a way of verifying the identity of a user is required. Patients have limited access and cannot access all the functions a doctor can. Meanwhile, a level 1 doctor can access functions and screens a level 2 or 3 doctor cannot. This is inbuilt into the application with Jinja2 and Python back-end selection statements.

Resetting the password requires two-step authentication – a user logs out, requests a password reset link to their email, and the link allows the user to change their account's password.

Validation

FlaskForm is used to ensure that the majority of user inputs are validated before being sent to the back-end. The validations include length, range, equal to, presence, data type, and format checks. This ensures that if the data passes further validity checks, it will not cause any unexpected results if queried for or added to the database.

Furthermore, every form submitted has a check for duplicate information before any database tables are updated, to ensure normalisation.

Sanitisation

Passwords are encrypted and hashed using the Werkzeug security module, by a user object method. This means the only time a password is requested by the API is when it is being hashed, minimising the amount of opportunities for interception

All other strings being saved to the database are saved in either all caps or lower case characters. This reduces the chances of case-sensitive results causing unexpected outcomes during runtime.

When NI Numbers are displayed on doctor profiles, asterisks cover 6 out of 9 characters to ensure security.

Application processes

Patient registration

For the registration and login sections, the SQLAlchemy model classes allowed me to take an object-oriented approach to reading, writing, and modifying the tables. All forms used in the program were generated by FlaskForm from flask_wtf. The credentials for users are detailed in the database system design section.

For the user tables (i.e., doctor and patient), the UserMixin class from Flask-Login was also inherited. The user classes are set attributes corresponding to each column in the table. A user's password is then hashed using the Werkzeug security module once it passes validation checks. Then it is written to the database, along with all the other fields.

The diagram below demonstrates how a patient would create an account to log in and view their record. The NHS numbers and dates of birth are also subject to additional validation checks for 'realism'. In the real world, the NHS number is a ten-digit number so requires a length check, and a child would not create or be authorised to create an account to view their records.

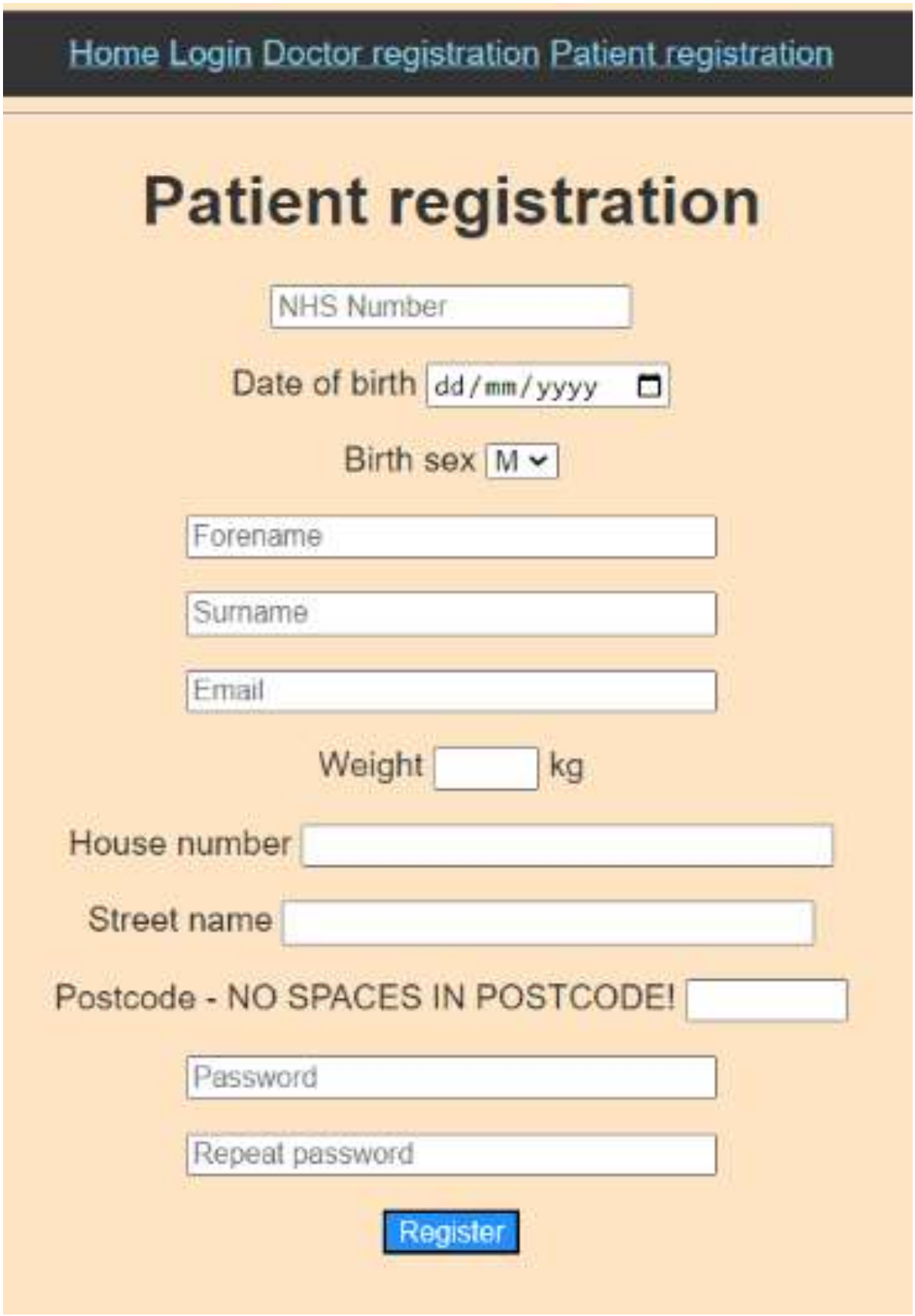


FIGURE: Screenshot of the patient registration form for a new patient.

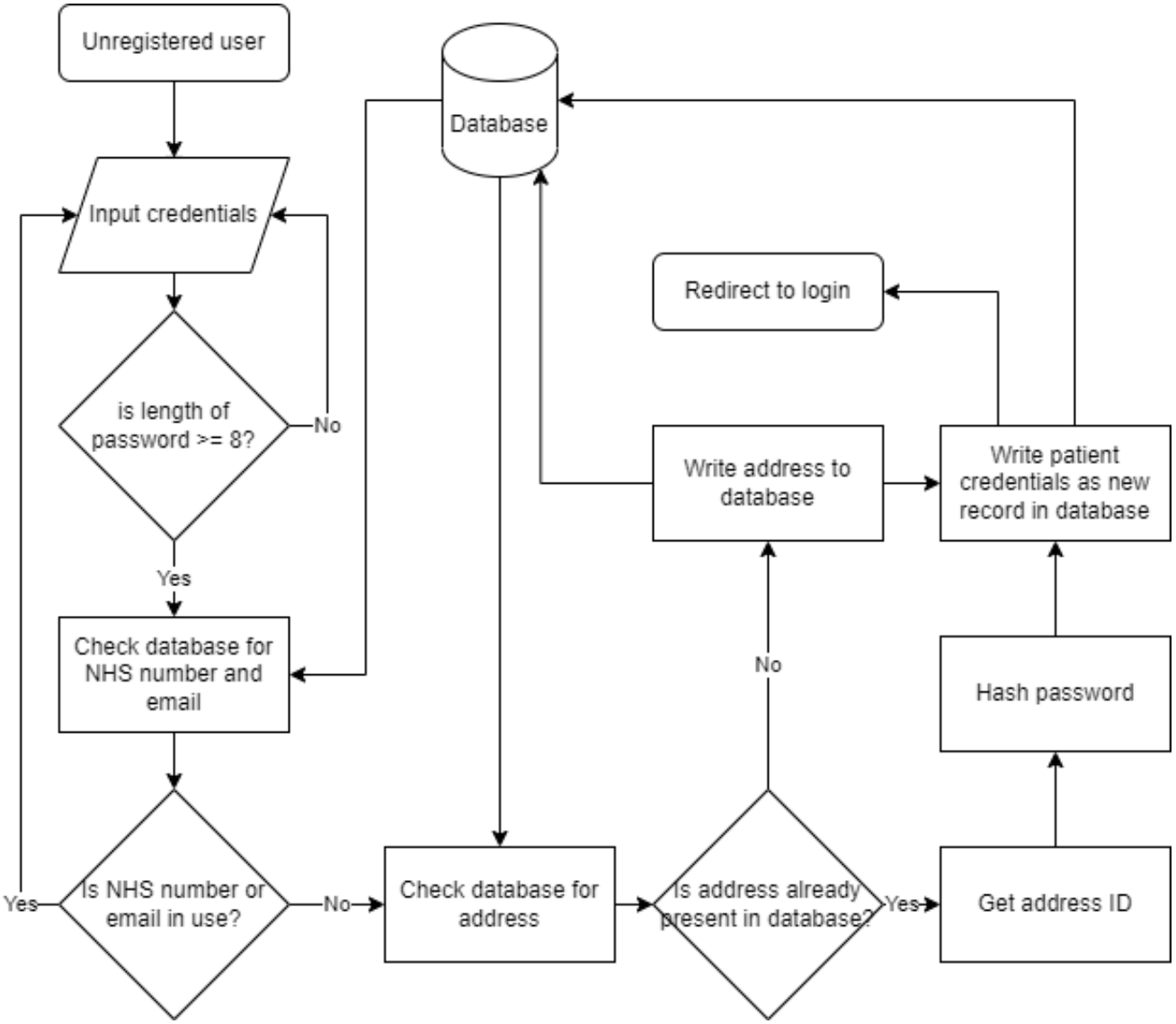


FIGURE: Flowchart demonstrating the patient registration process.

Doctor registration

The next figures demonstrate the similar process of registering as a doctor. Unlike the patients table, every doctor is assigned a unique ID, because the use of a National Insurance number as it would not be in the best interests of staff information security. The NI Number is also subject to format checks based on real-world parameters. [13] Doctors are also assigned an access level based on status in the GP - level 1: partner, level 2: senior doctor, level 3: junior doctor.

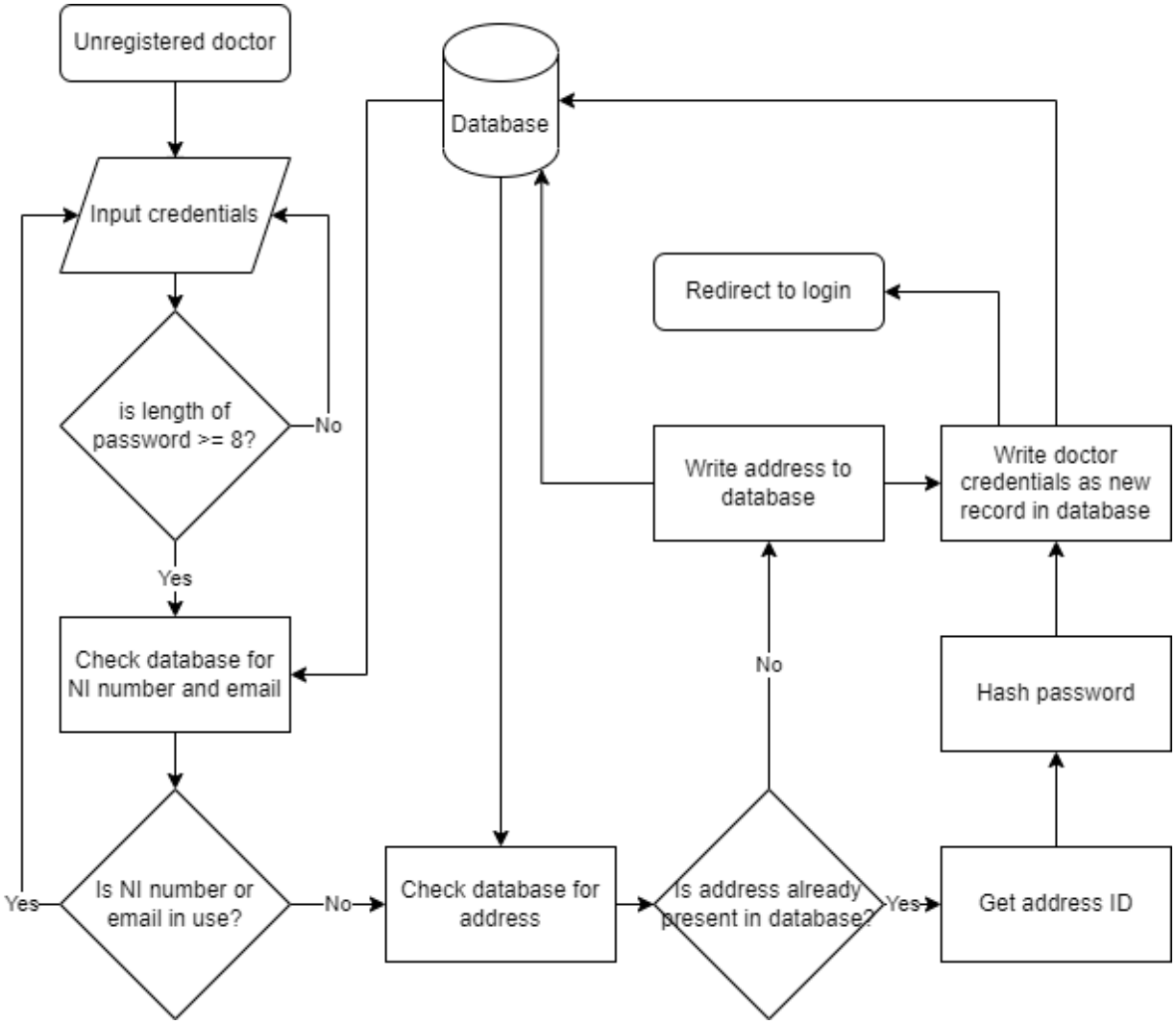


FIGURE: Flowchart demonstrating the doctor registration process.

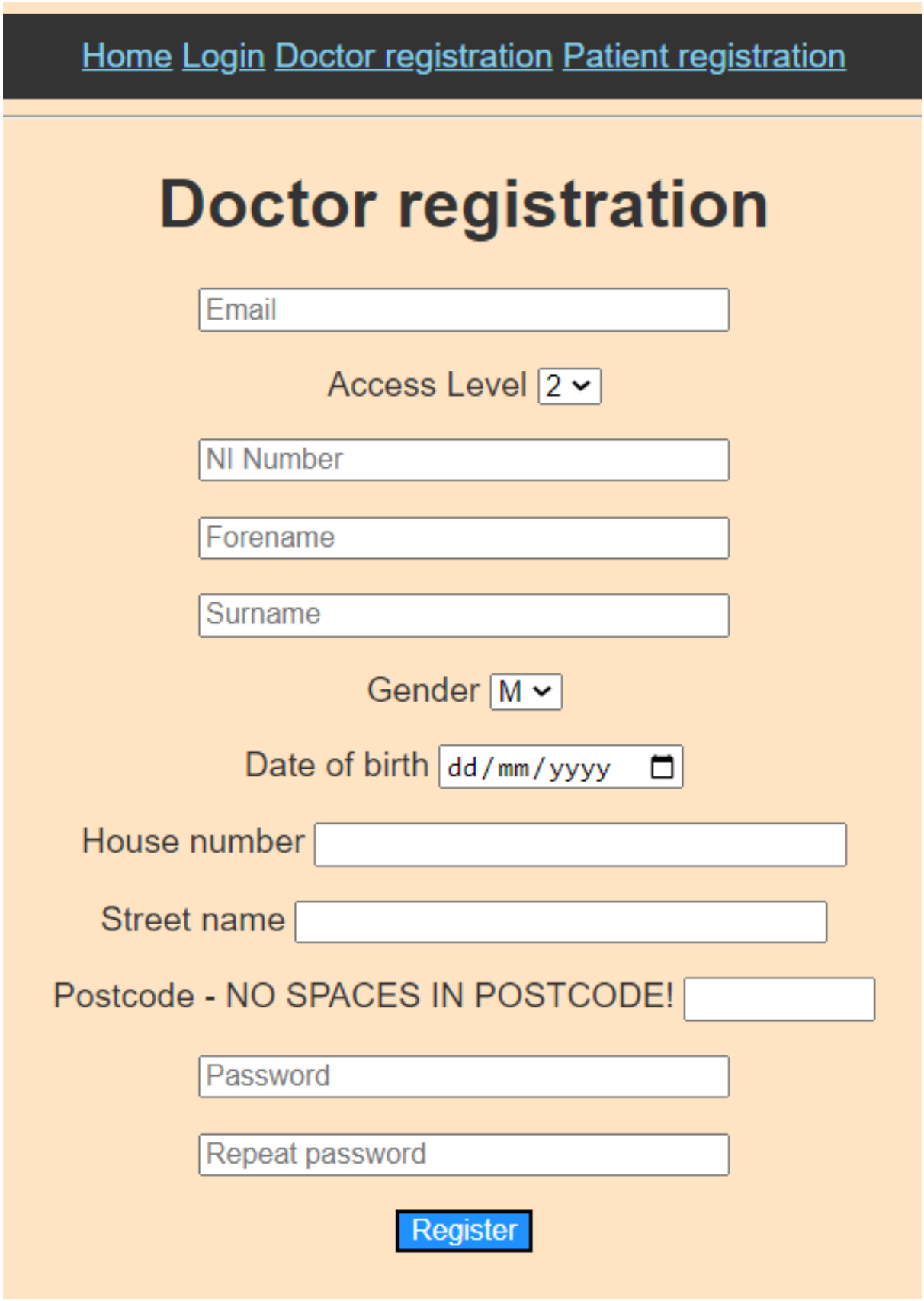
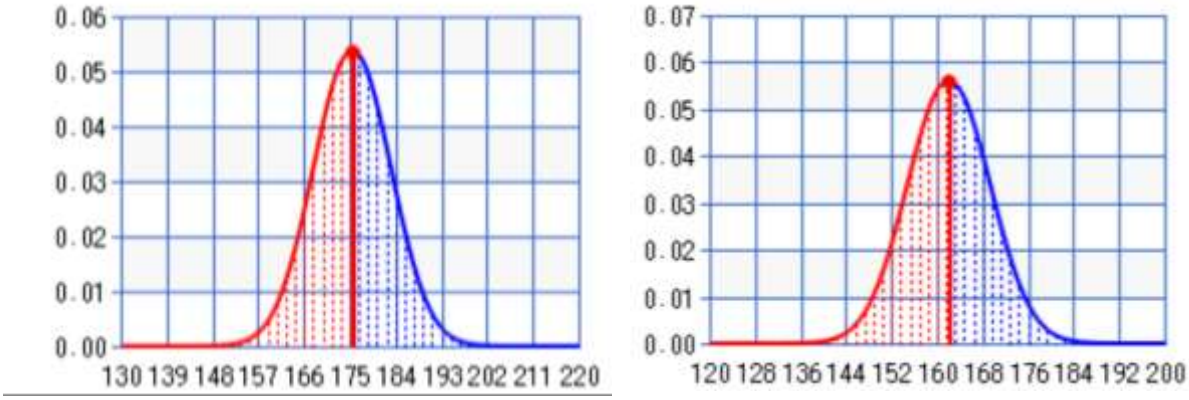


FIGURE: Screenshot of the doctor registration form for a doctor with no account.

Height, weight and blood type generation

To increase realism, as a GP would actually have this information about each user, the height, weight, and blood type of each patient is generated upon the creation of their account.



FIGURES: Normal distribution models of the heights of UK adult males (left) and females (right) in centimetres.

As height can be modelled by a normal distribution (or bell curve), so the height generator randomly selects a value from the model – the chance of being selected is based on how close the value is to the mean, which is shown by the highest point on the bell curve.

Weight distribution cannot be modelled easily due to its right skew, and therefore it is inputted by the user.

Blood type is generated randomly for each user off the probabilities detailed in the analysis section.

Login

After the patient has registered, they are redirected to a login page. The page's login functions (including the 'remember me' function) are handled by the Flask-Login module [14], which loads the 'email' and 'password hash' fields from the doctors and patients tables.

FIGURE: Screenshot of the login page for all registered users.

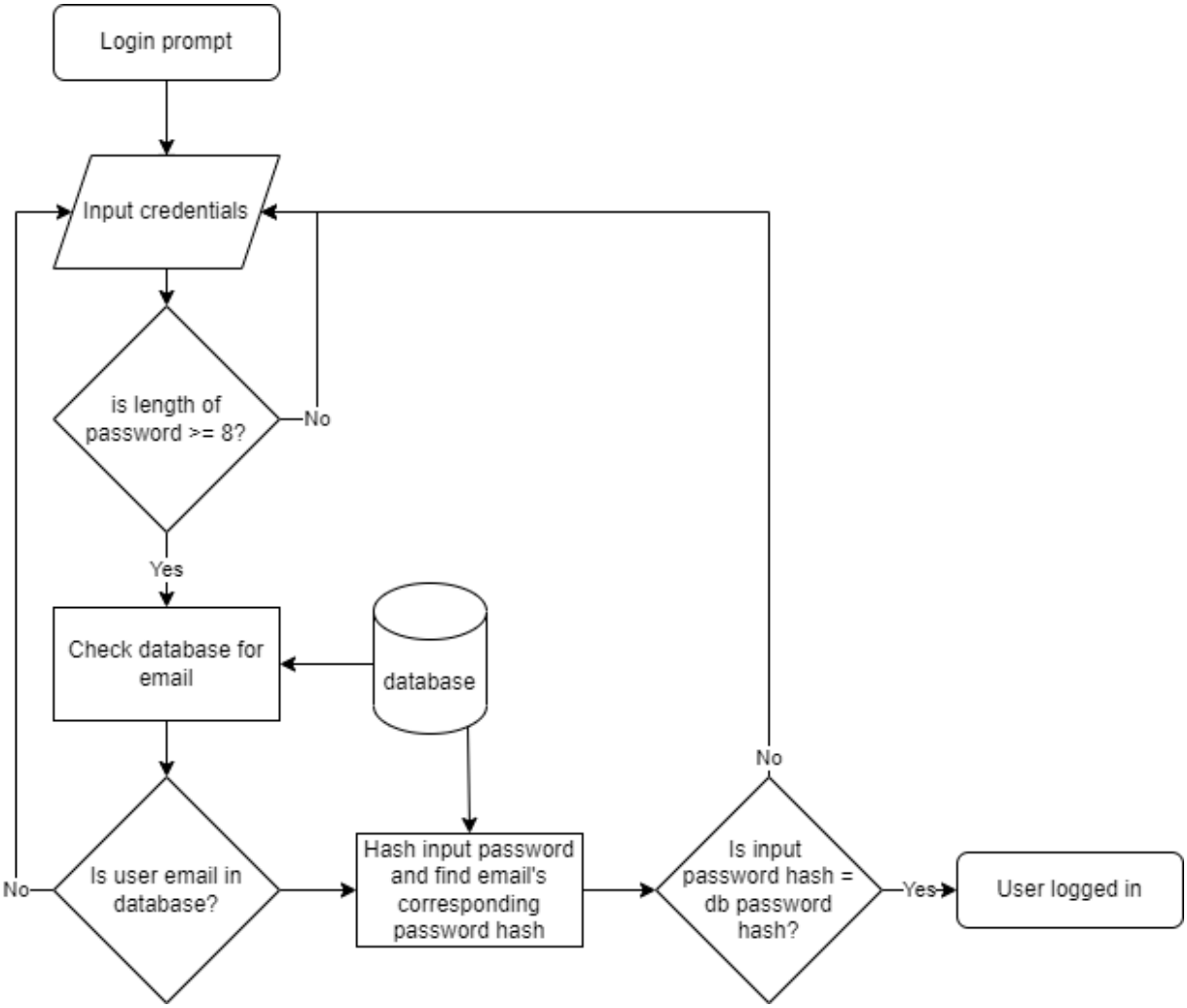


FIGURE: Flowchart demonstrating the login process.

Resetting user password

Users will need to reset their passwords from time to time, for security reasons or to regain account access. In this instance, there is an in-app tool that allows a user to enter the previous password and the new one twice, to change it.

However, sometimes a user may completely forget their password, in which case a reset link must be sent to their email to create a new password. The flowchart below outlines the email reset process.

The email is sent asynchronously to ensure minimal slowing of the application while it is running, using a thread.

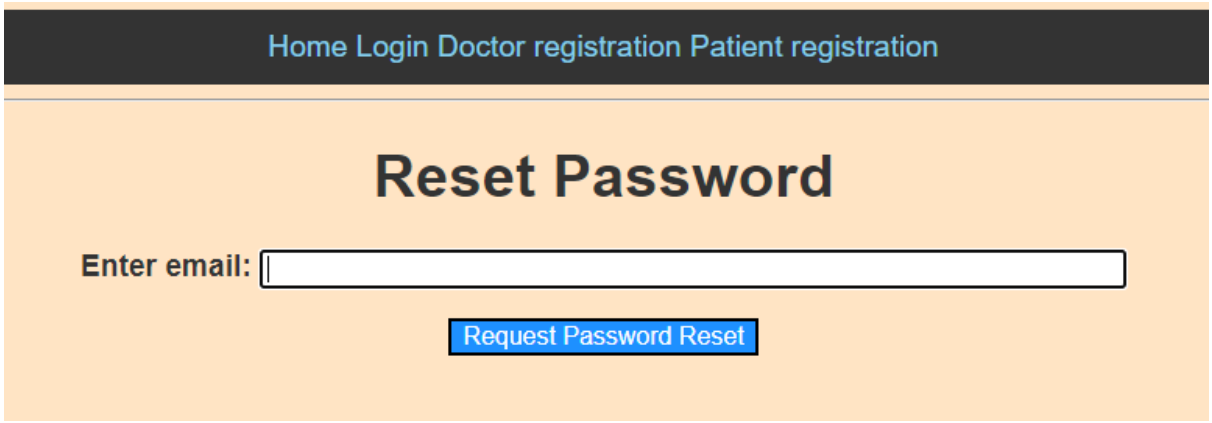


FIGURE: Screenshot of the reset password request page where a user enters their email.

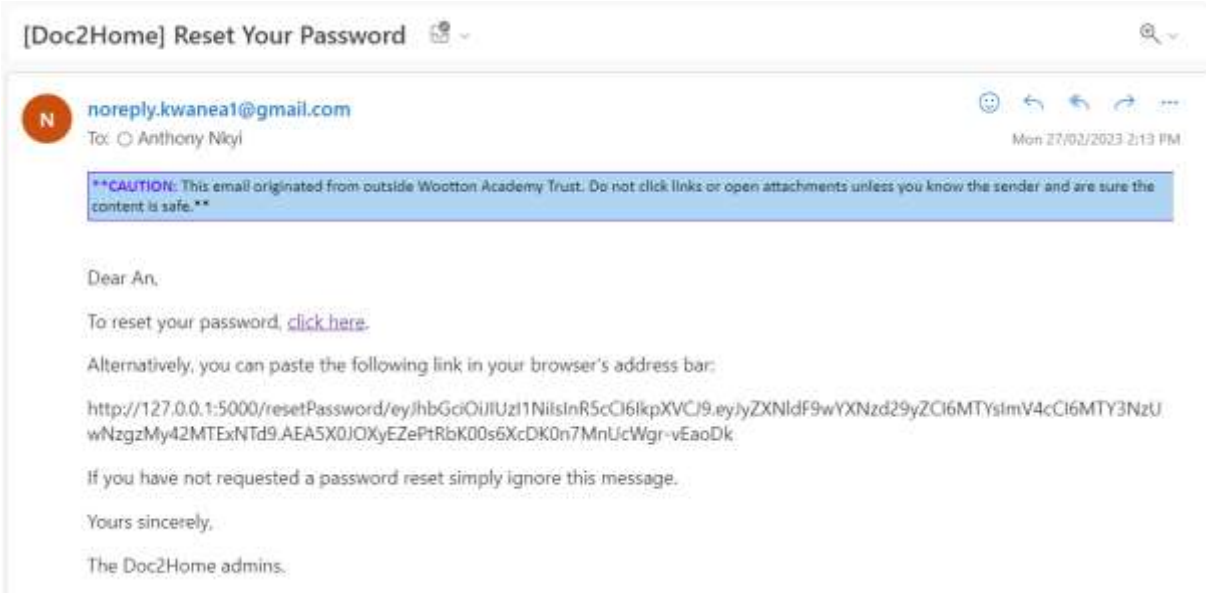


FIGURE: Reset password email sent after entering email.

FIGURE: Screenshot of the reset password page, where a user is directed after clicking the emailed link.

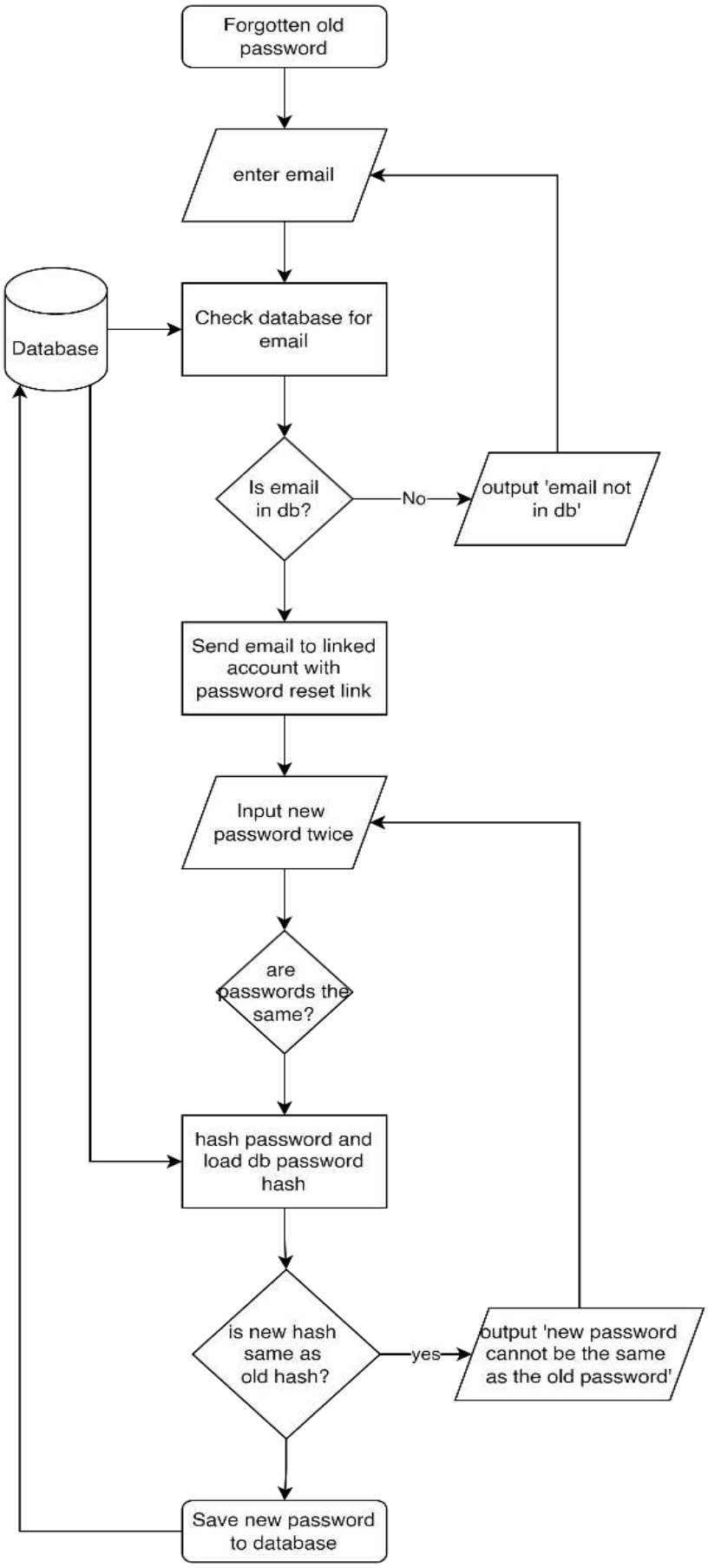


FIGURE: Flowchart demonstrating the process of resetting a password with a valid email.

Appointments

Once logged in, a patient may request an appointment by navigating to the ‘book appointment’ screen.

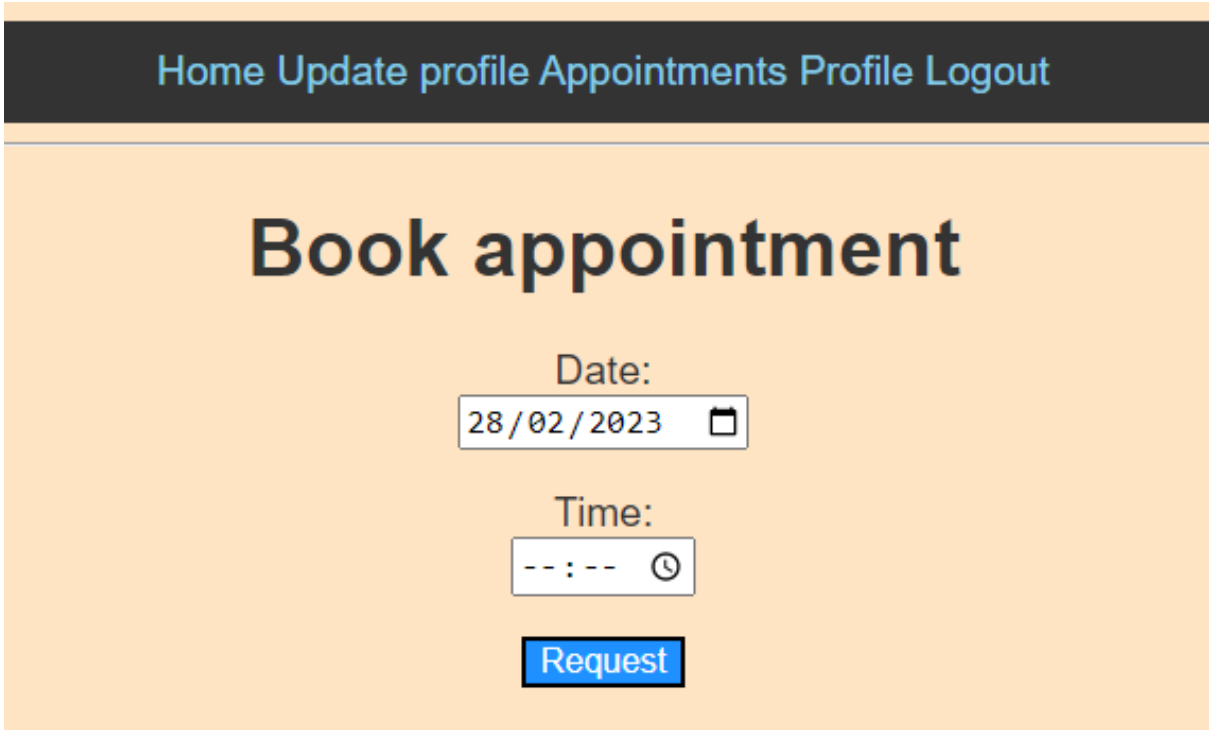


FIGURE: Screenshot of the appointment booking page for a patient.

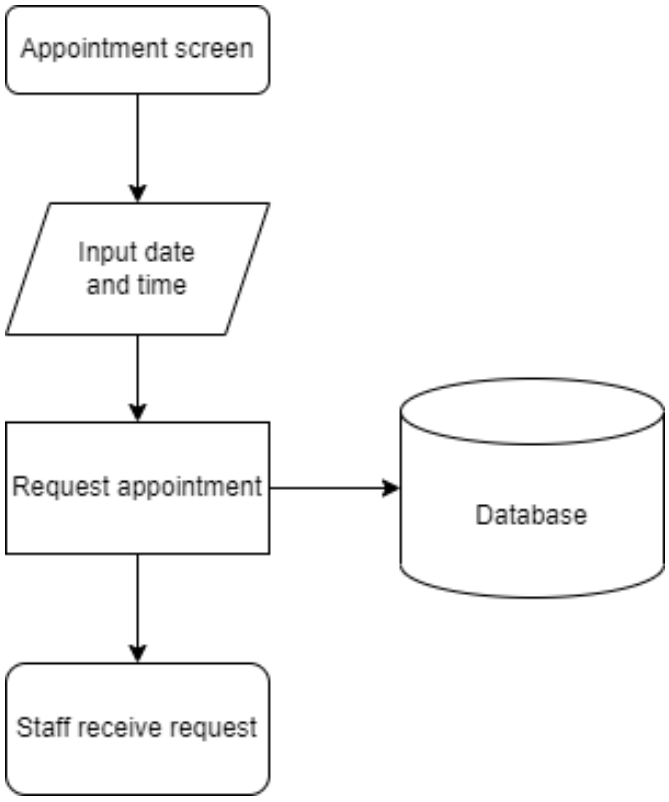


FIGURE: Flowchart demonstrating how a patient may request an appointment.

On the doctors' side, they receive the request and have the chance to respond to the request by either confirming the details or making edits to the date/time. This is subject to validation checks, as shown in the flowchart and pseudocode below.

```
SUB
check_clash(doctorID,room,appointment_date,appointment_time)
query1 = SELECT apptime FROM tblAppointment
WHERE (appdate = appointment_date) AND ((doctor_id = doctorID)
OR (approom = room))
    IF query1 != NULL THEN
        FOR record in query1
            IF |appointment_time - apptime| < 15 minutes THEN
                RETURN FALSE
    RETURN TRUE
```

TEXT: Pseudocode checking for any appointment clashes involving a doctor or room at a given date and time, within 15 minutes of the proposed appointment.

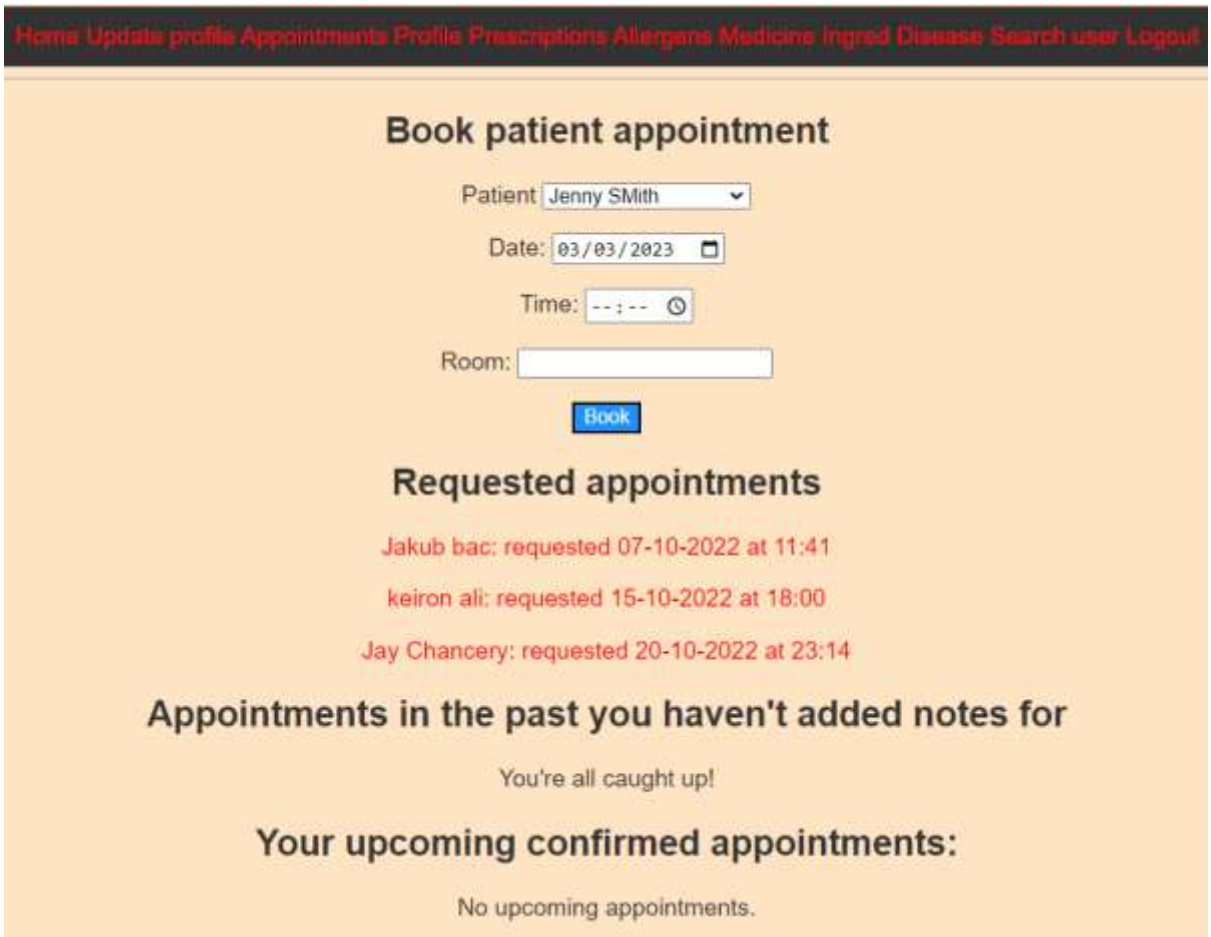


FIGURE: Screenshot of the appointment booking page for a doctor. It displays a creation form, requested appointments, past appointments with action required and upcoming appointments.

The appointment screen for doctors also uses a queue class I built to display an ordered queue for requests, upcoming appointments, and post-appointment feedback.

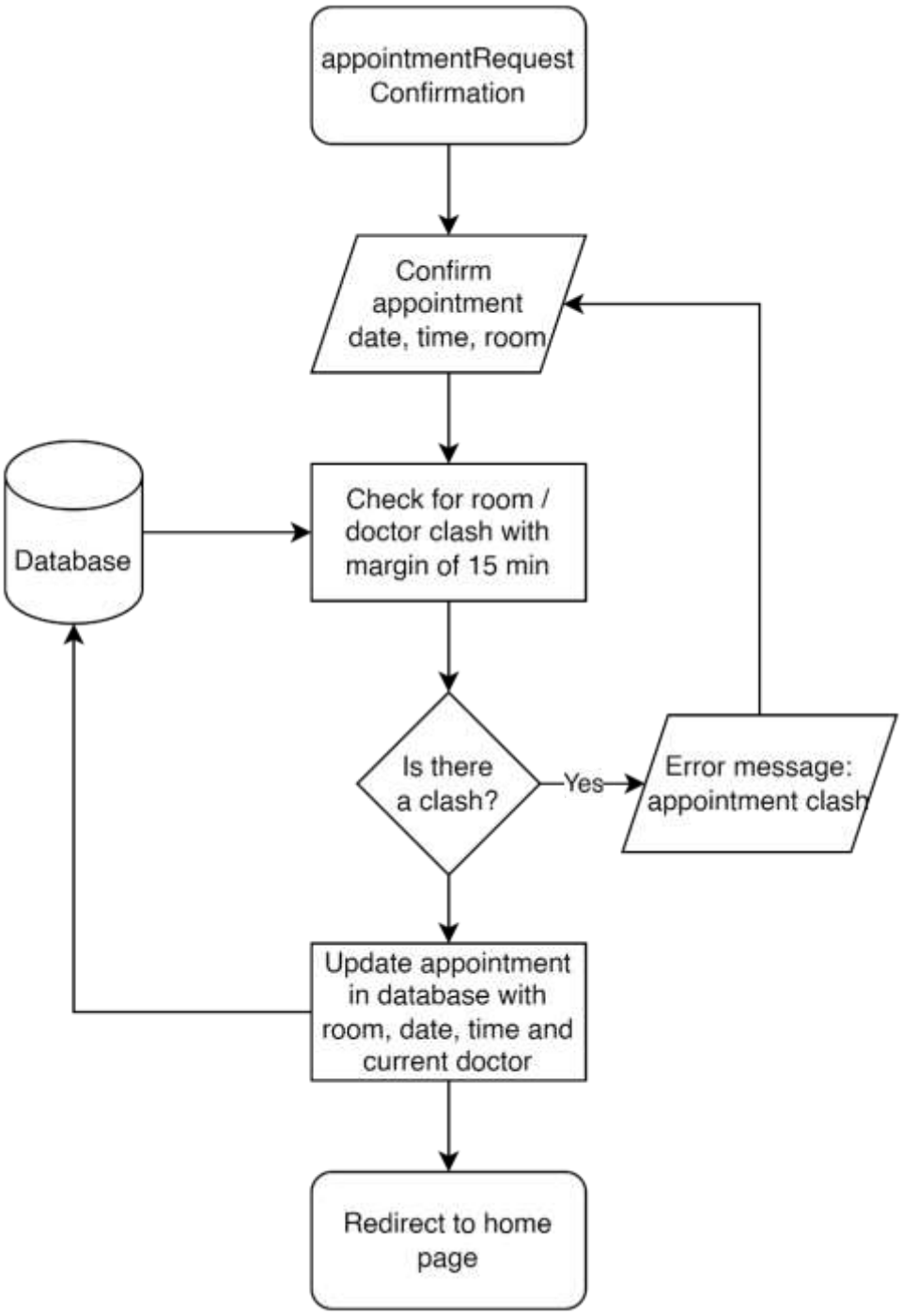


FIGURE: Flowchart demonstrating how a doctor would respond to a patient's request.

Assuming that the patient does not cancel the appointment (which may be done at the click of a button), from the day of the appointment the doctor may complete a post-check-up form. This would include answering whether the patient fulfilled the appointment, practice notes, and feedback viewable by the patient.

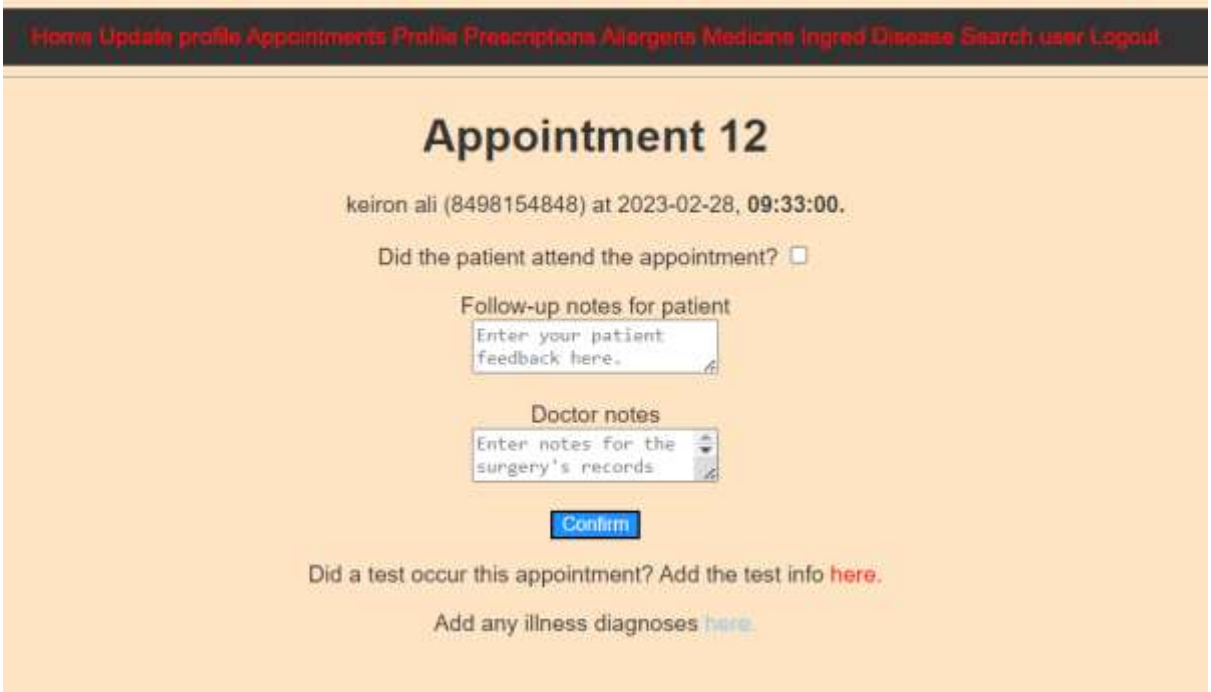


FIGURE: Screenshot illustrating how a doctor would add further notes to an appointment, with links to add test and illness diagnosis information here.

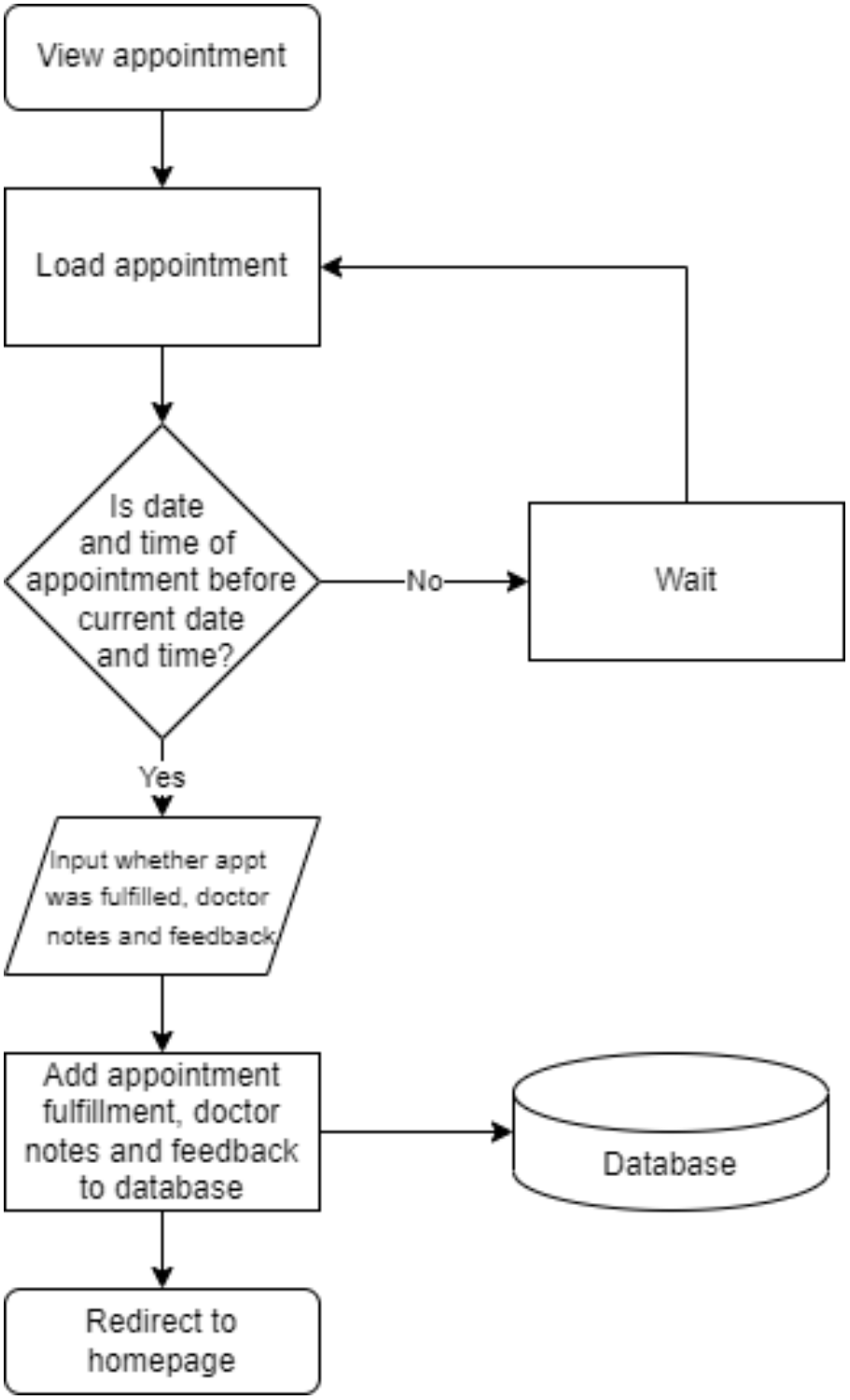


FIGURE: Flowchart demonstrating the post-check-up form.

Test results

Test results may also be added as part of the post-check-up process, in instances where tests are performed alongside an appointment.

FIGURE: Screenshot of the test addition section.

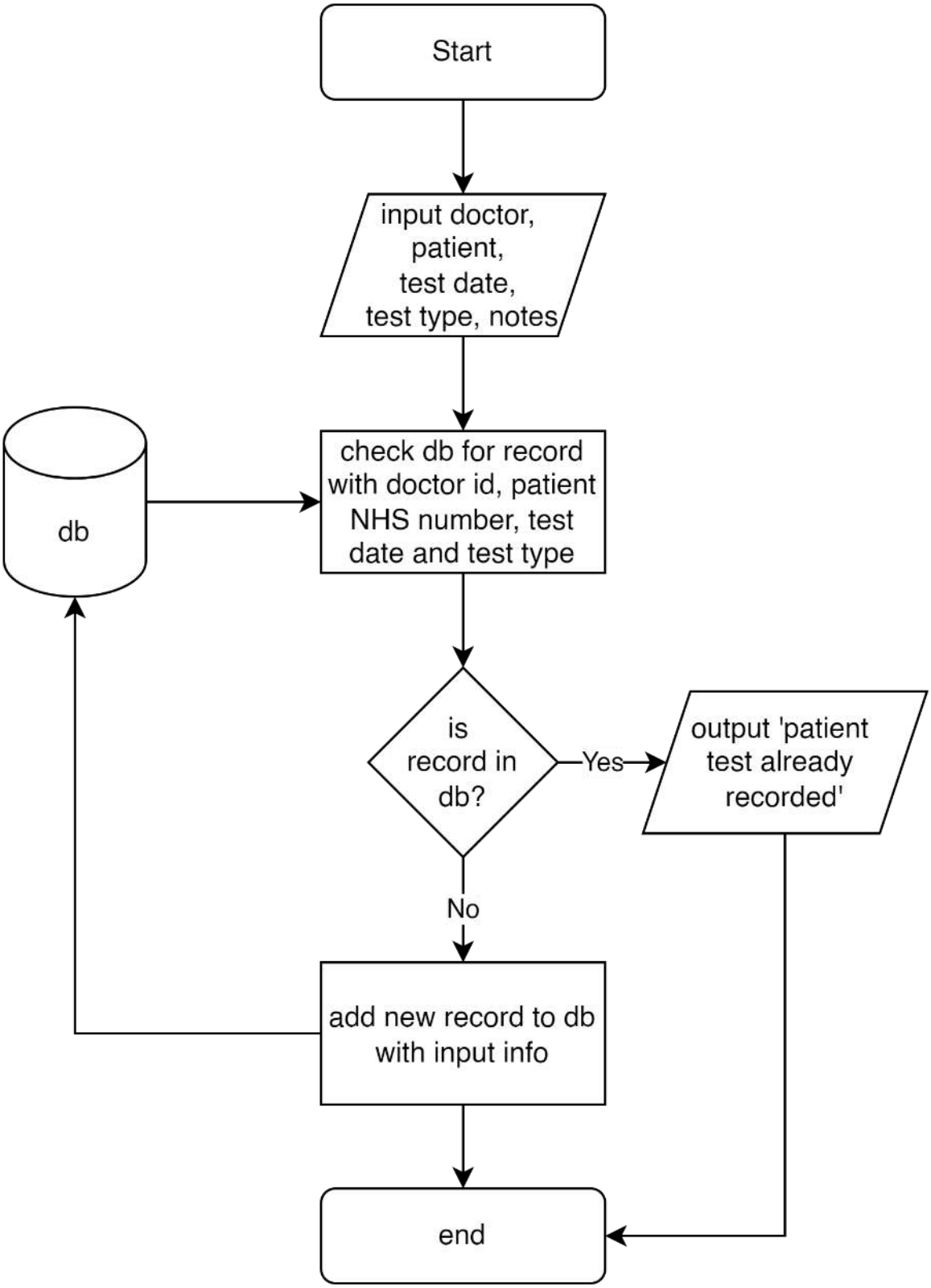


FIGURE: Flowchart describing the process of adding test information about a patient to records.

Prescriptions

After an appointment, a doctor may decide to prescribe a patient some medicine based on their own preliminary diagnosis of the sickness. First though, the medicine needs to be present in the database. Only doctors of the highest access level (1) can create or modify a medicine, to prevent spam or errors.

FIGURE: Screenshot of the medicine form, which redirects to the allergy addition after successful addition.

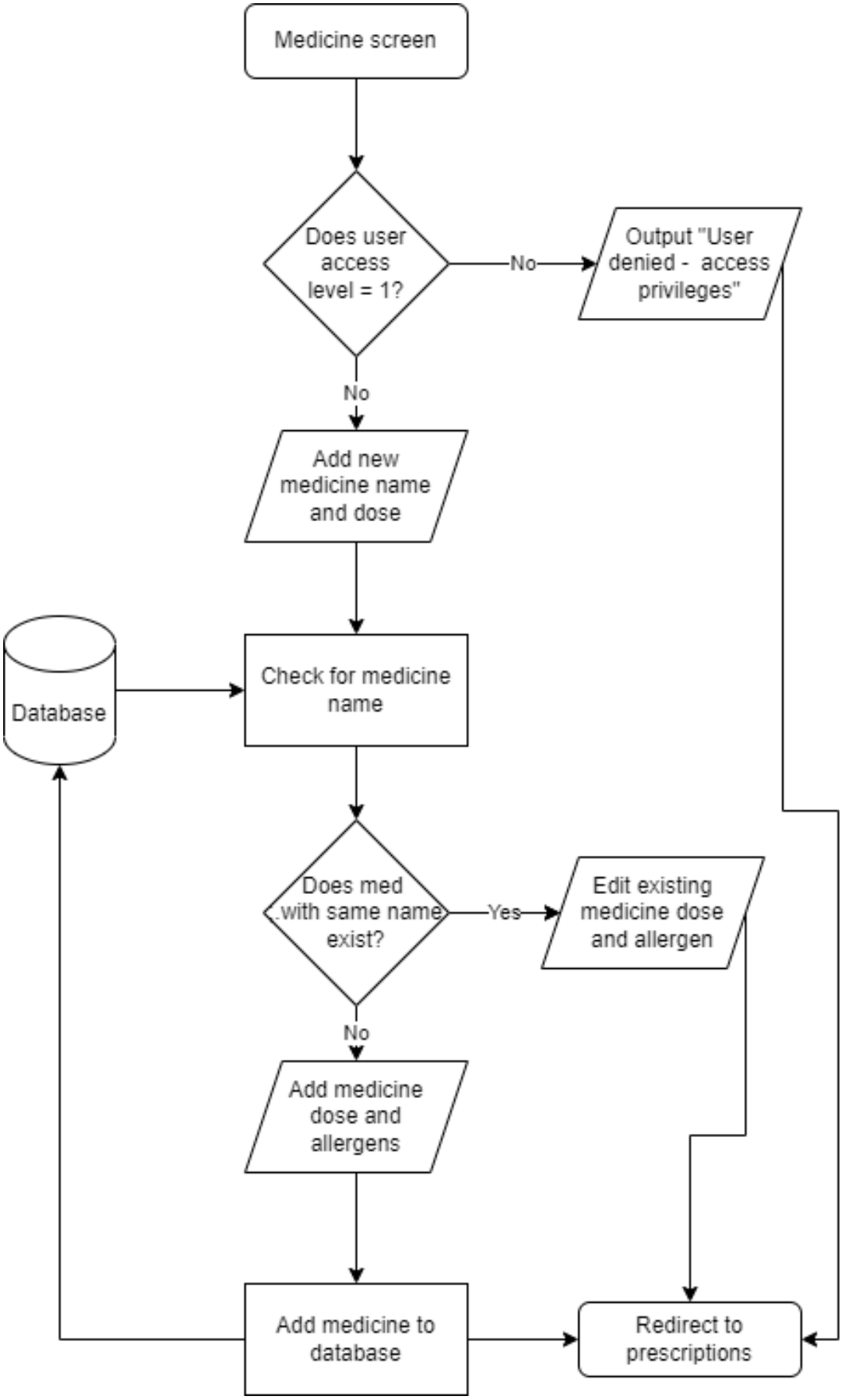


FIGURE: Flowchart showing how a level 1 doctor may add a new medicine to the system.

When viewing a medicine's information, a comprehensive profile is generated, including the patients currently prescribed it and its allergens. To find the allergens, the following complex query is required:

```
SELECT Allergen.allergenName FROM Allergen, Ingredient, Medicine WHERE Allergen.allergen_id=Ingredient.allergen_id AND Ingredient.med_id=Medicine.med_id AND Medicine.medicineName="{name}"
```

TEXT: SQL statement for selecting the names of allergens that have an id in the same row as the id of the medicine with the name {name} ingredients table.

If an allergy is attributed to a patient after they have been prescribed a medication with this allergen, the prescription is cancelled – the end date is set as today.

```
SELECT prescription.prescription_id FROM Prescription inner JOIN Ingredient ON (Prescription.med_id=Ingredient.med_id) where (ingredient.allergen_id={allergen_id} and prescription.NHSNumber={NHSNum})
```

TEXT: SQL statement that selects the prescription IDs for prescriptions attributed to the patient that contain a medicine that has the same allergen as one they are allergic to.

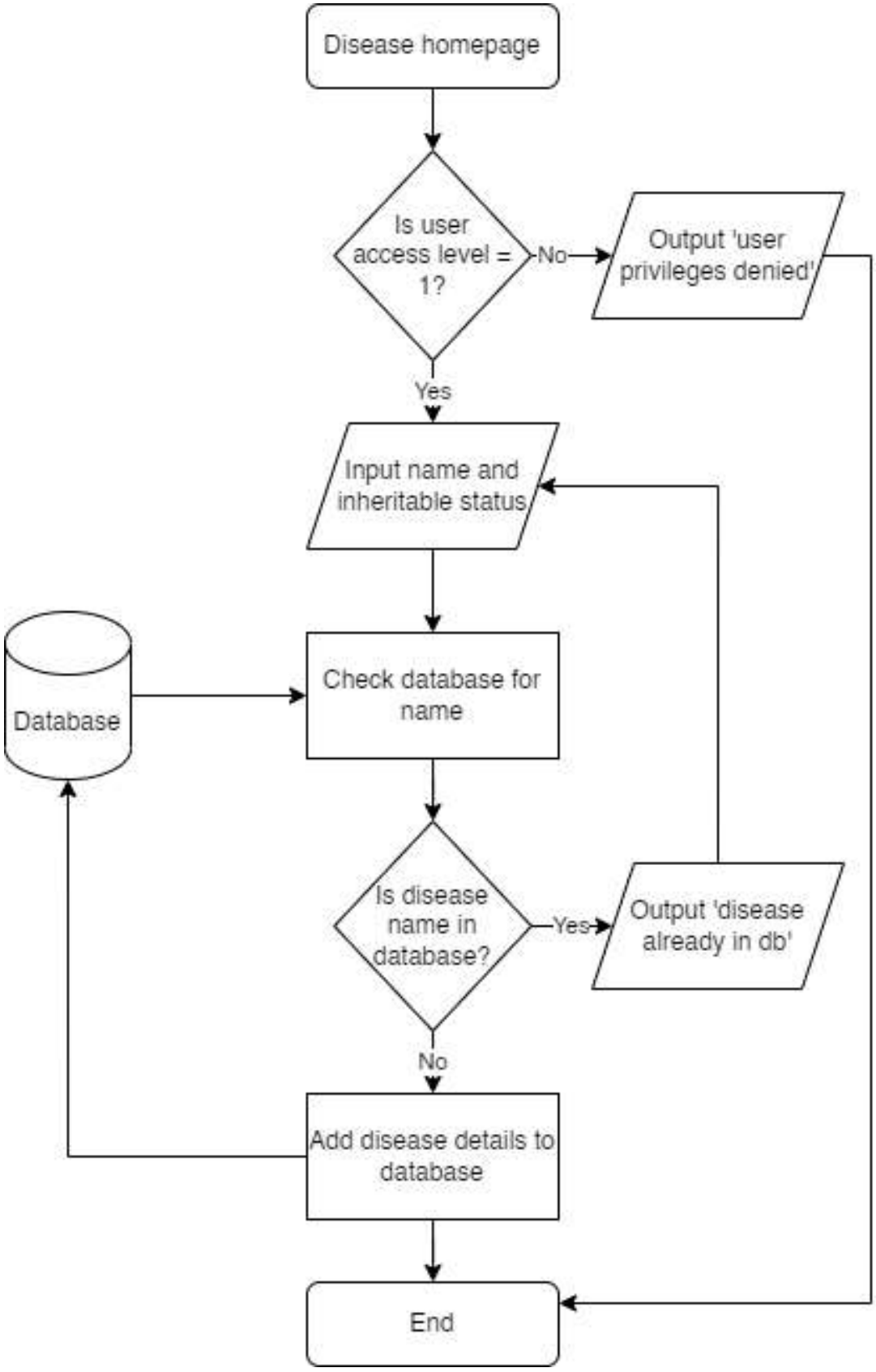


FIGURE: Flowchart demonstrating the addition of a disease to the database.

The prescription must also be linked to the disease the doctor has diagnosed the patient with. This means they may need to add a disease record to the table. Once again, this may only be done by a doctor of access level 1, to prevent misinformation being recorded.

Using the added medicine and disease, a doctor may now create a prescription for the patient. The system will not allow the doctor to prescribe a medicine the patient has an allergy to. It will also create a new prescription every time a doctor modifies the current one.

FIGURE: Screenshot of the prescription creation page. The medicine and disease entry boxes use an autocomplete, sourced from previous additions to the database.

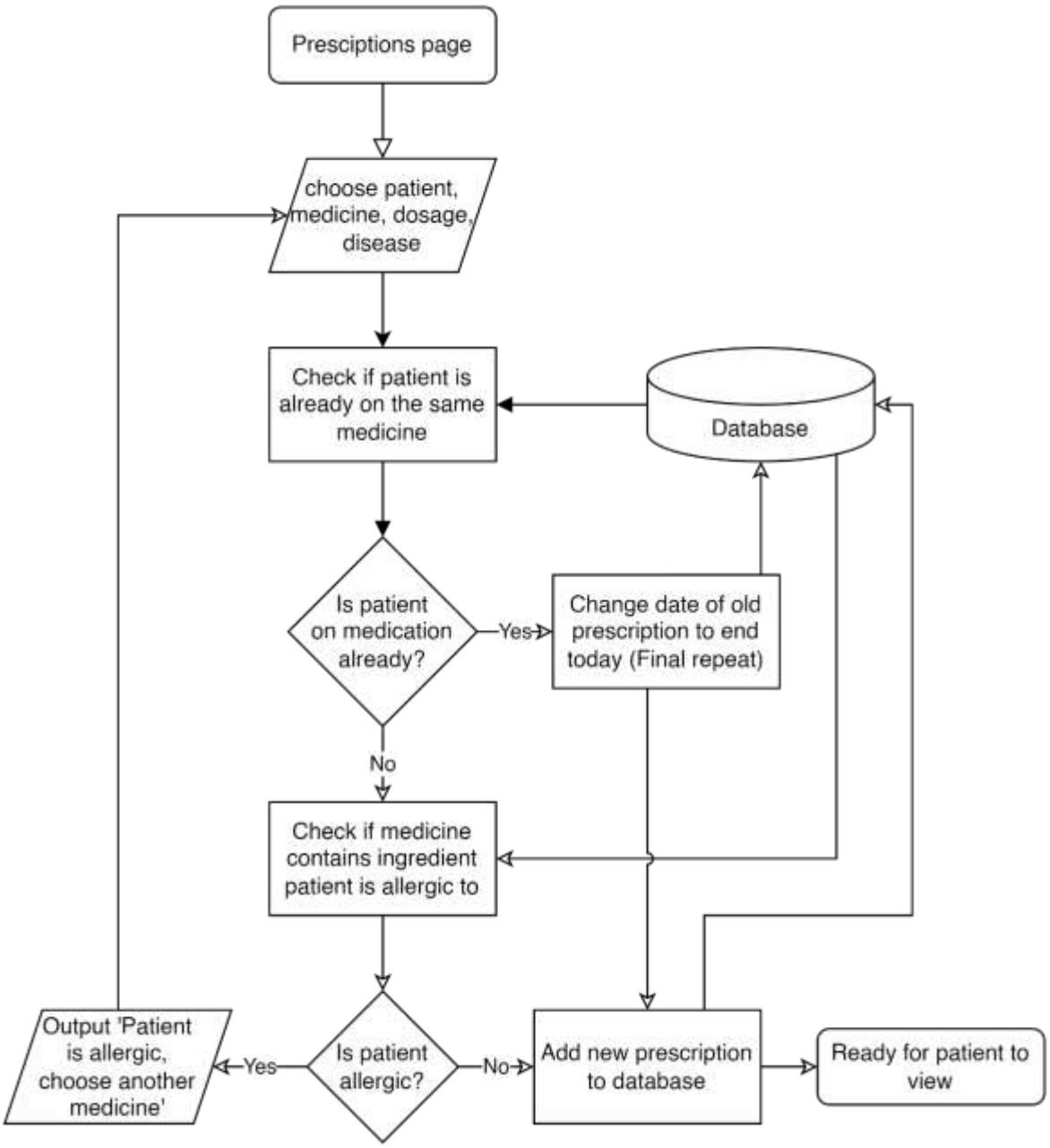


FIGURE: Flowchart demonstrating how a patient prescription is produced by a doctor.

Allergies

To create an allergen, a doctor with access level 1 or 2 may add its name to the allergens table.

FIGURE: Screenshot showing the allergen creation page.

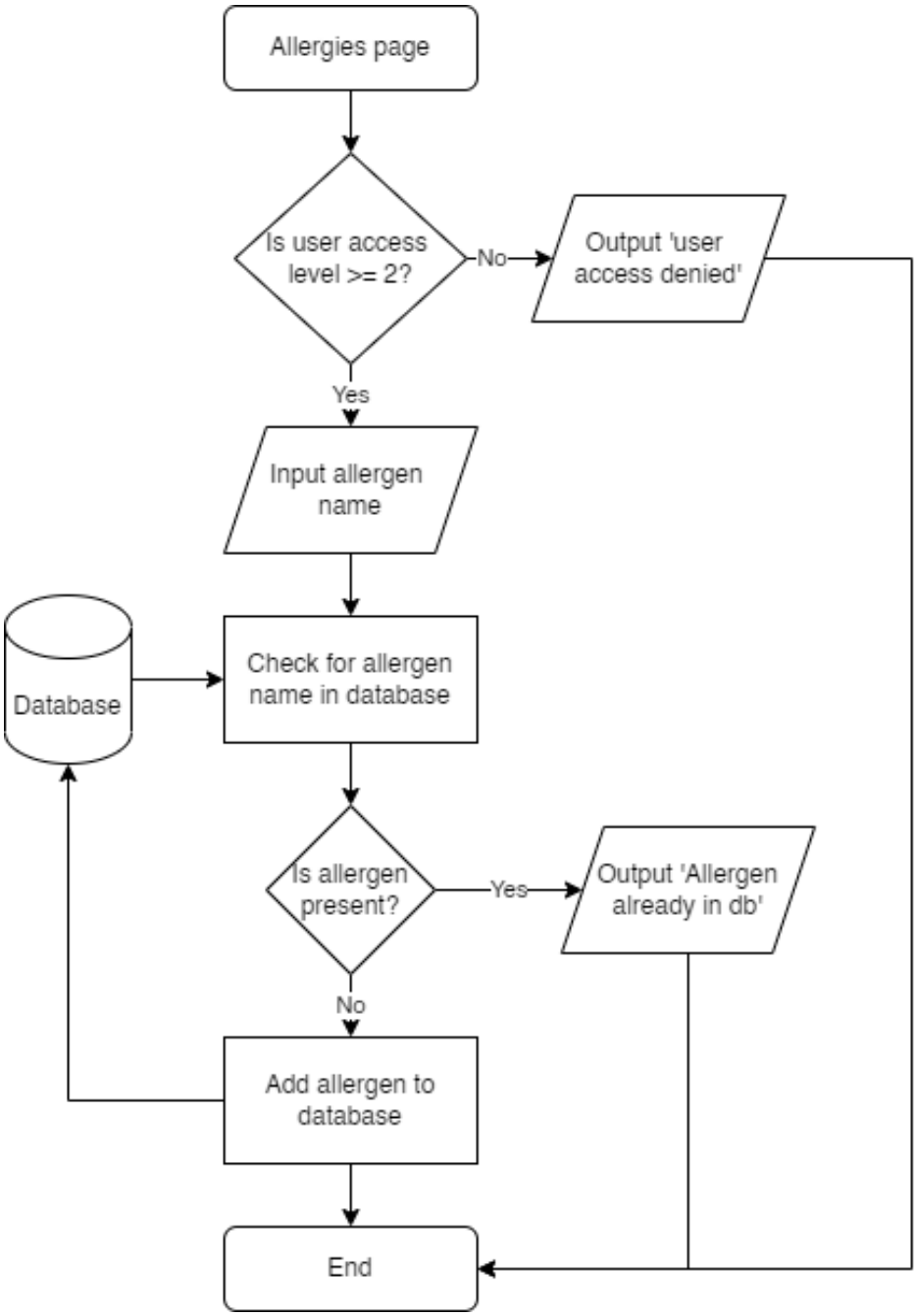


FIGURE: Flowchart showing the process of creating an allergen.

The doctor may then link the allergen to patients and medicines (on the ingredients screen). To add a patient's allergy, the search function is used to take you to a specific profile, and then the allergy addition.

FIGURE: Screenshot showing the patient profile search tool.

FIGURE: Screenshot of the allergy addition screen, which one would access via visiting the patient's profile directly. The allergy entry utilises an autocomplete function based on added allergens.

FIGURE: Screenshot of the page linking medicines to ingredients that may cause allergic reaction. Both entry fields use autocomplete.

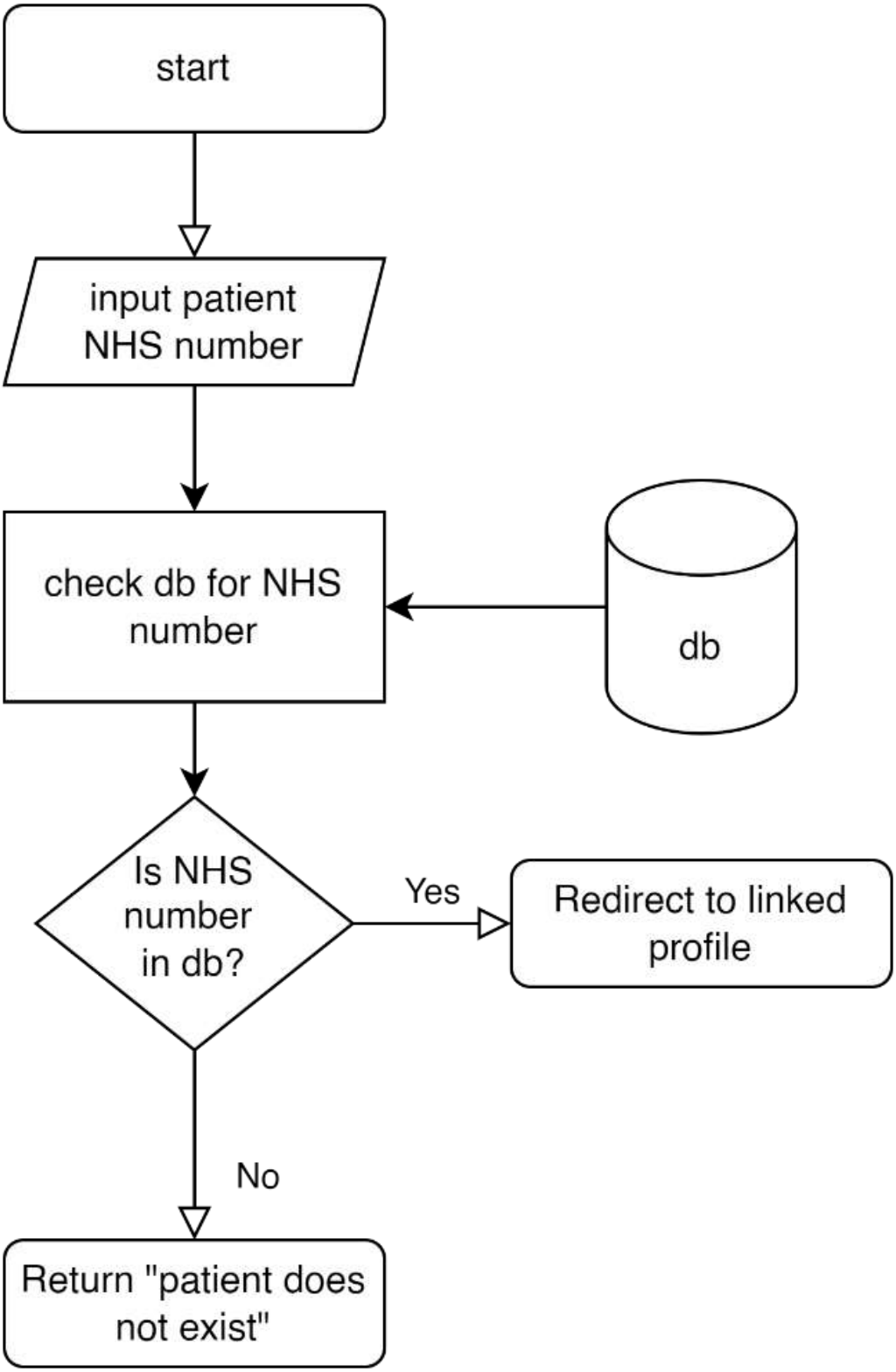


FIGURE: Flowchart demonstrating how the program searches for a patient.

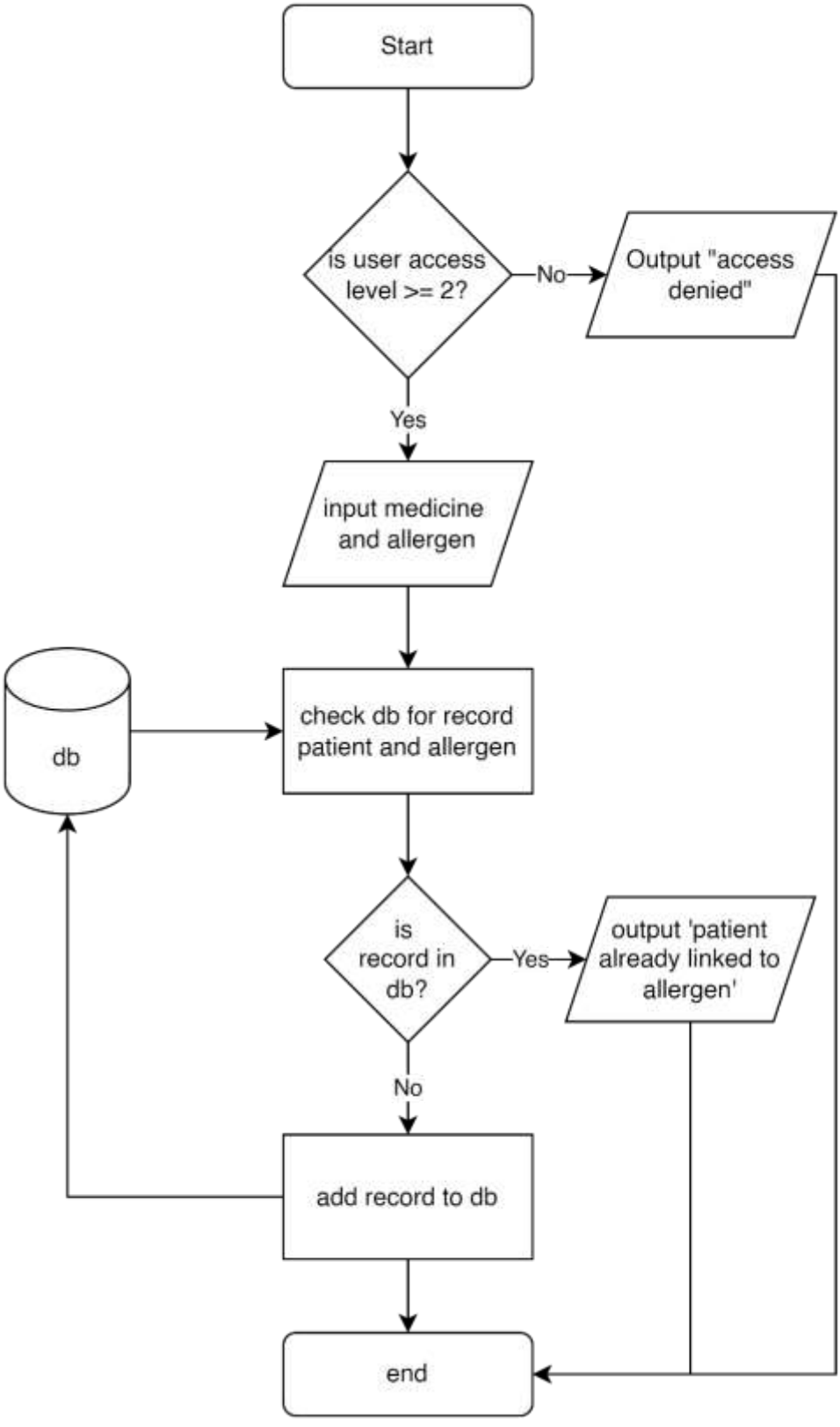


FIGURE: Flowchart demonstrating the linking between a patient and a known allergen.

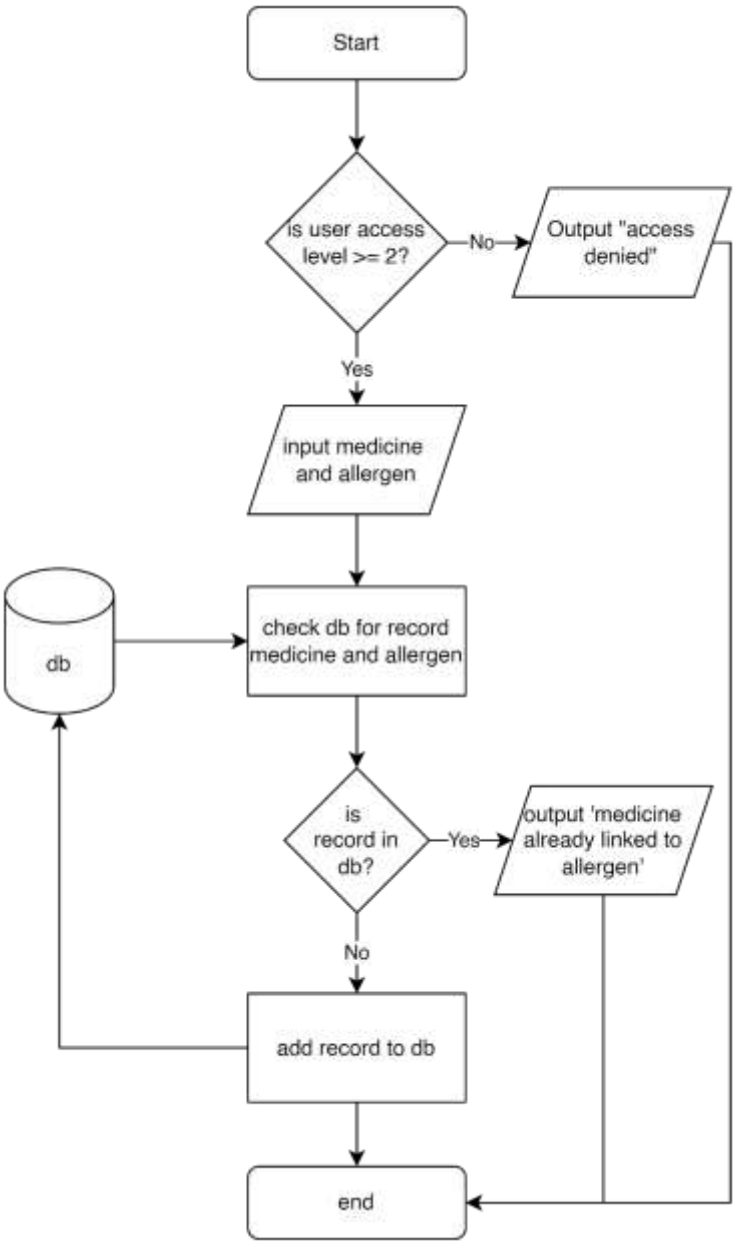


FIGURE: Flowchart demonstrating the linking between a medicine and a potential allergen ingredient.

Illness history

After a patient profile is created, the doctor may link an inherited disease to a patient who carries it. The doctor first inputs the patient and the disease (which can only be inheritable), each from a drop-down list.

FIGURE: Screenshot of the inheritable disease carrier page.

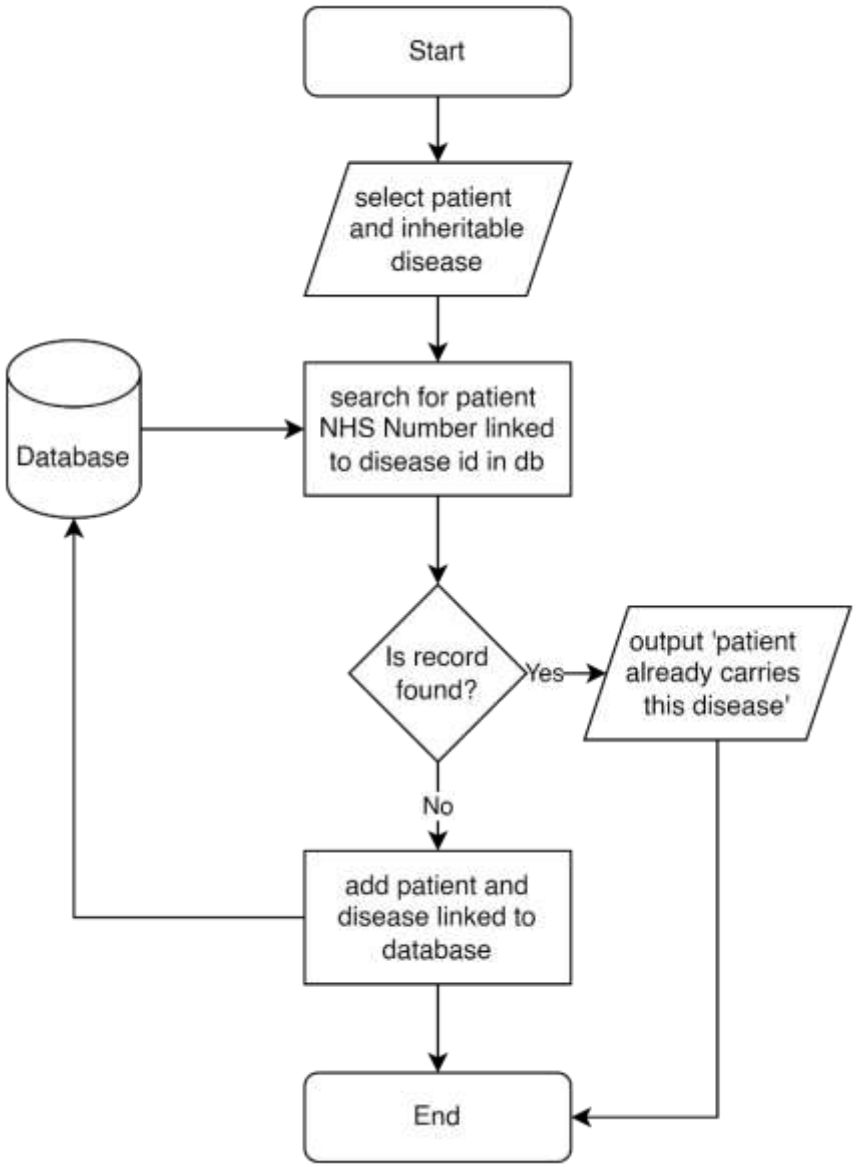


FIGURE: Flowchart demonstrating the process of linking an inherited disease to a patient profile.

They can also add an illness history relating to the patient to the database to aid in giving the best contextual care to each patient. As per usual, it is checked for duplicates before being added.

```
newIllness = INPUT patient, disease, diagnosisDate, notes

query1 = SELECT unhealthy_id FROM tblIllness WHERE
NHSNumber=newIllness.patient.NHSNumber AND
disease_id=newIllness.disease.disease_id AND
diagnosisDate=newIllness.diagnosisDate

IF query1 NULL THEN

     query2 = INSERT INTO tblIllness (NHSNumber, disease_id,
diagnosisDate, notes) VALUES (newIllness.patient.NHSNumber,
newIllness.diagnosisDate, newIllness.notes)

     db.EXECUTE(query2)

     db.COMMIT

ELSE THEN

     OUTPUT 'Illness has already been created'
```

TEXT: pseudocode of the process of creating an illness history.

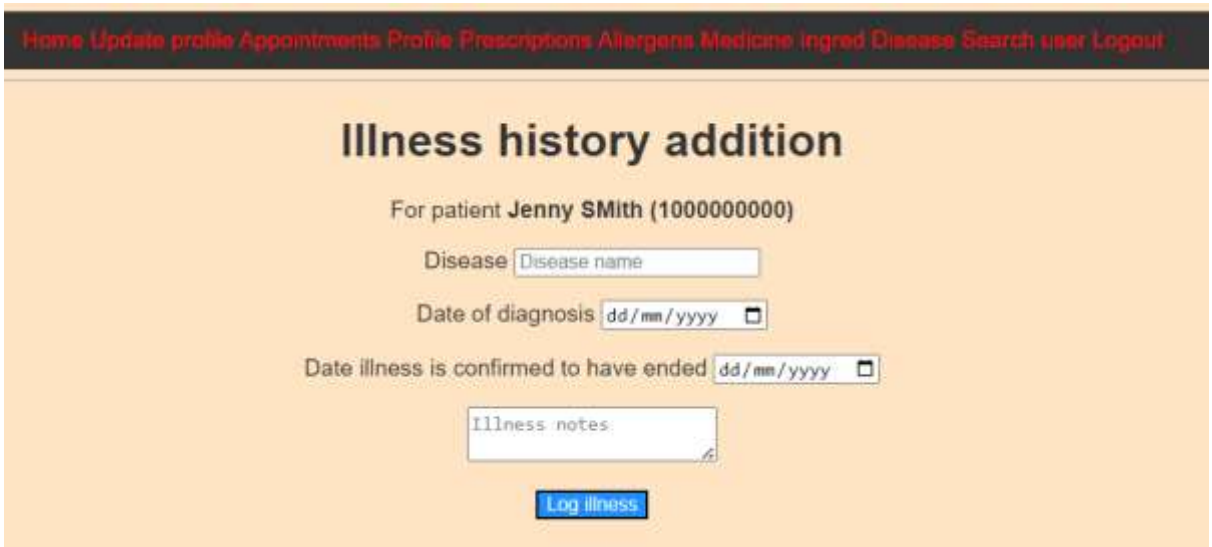


FIGURE: Screenshot of the illness history addition page, which uses autocomplete for the disease entry.

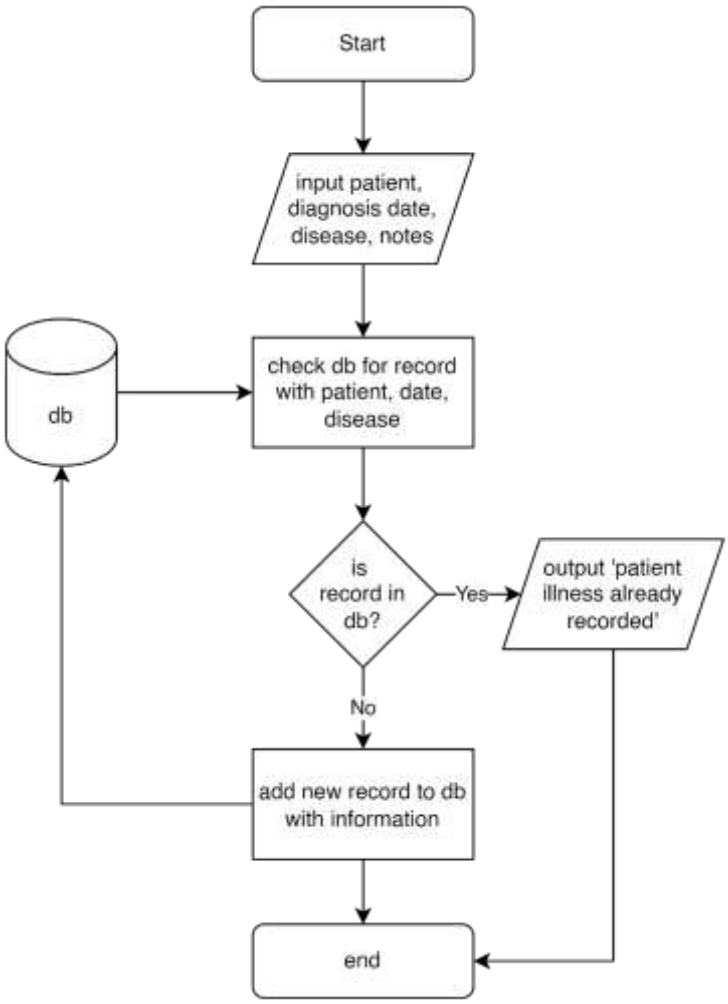


FIGURE: Flowchart demonstrating process of creating an illness history.

Patient referrals

Patient referrals such as hospitalisations, physiotherapy visits or accident & emergency visits) are also recorded in the ‘referral’ table, once again for the purpose of building a complete medical profile for every patient.

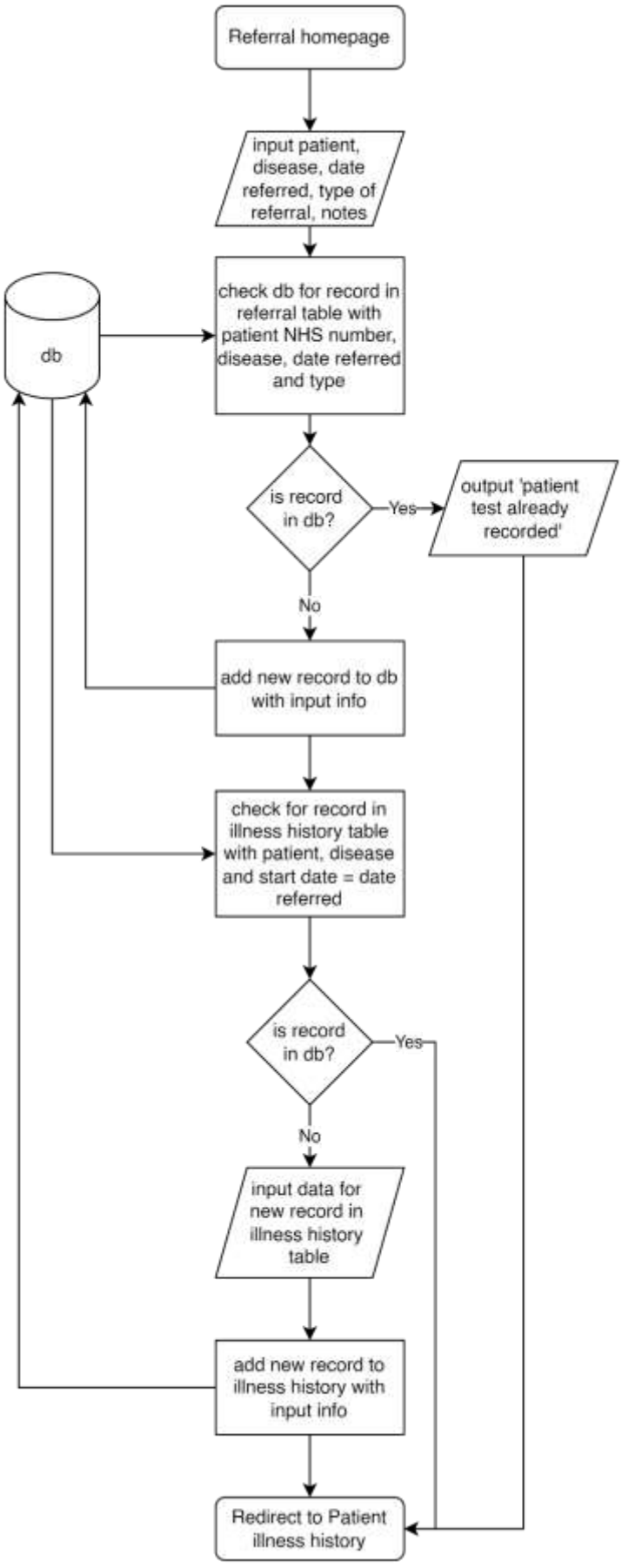


FIGURE: Flowchart demonstrating addition of referrals.

Hours spent

As detailed in the analysis, real world doctors tend to spend much more time at work than the average UK employee. The hours-spent table aims to record every instance of time spent at work by a doctor, to ensure that doctors are not spending more than the recommended amount every day.

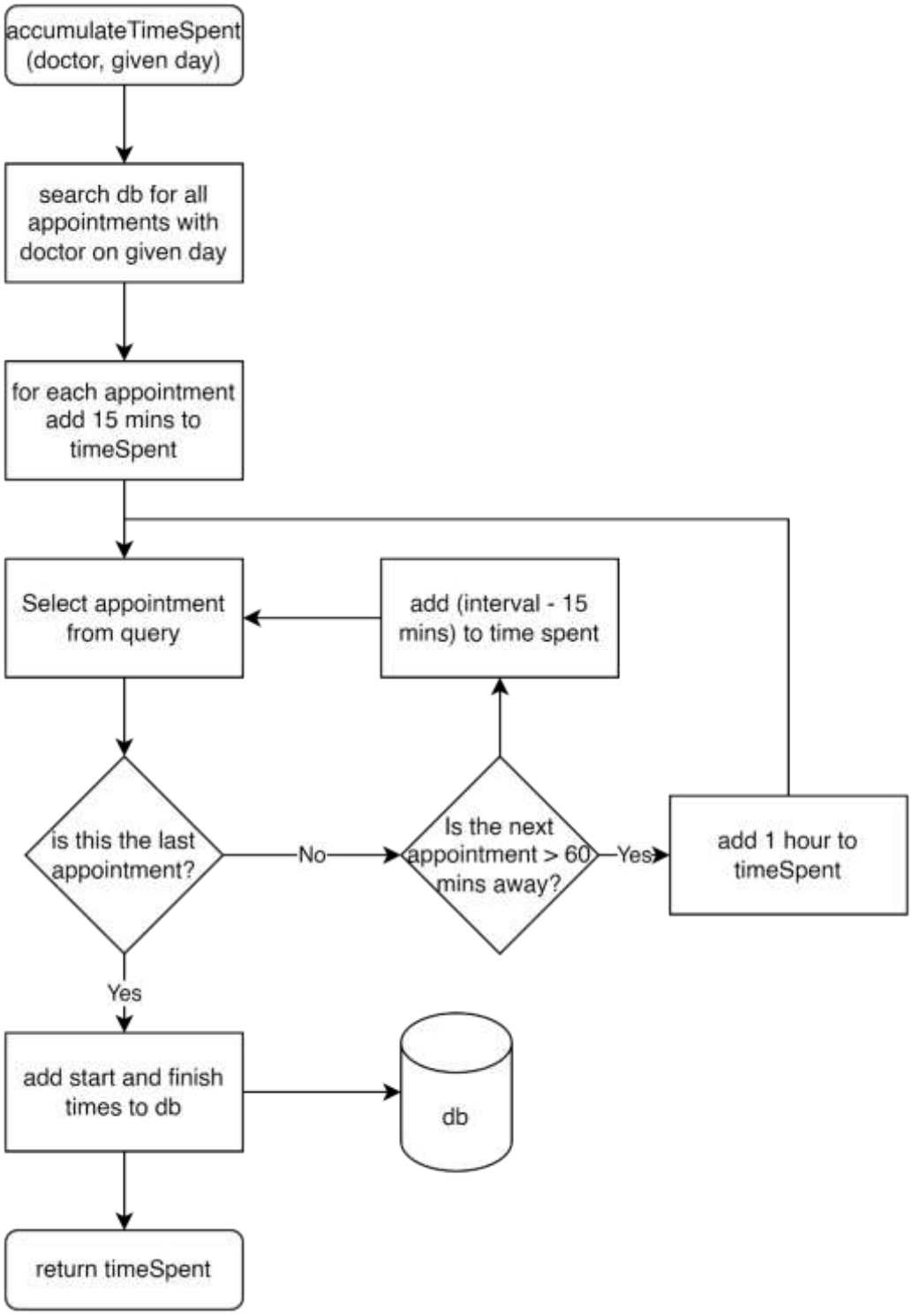


FIGURE: Flowchart demonstrating the process of calculating the time spent on a given day.

Calculation of the time spent at the surgery allows the system to deduce whether the doctor is working over their recommended weekly hours (i.e. the previous seven days), or their daily hour limit. After booking an appointment, this check lets the doctor know if they are overworking.

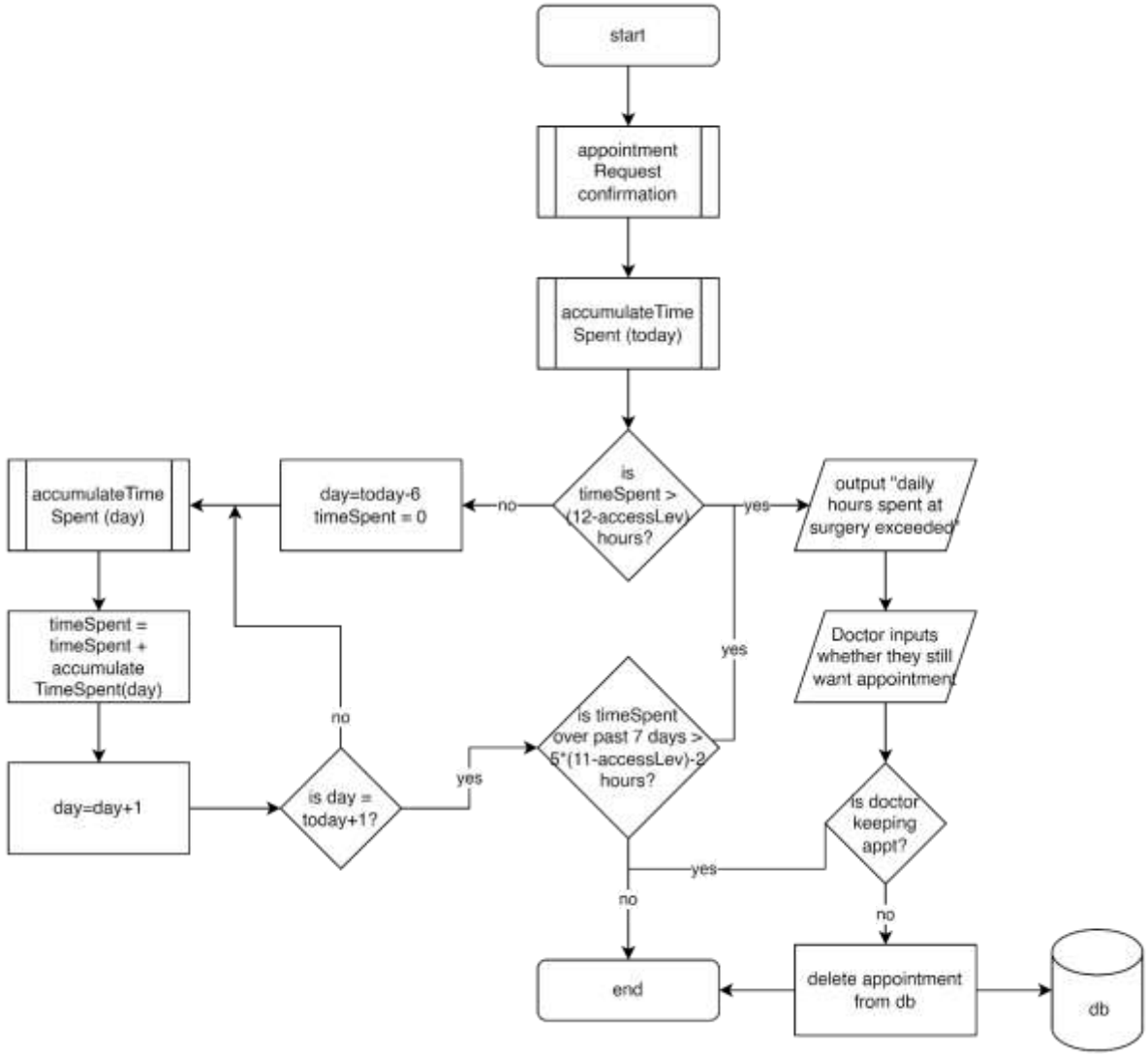


FIGURE: Flowchart demonstrating how an appointment booking must inform the doctor if it will lead to them working overtime.

Profile

Patient profiles will display all characteristics linked to the patient, including email, date of birth, birth sex, height, weight, blood type, address, prescription, allergies, illness history and inherited diseases. Doctor profiles will display email, date of birth, gender, address, and access level.

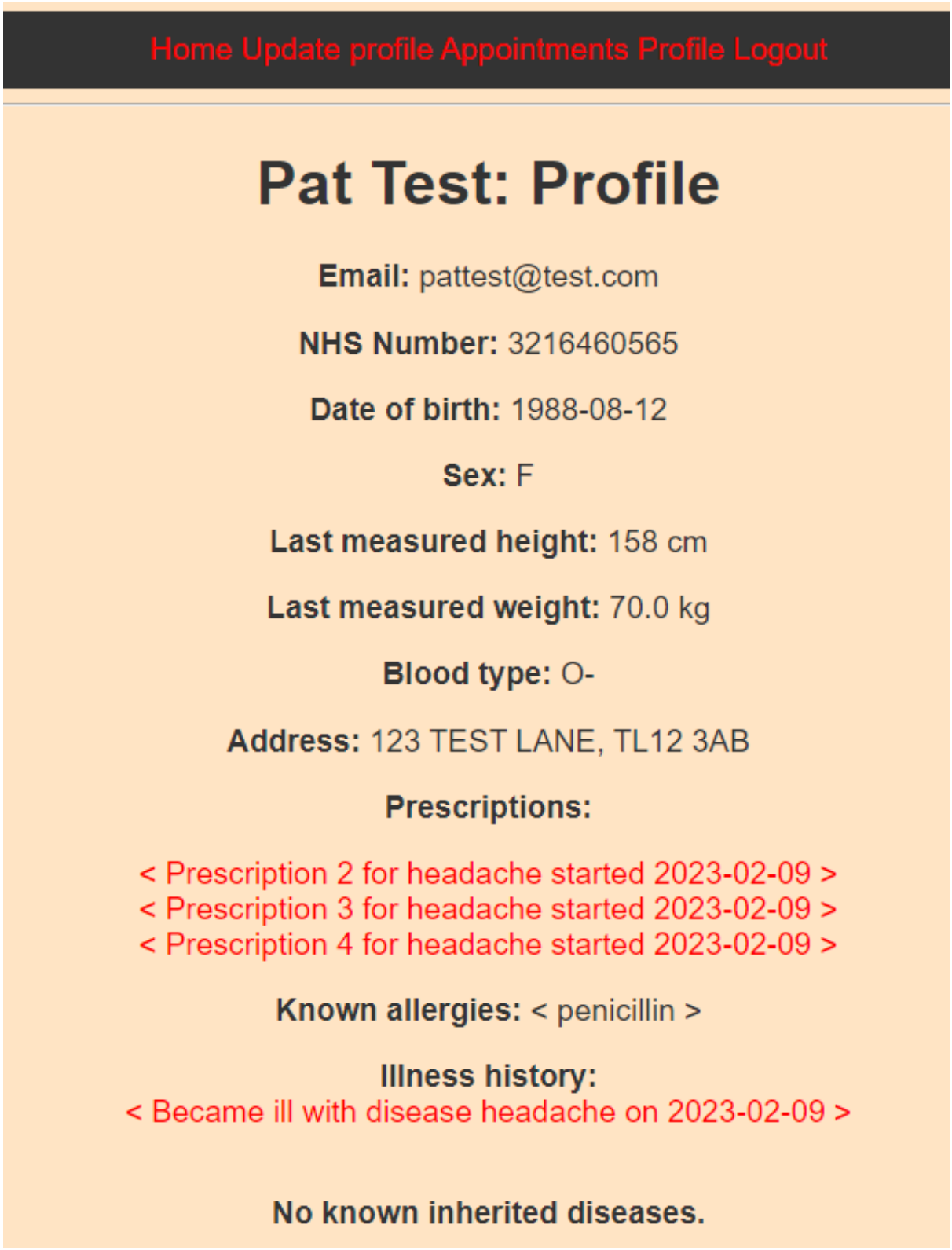


FIGURE: What a patient would view when visiting their own profile.

FIGURE: What a doctor would see when visiting the same patient's profile: note the inclusion of links to adding illness history, appointments, allergies and inherited disease.

FIGURE: Screenshot of what a doctor would see on their own profile – email, part of their NI number, access level, gender and their address.

TECHNICAL SOLUTION

All code is up to date as of Wednesday 9th March 2023.

\config.py

```
import os

basedir = os.path.abspath(os.path.dirname(__file__))


#allows the object to communicate over webpage and link to db

class Config(object):#these are all the parameters passed through the venv: "set CONSTANT=whatever"
    SECRET_KEY = os.environ.get('SECRET_KEY') or "unguessable_secret_key"
    SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL') or \
        'sqlite:///' + os.path.join(basedir,'app.db')
    SQLALCHEMY_TRACK_MODIFICATIONS = False
    MAIL_SERVER = 'smtp.googlemail.com'
    MAIL_PORT = int(587)
    MAIL_USE_TLS = True
    MAIL_USE_SSL = False
    MAIL_USERNAME = 'noreply.kwanea1@Gmail.com'
    MAIL_PASSWORD = 'wnxkdiejrtpkhlot'
    ADMINS = ['noreply.kwanea1@gmail.com']
```

SECRET_KEY is used to provide a ‘signature’ for each flaskform interaction

These settings are needed for communication with the email server

\nea.py

```
from app import app, db

from app.models import Doctor, HoursSpent, Address, Allergen, Allergy, Appointment, Patient, Prescription, Disease, Medicine

from datetime import datetime, date, time

from app.numberGen import nhsgen, insurancegen, findAge, heightGen, bloodGen
```

This imports the Flask app in the constructor from inside the ‘app’ folder, and the database

Second line onwards is required for python shell testing

\app__init__.py

```
#constructor for app
from flask import Flask
from config import Config
from flask_sqlalchemy import SQLAlchemy
from flask_migrate import Migrate
from flask_login import LoginManager
from flask_mail import Mail
import logging
import os
from logging.handlers import SMTPHandler, RotatingFileHandler
```

The constructor imports all these classes for the running of the application in the virtual environment, for the server and the user can access it

```
#initiating flask app
app = Flask(__name__)
#config.py for configurations
app.config.from_object(Config)
#SQLAlchemy for db managing of the app
db = SQLAlchemy(app)
#Flask-Migrate for database migration management
migrate = Migrate(app,db,render_as_batch=True)
#flask-login for login forms
login = LoginManager(app)
login.login_view = 'loginDoc'
#flask-mail for email handling
mail = Mail(app)
```

Line by line:

Flask app instance created

Config object from config.py imported for app settings

SQLAlchemy database instance created using Flask app for queries

Migration instance with the Flask app and SQLAlchemy class created for database structure modifications

Login manager class created using the Flask app for the login function

Mail class instance using Flask app created for the sending of emails

```
#error logging via file and email
if not app.debug:
    if not os.path.exists('logs'):
        os.mkdir('logs')
    file_handler = RotatingFileHandler('logs/nea2.log', maxBytes=10240,
```

```
                                backupCount=10)
    file_handler.setFormatter(logging.Formatter(
        '%(asctime)s %(levelname)s: %(message)s [in %(pathname)s:%(lineno)d]'))
    file_handler.setLevel(logging.INFO)
    app.logger.addHandler(file_handler)

    app.logger.setLevel(logging.INFO)
    app.logger.info('NEA2 startup')

    if app.config['MAIL_SERVER']:
        auth = None
        if app.config['MAIL_USERNAME'] or app.config['MAIL_PASSWORD']:
            auth = (app.config['MAIL_USERNAME'], app.config['MAIL_PASSWORD'])
        secure = None
        if app.config['MAIL_USE_TLS']:
            secure = ()
        mail_handler = SMTPHandler(
            mailhost=(app.config['MAIL_SERVER'], app.config['MAIL_PORT']),
            fromaddr='no-reply@' + app.config['MAIL_SERVER'],
            toaddrs=app.config['ADMINS'], subject='NEA Failure',
            credentials=auth, secure=secure)
        mail_handler.setLevel(logging.ERROR)
        app.logger.addHandler(mail_handler)

from app import routes, models
```

When app is not in debug mode, any runtime errors or timeouts are recorded in the file logs\nea2.log, and sent to the admin emails (stored in the Config object).

\app\appoint_val.py

```
from app import app, db

from app.models import Appointment, HoursSpent

from datetime import datetime, date, time, timedelta

from app.sql_connect import DB_Execute

from app.query_count import queryCount


def accTimeSpent(doctor_id, day):
    #calculates how much time was spent by a doctor on one day


q1=Appointment.query.filter_by(appdate=day,doctor_id=doctor_id).order_by(Appointment.apptime.asc()) #day in format date(yyyy,mm,dd)
    timeSpent = timedelta(minutes=0)
    length=queryCount(q1)
    print(length, "appointments on", day,"\n")
    if length > 0:
        start=q1[0].apptime
        end=q1[length-1].apptime
        print("First appointment at",start,"last appointment at",end)
        for x in range (length):
            #for each appointment 15 minutes is added
            timeSpent += timedelta(minutes=15)
            appt_time = datetime.combine(q1[x].appdate,q1[x].apptime)
            try:
                nextappt_time = datetime.combine(q1[x+1].appdate,q1[x+1].apptime)
                #is gap between appointments longer than an hour?
                if (nextappt_time-(appt_time+ timedelta(minutes=15))) > (timedelta(hours=1)):
                    timeSpent += timedelta(hours=1)
                else:
                    timeSpent+=((nextappt_time-appt_time)-timedelta(minutes=15))
            except:
```

For Objective 11

```
            pass
        #adds to db
        if HoursSpent.query.filter_by(doctor_id=doctor_id,date=day,leaveTime=end).first() is None:
            x=HoursSpent(doctor_id=doctor_id,date=day,entryTime=start,leaveTime=end)
            db.session.add(x)
            db.session.commit()
            print('added today\'s record to db')
    print(f"time spent on {day}:",timeSpent,"\n")
    return timeSpent


def checkOvertime(doctor_id, accessLev, appt_date=date.today()):#returns false if overtime limit exceeded
    todayTimeSpent = accTimeSpent(doctor_id,appt_date)
    print("Max time allowed on one day for level",accessLev,"doctor:", timedelta(hours=(12-accessLev)),"\n")
    if todayTimeSpent > timedelta(hours=(12-accessLev)): #higher access level / status means more time is usually spent at the surgery
        print("Recommended daily hours spent at surgery exceeded.")
        return True
        #user doctor proceeds to input whether they want to continue booking appointment
    day=appt_date-timedelta(days=6)
    totalTimeSpent = todayTimeSpent
    while day != appt_date:
        #accumulating time spent over the past week including today
        totalTimeSpent += accTimeSpent(doctor_id, day)
        day+=timedelta(days=1)
    print("\ntotal time spent over past 7 days is",totalTimeSpent)
    print("total time allowed",timedelta(hours=(((11-accessLev)*5)-2)))
    if totalTimeSpent > timedelta(hours=(((11-accessLev)*5)-2)):
        print("Recommended weekly hours spent at surgery exceeded.")
        return True
```

Sum of previous 7 days appointments

```
    return False


#checks for same day clashes - 15 minute margin
def checkSlotFree(doctor_id: int, day: date, apptime: time, room):
    before=datetime.combine(day,apptime)-timedelta(minutes=15)
    print(before)
    after=datetime.combine(day,apptime)+timedelta(minutes=15)
    print(after)
    q=DB_Execute('app.db')
    #searches for appointments same day with either
    query1=q.select(f'SELECT apptime FROM Appointment WHERE appdate="{day}" AND (doctor_id="{doctor_id}" OR room="{room}")')
    print(query1)
    if query1 is not None:
        for record in query1:
            #converting each query result to a python-readable time format
            time_obj= datetime.strptime(query1[0][0][:5], '%H:%M').time()
            print("time obj", time_obj)
            record_time=datetime.combine(day,time_obj)
            if (record_time > before) and (record_time < after):
                return False, f"You and/or room {room} are not free at {apptime} on {day}."
    return True, f"Room {room} and doctor are free at {apptime} on {day}."
```

Checks clashes for objective 4

\app\createdClass.py

```
class myQueue:
    def __init__(self,size):#sets up variables required for circular queue structure
        self.size = size
        self.queue = [None]*(self.size)

        #position of first item in queue, position of first item after end of queue
        self.start = -1
        self.end = 0
    def deQueue(self):
        x = self.isEmpty()
        if not x:
            #removes and returns the value at queue[start]
            self.val = self.queue.pop(self.start)
            #changes front of queue to next item waiting
            self.start = (self.start+1) % (self.size)
            #maintains size of the queue
            self.queue.insert((self.start-1),None)
            print("Value removed from queue.")
    def enQueue(self, value):
        self.val = value
        if None in self.queue:#checks for space in queue
            #inserts value at the back of queue and removes a None (keep size)
            self.queue.insert((self.end),self.val)
            self.queue.remove(None)
            #adjusting size for queue
            if self.start == -1:
                self.start = 0
                self.end = 1
            else:
                self.end = (self.end+1) % (self.size)
```

```
            print("Value added to queue.")
        else:#queue must be full
            print("Not possible.")
    def isEmpty(self):
        for element in self.queue: #if any element != None, list is not empty
            if element != None:
                return False
            else:
                pass
        print("Queue empty.")
        return True
    def isFull(self):
        for element in self.queue:#if any element = None, list is not full
            if element == None:
                return False
            else:
                pass
        return True


class HiddenValues:
    def __init__(self,item):
        self.item = item


    def hideValues(self,startVal,endVal):
        self._hiddenVal = list(self.item)
        #replaces values up to endval with *
        for i in range (startVal,endVal+1):
            (self._hiddenVal)[i-1]='*'
        self.hiddenWord = ''.join(self._hiddenVal)
        return self.hiddenWord
```

\app\email.py

```
from flask_mail import Message
from app import mail, app
from flask import render_template
from threading import Thread
#asynchronous processing
def send_async_email(app, msg):
    with app.app_context():
        mail.send(msg)
def send_email(subject, sender, recipients, text_body, html_body):
    msg = Message(subject, sender=sender, recipients=recipients)
    msg.body = text_body
    msg.html = html_body
    #sends email asynchronously as not to slow processing of the program
    Thread(target=send_async_email, args=(app, msg)).start()
    print("email sent")
def sendPasswordResetEmail(user):
    token = user.get_reset_password_token()
    send_email('[Doc2Home] Reset Your Password',
               sender=app.config['ADMINS'][0],
               recipients=[user.email],
text_body=render_template('email/reset_password_email.txt',user=user,token=token),
               html_body=render_template('email/reset_password_email.html',              user=user,
token=token))
```

Thread schedules email in the background so the app can continue running while the email is sent

Email contains token based on user primary key to ensure that the correct user is being sent to the right reset password page

Email template is imported from web templates to employ inline HTML for links

\app\forms.py

```
from app.models import Doctor, Patient

from flask_wtf import FlaskForm

from wtforms import StringField, PasswordField, BooleanField, SubmitField, SelectField, FloatField, IntegerRangeField, DateField, TimeField, TextAreaField, IntegerField, SelectMultipleField

from wtforms.validators import InputRequired, Length, Email, EqualTo, ValidationError, DataRequired, NumberRange

from datetime import date, time, datetime, timedelta
```

Login form for objective 1.4

```
#login form for webpage

class LoginForm(FlaskForm):

    email = StringField('Email', validators=[InputRequired(),Length(min=3,max=32),Email()])

    password = PasswordField('Password', validators=[InputRequired(),Length(min=8,max=80)])

    remember = BooleanField('Remember me?')

    submit = SubmitField('Sign in')
```

Appointment request form for objective 4.1

```
#book appointment: default is next day

class AppointmentForm(FlaskForm):

    appdate = DateField('Date:',default=(date.today()+timedelta(days=1)),format='%Y-%m-%d',validators=[InputRequired()])

    apptime = TimeField('Time:',format='%H:%M',validators=[InputRequired()])

    submit = SubmitField('Request')


    def validate_appdate(form,field):

        if (field.data) <= date.today():

            print('ok')

            raise ValidationError('Appointment cannot be booked in the past.')


class DoctorAppointmentForm(FlaskForm):

    patient = SelectField('Patient', choices=[])
```

```
    appdate = DateField('Date:',default=(date.today()+timedelta(days=1)),format='%Y-%m-%d',validators=[InputRequired()])
    apptime = TimeField('Time:',format='%H:%M',validators=[InputRequired()])
    approom = StringField('Room:',validators=[InputRequired(),Length(min=1,max=4)])
    submit = SubmitField('Book')


    def validate_appdate(form,field):
        if (field.data) <= date.today():
            print('ok')
            raise ValidationError('Appointment cannot be booked in the past.')
```

Form for objective 6.1

```
class PatientAllergyForm(FlaskForm):
    allergy = StringField('Allergy', validators=[InputRequired()])
    severity = IntegerField('Severity',validators=[InputRequired(),NumberRange(min=1,max=5)])
    submit = SubmitField('Submit')
```

Form for objective 5.1.2

```
class MedicineAllergyForm(FlaskForm):
    medicine = StringField('Medicine', validators=[InputRequired()])
    allergen = StringField('Allergen', validators=[InputRequired()])
    submit = SubmitField('Submit')
```

Form for objective 10.1

```
class PrescriptionForm(FlaskForm):
    patient = SelectField('Patient', choices=[], validators=[InputRequired()])
    medicine = StringField('Medicine', validators=[InputRequired()])
    dosage = TextAreaField('Dosage', validators=[InputRequired(),Length(max=2048)])
    disease = StringField('Disease', validators=[InputRequired()])
    submit = SubmitField("Create prescription")



#confirming appointment
```

```
class AppointmentConfirmForm(FlaskForm):
    #default date for appointment booking is tomorrow
    appdate = DateField('Date:',format='%Y-%m-%d',validators=[DataRequired()])
    apptime = TimeField('Time:',format='%H:%M',validators=[DataRequired()])
    approom = StringField('Room:',validators=[DataRequired(),Length(min=1,max=4)])
    submit = SubmitField('Confirm')

    #checks if appointment has been booked before today
    def validate_appdate(self,appdate):
        if (appdate.data) <= date.today():
            print('ok')
            raise ValidationError('Appointment cannot be booked in the past.')

class AppointmentFollowUpForm(FlaskForm):
    fulfilled = BooleanField('Did the patient attend the appointment?')
    patient_feedback = TextAreaField('Follow-up notes for patient',validators=[InputRequired(),Length(max=1024)])
    doctor_notes = TextAreaField('Doctor notes',validators=[InputRequired(),Length(max=2048)])
    submit = SubmitField('Confirm')


class updateForm(FlaskForm):
    email = StringField('Email', validators=[InputRequired(),Length(min=3,max=32),Email()])
    houseNo = StringField('House number',validators=[InputRequired(),Length(max=5)])
    streetName = StringField('Street name',validators=[InputRequired(),Length(min=3,max=32)])
    postcode = StringField('Postcode',validators=[InputRequired(),Length(min=5,max=7)])
    submit = SubmitField("Register")

class patientRegistrationForm(FlaskForm):
    NHSNumber = StringField('NHS Number', validators=[InputRequired(),Length(min=10,max=10)])
```

How a doctor can confirm the appointment – objective 4.1

Form for objective 4.5

Patient registration form for objective 1.1

```
    dob = DateField('Date of birth',format='%Y-%m-%d',validators=[InputRequired()])
    birth_sex = SelectField('Birth sex',choices=[('M'),('F')],validators=[DataRequired()])
    forename = StringField('Forename',validators=[InputRequired()])
    surname = StringField('Surname',validators=[InputRequired()])
    email = StringField('Email', validators=[InputRequired(),Length(min=3,max=32),Email()])
    weight = FloatField('Weight',validators=[InputRequired(),NumberRange(min=0.0,max=500.0)])
    houseNo = StringField('House number',validators=[InputRequired(),Length(max=5)])
    streetName = StringField('Street name',validators=[InputRequired(),Length(min=3,max=32)])
    postcode = StringField('Postcode',validators=[InputRequired(),Length(min=5,max=7)])
    password = PasswordField('Password', validators=[InputRequired(),Length(min=8,max=80)])
    password2 = PasswordField('Repeat password',validators=[InputRequired(), EqualTo('password')])
    submit = SubmitField("Register")


    def validateNHSNumber(self, NHSNumber):
        try:
            int(NHSNumber)
        except:
            raise ValidationError("NHS Number is not a number.")
        user = Patient.query.filter_by(NHSNumber=NHSNumber.data).first()
        if user is not None:
            raise ValidationError("NHS Number already in use.")


    def validateEmail(self, email):
        user = Patient.query.filter_by(email=email.data).first()
        if user is not None:
            raise ValidationError("Email already in use.")


#example for user creation
```

Doctor registration form for objective 1.2

```
class doctorRegistrationForm(FlaskForm):
    email = StringField('Email', validators=[InputRequired(),Length(min=3,max=32),Email()])
    accessLev = SelectField('Access Level',default=2,choices=[(1),(2),(3)])
    NINumber = StringField('NI Number', validators=[InputRequired(),Length(min=9,max=9)])
    forename = StringField('Forename',validators=[InputRequired()])
    surname = StringField('Surname',validators=[InputRequired()])
    gender = SelectField('Gender',choices=[('M'),('F'),('O')],validators=[DataRequired()])
    dob = DateField('Date of birth',format='%Y-%m-%d',validators=[InputRequired()])
    houseNo = StringField('House number',validators=[InputRequired(),Length(max=5)])
    streetName = StringField('Street name',validators=[InputRequired(),Length(min=3,max=32)])
    postcode = StringField('Postcode',validators=[InputRequired(),Length(min=5,max=7)])
    password = PasswordField('Password', validators=[InputRequired(),Length(min=8,max=80)])
    password2 = PasswordField('Repeat password',validators=[InputRequired(), EqualTo('password')])
    submit = SubmitField("Register")


    def validate_email(self,email):
        user = Doctor.query.filter_by(email=email.data).first()
        if user is not None:
            raise ValidationError("Email already in use.")


    def validate_NINumber(self,NINumber):
        user = Doctor.query.filter_by(NINumber=NINumber.data).first()
        if user is not None:
            raise ValidationError("NI Number already in use.")


class RequestPasswordResetForm(FlaskForm):
    email = StringField('Email', validators=[DataRequired(), Email()])
    submit = SubmitField('Request Password Reset')
```

Reset password request form for objective 2

Resetting password for objective 2.2

```
class ResetPasswordForm(FlaskForm):
    password = PasswordField('Password', validators=[DataRequired()])
    password2 = PasswordField(
        'Repeat Password', validators=[DataRequired(), EqualTo('password')])
    submit = SubmitField('Request Password Reset')
```

Referral form for objective 9.2

```
class ReferralForm(FlaskForm):
    disease = StringField('Disease',validators=[InputRequired()])
    refDate = DateField('Date of referral',validators=[InputRequired()])
    refType = StringField('Type of referral',validators=[InputRequired()])
    ref_notes = TextAreaField('Referral notes',validators=[InputRequired(),Length(max=2048)])
    submit = SubmitField('Add referral')
```

Test record form for objective 4.6, 4.7

```
class TestForm(FlaskForm):
    testDate = DateField('Date of test',validators=[DataRequired()])
    patient = SelectField('Patient', choices=[], validators=[DataRequired()])
    testType = StringField('Type of test',validators=[InputRequired()])
    test_notes = TextAreaField('Doctor notes',validators=[InputRequired(),Length(max=2048)])
    submit = SubmitField('Log test')
```

Used to find patient profiles

```
class SearchForm(FlaskForm):
    NHSNumber = StringField('NHS Number', validators=[InputRequired(),Length(min=10,max=10)])
    submit = SubmitField('Search')


    def validate_NHSNumber(self,NHSNumber):
        user = Patient.query.filter_by(NHSNumber=int(NHSNumber.data)).first()
        if user is None:
```

```
        raise ValidationError("NHS Number not attached to a user.")
```

Illness history form for objective 9

```
class IllnessForm(FlaskForm):
    #nhs number included separately
    disease=StringField('Disease',validators=[InputRequired()])
    diagDate = DateField('Date of diagnosis',validators=[InputRequired()])
    endDate = DateField('Date illness is confirmed to have ended',validators=[])
    notes=TextAreaField('Doctor notes',validators=[InputRequired(),Length(max=2048)])
    submit = SubmitField('Log illness')

    def validate_diagDate(self,diagDate):
        if (diagDate.data) > date.today():
            raise ValidationError("Date cannot be in the future.")
```

\app\models.py – classes used to access database tables

```
from datetime import datetime, date, time
from time import time
import jwt
from werkzeug.security import generate_password_hash, check_password_hash
from app import app, db, login
from flask_login import UserMixin


class Doctor(UserMixin,db.Model):
#attributes to be added to db
    doctor_id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(50), index=True, unique=True)
    accessLev = db.Column(db.Integer, index=True)
    NINumber = db.Column(db.String(9), index=True, unique=True)
    forename = db.Column(db.String(30), index=True)
    surname = db.Column(db.String(50), index=True)
    gender = db.Column(db.String, index=True)
    dob = db.Column(db.Date)
    passwordHash = db.Column(db.String(128), index=True)
    address_id = db.Column(db.Integer, db.ForeignKey('address.address_id'))


    def setPassword(self, password):
        self.passwordHash = generate_password_hash(password)


    def checkPassword(self, password):
        return check_password_hash(self.passwordHash,password)


    def get_id(self):
        return (self.doctor_id)


    def get_reset_password_token(self, expires_in=600):
```

Each class in this file contains a model used to interact with the database – each attribute contains the table constraints

Required for login – also in patient class

Expiry for objective 2.1 – same method in the patient class

```
        return jwt.encode(
            {'reset_password': self.doctor_id, 'exp': time() + expires_in},
            app.config['SECRET_KEY'], algorithm='HS256')

    @staticmethod
    #reset password key contained in email that is hashed for the id
    def verify_reset_password_token(token):
        try:
            doctor_id = jwt.decode(token, app.config['SECRET_KEY'],
                        algorithms=['HS256'])['reset_password']
        except:
            print("\n failure")
            return
        return Doctor.query.get(doctor_id)



    def __repr__(self):#how the object is represented when called with no function
        return '<Dr. {}. {} born {}>'.format((self.forename)[0],self.surname,self.dob)


class HoursSpent(db.Model):
    hours_id = db.Column(db.Integer, primary_key=True)
    doctor_id = db.Column(db.Integer,db.ForeignKey('doctor.doctor_id'))
    date = db.Column(db.Date)
    entryTime = db.Column(db.Time)
    leaveTime = db.Column(db.Time)


    def __repr__(self):
        return '<Doctor id {} entered the surgery at {} and left at {} on {}.'.format(self.id,self.entryTime,self.leaveTime,self.date)


class Medicine(db.Model):
```

```
    med_id = db.Column(db.Integer, primary_key=True)
    medicineName = db.Column(db.String(64), index=True, unique=True)
    recommendedDose = db.Column(db.String(128))

    def __repr__(self):
        return f'<Medicine {self.medicineName} id {self.med_id} has recommended dosage {self.recommendedDose}.'

class Disease(db.Model):
    disease_id = db.Column(db.Integer, primary_key=True)
    diseaseName = db.Column(db.String(128), index=True)
    inheritability = db.Column(db.Boolean, default=False)

    def __repr__(self):
        return f'<Disease {self.diseaseName} id {self.disease_id} has inheritability status {self.inheritability}.'

class Allergen(db.Model):
    allergen_id = db.Column(db.Integer, primary_key=True)
    allergenName = db.Column(db.String(128), unique=True)

    def __repr__(self):
        return f'< {self.allergenName} >'

class Patient(UserMixin,db.Model):
    NHSNumber = db.Column(db.Integer, primary_key=True)
    forename = db.Column(db.String(30), index=True)
    surname = db.Column(db.String(50), index=True)
    email = db.Column(db.String(50), index=True, unique=True)
    dob = db.Column(db.Date, index=True)
    sex = db.Column(db.String(1), index=True)
```

Each model class inherits from the db.Model base class; users doctor and patient also inherit from UserMixin

```
    height = db.Column(db.Integer, index=True)#randomised
    weight = db.Column(db.Float, index=True)
    bloodType = db.Column(db.String(3), index=True)#randomised
    address_id = db.Column(db.Integer, db.ForeignKey('address.address_id'))
    passwordHash = db.Column(db.String(128), index=True)

    def setPassword(self, password):
        self.passwordHash = generate_password_hash(password)

    def checkPassword(self, password):
        return check_password_hash(self.passwordHash,password)

    def get_reset_password_token(self, expires_in=600):
        return jwt.encode(
            {'reset_password': self.NHSNumber, 'exp': time() + expires_in},
            app.config['SECRET_KEY'], algorithm='HS256')

    @staticmethod
    def verify_reset_password_token(token):
        try:
            NHSNumber = jwt.decode(token, app.config['SECRET_KEY'],
                                   algorithms=['HS256'])['reset_password']
        except:
            return
        return Patient.query.get(NHSNumber)

    def get_id(self):
        return (self.NHSNumber)

#address in separate table for speed purposes + because multiple individuals may live in the same address
```

Use of foreign key to link patient to address – also used by doctor

```
class Address(db.Model):
    address_id = db.Column(db.Integer, primary_key=True)
    houseNo = db.Column(db.String(5))
    streetName = db.Column(db.String(36), index=True)
    postcode = db.Column(db.String(7), index=True)

    def __repr__(self):
        return f'<{self.houseNo} {self.streetName}, {self.postcode}.>'

    def search(self, houseNumber, street, post_code):
        #searches for address passed as parameter
        address = Address.query.filter_by(houseNo=(houseNumber.lower()),streetName=(street.upper()),postcode=(post_code.upper())).first()
        if address is None: #if address has not been found, it is added to database
            address = Address(houseNo=(houseNumber.lower()),streetName=(street.upper()),postcode=(post_code.upper()))
            db.session.add(address)
            db.session.commit()
        return address.address_id

#medicines that contain a potential allergen
class Ingredient(db.Model):
    ingredient_id = db.Column(db.Integer, primary_key=True)
    med_id = db.Column(db.Integer, db.ForeignKey('medicine.med_id'))
    allergen_id = db.Column(db.Integer, db.ForeignKey('allergen.allergen_id'))

#carrier of inheritable disease
class Carrier(db.Model):
    carrier_id = db.Column(db.Integer, primary_key=True)
    NHSNumber = db.Column(db.Integer, db.ForeignKey('patient.NHSNumber'))
```

Method used to check if a new address needs to be added

```
    disease_id = db.Column(db.Integer, db.ForeignKey('disease.disease_id'))


    def __repr__(self) -> str:
        return f"< {Disease.query.filter_by(disease_id=self.disease_id).first().diseaseName} >"


class Allergy(db.Model):
    patientAllergy_id = db.Column(db.Integer, primary_key=True)
    NHSNumber = db.Column(db.Integer, db.ForeignKey('patient.NHSNumber'))
    allergen_id = db.Column(db.Integer, db.ForeignKey('allergen.allergen_id'))
    severity = db.Column(db.Integer, default=1)#severity of allergy from extreme to minimal (1-5)


    def __repr__(self):
        return f"< {Allergen.query.filter_by(allergen_id=self.allergen_id).first().allergenName} >"


class Prescription(db.Model):
    prescription_id = db.Column(db.Integer, primary_key=True)
    med_id = db.Column(db.Integer, db.ForeignKey('medicine.med_id'))
    NHSNumber = db.Column(db.Integer, db.ForeignKey('patient.NHSNumber'))
    doctor_id = db.Column(db.Integer, db.ForeignKey('doctor.doctor_id'))
    disease_id = db.Column(db.Integer, db.ForeignKey('disease.disease_id'))
    startDate = db.Column(db.Date, index=True)
    dose = db.Column(db.String(2048))
    finalRepeatDate = db.Column(db.Date, index=True)


    def __repr__(self):
        return f"< Prescription {self.prescription_id} for {Disease.query.filter_by(disease_id=self.disease_id).first().diseaseName} started {self.startDate} >"


class Appointment(db.Model):
```

```
    appointment_id = db.Column(db.Integer, primary_key=True)
    doctor_id = db.Column(db.Integer,db.ForeignKey('doctor.doctor_id'))
    NHSNumber = db.Column(db.Integer,db.ForeignKey('patient.NHSNumber'))
    appdate = db.Column(db.Date, index=True)
    apptime = db.Column(db.Time, index=True)
    room = db.Column(db.String(64))
    fulfilled = db.Column(db.Boolean, default=False)
    patient_feedback = db.Column(db.String(1024))#
    doctorNotes = db.Column(db.String(2048))

    def __repr__(self):
        return f'Appointment {self.appointment_id} date {self.appdate} time {self.apptime} in room {self.room}.'


class Test(db.Model):
    test_id = db.Column(db.Integer, primary_key=True)
    doctor_id = db.Column(db.Integer,db.ForeignKey('doctor.doctor_id'))
    NHSNumber = db.Column(db.Integer,db.ForeignKey('patient.NHSNumber'))
    testDate = db.Column(db.Date, index=True)
    testType = db.Column(db.String(64), index=True)
    testDoctorNotes = db.Column(db.String(2048))


#illnesshistory
class IllnessHistory(db.Model):
    unhealthy_id = db.Column(db.Integer, primary_key=True)
    NHSNumber = db.Column(db.Integer,db.ForeignKey('patient.NHSNumber'))
    disease_id = db.Column(db.Integer, db.ForeignKey('disease.disease_id'))
    diagnosisDate = db.Column(db.Date, index=True)
    endDate = db.Column(db.Date, index=True)
    notes = db.Column(db.String(4096))
```

```
    def __repr__(self):
        if self.endDate is None:
            return f'< Became ill with disease {Disease.query.filter_by(disease_id=self.disease_id).first().diseaseName} on {self.diagnosisDate} >'
        else:
            return f'< Became ill with disease {Disease.query.filter_by(disease_id=self.disease_id).first().diseaseName} on {self.diagnosisDate} and healed fully on {self.endDate} >'


class Referral(db.Model):
    referral_id = db.Column(db.Integer, primary_key=True)
    NHSNumber = db.Column(db.Integer, db.ForeignKey('patient.NHSNumber'))
    disease_id = db.Column(db.Integer, db.ForeignKey('disease.disease_id'))
    diagnosisDate = db.Column(db.Date, index=True)
    type = db.Column(db.String(64), index=True)
    notes = db.Column(db.String(2048))


    def __repr__(self):
        return f"< Referral of type {self.type} for disease {Disease.query.filter_by(disease_id=self.disease_id).first().diseaseName} took place on {self.diagnosisDate}. >"


@login.user_loader
def load_user(id):
    #searches for doctor/patient in database using
    if Patient.query.get(int(id)) != None:
        return Patient.query.get(int(id))
    else:
        return Doctor.query.get(int(id))
```

\app\numberGen.py

```
from random import randint, choice

from string import ascii_uppercase

from datetime import date, time, datetime

import numpy as np


#mean height- age:height(cm)

MEDIAN_M_HEIGHT =
{0:50,1:77,2:87,3:96,4:103,5:110,6:116,7:122,8:128,9:133,10:137,11:143,12:148,13:155,14:
162,15:169,16:173,17:176,'adult':175.3}

MEDIAN_F_HEIGHT =
{0:49,1:74,2:86,3:95,4:102,5:109,6:115,7:121,8:127,9:133,10:139,11:144,12:150,13:155,14:
160,15:162,16:163,17:163,'adult':161.9}

#Standard deviation under-18 height - age:cm

SD_M_HEIGHT =
{0:1.87,1:2.37,2:3.08,3:4.03,4:4.34,5:4.61,6:4.94,7:5.36,8:5.86,9:6.38,10:6.88,11:7.30,12:7.5
8,13:7.72,14:7.70,15:7.55,16:7.34,17:7.11,'adult':7.42}

SD_F_HEIGHT =
{0:1.88,1:2.58,2:3.24,3:4.09,4:4.53,5:4.91,6:5.33,7:5.79,8:6.26,9:6.67,10:6.97,11:7.12,12:7.1
1,13:6.97,14:6.76,15:6.55,16:6.41,17:6.39,'adult':7.11}


def checkPatient(user):
    try:
        if user.NHSNumber is not None:
            return True
    except:
        return False


def findAge(birthdate):
    today = date.today()
    #is birthday's calendar date before today's?
    beforeToday = ((today.month, today.day) < (birthdate.month, birthdate.day))
    year_difference = today.year - birthdate.year
```

Height data for random generation: objective 1.1.4

```
    age = year_difference - beforeToday
    return age
```

Objective 1.1.6

```
def heightGen(age,sex):
    if sex.upper() == ('M' or 'MALE'):
        if age < 18:
            mu = MEDIAN_M_HEIGHT[age]
            sigma = SD_M_HEIGHT[age]
        else:
            mu = MEDIAN_M_HEIGHT['adult']
            sigma = SD_M_HEIGHT['adult']
        height = np.random.normal(loc=mu,scale=sigma)
        return height
    elif sex.upper() == ("F" or "FEMALE"):
        if age < 18:
            mu = MEDIAN_F_HEIGHT[age]
            sigma = SD_F_HEIGHT[age]
        else:
            mu = MEDIAN_F_HEIGHT['adult']
            sigma = SD_F_HEIGHT['adult']
        height = np.random.normal(loc=mu,scale=sigma)
        return height
    else:
        return -1


def nhsgen():
    number = randint(1000000000,9999999999)
    return number


def insurancegen():
```

```
    #valid characters for each part of the NI number
    array1 = ['A','B','C','E','G','H','J','K','L','M','N','O','P','R','S','T','W','X','Y','Z']
    array2 = ['A','B','C','E','G','H','J','K','L','M','N','P','R','S','T','W','X','Y','Z']
    array3 = ['A','B','C','D']
    number = ''
    number = choice(array1)
    number += choice(array2)
    #first two letters of NI Number cannot be in given array
    while number in ['GB','BG','NK','KN','TN','NT','ZZ']:
        number = choice(array1)
        number += choice(array2)
    #middle part of NI number
    number += str(randint(100000,999999))
    #final letter of NI number
    number += choice(array3)
    return number
```

Random generation of blood type for objective 1.1.5

```
def bloodGen():
    bloodTypeArr = ['O+','O-','A+','A-','B+','B-','AB+','AB-']
    #random choice of weighted probabilities
    x=np.random.choice(bloodTypeArr, size=1, p=[(0.35/0.99),(0.13/0.99),(0.3/0.99),(0.08/0.99),(0.08/0.99),(0.02/0.99),(0.02/0.99),(0.01/0.99)])
    x=str(x)
    x=x[2:len(x)-2]
    return str(x)
```

\app\query_count.py

```
from app.models import Appointment
from datetime import datetime
from app.createdClass import myQueue
from sqlalchemy import *


def createQueryQueue(query):
    queue = None
    if query != None:#if there are requested appointments, creates queue to store them in order of date-time ascending
        if queryCount(query) > 10:
            queue = myQueue(10)
        else:
            queue = myQueue(queryCount(query))
        for i in range (queue.size):
            queue.enQueue(query[i])
    return queue


def queryCount(query):
    x=0
    if query is not None:
        for item in query:
            x+=1
    return x
```

\app\routes.py

```
from datetime import date, datetime
from time import localtime
from flask import render_template, flash, redirect, url_for, request, json
from flask_login import current_user, login_user, logout_user
from app import app, db
from app.forms import *
from app.email import sendPasswordResetEmail
from app.models import *
from app.appoint_val import checkOvertime, checkSlotFree
from app.createdClass import myQueue, HiddenValues
from app.numberGen import bloodGen, checkPatient, heightGen, findAge
from app.query_count import createQueryQueue, queryCount
from flask_login import login_required
from werkzeug.urls import url_parse
from werkzeug.security import generate_password_hash
from app.sql_connect import DB_Execute


GENDERS = ['M','F','O']
ACCESSLEV = [1,2,3]

def recent_apps(doctorID):
```

Gets recent patients for obj 10.1

```
    #searches for all appointments with the given doctor that take place after 7 days age - recent appointments
    recent_apps = (Appointment.query.filter(Appointment.doctor_id==doctorID,Appointment.appdate>=(date.today()-timedelta(days=7)))).order_by(Appointment.appdate.desc())
    if recent_apps is not None:
        choices=[]
        for item in recent_apps:
            #gets name for each user
            forename=Patient.query.filter_by(NHSNumber=item.NHSNumber).first().forename
```

```
        surname=Patient.query.filter_by(NHSNumber=item.NHSNumber).first().surname
        if (item.NHSNumber, (forename +' '+ surname)) not in choices:#no duplicates
            choices.append((item.NHSNumber, (forename +' '+ surname)))
    return choices


#homepage
@login_required
@app.route('/')
def index():
    try:
        isP=checkPatient(current_user) #is user a patient?
        if isP==False:
            #appointments not yet fulfilled

app=Appointment.query.filter_by(doctor_id=current_user.doctor_id,fulfilled=False).order_by(Appointment.appdate.asc(),Appointment.apptime.asc())
        arr=createQueryQueue(app)

        today=date.today().strftime("%Y-%m-%d")
        timenow=datetime.now().strftime("%H:%M")
        q=DB_Execute('app.db')


        #get appointments that have been attended but have not had notes added for
        query1=q.select(f"SELECT * FROM Appointment WHERE (fulfilled=True AND (doctorNotes or patient_feedback IS NULL)) and (appdate<'{today}' OR (appdate='{today}' and apptime<'{timenow}'))")
        if query1 is not None:
            app_no_notes=[]
            for item in query1:
                app_no_notes.append(item)
```

Route for homepage – objective 3

This queue stores appointments not yet fulfilled

```
            return render_template('index.html',title="Titlepage",isP=isP,nextapp=arr.queue,Med=Medicine,Pat=Patient,app_no_notes=app_no_notes,today=date.today())
        elif isP==True:
            #gets earliest future appointment and all current prescriptions that have not been ended yet

next_app=Appointment.query.filter(Appointment.NHSNumber==current_user.NHSNumber,Appointment.fulfilled==False,Appointment.appdate>=date.today()).order_by(Appointment.appdate.desc()).first()

current_pres=Prescription.query.filter(Prescription.NHSNumber==current_user.NHSNumber,Prescription.finalRepeatDate==None)
            return render_template('index.html',title="Titlepage",isP=isP,nextapp=next_app,current_pres=current_pres,Med=Medicine,today=date.today())
    except:
        return render_template('index.html',title="Titlepage",isP=isP)



#login page
@app.route('/login',methods=['GET','POST'])
def login():
    if current_user.is_authenticated:#is user is already logged in?
        return redirect(url_for('index'))


    form = LoginForm()
    if form.validate_on_submit():#searches for user in database
        user = Doctor.query.filter_by(email=(form.email.data).lower()).first()


        if user is None or not user.checkPassword(form.password.data):
            user = Patient.query.filter_by(email=(form.email.data).lower()).first()
            if user is None or not user.checkPassword(form.password.data):
                flash('Invalid username or password')
```

Route for objective 1.4

```
            return redirect(url_for('login'))

        login_user(user,remember=form.remember.data)
        #gets next page and redirects
        next_page = request.args.get('next')
        #is next page existing?
        if not next_page or url_parse(next_page).netloc != "":
            next_page = url_for('index')

        return redirect(next_page)
    return render_template('loginDoc.html', title='Sign in',form=form)

#patient registration page
@app.route('/registerPatient',methods=['GET','POST'])
def registerPatient():
    if current_user.is_authenticated:#is user already logged in?
        return redirect(url_for('index'))
    form = patientRegistrationForm()
    if form.validate_on_submit():
        #additional validations
        age = findAge(form.dob.data)
        #checks if user number or email is in the database already
        userNHS = Patient.query.filter_by(NHSNumber=int(form.NHSNumber.data)).first()
        userEm = Patient.query.filter_by(email=(form.email.data).lower()).first()
        docEm = Doctor.query.filter_by(email=(form.email.data).lower()).first()
        if (userNHS or userEm or docEm) is not None:
            flash('Email or NHS number already in use.')
            return redirect(url_for('registerPatient'))
        #ensures height generated is reasonable
        height = round(heightGen(age, (form.birth_sex.data)))
```

Route for objective 1.1

```
        if (height == -1) or ((age > 11) and (height < 101)):
            flash('Input error.')
            return redirect(url_for('registerPatient'))


        if age < 16:
            flash('You must be 16 or over to register for this service.')
            return redirect(url_for('registerPatient'))
        bloodType = bloodGen()
        #searches for address to see if it's already stored in the address db
        x=Address()
```

User and address created

```
addressID=x.search(form.houseNo.data.upper(),form.streetName.data.upper(),form.postcode.data.upper())
        user = Patient(NHSNumber=form.NHSNumber.data,forename=form.forename.data,surname=form.surname.data,email=(form.email.data).lower(),dob=form.dob.data,sex=form.birth_sex.data,height=height,weight=form.weight.data,bloodType=bloodType,address_id=addressID)
        #hashes password then adds user
        user.setPassword(form.password.data)
        db.session.add(user)
        flash(f'{form.email.data} Registered!')
        db.session.commit()
        return redirect(url_for('login'))
    return render_template('registerPatient.html', title='Patient registration', form=form)


#doctor registration page
@app.route('/registerDoc', methods=['GET', 'POST'])
def registerDoc():
    if current_user.is_authenticated:#user already logged in?
        return redirect(url_for('index'))
    form = doctorRegistrationForm()
    if form.validate_on_submit():
```

Registration for objective 1.2

```
        print("searching")
        #checking for presence of email or NI number already
        userEm = Doctor.query.filter_by(email=(form.email.data).lower()).first()
        patEm = Patient.query.filter_by(email=(form.email.data).lower()).first()
        userNI = Doctor.query.filter_by(NINumber=(form.NINumber.data).upper()).first()
        if (userEm or patEm or userNI or patEm) is not None:
            flash('Email or NI number already in use.')
            return redirect(url_for('registerDoc'))

        if form.dob.data > date(2000,1,1):
            flash('Are you sure you entered the correct birthday?')
            print('Are you sure you entered the correct birthday?')
            return redirect(url_for('registerDoc'))

        print("searching for address")
        #searching for address
        x = Address()
        addressID = x.search(form.houseNo.data.lower(),form.streetName.data.upper(),form.postcode.data.upper())
        #creating user
        user = Doctor(email=(form.email.data).lower(),accessLev=form.accessLev.data,NINumber=(form.NINumber.data).upper(),forename=form.forename.data,surname=form.surname.data,gender=form.gender.data,dob=form.dob.data,address_id=addressID)
        user.setPassword(form.password.data)

        db.session.add(user)
        flash('Registered!');print("Registered!")
        db.session.commit()
        return redirect(url_for('login'))
    else:
```

Objective 1.3 – also in patient registration

```
        print('errors',form.errors)
    return render_template('registerDoc.html', title='Doctor registration', form=form, accessLev=ACCESSLEV, genders=GENDERS)


#reset password
@app.route('/resetPasswordRequest', methods=['GET', 'POST'])
def resetPasswordRequest():
    if current_user.is_authenticated:
        flash("You must log out to reset your password.")
        return redirect(url_for('index'))
    form = RequestPasswordResetForm()
    if form.validate_on_submit():
        #checks to see if email is valid.
        pat = Patient.query.filter_by(email=form.email.data).first()
        doc = Doctor.query.filter_by(email=form.email.data).first()
        if pat:
            sendPasswordResetEmail(pat)
        elif doc:
            sendPasswordResetEmail(doc)
        else:
            flash('Email is not linked to an account')
            return redirect(url_for('resetPasswordRequest'))
        flash('Visit your email for the instructions to reset your password. Link expires in 10 minutes, be fast!')
        return redirect(url_for('login'))
    return render_template('reset_password_request.html', title='Reset Password', form=form)


@app.route('/resetPassword/<token>', methods=['GET', 'POST'])
def resetPassword(token):
    if current_user.is_authenticated:
        flash("Log out to reset password")
```

Objective 2

```
        return redirect(url_for('index'))
    #checks to see if user id hashes to same val as token before allowing reset
    user = Patient.verify_reset_password_token(token)
    if user is None:
        user = Doctor.verify_reset_password_token(token)
    if user is None:
        print('error')
        return redirect(url_for('index'))
    form = ResetPasswordForm()
    if form.validate_on_submit():
        #checks if password is still the same
        hasnotchanged=user.checkPassword(form.password.data)
        if hasnotchanged==False:
            user.setPassword(form.password.data)
            db.session.commit()
            flash('Your password has successfully been reset.')
            return redirect(url_for('login'))
        else:
            flash("Password has not changed! Enter a new password")
    return render_template('reset_password.html', form=form, token=token)


#profile view page
@app.route('/profile/<user_id>', methods=['GET','POST'])
def viewProfile(user_id):
    try:
        all_pres=None
        allergies=None
        ill_hist=None
        inherited=None
        Ninum=None
```

Viewing profile: getting prescriptions, allergies, illnesses, inherited diseases and NI number

```
    #is the user being viewed a doctor or patient
    user=Patient.query.filter_by(NHSNumber=int(user_id)).first()
    if user is None:
        user=Doctor.query.filter_by(doctor_id=int(user_id)).first()
        #hides ni num for security reason
        Ninum=HiddenValues(user.NINumber).hideValues(2,7)
    else:
        #gets prescriptions allergies illness history and carried diseases for profile

all_pres=createQueryQueue(Prescription.query.filter_by(NHSNumber=user.NHSNumber).or
der_by(Prescription.startDate.asc()))

allergies=createQueryQueue(Allergy.query.filter_by(NHSNumber=user.NHSNumber))

ill_hist=createQueryQueue(IllnessHistory.query.filter_by(NHSNumber=user.NHSNumber).o
rder_by(IllnessHistory.diagnosisDate.asc()))

inherited=createQueryQueue(Carrier.query.filter_by(NHSNumber=user.NHSNumber))
    isP_prof=checkPatient(user)#is the profile patient or doctor?
    isP=checkPatient(current_user)#is the user viewing the profile patient or doctor?
    address = Address.query.filter_by(address_id=(user.address_id)).first()
    return render_template('profile.html',account=user,houseNo=address.houseNo,
streetName=address.streetName,
postcode=address.postcode,isP=isP,isP_prof=isP_prof,all_pres=all_pres,allergies=allergies,ill
_hist=ill_hist,inherited=inherited,number=Ninum,spl=(len(address.postcode))-3)
  except:
    print("profile didn't work")
    flash('profile could not be loaded')
    return redirect(url_for('index'))


#update profile with email and address
@app.route('/update',methods=['GET','POST'])
def updateProfile():
  address = Address.query.filter_by(address_id=(current_user.address_id)).first()
```

NI number edited before passing through to prevent identity fraud

```
    form=updateForm()
    #checks if patient or doctor
    if checkPatient(current_user)==False:
        updatedUser = Doctor.query.filter_by(doctor_id=current_user.doctor_id).first()
    elif checkPatient(current_user)==True:
        updatedUser = Patient.query.filter_by(NHSNumber=current_user.NHSNumber).first()
    if form.validate_on_submit():
        print('form validated')
        updatedUser.email=form.email.data

check_add=Address.query.filter_by(houseNo=form.houseNo.data.lower(),streetName=(form.
streetName.data).upper(),postcode=(form.postcode.data).upper()).first()
        #checks if address submitted has changed
        if check_add is None:
            address.houseNo=form.houseNo.data.lower()
            address.streetName=form.streetName.data.upper()
            address.postcode=form.postcode.data.upper()
        else:
            updatedUser.address_id=check_add.address_id
        #updates user and address
        db.session.commit()
        flash("Visit profile to see effected changes.")
        return redirect(url_for('index'))
    return
render_template('update.html',form=form,updatedUser=updatedUser,address=address)
```

Allows doctors to visit profiles of patients

```
@app.route('/search', methods=['GET','POST'])
def search():#searches for patient by NHS number
    form=SearchForm()
    if form.validate_on_submit():
```

```
        user=Patient.query.filter_by(NHSNumber=int(form.NHSNumber.data)).first()
        return redirect(url_for('viewProfile', user_id=int(form.NHSNumber.data)))
    return render_template('search.html',isP=checkPatient(current_user),form=form)
```

Objective 4-4.5

```
#create appointment
@app.route('/createappointment',methods=['GET','POST'])
def createApp():
    isP = checkPatient(current_user)
    title='Book appointment'
    appointment = None
    if isP==True:#if you are a patient, you get the create-appointment form
        form = AppointmentForm()
        try:
            #gets previous appointments
```

How an appointment can be requested by a patient

```
prev_app=Appointment.query.filter(Appointment.NHSNumber==current_user.NHSNumber,Appointment.fulfilled==True).order_by(Appointment.appdate.asc(),Appointment.apptime.asc())
            for item in prev_app:
                print(item)
            prev_queue=createQueryQueue(prev_app,size=99)
            print('done')
        except:
            pass
        #appointment can only be booked if they do not have an upcoming appointment
        appointment = Appointment.query.filter(Appointment.NHSNumber==current_user.NHSNumber,Appointment.fulfilled==False,Appointment.appdate>=date.today()).first()
        if appointment == None:
            if form.validate_on_submit():
                #checking if appointment doesn't clash
```

```
            appointment = Appointment(NHSNumber=current_user.NHSNumber,appdate=(form.appdate.data),apptime=(form.apptime.data))
            db.session.add(appointment)
            db.session.commit()
            print('added to db')
            return redirect(url_for('index'))
        #shows them upcoming appointment
        else:
            return render_template('appointments.html',title='Book appointment',form=form,appointment=appointment,isP=isP,prev_queue=prev_queue.queue,doc=Doctor)

    elif isP == False:# user is a doctor allowed to view appointments
        form = AppointmentConfirmForm()
        title='Requested appointments'
        doctorForm=DoctorAppointmentForm()
        doctorForm.patient.choices = [(pat.NHSNumber, (pat.forename +' '+ pat.surname)) for pat in Patient.query.all()]
        #searching for appointments that haven't been confirmed yet (i.e. no room)

q1=Appointment.query.filter_by(room=None).order_by(Appointment.appdate.asc(),Appointment.apptime.asc()).all()
        queue1=createQueryQueue(q1)

        #searching for appointments in the past that were fulfilled and have no notes

q2=Appointment.query.filter(Appointment.doctorNotes==None,Appointment.appdate<=date.today(),(Appointment.doctor_id==current_user.doctor_id)).order_by(Appointment.appdate.asc(),Appointment.apptime.asc())
        queue2=createQueryQueue(q2)

        #searching for appointments that are happening in the future and have been confirmed by said doctor.
```

```
q3=Appointment.query.filter(Appointment.doctorNotes==None,(Appointment.room!=None),(Appointment.doctor_id==current_user.doctor_id)).order_by(Appointment.appdate.asc(),Appointment.apptime.asc())

    queue3=createQueryQueue(q3)


    if doctorForm.validate_on_submit():
        #checks if slot is free
        isFree, message = checkSlotFree(current_user.doctor_id, doctorForm.appdate.data, doctorForm.apptime.data, doctorForm.approom.data)
        flash(message)
        if isFree == True:
            #creates apppointment that is only added if overtime is not exceeded
            newAppointment = Appointment(doctor_id=current_user.doctor_id,NHSNumber=(doctorForm.patient.data),appdate=doctorForm.appdate.data,apptime=doctorForm.apptime.data,room=doctorForm.approom.data)
            if checkOvertime(current_user.doctor_id,current_user.accessLev,appt_date=doctorForm.appdate.data) == True:#redirects to appointment viewer for confirmation of appointment if user is going into overtime


item=(Appointment.query.filter_by(doctor_id=current_user.doctor_id,NHSNumber=doctorForm.patient.data,appdate=doctorForm.appdate.data,apptime=doctorForm.apptime.data)).first()


forename=(Patient.query.filter_by(NHSNumber=item.NHSNumber).first()).forename


surname=(Patient.query.filter_by(NHSNumber=item.NHSNumber).first()).surname

                return render_template('appointment_viewer.html',id=item.appointment_id, form=form, message="Overtime limits exceeded...",isP=isP,fn=forename,sn=surname,item=item,today=date.today(),newApp=newAppointment)
            db.session.add(newAppointment)
            db.session.commit()
            return redirect(url_for('index'))
```

Objective 4.3

Objective 11

```
        return
render_template('appointments.html',title=title,queue1=queue1.queue,queue2=queue2.queue,
queue3=queue3.queue,isP=isP,doc=Doctor,pat=Patient,form=form,doctorForm=doctorForm)

    return
render_template('appointments.html',title=title,form=form,isP=isP,prev_queue=prev_queue.q
ueue,doc=Doctor)


@app.route('/referral/<NHSNum>',methods=['GET','POST'])

def createRef(NHSNum):

    form=ReferralForm()

    #shows all previous referrals


prev_ill=createQueryQueue(Referral.query.filter_by(NHSNumber=NHSNum).order_by(Refe
rral.diagnosisDate.asc()),99)

    if form.validate_on_submit():

        #checks to ensure disease is in database to get id to add referral

        dis_id=Disease.query.filter_by(diseaseName=form.disease.data).first().disease_id


ref_search=Referral.query.filter_by(NHSNumber=NHSNum,diagnosisDate=form.refDate.dat
a,type=form.refType.data).first()

        if ref_search is None:


new_ref=Referral(NHSNumber=NHSNum,diagnosisDate=form.refDate.data,disease_id=dis_
id,type=form.refType.data,notes=form.ref_notes.data)

            db.session.add(new_ref)

            db.session.commit()

            flash('referral added')

        else:

            flash('referral already recorded')

    return
render_template('referral.html',form=form,isP=checkPatient(current_user),title='Create
referral',prev_ill=prev_ill.queue)


@app.route('/viewreferral/<id>',methods=['GET','POST'])

def viewRef(id):
```

Objective 9.2

```
    refer=Referral.query.filter_by(referral_id=id).first()
    allowed=True
    isP=checkPatient(current_user)
    #doctors need to have had or be having an appointment with patient in next two weeks before seeing
    if isP==False:
        if current_user.accessLev > 1:
            if (Appointment.query.filter(Appointment.appdate<=(date.today()+timedelta(days=14)),Appointment.doctor_id==current_user.doctor_id)) is None:
                allowed=False
    elif isP==True:#patients cannot view other users' referrals
        if refer.NHSNumber!=current_user.NHSNumber:
            allowed=False
    return render_template('ref_viewer.html',allowed=allowed,ref=refer,doc=Doctor,pat=Patient,id=id,isP=isP)


@app.route('/test/<appoint>',methods=['GET','POST'])
def createTest(appoint=None):#creating test linked to appointment by id
    form=TestForm()
    print(form.errors)
```

Objective 4.6, 4.7

```
    NHSNum=Appointment.query.filter_by(appointment_id=appoint).first().NHSNumber
    day=Appointment.query.filter_by(appointment_id=appoint).first().appdate
    forename=Patient.query.filter_by(NHSNumber=NHSNum).first().forename
    surname=Patient.query.filter_by(NHSNumber=NHSNum).first().surname
    form.patient.choices=[(NHSNum,forename+' '+surname)]


    if form.validate_on_submit():
        print('sub')
```

```
test_search=Test.query.filter_by(NHSNumber=form.patient.data,testDate=form.testDate.data
,testType=form.testType.data).first()
        if test_search is None:
            print('ok')

new_test=Test(NHSNumber=form.patient.data,testDate=form.testDate.data,doctor_id=curren
t_user.doctor_id,testType=form.testType.data,testDoctorNotes=form.test_notes.data)
            db.session.add(new_test)
            db.session.commit()
            print('test')
            return redirect(url_for('viewApp', id=appoint))


    return    render_template('test.html',form=form,isP=checkPatient(current_user),title='Create
test',fn=forename,sn=surname,day=day,NHSNum=NHSNum)


@app.route('/viewtest/<id>',methods=['GET','POST'])
def viewTest(id):
    test=Test.query.filter_by(test_id=id).first()
    allowed=True
    isP=checkPatient(current_user)
    if isP==False:#same viewing criteria as referral
        if current_user.accessLev > 1:
            if
(Appointment.query.filter(Appointment.appdate<=(date.today()+timedelta(days=14)),Appoin
tment.doctor_id==current_user.doctor_id)) is None:
                allowed=False
    elif isP==True:
        if test.NHSNumber!=current_user.NHSNumber:
            allowed=False
    return
render_template('test_viewer.html',allowed=allowed,test=test,doc=Doctor,pat=Patient,id=id,i
sP=isP)
```

Objective 4.9

```
@app.route('/createIllness/<NHSNum>',methods=['GET','POST'])
def createIllness(NHSNum):
    form=IllnessForm()
    allowed=False
    #previous illness history

prev_ill=createQueryQueue(IllnessHistory.query.filter_by(NHSNumber=NHSNum).order_by(IllnessHistory.diagnosisDate.asc()),99)
    fn=Patient.query.filter_by(NHSNumber=NHSNum).first().forename
    sn=Patient.query.filter_by(NHSNumber=NHSNum).first().surname
    #gets appointments that the patient has within period four weeks ago and two weeks in the future
    four_weeks_ago=date.today()-timedelta(days=28)
    future_two_weeks=date.today()+timedelta(days=14)

next_apps=Appointment.query.filter(Appointment.appdate>=four_weeks_ago,Appointment.appdate<=future_two_weeks,Appointment.doctor_id==current_user.doctor_id,Appointment.NHSNumber==NHSNum)


    if form.validate_on_submit():
        try:
            #checks to see if disease is in database

dis_id=Disease.query.filter_by(diseaseName=(form.disease.data).lower()).first().disease_id
            #searches if illness has already been recorded

u=IllnessHistory.query.filter_by(NHSNumber=NHSNum,disease_id=dis_id,diagnosisDate=(form.diagDate.data)).first()
            if u is not None:
                flash('illness has already been recorded')
            else:
                print(NHSNum)
                print(form.diagDate.data)
```

Objective 9, 9.1

```
new_ill=IllnessHistory(NHSNumber=NHSNum,disease_id=dis_id,diagnosisDate=(form.diag
Date.data),notes=form.notes.data)
            if form.endDate.data is not None:
                if form.endDate.data < form.diagDate.data:
                    flash('disease may not end before it has started')
                    return redirect(url_for('createIllness', NHSNum=NHSNum))
                new_ill.endDate==form.endDate.data
            db.session.add(new_ill)
            db.session.commit()
            flash('illness added')
        except:
            flash('disease does not exist')
        return redirect(url_for('createIllness', NHSNum=NHSNum))


    if (current_user.accessLev==1) or (next_apps is not None):
        allowed=True
    return render_template('illness.html',title='Patient illness
history',prev_ill=prev_ill.queue,isP=checkPatient(current_user),NHSNum=NHSNum,form=f
orm,fn=fn,sn=sn,allowed=allowed,d=Disease)


@app.route('/viewHist/<id>',methods=['GET','POST'])
def viewHist(id):#viewing individual illness history
    illness_hist=IllnessHistory.query.filter_by(unhealthy_id=id).first()
    allowed=True
    isP=checkPatient(current_user)
    fn=Patient.query.filter_by(NHSNumber=illness_hist.NHSNumber).first().forename
    sn=Patient.query.filter_by(NHSNumber=illness_hist.NHSNumber).first().surname
    if isP==False:
        if current_user.accessLev > 1:
```

Patient illness history not accessible to all doctors

```
        if (Appointment.query.filter(Appointment.appdate<=(date.today()+timedelta(days=14)),Appointment.doctor_id==current_user.doctor_id)) is None:
            allowed=False
    elif isP==True:
        if illness_hist.NHSNumber!=current_user.NHSNumber:
            allowed=False
    return render_template('hist_viewer.html',allowed=allowed,illness_hist=illness_hist,id=id,isP=isP,fn=fn,sn=sn)


#view appointment
@app.route('/viewapp/<id>',methods=['GET','POST'])
def viewApp(id):
    #searching for appointment details
    item=Appointment.query.filter_by(appointment_id=id).first()
    forename=(Patient.query.filter_by(NHSNumber=item.NHSNumber).first()).forename
    surname=(Patient.query.filter_by(NHSNumber=item.NHSNumber).first()).surname
    #checking for a test on that day
    #checking to see if appointment can be reviewed
    if item.patient_feedback is not None:
        #displays information rather than a form
        forename=(Doctor.query.filter_by(doctor_id=item.doctor_id).first()).forename
        surname=(Doctor.query.filter_by(doctor_id=item.doctor_id).first()).surname
        post_app=True
        #gets tests linked to appointment by date to display
        linked_test=createQueryQueue(Test.query.filter_by(doctor_id=item.doctor_id,NHSNumber=item.NHSNumber,testDate=item.appdate))
        return render_template('appointment_viewer.html',id=id,isP=checkPatient(current_user),fn=forename,sn=surname,item=item,today=date.today(),post_app=post_app,linked_test=linked_test.queue)
    else:
```

Individual patient viewer for 4.2-4.5

Tests linked to appointment passed through – 4.9

```
    form = AppointmentConfirmForm()
    followUp=AppointmentFollowUpForm()
    post_app=False


  if form.validate_on_submit():
    #form checked after doctor confirms request
    if checkPatient(current_user)==False:
      #checks if slot is free, then if the user would be working overtime
      isFree, message = checkSlotFree(current_user.doctor_id, form.appdate.data, form.apptime.data, form.approom.data)
      if isFree==True:
        if checkOvertime(current_user.doctor_id,current_user.accessLev,appt_date=form.appdate.data)==True:
          flash('Overtime limit exceeded. Choose another date.')
          return render_template('appointment_viewer.html',id=id,isP=checkPatient(current_user),form=form,fn=forename,sn=surname,item=item,today=date.today(),followUp=followUp)
        flash(message)
        flash('appointment confirmed!')
      else:
        flash(message)
        flash('Please rebook appointment at different time or room.')
        return render_template('appointment_viewer.html',id=id,isP=checkPatient(current_user),form=form,fn=forename,sn=surname,item=item,today=date.today(),followUp=followUp)


    print('done')
    #adding the new form submitted data to the database to finially confirm
    appt = Appointment.query.filter_by(appointment_id=id).first()
    appt.appdate = form.appdate.data
    appt.apptime = form.apptime.data
    appt.room = form.approom.data
```

```
        appt.doctor_id = current_user.doctor_id
        db.session.commit()
        return redirect(url_for('createApp'))

    if followUp.validate_on_submit():
        #doctor has finished appointment and is adding the information for a patient to view
        appt=Appointment.query.filter_by(appointment_id=id).first()
        appt.fulfilled=followUp.fulfilled.data
        appt.patient_feedback=followUp.patient_feedback.data
        appt.doctorNotes=followUp.doctor_notes.data
        db.session.commit()
        return redirect(url_for('createApp'))


    return
render_template('appointment_viewer.html',id=id,isP=checkPatient(current_user),form=form
,fn=forename,sn=surname,item=item,today=date.today(),followUp=followUp)
```

Objective 4.5

```
@app.route('/cancelApp/<id>')
def cancelApp(id):#cancelling appointment through api
    Appointment.query.filter_by(appointment_id=id).delete()
    db.session.commit()
    flash("Your appointment has been deleted.")
    return redirect(url_for('index'))


#add/search for allergen
@app.route('/allergens',methods=['GET','POST'])
def createAllergen():
    indb=True
    #checks if user is level 1/2 doctor before they can add
    if (checkPatient(current_user) == False) and (current_user.accessLev < 3):
        #gets allergen name and checks in database to see if unique
```

Objective 4.4

Objective 6

```
        allergen_data = request.form.get('allergenName')
        if allergen_data is not None:
            query=Allergen.query.filter_by(allergenName=(allergen_data.lower())).first()
            if query==None:
                indb=False
                new_allergen=Allergen(allergenName=(allergen_data.lower()))
                db.session.add(new_allergen)
                db.session.commit()
            return
render_template('allergen.html',isP=checkPatient(current_user),indb=indb,allergen=allergen_
data)
    return render_template('allergen.html',isP=checkPatient(current_user))


@app.route('/allergies/<NHSNum>',methods=['GET','POST'])
def createPatientAllergy(NHSNum):#linking allergy to patient
    form = PatientAllergyForm()
    fn=Patient.query.filter_by(NHSNumber=NHSNum).first().forename
    sn=Patient.query.filter_by(NHSNumber=NHSNum).first().surname
    if form.validate_on_submit():
        #checks for allergen id then adds

allergen_id=(Allergen.query.filter_by(allergenName=(form.allergy.data.lower())).first()).aller
gen_id


new_patallergy=Allergy(NHSNumber=NHSNum,allergen_id=allergen_id,severity=form.sev
erity.data)
        db.session.add(new_patallergy)
        db.session.commit()
        q=DB_Execute('app.db')
        #searching the prescriptions that the patient has that contain this allergen in a medicine
        query1=q.select(f'SELECT prescription.prescription_id FROM Prescription inner JOIN
Ingredient ON (Prescription.med_id=Ingredient.med_id) where
(ingredient.allergen_id={allergen_id} and prescription.NHSNumber={NHSNum})')
```

Objective 6.1

```
        #removes all duplicates
        query1=list(set(query1))
        #ending these prescriptions
        if query1 is not None:
            for item in query1:
                x=Prescription.query.filter_by(prescription_id=item[0]).first()
                x.finalRepeatDate=date.today()
                flash(f'Prescription {item[0]} ended due to allergy clash')
            db.session.commit()
        flash('Allergy for patient added.')
        return redirect(url_for('viewProfile', user_id=NHSNum))


    return
render_template('allergy.html',form=form,isP=checkPatient(current_user),NHSNum=NHSNum,fn=fn,sn=sn)


@app.route('/medicine',methods=['GET','POST'])
def createMedicine():#searching/adding for medicines
    if (checkPatient(current_user) == False) and (current_user.accessLev < 3):
        #gets name and dose from form
        medicine_data = request.form.get('medicineName')
        med_dosage = request.form.get('dosage')
        if medicine_data is not None:
            #checks if medicine is in database
            query=Medicine.query.filter_by(medicineName=(medicine_data.lower())).first()
            if query==None:

new_medicine=Medicine(medicineName=(medicine_data.lower()),recommendedDose=(med_dosage.lower()))
                db.session.add(new_medicine)
                db.session.commit()
                flash(f"{medicine_data.lower()} added to database.")
```

Objective 6.4

Objective 5, 5.1

```
            #redirect to the link med-allergy page
            return redirect(url_for('createMedAllergy'))
        return render_template('medicine.html',isP=checkPatient(current_user),indb=True,medicine=medicine_data)
    return render_template('medicine.html',isP=checkPatient(current_user))


@app.route('/medicine/<name>',methods=['GET','POST'])
def viewMedicine(name):
    name=name
    allergies=[]
    med_id=Medicine.query.filter_by(medicineName=name).first()
    #searches for allergens that are linked to medicine
    q=DB_Execute('app.db')
    query1=q.select(f'SELECT Allergen.allergenName FROM Allergen, Ingredient, Medicine WHERE Allergen.allergen_id=Ingredient.allergen_id AND Ingredient.med_id=Medicine.med_id AND Medicine.medicineName="{name}"')
    if query1 is not None:#GETS LIST OF ALLERGENS INSIDE MEDICINE
        for item in query1:
            allergies.append(item[0])
    return render_template('medicine_viewer.html',isP=False,name=name,allergies=allergies)


@app.route('/medicineAllergy',methods=['GET','POST'])
def createMedAllergy():
    form = MedicineAllergyForm()
    if form.validate_on_submit():
        indb=True
        #checks if patient is access lev 1 or 2
        if (checkPatient(current_user) == False) and (current_user.accessLev < 3):
            med_id = (Medicine.query.filter_by(medicineName=(form.medicine.data).lower()).first()).med_id
```

Objective 5.1.3

Complex multi-table query used to find allergens in the medicine

Objective 5.1.2

```
        allergen_id = (Allergen.query.filter_by(allergenName=(form.allergen.data).lower()).first()).allergen_id
        #if medicine + allergen is not stored as ingredient already
        if Ingredient.query.filter_by(med_id=med_id,allergen_id=allergen_id).first() == None:
            newIngredient = Ingredient(med_id=med_id,allergen_id=allergen_id)
            db.session.add(newIngredient)
            db.session.commit()
            indb=False
        return render_template('ingred.html',form=form,isP=checkPatient(current_user),indb=indb,medicine=form.medicine.data)
    else:
        flash("unauthorised action")
        return render_template('ingred.html',form=form,isP=checkPatient(current_user))
  return render_template("ingred.html",form=form,isP=checkPatient(current_user))


@app.route('/disease',methods=['GET','POST'])
def createDisease():#add disease to system
  if request.method == 'POST':
    indb=True
    disease_name = request.form.get('disease_name')
    inheritable = request.form.get('ch1')
    #checks if disease is already recorded
    query=Disease.query.filter_by(diseaseName=(disease_name.lower())).first()
    if query is None:#disease doesn't exist
        new_disease = Disease(diseaseName=(disease_name.lower()),inheritability=False)
        if inheritable is not None: #the disease is inheritable if it is ticked on
            new_disease.inheritability = True
        db.session.add(new_disease)
        db.session.commit()
        indb=False
```

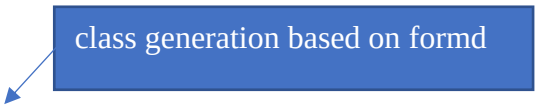
Objective 7

```
        return      render_template('disease_add.html',isP=checkPatient(current_user),indb=indb,
message="Disease added", disease=disease_name)
    return render_template('disease_add.html',isP=checkPatient(current_user))


@app.route('/inheritable/<NHSNum>',methods=['GET','POST'])
def createInherit(NHSNum):#add carried disease to a patient
    if request.method == 'POST':
        indb=True
        disease_name=request.form.get('disease_name')
        #is the disease actually inheritable?
        dis_id=Disease.query.filter_by(diseaseName=disease_name,inheritability=True).first()
        if dis_id is not None:
            #has inheritable disease been recorded already

carrier=Carrier.query.filter_by(disease_id=dis_id.disease_id,NHSNumber=NHSNum).first()
            if carrier is None:
                new_carrier=Carrier(disease_id=dis_id.disease_id,NHSNumber=NHSNum)
                db.session.add(new_carrier)
                db.session.commit()
                indb=False


            return
render_template('carrier.html',isP=checkPatient(current_user),indb=indb,NHSNum=NHSNu
m,patient=Patient.query.filter_by(NHSNumber=NHSNum).first(),disease=disease_name)
        else:
            flash('Disease does not exist, enter a valid disease name')
    return
render_template('carrier.html',isP=checkPatient(current_user),NHSNum=NHSNum,patient=P
atient.query.filter_by(NHSNumber=NHSNum).first())


@app.route('/prescription',methods=['GET','POST'])
def createPres():#creating prescription
```

Objective 8

Objective 10, 10.1

```
    if checkPatient(current_user)==False:
        form = PrescriptionForm()
        #doctor may only create prescriptions for patients he is due to interact with
        form.patient.choices=recent_apps(current_user.doctor_id)

        if form.validate_on_submit():
            #searching for prescription with same patient and medicine that hasn't ended yet
            medid = Medicine.query.filter_by(medicineName=form.medicine.data).first()
            disid = Disease.query.filter_by(diseaseName=form.disease.data).first()
            if (medid or disid) is None:
                flash("Disease or medicine does not exist. Ensure you have spelt the name correctly and are using the autocomplete.")
                return render_template('prescription.html', isP=checkPatient(current_user), form=form)
            else:
                #checks if prescription already exists - if it does it is updated
                q1 = Prescription.query.filter_by(NHSNumber=form.patient.data,med_id=medid.med_id,disease_id=disid.disease_id,finalRepeatDate=None).first()
                if q1 is not None:
                    if q1.startDate == date.today():
                        q1.dose = form.dosage.data
                    else:
                        new_pres = Prescription(med_id=medid.med_id,NHSNumber=form.patient.data,doctor_id=current_user.doctor_id,disease_id=disid.disease_id,startDate=date.today(),dose=form.dosage.data)
                        q1.finalRepeatDate=date.today()
                    db.session.commit()
                    flash("Prescription updated")
                    return render_template('prescription.html', isP=checkPatient(current_user), form=form)
                else:
                    #searching illness history to see if the patient is ill and it has been recorded
```



```
q2=IllnessHistory.query.filter(IllnessHistory.NHSNumber==form.patient.data,IllnessHistory.
disease_id==disid.disease_id,IllnessHistory.diagnosisDate<=date.today(),IllnessHistory.end
Date==None).first()
                if queryCount(q2)>0:
                    pass
                else:
new_illness=IllnessHistory(NHSNumber=form.patient.data,disease_id=disid.disease_id,diag
nosisDate=date.today())
                    db.session.add(new_illness)
                    db.session.commit()
                #searching for allergens found in medicine
                allergens_in_med = Ingredient.query.filter_by(med_id=medid.med_id)
                for x in allergens_in_med:
                    all_id=x.allergen_id
                    #search allergy table to see if patient has the allergic reaction
                    if
Allergy.query.filter_by(NHSNumber=form.patient.data,allergen_id=all_id).first() is not
None:
```

Objective 6.4

```
flash(f"{Medicine.query.filter_by(med_id=medid.med_id).first().medicineName} contains
the allergen {Allergen.query.filter_by(allergen_id=all_id).first().allergenName} and thus
cannot be prescribed to the patient, due to risk of allergic reaction.")
                    return render_template('prescription.html', isP=checkPatient(current_user),
form=form)

                new_pres =
Prescription(med_id=medid.med_id,NHSNumber=form.patient.data,doctor_id=current_user.
doctor_id,disease_id=disid.disease_id,startDate=date.today(),dose=form.dosage.data)
                db.session.add(new_pres)
                db.session.commit()
                print('added pres')
                flash('prescription added')
                return redirect((url_for('index')))
        return render_template('prescription.html', isP=checkPatient(current_user), form=form)
```

```
@app.route('/viewPrescription/<id>',methods=['GET','POST'])
def viewPres(id):#view prescription
    pres=Prescription.query.filter_by(prescription_id=id).first()
    medName=Medicine.query.filter_by(med_id=pres.med_id).first().medicineName
    doc=Doctor.query.filter_by(doctor_id=pres.doctor_id).first()
    docName=doc.forename+' '+doc.surname
    disName=Disease.query.filter_by(disease_id=pres.disease_id).first().diseaseName
    print(disName)
    return render_template('prescription_viewer.html', isP=checkPatient(current_user),
title='View
Prescription',medName=medName,docName=docName,disName=disName,pres=pres)


@app.route('/api/<name>')
def api_get(name):#api returning lists for each autocompleted function
    if name == 'allergens':
        return json.jsonify({
            'allergens': [(substance.allergenName) for substance in Allergen.query.all()]
        })
    elif name == 'medicine':
        return json.jsonify({
            'medicine': [(substance.medicineName) for substance in Medicine.query.all()]
        })
    elif name == 'disease':
        return json.jsonify({
            'disease': [(substance.diseaseName) for substance in Disease.query.all()]
        })
    elif name == 'carrier':
        return json.jsonify({
            'carrier': [(substance.diseaseName) for substance in
Disease.query.filter(Disease.inheritability==True)]
        })
```

Objective 10.3

Gets autocomplete for multiple objectives incl. 4, 5, 8, 9, 10

```
    elif name == 'referral_type':
        if Referral.query.all() is not None:
            x={'referral_type': [(t.type) for t in Referral.query.all()]}
            for i in range (len(x['referral_type'])):
                #checks for duplicates - if the index is not the first returned then there is more than one of the value in the array
                if x['referral_type'].index(x['referral_type'][i]) != i:
                    x['referral_type'].pop(i)
        else:
            x={'referral_type': []}
        return json.jsonify(x)
    elif name == 'test_type':
        if Test.query.all() is not None:
            x={'test_type': [(t.testType) for t in Test.query.all()]}
            for i in range (len(x['test_type'])):
                #checks for duplicates - if the index is not the first returned then there is more than one of the value in the array
                if x['test_type'].index(x['test_type'][i]) != i:
                    x['test_type'].pop(i)
        else:
            x={'test_type': []}
        return json.jsonify(x)
```

Each of these api calls return JSON arrays to the AJAX request

```
#logging out of account
@app.route('/logout')
def logout():
    logout_user()
    print("logged out")
    return redirect(url_for('index'))
```

\app\sql_connect.py

```
import sqlite3

def dbComm(database):
    #database param should be string type, i.e. relative path to db
    conn = sqlite3.connect(database)
    cursor = conn.cursor()
    command = input(f"\nEnter your SQL command for {database}: ")
    while command.lower() != "exit":
        try:
            for row in cursor.execute(command):
                print(row)
            conn.commit()
        except:
            print("invalid request")
        command = input(f"\nEnter your SQL command for {database}: ")


class DB_Execute():
    def __init__(self,db):
        self.db = db
        self.conn=sqlite3.connect(self.db)
        self.cursor=self.conn.cursor()

    def select(self,command):
        try:
            self.cursor.execute(command)
            self.items=self.cursor.fetchall()
            print(self.items)
            self.conn.commit()
            return self.items
        except:
            print(f'The command {command} could not be executed, or it returned no result.')
```

Used to communicate with db for raw sql commands

```
            return None


    def other_com(self,command):
        try:
            self.cursor.execute(command)
            self.conn.commit()
            print(f"The command {command} has been executed.")
        except:
            print(f'The command {command} could not be executed.')
```

\app\static\style.css

```
.topnav {
	background-color: #333;
	overflow: hidden;
	line-height: 2.5;
	}

a:link {
	text-decoration: none;
	color: red;
	}

a:visited {
text-decoration: none;
color: skyblue;
	}

body { background-color: bisque;
	color: #333;
	font: 16px Arial;
	text-align: center;
	}

	* {
	box-sizing: border-box;
	}

	/*the container must be positioned relative:*/
	.autocomplete {
	position: relative;
	display: inline-block;
```

```
}

input[type=submit] {
background-color: DodgerBlue;
color: #fff;
cursor: pointer;
}

.autocomplete-items {
position: absolute;
border: 1px solid #d4d4d4;
border-bottom: none;
border-top: none;
z-index: 99;
/*position the autocomplete items to be the same width as the container:*/
top: 100%;
left: 0;
right: 0;
}

.autocomplete-items div {
cursor: pointer;
background-color: #fff;
border-bottom: 1px solid #d4d4d4;
}

/*when navigating through the items using the arrow keys:*/
.autocomplete-active {
background-color: DodgerBlue !important;
color: #ffffff; }
```

\app\templates\allergen.html – objective 6

```
{% extends 'base.html' %}


{% block content %}
    <h1>Allergen addition</h1>
  {% if (current_user.is_authenticated) and (isP==False) and (current_user.accessLev < 3) %}
    <form method="POST">
      <label for="allergenName">
        <input type="text" id="allergenName" name="allergenName" required pattern="\S(.*\S)?">
      </label>
      <input type="submit" value="search">
        <br><div label="a">
          {% if indb == True %}
            {{allergen}} is already present in database.
          {% elif indb == False %}
            {% if allergen %}
              {{allergen}} has been added to database.
            {% endif %}
          {% endif %}
        </div>
    </form>
  {% else %}
    <p>404 error: page not found.</p>
  {% endif %}


{% endblock %}
```

\app\templates\allergy.html – objective 6

```
{% extends 'base.html' %}


{% block content %}
   <h1>Create patient allergy</h1>
   {% if (current_user.is_authenticated) and (isP==False) and (current_user.accessLev < 3) %}
      <p>
         For <b>{{fn}} {{sn}}: ({{NHSNum}})</b>
      </p>
      <form action="" method="POST" autocomplete="off">
         {{ form.hidden_tag() }}
         <div class="autocomplete">
            {{ form.allergy.label }} {{ form.allergy(id="myInput",placeholder="Allergy") }}
         </div>
         <div><br>
            {{ form.severity.label }}
            {{ form.severity }}
         </div>
         <br>
         <p><b>Note: severity 1 is the highest severity allergy while severity 5 is the
mildest.</b></p>
         {{ form.submit }}
      </form>
      <script>
         $.ajax({
            url: "/api/allergens"
         }).done(function(res) {
            const allergies = res.allergens
            console.log(allergies)
            autocomplete(document.getElementById("myInput"), allergies)
         })
```

Example of using AJAX to get a list of allergens for the autocomplete

```
        </script>
    {% else %}
        <p>404 error: page not found.</p>
    {% endif %}

{% endblock %}
```

\app\templates\appointment_viewer.html – objective 4

```
{% extends 'base.html' %}


{% block content %}

<h1>Appointment {{id}}</h1>


{% if current_user.is_authenticated %}

{% if message %}

<p><b>{{message}}</b></p>

<p>Are you sure you want to book the appointment? </p>

<div>

   <b>Date:</b> {{item.appdate}} --- <b>Time:</b> {{item.apptime}} --- with <b>{{fn}} {{sn}}</b>

</div>

<form action="" method="post">

   Room: <input type="text" label="approom" id="approom">

   <br><input type="submit"><b><input type="submit" value="No, I won't book" id="reject" label="reject"></b>

</form>

{% else %}

      {% if post_app==True %}

         <p>Your appointment was with <b>Dr. {{ fn }} {{ sn }}</b> on <b>{{ item.appdate }}, {{ item.apptime }}.</b></p>

         <br><p><b>What your doctor said about it</b></p>

         <p>{{item.patient_feedback}}</p>

         {% if isP==False %}

            <p><b>Doctor notes</b></p>

            <p>{{item.doctorNotes}}</p>

         {% endif %}

         {% for item in linked_test %}

         <div>Test type <b>{{item.testType}}</b> referred to on <b>{{item.testDate}}.</b>

         <br><b>Doctor comments:</b><p>{{item.testDoctorNotes}}</p>
```

```
        </div><br>
        {% endfor %}
    {% elif ((isP==False) and (item.appdate<=today and item.room)) %}
        <p>{{ fn }} {{ sn }} ({{item.NHSNumber}}) at {{ item.appdate }}, <b>{{ item.apptime }}.</b></p>
          <form action="" method="post">
            {{ followUp.hidden_tag() }}
            <p>
              {{ followUp.fulfilled.label }} {{ followUp.fulfilled }}<br>
            </p>
            <p>
              {{ followUp.patient_feedback.label }}<br>
              {{ followUp.patient_feedback(placeholder='Enter your patient feedback here.') }}
            </p>
            <p>
              {{ followUp.doctor_notes.label }}<br>
              {{ followUp.doctor_notes(placeholder=("Enter notes for the surgery's records here.")) }}
            </p>
            <p>{{followUp.submit}}</p>
          </form>
        <!--label for notes-->
        <p>Did a test occur this appointment? Add the test info <a href="{{ url_for('createTest', appoint=item.appointment_id ) }}">here.</a></p>
        <p>Add any illness diagnoses <a href="">here.</a></p>
        <!--label for feedback -->
    {% elif ((isP==False) or current_user.NHSNumber==item.NHSNumber) %}
        <p><b>{{ fn }} {{ sn }} ({{item.NHSNumber}})</b></p>
        <p><b>REQUESTED DATE AND TIME: {{ item.appdate }} {{ item.apptime }}.</b></p>
        <form action="" method="post">
```

```
            {{ form.hidden_tag() }}
            <p>
               {{ form.appdate.label }}<br>
               {{ form.appdate(value=item.appdate) }}<br>
            </p>
            {% for errors in form.appdate.errors %}
            <span style="color: red;">Appointment cannot be booked in the past.</span>
            {% endfor %}
            <p>
               {{ form.apptime.label }}<br>
               {{ form.apptime(value=item.apptime,size=12) }}<br>
            </p>
            <p>
               {{ form.approom.label }}<br>
               {{ form.approom(size=1) }}
            </p>
            {{ form.submit() }}
            <p><a href="{{url_for('cancelApp', id=item.appointment_id)}}">Cancel
appointment</a></p>
         </form>
      {% endif %}
{% endif %}
{% else %}<p><B>Login please</B></p>
{% endif %}
{% endblock %}
```

\app\templates\appointments.html – objective 4

```
{% extends 'base.html' %}


{% block content %}
   {% if current_user.is_authenticated %}
      {% if isP==False %}<!-- doctor section -->
         <h2>Book patient appointment</h2>
            <form action="" method="post">
               {{ doctorForm.hidden_tag() }}
               <p>
                  {{ doctorForm.patient.label }}
                  {{ doctorForm.patient }}
               </p>
                  {{ doctorForm.appdate.label }}
                  {{ doctorForm.appdate }}
               <p>
                  {{ doctorForm.apptime.label }}
                  {{ doctorForm.apptime() }}
               </p>
               <p>
                  {{ doctorForm.approom.label }}
                  {{ doctorForm.approom }}
               </p>
               <p>
                  {{ doctorForm.submit }}
               </p>
            </form>


         <h2>Requested appointments</h2>
         {% for item in queue1 %}
            <a href="{{ url_for('viewApp', id=item.appointment_id ) }}"><p class="dotted">
```

```
            {{ (pat.query.filter_by(NHSNumber=item.NHSNumber).first()).forename }}
            {{ (pat.query.filter_by(NHSNumber=item.NHSNumber).first()).surname }}:
requested
            {{ item.appdate.strftime('%d-%m-%Y')}} at {{
item.apptime.strftime('%H:%M')}} {{ form.hidden_tag() }}
          </p></a>
        {% endfor %}
        <h2>Appointments in the past you haven't added notes for</h2>
        {% if queue2 %}
          {% for item in queue2 %}
            <a href="{{ url_for('viewApp', id=item.appointment_id ) }}"><p class="dotted">
              {{ (pat.query.filter_by(NHSNumber=item.NHSNumber).first()).forename }}
              {{ (pat.query.filter_by(NHSNumber=item.NHSNumber).first()).surname }}:
              {{ item.appdate.strftime('%d-%m-%Y')}} at {{
item.apptime.strftime('%H:%M')}}
              with {{ (doc.query.filter_by(doctor_id=item.doctor_id).first()) }}
              {{ form.hidden_tag() }}
            </p></a>
          {% endfor %}
        {% else %}
          <p>You're all caught up!</p>
        {% endif %}
        <h2>Your upcoming confirmed appointments:</h2>
        {% if queue3 %}
        {% for item in queue3 %}
          <a href="{{ url_for('viewApp', id=item.appointment_id ) }}"><p class="dotted">
            {{ (pat.query.filter_by(NHSNumber=item.NHSNumber).first()).forename }}
            {{ (pat.query.filter_by(NHSNumber=item.NHSNumber).first()).surname }}:
            {{ item.appdate.strftime('%d-%m-%Y')}} at {{
item.apptime.strftime('%H:%M')}}
            in room {{ item.room }} {{ form.hidden_tag() }}
          </p></a>
```

```
    {% endfor %}
    {% else %}
    <p>No upcoming appointments.</p>{%endif%}
  {% else %}<!--Patient section-->
    {% if appointment %}
      <h1>Your next appointment</h1>
      <p>
        {{ appointment.appdate }}
        {{ appointment.apptime }}
        in room {{ appointment.room}}
      </p>
    {% else %}
      <h1>Book appointment</h1>
      <form action="" method="post">
        {{ form.hidden_tag() }}
        <p>
          {{ form.appdate.label }}<br>
          {{ form.appdate }}<br>
        </p>
        <p>
          {{ form.apptime.label }}<br>
          {{ form.apptime }}<br>
        </p>
        <p>
          {% for error in form.appdate.errors %}
            <span style="color: red;">Error: {{ error }}</span>
          {% endfor %}
        </p>
        <p>{{ form.submit() }}</p>
      </form>
```

```
        {% endif %}
        {% if prev_queue %}
          <h1>Previous appointments</h1>
          {% for item in prev_queue %}
          <a href="{{ url_for('viewApp', id=item.appointment_id ) }}"><p class="dotted">
            With Dr. {{ (doc.query.filter_by(doctor_id=item.doctor_id).first()).forename }}
            {{ (doc.query.filter_by(doctor_id=item.doctor_id).first()).surname }}:
            {{ item.appdate.strftime('%d-%m-%Y')}} at {{ item.apptime.strftime('%H:%M')}}
            in room {{ item.room }} {{ form.hidden_tag() }}
          </p></a>
        {% endfor %}
        {% endif %}
      {% endif %}
    {% else %}
      <p>Log in to book an appointment.</p>
    {% endif %}
{% endblock %}
```

\app\templates\base.html

Base template for every webpage

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    {% if title %}
      <title>{{ title }}</title>
    {% else %}
      <title>Anthony Nkyi project</title>
    {% endif %}
    <div class="topnav">
      <a href="{{ url_for('index') }}">Home</a>
      {% if current_user.is_anonymous %}
        <a href="{{ url_for('login') }}">Login</a>
        <a href="{{ url_for('registerDoc') }}">Doctor registration</a>
        <a href="{{ url_for('registerPatient') }}">Patient registration</a>
      {% else %}
        <a href="{{ url_for('updateProfile') }}">Update profile</a>
        <a href="{{ url_for('createApp') }}">Appointments</a>
        {% if isP==False %}
          <a href="{{ url_for('viewProfile', user_id=current_user.doctor_id) }}">Profile</a>
          <a href="{{ url_for('createPres') }}">Prescriptions</a>
          <a href="{{ url_for('createAllergen') }}">Allergens</a>
          <a href="{{ url_for('createMedicine') }}">Medicine</a>
          <a href="{{ url_for('createMedAllergy') }}">Ingred</a>
          <a href="{{ url_for('createDisease') }}">Disease</a>
          <a href="{{ url_for('search')}}">Search user</a>
        {% else %}
```

```
            <a href="{{ url_for('viewProfile', user_id=current_user.NHSNumber) }}">Profile</a>
            {% endif %}
            <a href="{{ url_for('logout') }}">Logout</a>
        {% endif %}
    </div>
    <hr>
    <link rel="stylesheet" href="../static/style.css">
    <!-- autocomplete is sourced from https://www.w3schools.com/howto/howto_js_autocomplete.asp -->
    <script>
        function autocomplete(inp, arr) {
            /*the autocomplete function takes two arguments,
            the text field element and an array of possible autocompleted values:*/
            var currentFocus;
            /*execute a function when someone writes in the text field:*/
            inp.addEventListener("input", function(e) {
                var a, b, i, val = this.value;
                /*close any already open lists of autocompleted values*/
                closeAllLists();
                if (!val) { return false;}
                currentFocus = -1;
                /*create a DIV element that will contain the items (values):*/
                a = document.createElement("DIV");
                a.setAttribute("id", this.id + "autocomplete-list");
                a.setAttribute("class", "autocomplete-items");
                /*append the DIV element as a child of the autocomplete container:*/
                this.parentNode.appendChild(a);
                /*for each item in the array...*/
                for (i = 0; i < arr.length; i++) {
                  /*check if the item starts with the same letters as the text field value:*/
```

Autocomplete function source

```
        if (arr[i].substr(0, val.length).toUpperCase() == val.toUpperCase()) {
          /*create a DIV element for each matching element:*/
          b = document.createElement("DIV");
          /*make the matching letters bold:*/
          b.innerHTML = "<strong>" + arr[i].substr(0, val.length) + "</strong>";
          b.innerHTML += arr[i].substr(val.length);
          /*insert a input field that will hold the current array item's value:*/
          b.innerHTML += "<input type='hidden' value='" + arr[i] + "'>";
          /*execute a function when someone clicks on the item value (DIV element):*/
              b.addEventListener("click", function(e) {
              /*insert the value for the autocomplete text field:*/
              inp.value = this.getElementsByTagName("input")[0].value;
              /*close the list of autocompleted values,
              (or any other open lists of autocompleted values:*/
              closeAllLists();
          });
          a.appendChild(b);
        }
      }
    });
    /*execute a function presses a key on the keyboard:*/
    inp.addEventListener("keydown", function(e) {
      var x = document.getElementById(this.id + "autocomplete-list");
      if (x) x = x.getElementsByTagName("div");
      if (e.keyCode == 40) {
        /*If the arrow DOWN key is pressed,
        increase the currentFocus variable:*/
        currentFocus++;
        /*and and make the current item more visible:*/
        addActive(x);
```

```
    } else if (e.keyCode == 38) { //up
      /*If the arrow UP key is pressed,
      decrease the currentFocus variable:*/
      currentFocus--;
      /*and and make the current item more visible:*/
      addActive(x);
    } else if (e.keyCode == 13) {
      /*If the ENTER key is pressed, prevent the form from being submitted,*/
      e.preventDefault();
      if (currentFocus > -1) {
        /*and simulate a click on the "active" item:*/
        if (x) x[currentFocus].click();
      }
    }
});
function addActive(x) {
  /*a function to classify an item as "active":*/
  if (!x) return false;
  /*start by removing the "active" class on all items:*/
  removeActive(x);
  if (currentFocus >= x.length) currentFocus = 0;
  if (currentFocus < 0) currentFocus = (x.length - 1);
  /*add class "autocomplete-active":*/
  x[currentFocus].classList.add("autocomplete-active");
}
function removeActive(x) {
  /*a function to remove the "active" class from all autocomplete items:*/
  for (var i = 0; i < x.length; i++) {
    x[i].classList.remove("autocomplete-active");
  }
```

```
        }
        function closeAllLists(elmnt) {
          /*close all autocomplete lists in the document,
          except the one passed as an argument:*/
          var x = document.getElementsByClassName("autocomplete-items");
          for (var i = 0; i < x.length; i++) {
            if (elmnt != x[i] && elmnt != inp) {
            x[i].parentNode.removeChild(x[i]);
            }
          }
        }
        /*execute a function when someone clicks in the document:*/
        document.addEventListener("click", function (e) {
            closeAllLists(e.target);
        });
        }
    </script>
  </head>
  <body><!-- displays messages from redirects -->
    {% with messages = get_flashed_messages () %}
    {% if messages %}
    <ul>
      {% for message in messages %}<li><b>{{ message }}</b></li>{% endfor %}
    </ul>
    {% endif %}
    {% endwith %}
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
    <!-- content using the template -->
    {% block content %}{% endblock %}
  </body></html>
```

\app\templates\carrier.html – objective 8

```
{% extends 'base.html' %}


{% block content %}
{% if (current_user.is_authenticated) and (isP==False) %}
   <script>
      $.ajax({
         url: "/api/carrier"
      }).done(function(res) {
         const carrier = res.carrier
         console.log(carrier)
         autocomplete(document.getElementById("carrier"), carrier)
      })
   </script>
   <h1>Inheritable disease carrier addition</h1>
   <b>For patient {{patient.forename}} {{patient.surname}} ({{NHSNum}})</b>


      <form action=" method="POST">
         <div class="autocomplete">
            Disease <input type="string" name="disease_name" id="carrier"
placeholder="Disease name">
         </div>
         <div>
            <input type="submit">
         </div>
         <div label="a">
            {% if indb == True %}
               This inherited disease is already recorded, or the disease itself has not been
documented yet.
            {% elif indb == False %}
               <b>User {{NHSNum}}'s {{disease}} trait</b> has been added to database.
```

```
            {% endif %}
          </div>
        </form>
    {% else %}
        <p>404 error: page not found.</p>
    {% endif %}

{% endblock %}
```

\app\templates\disease_add.html – objective 8

```
{% extends 'base.html' %}

{% block content %}
        <h1>Disease addition</h1>
    {% if (current_user.is_authenticated) and (isP==False) and (current_user.accessLev < 2) %}
        <form method="POST">
            <div>
                Disease <input type="text" name="disease_name" placeholder="Disease name">
            </div><br>
            <div>
                Is the disease inheritable? <input type="checkbox" name="ch1">
            </div><br>
            <div><input type="submit"></div>
            <div label="a">
                {% if indb == True %}
                    <b>{{disease}}</b> is already in database.
                {% elif indb == False %}
                    <b>{{disease}}</b> has been added to database.
                {% endif %}
            </div></form>
    {% else %}<p>404 error: page not found.</p> {% endif %} {% endblock %}
```

\app\templates\hist_viewer.html – objective 9

```
{% extends 'base.html' %}


{% block content %}
   <h1>View illness history</h1>
   {% if (allowed==True or (illness_hist.NHSNumber==current_user.NHSNumber)) %}
      <p><b>Illness for patient {{fn}} {{sn}} ({{illness_hist.NHSNumber}})</b></p>
      <p><b>Info:</b> {{illness_hist}} </p>
      {% if illness_hist.endDate %}
      <p><b>End date:</b> {{illness_hist.endDate}}</p>
      {% endif %}
      <p><b>Doctor notes:</b> {{illness_hist.notes}} </p>
   {% else %}
      <p>404 error. Page not found.</p>
   {% endif %}
{% endblock %}
```

\app\templates\illness.html – objective 9

```
{% extends 'base.html' %}


{% block content %}
<script>
   $.ajax({
      url: "/api/disease"
   }).done(function(res) {
      const disease = res.disease
      console.log(disease)
      autocomplete(document.getElementById("disease"), disease)
   })
</script>
   {% if isP==False %}
      <h1>Illness history addition</h1>
      <p><B><A href="{{url_for('createRef', NHSNum=NHSNum)}}">Add referral</A></B></p>
      <p>For patient <b>{{fn}} {{sn}} ({{NHSNum}})</b></p>
      <form action="" method="POST">
         {{form.hidden_tag()}}
         <div class="autocomplete">
            {{form.disease.label}} {{form.disease(placeholder="Disease name",id="disease")}}
         </div>
         <p>
            {{form.diagDate.label}} {{form.diagDate}}
         </p>
         {% for error in form.diagDate.errors %}
         <span style="color: red;">{{ error }}</span>
         {% endfor %}
         <p>
```

```
            {{form.endDate.label}} {{form.endDate}}
          </p>
          <p>
            {{form.notes(placeholder='Illness notes')}}
          </p>
          <p>
            {{form.submit()}}
          </p>
        </form>
    {% endif %}
    {% if allowed==True %}
        {% for item in prev_ill %}
          <div>Diagnosed with <b>{{d.query.filter_by(disease_id=item.disease_id).first().diseaseName}}</b> on <b>{{item.diagnosisDate}}.</b>
          {% if item.endDate %}
          <br>Confirmed free of illness on <b>{{item.endDate}}.</b>
          {% endif %}
          <br><b>Doctor comments:</b><p>{{item.notes}}</p>
          </div><br>
        {% endfor %}
    {% endif %}
{% endblock %}
```

\app\templates\index.html – objective 3

```
{% extends "base.html" %}


{% block content %}
   {% if current_user.is_authenticated %}
      <h1>Hello, {{ current_user.forename }}!</h1>
      {% if (isP==True) %}
         <p><b>Your NHS Number:</b> {{current_user.NHSNumber}}</p>
         <p><b>Your date of birth:</b> {{current_user.dob}} </p>
         {% if nextapp %}
            <h3><b>Your next appointment: </b></h3>
            {% if nextapp.room %}
               <p><b>Confirmed:</b> {{nextapp.apptime}} on {{nextapp.appdate}} in room {{nextapp.room}}.</p>
               {% if nextapp.appdate!=today or nextapp.appdate<today %}
                  <p><a href="{{url_for('cancelApp', id=nextapp.appointment_id)}}"><b>Cancel your appointment</b></a></p>
               {% endif %}
            {% else %}
               <p><b>You have requested:</b> {{nextapp.apptime}} on {{nextapp.appdate}}.</p>
            {% endif %}
         {% else %}
            <p><b>Book an <a href="{{ url_for('createApp') }}">appointment</a></b></p>
         {% endif %}


         <h3>Current prescriptions:</h3>
         {% for pres in current_pres %}
            <a href="{{ url_for('viewPres', id=pres.prescription_id )}}"<p><b>{{ Med.query.filter_by(med_id=pres.med_id).first().medicineName }}</b> since {{pres.startDate}}.</p>
         {% endfor %}
```

Objective 3.4

Objective 3.2

Objective 3.5


```
    {% else %}
      {% if app_no_notes %}
        <p><b>Appointments with no notes: </b></p>
        {% for appointment in app_no_notes %}
          <a href="{{ url_for('viewApp', id=appointment[0] ) }}"><p>With
{{(Pat.query.filter_by(NHSNumber=appointment[2]).first()).forename}}
{{(Pat.query.filter_by(NHSNumber=appointment[2]).first()).surname}} on
{{appointment[7]}} at {{appointment[8]}}</p></a>
        {% endfor %}
      {% endif %}
      <p><b>Your upcoming appointments: </b></p>
      {% if nextapp %}
        {% for appointment in nextapp %}
          <a href="{{ url_for('viewApp', id=appointment.appointment_id ) }}"><p>With
{{(Pat.query.filter_by(NHSNumber=appointment.NHSNumber).first()).forename}}
{{(Pat.query.filter_by(NHSNumber=appointment.NHSNumber).first()).surname}} on
{{appointment.appdate}} at {{appointment.apptime}} in room
{{appointment.room}}.</p></a>
        {% endfor %}
      {% endif %}
    {% endif %}
  {% else %}
    <h1>Welcome visitor!</h1>
    <p>Login or register to access the surgery's application.</p>
  {% endif %}
{% endblock %}
```

\app\templates\ingred.html – objective 5

```
{% extends 'base.html' %}


{% block content %}
   <h1>Link medicine to allergen</h1>
   {% if (current_user.is_authenticated) and (current_user.accessLev<3) %}
      <!-- autocomplete is sourced from https://www.w3schools.com/howto/howto_js_autocomplete.asp -->
      <style>
         * {
         box-sizing: border-box;
         }

         /*the container must be positioned relative:*/
         .autocomplete {
         position: relative;
         display: inline-block;
         }

         input[type=text] {
         background-color: #f1f1f1;
         width: 100%;
         }

         input[type=submit] {
         background-color: DodgerBlue;
         color: #fff;
         cursor: pointer;
         }

         .autocomplete-items {
```

```
        position: absolute;
        border: 1px solid #d4d4d4;
        border-bottom: none;
        border-top: none;
        z-index: 99;
        /*position the autocomplete items to be the same width as the container:*/
        top: 100%;
        left: 0;
        right: 0;
        }

        .autocomplete-items div {
        cursor: pointer;
        background-color: #fff;
        border-bottom: 1px solid #d4d4d4;
        }

        /*when hovering an item:*/
        .autocomplete-items div:hover {
        background-color: #e9e9e9;
        }

        /*when navigating through the items using the arrow keys:*/
        .autocomplete-active {
        background-color: DodgerBlue !important;
        color: #ffffff;
        }
    </style>

    <form action="" method="POST" autocomplete="off">
```

```
        {{ form.hidden_tag() }}
        <div class="autocomplete">
            <p>
                {{ form.medicine.label }}
                {{ form.medicine(id='med',placeholder='medicine',size=24) }}
            </p>
        </div>
        <div class="autocomplete">
            {{ form.allergen.label }} {{ form.allergen(id="all",placeholder="allergen") }}
        </div>
        <div label="a"><b>
            {% if indb == True %}
                This {{medicine}} allergy is already in database.
            {% elif indb == False %}
                {% if medicine %}
                    {{medicine}} allergy has been added to database.
                {% endif %}
            {% endif %}
        </div></b>
        {{ form.submit }}
    </form>
    <script>
        $.ajax({
            url: "/api/medicine"
        }).done(function(res) {
            const medicine = res.medicine
            console.log(medicine)
            autocomplete(document.getElementById("med"), medicine)
        })
        $.ajax({
```

```
            url: "/api/allergens"
        }).done(function(res) {
            const allergies = res.allergens
            console.log(allergies)
            autocomplete(document.getElementById("all"), allergies)
        })


    </script>
{% else %}
    <p>404 error: page not found.</p>
{% endif %}

{% endblock %}
```

\app\templates\loginDoc.html – objective 1

```
{% extends "base.html" %}


{% block content %}
<div class="container"><h2 class="form-signin-heading">Login</h2>
   <div class="input-group" style="margin:auto;">
   <form action="" method="post" novalidate>
      {{ form.hidden_tag() }}
      <p>
         {{ form.email(placeholder="Email") }}<br>
         {% for errors in form.email.errors %}
         <span style="color: red;">Email must be between 8 and 64 characters.</span>
         {% endfor %}
      </p>
      <p>
         {{ form.password(placeholder="Password") }}<br>
         {% for errors in form.password.errors %}
         <span style="color: red;">Password must be between 8 and 80 characters.</span>
         {% endfor %}
      </p>
      <p>{{ form.remember.label }} {{ form.remember() }}</p>
      <p>{{ form.submit() }}</p>
   </form></div></div>
   <p>New User? Click to register<a href="{{ url_for('registerDoc') }}"> as a doctor</a> or
<a href="{{ url_for('registerPatient') }}`"> as a patient</a>.</p>
   <p>
      Forgot Your Password?
      <a href="{{ url_for('resetPasswordRequest') }}">Click to Reset It</a>
   </p>
{% endblock %}
```

\app\templates\medicine.html – objective 5

```
{% extends 'base.html' %}


{% block content %}
   <script>
      function medicineCheck() {
         $.ajax({
            url: "/api/medicine"
         }).done(function(res) {
            const medicine = res.medicine
            var med = document.getElementById("medicineName").value;
            var para = document.getElementById("db_check")
            var button = document.getElementById("but")
            if (medicine.includes(med)) {
               var para = document.getElementById("in_db")
               para.textContent += "Item is already present in database";
               window.location.href = "/medicine/"+med
            } else {
                  para.style.display = 'block'
                  button.style.display = 'none'
            }
         })
      }
   </script>
   <h1>Medicine addition</h1>
   {% if (current_user.is_authenticated) and (isP==False) and (current_user.accessLev == 1)
%}
      <form method="POST">
         <div>
            <label for="medicineName">
```

```
            Name: <input required pattern="\S(.*\S)?" type="text" id="medicineName" name="medicineName" placeholder="Medicine" maxlength="64" required>
          </label>
        </div><br>

        <b><div id="in_db"></div></b>

        <div id="db_check" style="display:none">
          <Div>
            <label for="dosage">
              Recommended dosage: <input required pattern="\S(.*\S)?" type="text" id="dosage" name="dosage" maxlength="128" placeholder="Dosage">
            </label>
          </Div>
          <br><input type="submit">
        </div>

        <div id="but" style="display:block">
          <br><button onclick="medicineCheck()">Submit</button>
        </div>
      </form>
    {% else %}
      <p>404 error: page not found.</p>
    {% endif %}

{% endblock %}
```

\app\templates\medicine_viewer.html – objective 5

```
{% extends 'base.html' %}
{% block content %}
   {% if (isP==False) and (current_user.accessLev < 3) %}
      <h1><b>Medicine: {{name}}</b></h1>
      <h2><B>Known allergens in {{name}}:</B></h2>
      <h3><a href="{{ url_for('createMedAllergy') }}">Add new allergen</a></h3>
      {% for item in allergies %}
         <h3><li>{{item}}</li></h3>
      {% endfor %}
      <h2>Patients currently prescribed:</h2>
   <!--url to add allergen / new med-->
   {% else %}
      <p>404 error</p>
   {% endif %}{% endblock %}
```

\app\templates\prescription_viewer.html – objective 10

```
{% extends 'base.html' %}
{% block content %}
   <h1>View prescription</h1>
   {% if ((current_user.NHSNumber==pres.NHSNumber) or (isP==False)) %}
      <p><b>Prescription id{{pres.prescription_id}}</b></p>
      <p><b>Medicine:</b> {{medName}}</p>
      <p><b>Instructions:</b> {{pres.dose}} </p>
      <p><b>Prescribed for:</b> {{disName}} </p>
      <p><b>Start date:</b> {{pres.startDate}} </p>
      {% if pres.finalRepeatDate %}
         <p><b>Prescription ended:</b> {{pres.finalRepeatDate}} </p>
      {% endif %}
      <p><b>Doctor:</b> {{docName}}</p>
   {% else %}{% endif %}{% endblock %}
```

\app\templates\prescription.html – objective 10

```
{% extends 'base.html' %}


{% block content %}
   <script>
      $.ajax({
         url: "/api/medicine"
      }).done(function(res) {
         const medicine = res.medicine
         console.log(medicine)
         autocomplete(document.getElementById("medicine"), medicine)
      })
      $.ajax({
         url: "/api/disease"
      }).done(function(res) {
         const disease = res.disease
         console.log(disease)
         autocomplete(document.getElementById("disease"), disease)
      })
   </script>
   <h1>Prescription addition</h1>
   <form action="" method="post">
      {{ form.hidden_tag() }}
      <p>
         <div>
            {{form.patient.label}} {{ form.patient }}
         </div>
         <div class="autocomplete">
            {{form.medicine.label}}                                                {{
form.medicine(id="medicine",placeholder="Medicine") }}
         </div>
```

```
        <div>
            {{form.dosage.label}} {{ form.dosage }}
        </div>
        <div class="autocomplete">
            {{form.disease.label}} {{ form.disease(id="disease",placeholder="Disease") }}
        </div>
        <div>
            {{ form.submit() }}
        </div>
      </p>
   </form>
{% endblock %}
```

\app\templates\profile.html

```
{% extends "base.html" %}

{% block content %}

{% if current_user.is_authenticated %}

  <h1>{{ account.forename }} {{ account.surname }}: Profile</h1>

  {% if (isP==False and isP_prof==True) %}

  <a href="{{ url_for('createIllness', NHSNum=account.NHSNumber) }}">Add illness history</a> - <a href="{{ url_for('createApp') }}">Create appointment</a> - <a href="{{ url_for('createPatientAllergy', NHSNum=account.NHSNumber) }}">Create patient allergy</a> - <a href="{{ url_for('createInherit', NHSNum=account.NHSNumber) }}">Record inherited disease</a>

  {% endif %}

  <p><b>Email:</b> {{account.email}}</p>

  {% if isP_prof==False %}

    <p><b>NI Number:</b> {{ number }} </p>

    <p><b>Access level:</b> {{ current_user.accessLev }}</p>

    <p><b>Gender:</b> {{ current_user.gender }}</p>

  {% else %}

    <p><b>NHS Number:</b> {{ account.NHSNumber }}</p>

    <p><b>Date of birth:</b> {{ account.dob }}</p>

    <p><b>Sex:</b> {{ account.sex }}</p>

    <p><b>Last measured height:</b> {{ account.height }} cm</p>

    <p><b>Last measured weight:</b> {{ account.weight }} kg</p>

    <p><b>Blood type:</b> {{ account.bloodType }}</p>

  {% endif %}

  <p><b>Address:</b> {{ houseNo }} {{ streetName }}, {{ postcode[:spl] }} {{ postcode[spl:]}}</p>


  {% if all_pres.queue %}

  <p><B>Prescriptions:</B></p>

    {% for item in all_pres.queue %}

      <div>
```

```
            <a href="{{ url_for('viewPres', id=item.prescription_id )}}">{{item}}</a><br>
        </div>
    {% endfor %}
{% elif isP_prof==True %}
    <p><b>No current prescriptions.</b></p>
{% endif %}
{% if allergies.queue %}
    <div>
        <p><b>Known allergies: </b>{% for item in allergies.queue %} {{item}} {% endfor %}</p>
    </div>
{% elif isP_prof==True %}
    <p><b>No known allergies.</b></p>
{% endif %}
{% if (ill_hist.queue) %}
<b>Illness history:</b><br>
    {% for item in ill_hist.queue %}
    <div>
        <a href="{{ url_for('viewHist', id=item.unhealthy_id )}}">{{item}}</a><br>
    </div>
    {% endfor %}<br>
{% elif isP_prof==True %}
    <p><b>No illness history recorded.</b></p>
{% endif %}
{% if inherited.queue %}
<div>
    <p><b>Known inherited diseases: </b>{% for item in inherited.queue %} {{item}} {% endfor %}</p>
</div>
{% elif isP_prof==True %}
    <p><b>No known inherited diseases.</b></p>
```

Objective 6.1

Objective 8.1, 9.1

```
    {% endif %}<br>
{% else %}
    <h1 style="font-family:'Times New Roman', Times, serif">You must be logged in to view this page!</h1>
{% endif %}
{% endblock %}
```

\app\templates\ref_viewer.html – objective 9

```
{% extends 'base.html' %}

{% block content %}
    <h1>View referral</h1>
    {% if (allowed==True or (ref.NHSNumber=current_user.NHSNumber)) %}
        <p><b>Referral id {{id}} for {{ref.NHSNumber}}</b></p>
        <p><b>Info:</b> {{ref}} </p>
        <p><b>Doctor notes:</b> {{ref.notes}} </p>
    {% else %}
        <p>404 error. Page not found.</p>
    {% endif %}
{% endblock %}
```

\app\templates\referral.html – objective 9

```
{% extends "base.html" %}


{% block content %}
   <script>
      $.ajax({
         url: "/api/referral_type"
      }).done(function(res) {
         const referral_type = res.referral_type
         console.log(referral_type)
         autocomplete(document.getElementById("referral_type"), referral_type)
      })
      $.ajax({
         url: "/api/disease"
      }).done(function(res) {
         const disease = res.disease
         console.log(disease)
         autocomplete(document.getElementById("disease"), disease)
      })
   </script>
   <h1>Referral addition</h1>
{% if not isP %}
   <form action="" method="post">
      {{ form.hidden_tag() }}
      <p>
         <div class="autocomplete">
            {{form.refType.label}}  {{  form.refType(id="referral_type",placeholder="Referral
type") }}
         </div>
         <div>
            {{form.refDate.label}} {{ form.refDate }}
```

```
            </div>
            <div class="autocomplete">
               {{form.disease.label}} {{ form.disease(id="disease",placeholder="Disease") }}
            </div>
            <div>
               {{form.ref_notes.label}} {{ form.ref_notes }}
            </div>
            <div>
               {{ form.submit() }}
            </div>
         </p>
      </form>
      {% for item in prev_ill %}
         <div>Type <b>{{item.type}}</b> referred to on <b>{{item.diagnosisDate}}.</b>
         <br><b>Doctor comments:</b><p>{{item.notes}}</p>
         </div><br>
      {% endfor %}
{% endif %}
{% endblock %}
```

\app\templates\registerDoc.html – objective 1

```
{% extends "base.html" %}


{% block content %}
   <h1>Doctor registration</h1>
   <form action="" method="post">
      {{ form.hidden_tag() }}
      {{ form.email(size=32, placeholder='Email') }}<br>
      {% for error in form.email.errors %}
      <span style="color: red;">[{{ error }}]</span>
      {% endfor %}
      <p>
         {{ form.accessLev.label }}
         {{ form.accessLev }}
      </p>
      <p>
         {{ form.NINumber(size=32, placeholder='NI Number') }}<br>
         {% for error in form.NINumber.errors %}
         <span style="color: red;">{{ error }}</span>
         {% endfor %}
      </p>
      <p>
         {{ form.forename(size=32, placeholder='Forename') }}
      </p>
      <p>
         {{ form.surname(size=32, placeholder='Surname') }}
      </p>
      <p>
         {{ form.gender.label }}
         {{ form.gender }}
      </p>
```

```
    <p>
        {{ form.dob.label }}
        {{ form.dob(size=32) }}<br>
    </p>
    <p>
        {{ form.houseNo.label }}
        {{ form.houseNo(size=32) }}<br>
    </p>
    <p>
        {{ form.streetName.label }}
        {{ form.streetName(size=32) }}<br>
    </p>
    <p>
        {{ form.postcode.label }} - NO SPACES IN POSTCODE!
        {{ form.postcode(size=8) }}<br>
    </p>
    <p>
        {{ form.password(size=32, placeholder='Password') }}<br>
        {% for error in form.password.errors %}
        <span style="color: red;">[{{ error }}]</span>
        {% endfor %}
    </p>
    <p>
        {{ form.password2(size=32, placeholder='Repeat password') }}<br>
        {% for error in form.password2.errors %}
        <span style="color: red;">[{{ error }}]</span>
        {% endfor %}
    </p>
    <p>{{ form.submit() }}</p>
</form>{% endblock %}
```

\app\templates\registerPatient.html – objective 1

```
{% extends 'base.html' %}
{% block content %}
  <h1>Patient registration</h1>
  <form action="" method="post">
    {{ form.hidden_tag() }}
    {{ form.NHSNumber(size=32,placeholder='NHS Number') }}<br>
    {% for error in form.NHSNumber.errors %}
    <span style="color: red;">{{ error }}</span>
    {% endfor %}
    <p>
      {{ form.dob.label }}
      {{ form.dob(size=32) }}<br>
    </p>
    <p>
      {{ form.birth_sex.label }}
      {{ form.birth_sex }}
    </p>
    <p>
      {{ form.forename(size=32,placeholder='Forename') }}
    </p>
    <p>
      {{ form.surname(size=32,placeholder='Surname') }}
    </p>
    <p>
      {{ form.email(size=32, placeholder='Email') }}<br>
      {% for error in form.email.errors %}
        <span style="color: red;">[{{ error }}]</span>
      {% endfor %}
    </p>
    <p>
```

Registration page – objective 1.1

```
        {{ form.weight.label }}
        {{ form.weight(size=2) }} kg
    </p>
    <p>
        {{ form.houseNo.label }}
        {{ form.houseNo(size=32) }}<br>
    </p>
    <p>
        {{ form.streetName.label }}
        {{ form.streetName(size=32) }}<br>
    </p>
    <p>
        {{ form.postcode.label }} - NO SPACES IN POSTCODE!
        {{ form.postcode(size=6) }}<br>
    </p>
    <p>
        {{ form.password(size=32,placeholder='Password') }}<br>
        {% for error in form.password.errors %}
        <span style="color: red;">[{{ error }}]</span>
        {% endfor %}
    </p>
    <p>
        {{ form.password2(size=32,placeholder='Repeat password') }}<br>
        {% for error in form.password2.errors %}
        <span style="color: red;">[{{ error }}]</span>
        {% endfor %}
    </p>
    <p>{{ form.submit() }}</p>
{% endblock %}
```

\app\templates\reset_password.html – objective 2

```
{% extends "base.html" %}


{% block content %}
    <h1>Reset Your Password</h1>
    <form action="" method="post">
        {{ form.hidden_tag() }}
        <p>
            {{ form.password.label }}<br>
            {{ form.password(size=32) }}<br>
            {% for error in form.password.errors %}
                <span style="color: red;">[{{ error }}]</span>
            {% endfor %}
        </p>
        <p>
            {{ form.password2.label }}<br>
            {{ form.password2(size=32) }}<br>
            {% for error in form.password2.errors %}
                <span style="color: red;">[{{ error }}]</span>
            {% endfor %}
        </p>
        <p>{{ form.submit() }}</p>
    </form>
{% endblock %}
```

\app\templates\reset_password_request.html – objective 2

```
{% extends "base.html" %}


{% block content %}
    <h1>Reset Password</h1>
    <form action="" method="post">
        {{ form.hidden_tag() }}
        <p>
            <b>Enter email:</b>
            {{ form.email(size=64) }}<br>
            {% for error in form.email.errors %}
                <span style="color: red;">[{{ error }}]</span>
            {% endfor %}
        </p>
        <p>{{ form.submit() }}</p>
    </form>
{% endblock %}
```

\app\templates\search.html

```
{% extends "base.html" %}
{% block content %}
{% if (isP==False) %}
    <h1>Search patient by NHS number</h1>
    <form action="" method="POST">
        {{form.hidden_tag()}}
        {{form.NHSNumber(placeholder='Patient NHS Number')}}
        {{form.submit()}}
        <p>{% for errors in form.NHSNumber.errors %}
            <span style="color: red;"><b>Patient does not exist.</b></span>
        {% endfor %}</p></form>
{% else %}
    <p>404 error: Page not found.</p>{% endif %}{% endblock %}
```

\app\templates\test.html – objective 4

```
{% extends "base.html" %}


{% block content %}
   <script>
      $.ajax({
         url: "/api/test_type"
      }).done(function(res) {
         const test_type = res.test_type
         console.log(test_type)
         autocomplete(document.getElementById("test_type"), test_type)
      })
   </script>
   <h1>Test addition</h1>
   <form action="" method="post">
      {{ form.hidden_tag() }}
      <p>
         <div>
            {% if fn %}
               For <b>{{fn}} {{sn}}</b> - test occurred on <b>{{day}}</b>.
               <p hidden>{{ form.patient(value=NHSNum) }}</p>
            {%else%}
               {{form.patient.label}} {{ form.patient }}
            {%endif%}
         </div><br>
         <div class="autocomplete">
            {{form.testType.label}} {{ form.testType(id="test_type",placeholder="Test type")
}}
         </div>
         <div>
            <p hidden>{{ form.testDate(value=day) }}</p>
```

```
        </div>
        <p>
            {{form.test_notes.label}} {{ form.test_notes }}
        </p>
        <div>
            {{ form.submit() }}
        </div>
      </p>
   </form>
{% endblock %}
```

\app\templates\test_viewer.html – objective 4

```
{% extends 'base.html' %}


{% block content %}
   <h1>View test</h1>
   {% if (allowed==True or (test.NHSNumber=current_user.NHSNumber)) %}
      <p><b>Test id {{id}} for {{test.NHSNumber}}</b></p>
      <p><b>Performed by:</b> Dr
{{doc.query.filter_by(doctor_id=test.doctor_id).first().forename}}
{{doc.query.filter_by(doctor_id=test.doctor_id).first().surname}}</p>
      <p><b>Took place on:</b> {{test.testDate}} </p>
      <p><b>Test type:</b> {{test.testType}} </p>
      <p><b>Doctor notes:</b> {{test.testDoctorNotes}} </p>
   {% else %}
      <p>404 error. Page not found.</p>
   {% endif %}
{% endblock %}
```

\app\templates\update.html

Update profile - bonus

```
{% extends "base.html" %}

{% block content %}
{% if current_user.is_authenticated %}
   <h1>{{ current_user.forename }} {{ current_user.surname }}: Profile update</h1>
   <form action="" method="post">
      {{ form.hidden_tag() }}
      {{ form.email.label }}<br>
      {{ form.email(size=32, class='form-control', value=current_user.email) }}<br>
      {% for error in form.email.errors %}
      <span style="color: red;">[{{ error }}]</span>
      {% endfor %}
      <p>
         {{ form.houseNo.label }}<br>
         {{ form.houseNo(size=12,class='form-control', value=address.houseNo) }}<br>
      </p>
      <p>
         {{ form.streetName.label }}<br>
         {{ form.streetName(size=32,class='form-control', value=address.streetName) }}<br>
      </p>
      <p>
         {{ form.postcode.label }}<br>
         {{ form.postcode(size=32,class='form-control', value=address.postcode) }}<br>
      </p>
      <p><b>Date of birth: </b>{{current_user.dob}}</p>
      <p>{{ form.submit() }}</p>
   </form>
{% else %}
   <h1 style="font-family:'Times New Roman', Times, serif">You must be logged in to view this page!</h1>{% endif %}{% endblock %}
```

\app\templates\email\reset_password_email.html – objective 2

```
<p>Dear {{ user.forename }},</p>
<p>
    To reset your password,
    <a href="{{ url_for('resetPassword', token=token, _external=True) }}">
        click here</a>.
</p>
<p>Alternatively, you can paste the following link in your browser's address bar:</p>
<p>{{ url_for('resetPassword', token=token, _external=True) }}</p>
<p>If you have not requested a password reset simply ignore this message.</p>
<p>Yours sincerely,</p>
<p>The Doc2Home admins.</p>
```

\app\templates\email\reset_password_email.txt – objective 2

```
Dear {{ user.forename }},

To reset your password, click on the following link:

{{ url_for('resetPassword', token=token, _external=True) }}

If you have not requested a password reset simply ignore this message.

Yours sincerely,

The Doc2Home admins.
```

TESTING

Overview

This section aims to provide an overview of the final testing of my application. While a lot of testing was done iteratively and during the coding phase, it is only once a program nears completion that the most rigorous assessment can be performed. Furthermore, a successful test that took place could be deemed invalid if other parts of the program has changed enough since. Therefore, most iterative tests are excluded from this section as the results could be deemed invalid. Please note that in the table 'normal' test input types are not stated.

Video references

[15] A. Nkyi, "MyQueue test," 1 February 2023. [Online]. Available: https://www.youtube.com/watch?v=qD5JczFXN6I.

[16] A. Nkyi, "Doc2Home: Create patient account," 2023. [Online]. Available: https://www.youtube.com/watch?v=IA16D708B74&ab_channel=KwabenaNkyi. [Accessed 6 March 2023].

[17] A. Nkyi, "Doc2Home: General testing part 1," 2023. [Online]. Available: https://youtu.be/jxCaUOQrFVI. [Accessed 6 March 2023].

[18] A. Nkyi, "Doc2Home: Appointment handling," 2023. [Online]. Available: https://youtu.be/KJ8nsvEq7YM. [Accessed 6# March 2023].

[19] A. Nkyi, "Doc2Home: General testing part 2," 2023. [Online]. Available: https://youtu.be/qJVy0KKJGQw. [Accessed 6 March 2023].

[20] A. Nkyi, "Doc2Home: Prescription exception," 8 March 2023. [Online]. Available: https://www.youtube.com/watch?v=iA_EGF6_y7Q. [Accessed 2023].

Module testing

Module	Test no.	Inputs	Expected result	Evidence	Result	Type and comments
myQueue	1	Instantiate queue with max size 5. Enqueue 1 then check if queue is full	False	[15] – 0:59	Pass	
	2	Check if queue is empty	False	[15] – 1:29	Pass	
	3	Dequeue then check if queue is empty	True	[15] – 1:38	Pass	
	4	Enqueue values 'a', '33hd', 54, '@', 'f'', 'g'	Error message - queue will be full after 'f' is added.	[15] – 1:54	Pass	Boundary. Mix of type inputs added to the queue to demonstrate its adaptability.
	5	Dequeue then check pointers	Start pointer = 2	[15] – 3:43	Pass	

Objective testing

Objective	Inputs	Expected result	Evidence	Result	Type and comments
1.1	Create patient account	Patient account created and user can login	[16]	Pass	
	Same NHS Number and email as existing account	Prompted to change NHS Number and email	[17]-1:42	Pass	Erroneous
1.2, 1.3	Same address as account in test 1	Doctor account created and user can login. Both accounts have same address id	Test fig 1 - found in image references	Pass	
	Same NI Number as existing account	Prompted to change NHS Number and email	[17]-0:31	Pass	Erroneous
1.4	Login with patient account created earlier	Login success and redirect to homepage	[16]	Pass	
	Login with doctor	Login success and redirect to homepage	[17]-1:20	Pass	
2	Enter email not linked to an account	Error message returned	[17]-2:48	Pass	Erroneous
	Submit valid email from test 1	Email sent to account	[17]-1:58	Pass	
2.2.1	Enter the same old password at reset screen	Prompted to re-enter password	[17]-2:13	Pass	Erroneous
2.2.2	Enter valid new password then log in	Password reset and login success	[17]-2:26	Pass	
4.1	Patient from test 1 requests an appointment	Submitted and shown on home screen	[18]-0:20	Pass	
4.2, 4.3	Doctor rearranges appointment before today	Prompted to re-enter date	[18]-0:42	Pass	Erroneous
	User from test 2 rearranged to valid date and other appointment rearranged 14 minutes after this new time	Prompted to change date or time	[17]-4:40	Pass	Boundary

4.4	Cancel the first appointment.	Cancelled appointment	[18]-1:00	Pass	
4.5	Add notes to an appointment, then view appointment with the patient.	Notes can be added. They can be viewed by patient.	[19]-0:18, 0:53	Pass	
4.6, 4.7	Record test result from doctor who has had appointment with patient.	Test is successfully recorded.	[19]-6:36	Pass	
4.8, 4.9	View test from test 25 from the patient account.	Both users should be able to view the test and its notes.	[19]-6:52	Pass	Boundary
5	Level 1 doctor creates medicine, then view medicine screen with patient and level 3 doctor.	Medicine added to database but neither the patient nor level 3 doctor can view it.	[19]-1:17	Pass	
6.1, 6.3	Level 1 doctor creates allergen, then view with level 3 doctor.	Allergen added to database but the level 3 doctor cannot view.	[19]-1:57, 3:33	Pass	
6.2, 10.1, 10.1.2	Create prescription for patient with medicine they are allergic to.	Prescription not allowed.	[19]-3:48	Pass	Erroneous
6.4, 10.1.1	Create prescription for patient then add an allergy to the medicine prescribed.	Prescription ended.	[20]	Pass	
7	Level 3 doctor adds disease, then level 1 doctor adds inheritable disease.	Level 3 doctor's access to disease e is denied, but the level 1 doctor's addition is added.	[19]-4:26	Pass	Erroneous
8, 8.1	Doctor links inheritable disease to patient profile then views profile.	Link added to database, and disease trait shown on profile.	[19]-4:56	Pass	
	Doctor links same inheritable disease to patient.	Link rejected as it already exists.	[19]-5:19	Pass	Erroneous
9, 9.1	Create illness history that ended before it begun.	Logic error and prompt to change dates.	[19]-5:33	Pass	Erroneous

	Create illness history with valid date range.	Illness linked and can be viewed on profile.	[19]-5:48		
9.2	Record referral with valid inputs.	Referral added and can be viewed on referral section.	[19]-6:07	Pass	
10.2, 10.3	Create valid prescription for patient, then view prescription from patient's side.	Prescription created and can be viewed from patient home page.	[19]-7:03	Pass	
11	Accumulate time spent over a given seven-day period for one doctor of each access level.	Overtime exceeded: False for each doctor	Test fig 2, Test fig 3, Test fig 4, Test fig 5 – found in image references	Pass	

EVALUATION

Objectives

As demonstrated by the extensive testing in the previous section, it is clear that all the objectives I identified when defining my project have been achieved. The following sections will involve discussing how these were achieved and the potential improvements to said objectives.

Objective 1: Login and registration

Registration and login were the very first features implemented for use, and has been stable for a while. There have been no bugs discovered during final testing and the forms are very robustly designed to ensure that any unexpected input does not make it to the database, as shown in the test and my code.

In the analysis, I stated that this project assumes that every user is who they are logging in as. Unfortunately, if this project were to be used in a real GP surgery, that assumption would pose a major security risk for both doctors and patients, as anyone could create an account as either user type. A better system would be one based on internal admins allowing access to an account through two-step verification. For example, a patient account is already stored in the database. When they physically visit the GP, they are emailed a code which allows them to access their account and all the information. This also allows doctors to add information for patients with no internet access, as another assumption was that every single patient in the GP had login access.

The subroutines such as the blood type selector were just a way to simulate the proportions of real life, and for the use in a real surgery they would already have all of the patient information available. The patient would just need to confirm their identity to gain access to it.

In the future, these pages could be given a facelift to improve the look and feel of the login and registration. With bigger input boxes and modern styling (perhaps using Bootstrap) it may be a much more approachable website than even official GP websites. Images could also be uploaded to profiles.

Objective 2: Resetting password

The reset password feature provides a very secure option to regain access to an account when there was no way to do so without the application, prior to its addition. It fulfilled all given objectives. In the future, user accounts can be validated upon registration using similar email methods.

Objective 3: Homepage

The homepage also fulfilled all objectives. It does not overload users with information However it is a bit sparse, so the design and placing of sections (e.g. appointments, prescriptions) could be made larger or moved about the page using some styling. There is also

some client-side processing and while nothing more than getting user names from the database, this could have been implemented by passing the variable from the server thus keeping the client's processing thin.

Objective 4: Appointments and tests

The appointment booking system was very difficult to build validation for, and required a lot of Jinja IF statements to cycle between the appointment viewer screen states (confirm booking form, post-appointment form, view feedback). The styling of the appointments homepage for doctors could also be moved into a column structure.

For patients, the appointment booking is very straightforward and streamlined. The doctor page contains more information and is thus more clunky. Doctors of any access level cannot cancel appointments, and while this is good practice in principle, there would need to be external intervention to communicate to patients if the doctor cannot fulfil the appointment. This is where email or even text message communication can make a difference.

The clash checker worked very well on a variety of test cases, and during the running of the web application. The overtime checker was a feature added mainly for the benefit of larger practices, where a doctor may be fully booked the whole day with appointments. Both checks are extremely useful to ensuring smooth running and that every patient who requests a booking is guaranteed to be seen at some point. This could be more strictly implemented with automatic pool-based allocation.

It was quite difficult to find a specific appointment as a doctor after adding patient feedback and doctor notes – as they are not recorded on the patient profile, the doctor has to save links before submitting the post-appointment form. This could be rectified by showing previous appointments on each patient's profile when viewed by the doctor.

Email confirmation of appointments in the future would be a great step forward and would improve the user experience. This would mean they would not have to keep checking the website for a response to the booking. A system like this would not be difficult to implement seeing as the mail framework is already in place for password resets.

Tests are easy to add and view from both sides.

Objective 5 & 6: Medicines and allergies

Iterative testing during development and database constraints demonstrated the fact that each record must have a unique name identifier. The addition is unproblematic, and all information returned by the program about inputs is easy to interpret. There is, however, no way of deleting a medicine or allergen so any typos in submission are permanent. An updated application will provide

The ingredients addition page (linking medicine to allergy) utilises a clever autocomplete function to reduce the chances of unrecorded items being saved. It calls the website API using AJAX for a list of all medicines and allergens. The individual medicine viewer displays its

name, recommended dosage, and all allergens. There was a section created for displaying prescriptions of each medicine that could quickly be completed but has not been finished.

Meanwhile, patient allergies can be created by visiting the patient profile directly and clicking the relevant link. The patient profile may be accessed directly through a search function where the doctor inputs the NHS Number. While not a specific objective, the profile search is a game-changer for functions such as this.

Objective 7 & 8: Disease and inheritable disease

Like medicines and allergens, the disease creator simply cannot take a non-unique name as an argument. Marking a disease as an inheritable disease was as simple as ticking a box, and the linking of an inheritable disease to a patient was easily done through the profile and an autocomplete field. Diseases could not be deleted, which does make practical sense, but maybe the option to modify diseases' inheritability status or names could have been implemented. Furthermore, adding a disease category like the types mentioned in the analysis would allow for even more analysis of illness trends.

Objective 9: Illness history

Illness history and referrals are both very quick to add and view, from an individual patient's profile. The referral section was the latest to benefit from autocomplete; this time it was type (e.g., A&E, physiotherapist, etc.) based on previous submissions. The doctor could also create their own type when creating a referral.

A feature I think could be implemented is a search for illness history and referrals based on diseases, dates, and type of referral. With the autocomplete feature, it should not be too difficult to implement and would allow GPs to better analyse local trends.

Objective 10: Prescriptions

Finishing the prescriptions was a very time-consuming part of the program, no doubt in part to the amount of validation required especially when it came to allergens. It required a multi-table join in raw SQL to ensure that dangerous allergens would automatically be removed from allergic patients' prescriptions.

Overall, this objective came out well, especially the prescription viewer. The UI could be cleaner and once again, a drop-down box would be quite inefficient for large practices, so prescriptions may have to be assigned by visits to individual profiles. The prescription objective could have been developed into one which allowed patients to monitor when and where to collect medicines.

Objective 11: Hours spent

This objective was completed in its entirety, but the hours spent table in the database went to waste because it was not really used other than for being written to. It was not used in any other part of the program. This table would be more useful if it had more detail such as recording sign-in and sign-out times (e.g. with physical lanyards). The new information connected to a payroll calculating application could be used to generate payslips or calculate the gross pay of a staff member. The doctor may also want to override overtime limits.

Third party feedback

Some third-party feedback from multiple end users:

- **User journey was very well designed – sibling:** This was one of the most important aspects of my project, and it was good to see end users feeling comfortable with the web application.
- **Autocomplete feature very innovative – classmate:** Adding autocomplete to my project was a game-changer as it reduced the chance of invalid input for key sections, and increases ease of access for everyone.
- **Misleading to include sections inaccessible to users on navbar (e.g. level 1 sections for a level 3 doctor) – parent:** In future updates to the project, a simple Jinja2 script would ensure that this does not occur. While the current error handling is secure enough to block any unauthorised users, it may be a frustration for a doctor who misclicks or forgets themselves.
- **Record details about allergies like common reactions – sibling:** Adding more details about allergies is a very good idea and one that is also not difficult to implement. It would lead to better knowledge of sickness and improves healthcare provisions when a patient visits the GP or hospital.
- **Tab titles not informative – classmate:** Also a simple addition, and at production level this is definitely essential for quick multi-tab browsing. They were only implemented on a few pages.
- **More direct links (e.g. hyperlinking to a patient profile on their own prescription) – sibling:** Adding more hyperlinks would definitely improve web app browsing speed, and would be useful for data like patient names. However the current system results in increased security because you must know a patient's NHS number beforehand to make key changes to their files, while the hyperlinks would allow any doctor to find their details and do so.

Potential improvements

Verification for doctors creating account – potentially by admin

This feature would be quite similar in implementation to the reset password – maybe a token is sent to a newly-registered doctor's email, which they use to undergo further security checks. It would be a very useful feature if this program is used in a real practice.

Alternative methods of linking to patients

As touched upon earlier, drop-down boxes could be a massive pain for a doctor in a hurry when saving a prescription or appointment: imagine submitting it to the wrong person by accident! Thus, like with adding allergies and an illness history, you would visit a patient's profile first before being linked to create a prescription or appointment.

Navbar improvements

The navbar was quite cluttered and may have benefitted from the subgrouping of some pages, e.g. ingredients inside → medicine/allergen. However, its positioning in relation to the styling of the page contents was good as it allowed for quick switching. If the styling changes proposed for certain objectives came to fruition, the navbar position may need to be changed.

Final conclusion

I am very happy with how this project went as it helped me develop some new programming skills while also showcasing my ability to work on a project of such breadth and depth. I set challenging but realistic objectives, and I learnt a lot about web development, databases, and object-oriented programming along the way. The new techniques I learnt allowed me to space development of the program over a period of a few months. This time helped me nail down exactly what I wanted in my program and how I wanted it, when I wanted to add it in.

I expected my application to be a very difficult task but I am pleased with my perseverance especially at times where my objectives were not always so easy to successfully implement. This is the largest and most interesting project I have ever built, and since it ties in well to real-life experiences, this is potentially a base for me to build upon. In the future this could be seen as the very rough draft of the first version of a ready solution to real-life issues in healthcare communication and documentation.

REFERENCES

Image references

Test figure 1:

```
>>> x.select('select address_id from patient where NHSNumber=3000000000')
[(34,)]
[(34,)]
>>> y=x.select('select address_id from patient where NHSNumber=3000000000')
[(34,)]
>>> z=x.select('select * from doctor where address_id=34')
[(16, 'testvid1@test.com', 1, 'AB457754C', 'Test', 'Video', 'F', '1980-07-05',
'pbkdf2:sha256:260000$9iMZD5QAOtT9nYkB$2c7afe275db673de19aac7e81ae4bd8c58463c3c
893d9165590518f11bc57534', 34)]
```

Test figure 2:

```
>>> db.select('select accesslev from doctor where doctor_id=5')
[(2,)]
[(2,)]
>>> from datetime import date
>>> y=checkOvertime(5,2,date(2022,11,17))
1 appointments on 2022-11-17

First appointment at 13:00:00 last appointment at 13:00:00
time spent on 2022-11-17: 0:15:00

Max time allowed on one day for level 2 doctor: 10:00:00

0 appointments on 2022-11-11

time spent on 2022-11-11: 0:00:00

1 appointments on 2022-11-12

First appointment at 21:26:00 last appointment at 21:26:00
time spent on 2022-11-12: 0:15:00

0 appointments on 2022-11-13

time spent on 2022-11-13: 0:00:00

0 appointments on 2022-11-14

time spent on 2022-11-14: 0:00:00

0 appointments on 2022-11-15

time spent on 2022-11-15: 0:00:00

0 appointments on 2022-11-16

time spent on 2022-11-16: 0:00:00


total time spent over past 7 days is 0:30:00
total time allowed 1 day, 19:00:00
>>> y
False
```

Test figure 3:

```
>>> db.select('select accessLev from doctor where doctor_id=12')
[(1,)]
[(1,)]
>>> y=checkOvertime(12,1,date(2023,3,4))
1 appointments on 2023-03-04

First appointment at 12:00:00 last appointment at 12:00:00
time spent on 2023-03-04: 0:15:00

Max time allowed on one day for level 1 doctor: 11:00:00

0 appointments on 2023-02-26

time spent on 2023-02-26: 0:00:00

0 appointments on 2023-02-27

time spent on 2023-02-27: 0:00:00

2 appointments on 2023-02-28

First appointment at 09:33:00 last appointment at 10:46:00
time spent on 2023-02-28: 1:28:00

0 appointments on 2023-03-01

time spent on 2023-03-01: 0:00:00

0 appointments on 2023-03-02

time spent on 2023-03-02: 0:00:00

0 appointments on 2023-03-03

time spent on 2023-03-03: 0:00:00


total time spent over past 7 days 1:43:00
total time allowed 2 days, 0:00:00
>>> y
False
```

Test figure 4:

```
>>> db.select('select apptime from appointment where doctor_id=2')
[('18:00:00.000000',)]
[('18:00:00.000000',)]
>>> db.select('select appdate from appointment where doctor_id=2')
[('2022-10-15',)]
[('2022-10-15',)]
>>> y=checkOvertime(2,3,date(2022,11,17))
0 appointments on 2022-11-17

time spent on 2022-11-17: 0:00:00

Max time allowed on one day for level 3 doctor: 9:00:00

0 appointments on 2022-11-11

time spent on 2022-11-11: 0:00:00

0 appointments on 2022-11-12

time spent on 2022-11-12: 0:00:00

0 appointments on 2022-11-13

time spent on 2022-11-13: 0:00:00

0 appointments on 2022-11-14

time spent on 2022-11-14: 0:00:00

0 appointments on 2022-11-15

time spent on 2022-11-15: 0:00:00

0 appointments on 2022-11-16

time spent on 2022-11-16: 0:00:00


total time spent over past 7 days is 0:00:00
total time allowed 1 day, 14:00:00
>>> y
False
```

Test figure 5:

```
>>> y=checkOvertime(2,3,date(2022,10,15))
1 appointments on 2022-10-15

First appointment at 18:00:00 last appointment at 18:00:00
added today's record to db
time spent on 2022-10-15: 0:15:00

Max time allowed on one day for level 3 doctor: 9:00:00

0 appointments on 2022-10-09

time spent on 2022-10-09: 0:00:00

0 appointments on 2022-10-10

time spent on 2022-10-10: 0:00:00

0 appointments on 2022-10-11

time spent on 2022-10-11: 0:00:00

0 appointments on 2022-10-12

time spent on 2022-10-12: 0:00:00

0 appointments on 2022-10-13

time spent on 2022-10-13: 0:00:00

0 appointments on 2022-10-14

time spent on 2022-10-14: 0:00:00


total time spent over past 7 days is 0:15:00
total time allowed 1 day, 14:00:00
>>> y
False
```

Text references

Cover image source: “African American businessman shaking hands with a male doctor.” Available: https://www.directrecruiters.com/about-us/testimonials/attachment/businessman-shaking-hands-with-a-doctor-4 [Accessed 1 February 2023]

[1] “Job profile: General practice doctor,” Prospects, 2021. [Online]. Available: https://www.prospects.ac.uk/job-profiles/general-practice-doctor. [Accessed 11 June 2022].

[2] “London Road Surgery,” London Road Surgery, 2022. [Online]. Available: https://www.londonroadsurgery.co.uk/. [Accessed 13 June 2022].

[3] TPP, “Why choose SystmOne GP? Let our users tell you why...,” 13 January 2016. [Online]. Available: https://youtu.be/KG_u0txADOU. [Accessed 7 October 2022].

[4] “Your health records,” NHS, 2021. [Online]. Available: https://www.nhs.uk/using-the-nhs/about-the-nhs/your-health-records/ . [Accessed 10 June 2022].

[5] “A Guide to Confidentiality in Health and Social Care,” NHS, 9 March 2022. [Online]. Available: https://digital.nhs.uk/data-and-information/looking-after-information/data-security-and-information-governance/codes-of-practice-for-handling-information-in-health-and-care/a-guide-to-confidentiality-in-health-and-social-care. [Accessed 12 June 2022].

[6] UNICEF, “Non-communicable diseases,” [Online]. Available: https://www.unicef.org/health/non-communicable-diseases.

[7] L. Haynes, “Full-time GP partners work 50% more than standard working week,” GP Online, 16 September 2019. [Online]. Available: https://www.gponline.com/full-time-gp-partners-work-50-standard-working-week/article/1595772. [Accessed 11 June 2022].

[8] “Blood types,” NHS, [Online]. Available: https://www.blood.co.uk/why-give-blood/blood-types/. [Accessed 16 August 2022].

[9] [Online]. Available: https://tall.life/height-percentile-calculator-age-country/. [Accessed 15 August 2022].

[10] “GIRLS UK Growth Chart 2-18 years,” [Online]. Available: https://www.rcpch.ac.uk/sites/default/files/Girls_2-18_years_growth_chart.pdf. [Accessed 16 August 2022].

[11] “BOYS UK Growth Chart 2-18 years,” [Online]. Available: https://www.rcpch.ac.uk/sites/default/files/Boys_2-18_years_growth_chart.pdf. [Accessed 16 August 2022].

[12] “Height Percentile Calculator, by Age or Country,” [Online]. Available: https://tall.life/height-percentile-calculator-age-country/. [Accessed 17 August 2022].

[13] “What a NINO looks like,” gov.uk, 20 October 2022. [Online]. Available: https://www.gov.uk/hmrc-internal-manuals/national-insurance-manual/nim39110.

[14] “Flask-Login Documentation,” [Online]. Available: https://flask-login.readthedocs.io/en/latest/. [Accessed 2022].

Video references

[15] A. Nkyi, “MyQueue test,” 1 February 2023. [Online]. Available: https://www.youtube.com/watch?v=qD5JczFXN6I.

[16] A. Nkyi, “Doc2Home: Create patient account,” 2023. [Online]. Available: https://www.youtube.com/watch?v=IA16D708B74&ab_channel=KwabenaNkyi. [Accessed 6 March 2023].

[17] A. Nkyi, “Doc2Home: General testing part 1,” 2023. [Online]. Available: https://youtu.be/jxCaUOQrFVI. [Accessed 6 March 2023].

[18] A. Nkyi, “Doc2Home: Appointment handling,” 2023. [Online]. Available: https://youtu.be/KJ8nsvEq7YM. [Accessed 6# March 2023].

[19] A. Nkyi, “Doc2Home: General testing part 2,” 2023. [Online]. Available: https://youtu.be/qJVy0KKJGQw. [Accessed 6 March 2023].

[20] A. Nkyi, “Doc2Home: Prescription exception,” 8 March 2023. [Online]. Available: https://www.youtube.com/watch?v=iA_EGF6_y7Q. [Accessed 2023].