PIMS: FE

The Patient Information Management System: Flask Edition

PIMS-Demo.HostedBy.JamesDev.co.gg

Project Analysis

Contents:

Problem Recognition

The original version of PIMS was coded between March and July 2022. A passion project of mine, inspired by my work with St. John Ambulance, its purpose was to store hypothetical patient information, including personal details, injury and treatment details, as well as vital observations. The system was composed of a terminal-based server program that backed to an SQLite3 database and a Tkinter-based client software for accessing and adding to the server via TCP communication. This worked well, however, the UI was heavily constricted by the limitations of the Tk language which led to a clunky user experience and inconsistency between operating systems. Additionally, Tkinter does not work on mobile devices, and Python is often not the easiest to compile for these systems either - the thought of having to lug around a laptop in a hypothetical first-aid scenario was not true to life, where time is of the essence and you only can really carry a couple of things. No first aider would prioritise carrying a laptop over a grab bag. From this, I started searching for solutions; building it using Kivy, PyQT, or in a completely different language altogether such as Java or Swift. All of these seemed too difficult or would subtract too much from the original idea. In the end, another project I was working on required me to become familiar with the Python Flask module - this pushed me to recreate PIMS in Flask.

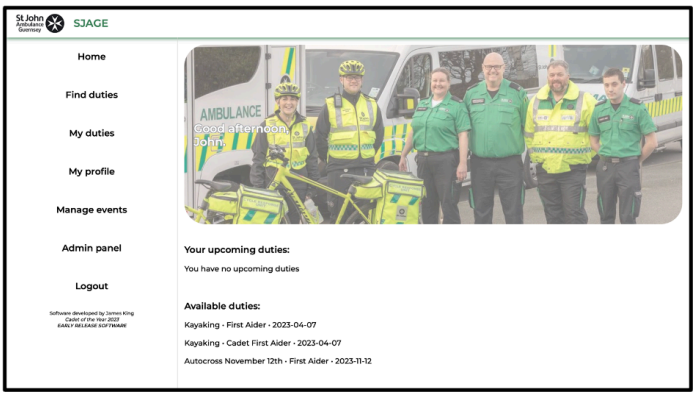
The new version of PIMS would have to meet a couple of requirements. Firstly, the new version of PIMS must be compatible with the old version. A user should expect to be able to simply link their existing database file with FE and have it run without issue - the two softwares should be able to read each other's edits and be used side by side. Secondly, the majority of the features present in the desktop edition of PIMS must be present in the Flask edition, the only time that a feature would not be included would be if it was simply technically impossible for it to run from a browser environment. Lastly, it must meet the requirements in the paragraph above - it must be able to be run on mobile devices and the UI must be easy to use and consistent across all devices - desktop and mobile.

Stakeholders

If PIMS was released to the public the main stakeholders would be first-aid groups. Typically, these people are a mix of ages, with many ranging from youth up to those aged 50+, additionally, the software would most likely be used during stressful treatment scenarios. As such, the interface must be simple and logical - buttons should be large to accommodate for touch screens and for inaccurate, rushed clicks. An uncomplicated design should also stick to a simple colour palette - with colours reused across the system to denote similar things, e.g. a colour to indicate which elements are clickable or elements that need a value. Other than that, the software does not necessarily need to be aesthetically pleasing, with medical personnel caring less about how their software looks and more about if it actually works or not. Therefore, greater attention should be placed on assuring the software is as functional and intuitive as possible, instead of aesthetically pleasing when it comes to UI design.

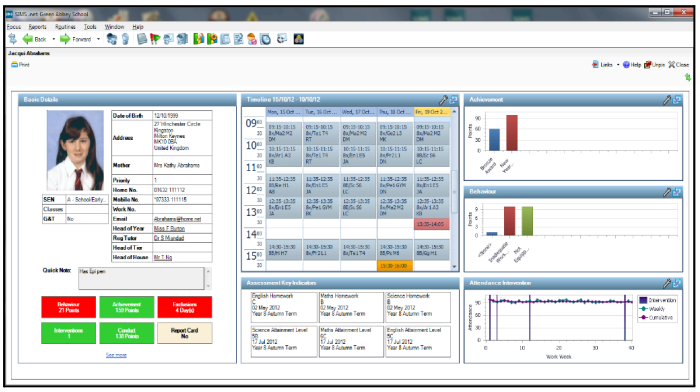
Research

The bulk of the UI elements are borrowed from a previous project of mine, the St. John Ambulance Guernsey Event System (SJAG Events) - an online duty booking system for personnel covering events I built for the local branch of St John Ambulance I volunteer for. When creating the UI for SJAG Events it was paramount that the UI would work well on desktop and mobile devices - something which is shared with PIMSFE. The UI features a large navigation bar on the left of the screen that is always visible - this allows the user to easily navigate the website while also being able to view the page contents without the need for a sliding navigation bar.



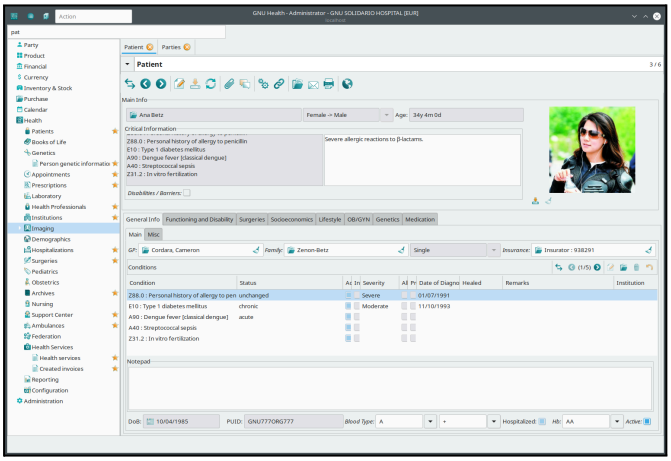
For help with HTML and CSS styling, the best two resources I have found online are W3Schools and the Mozilla Developer pages. Additionally, I also draw help from the book *'HTML & CSS'* by Jon Duckett (ISBN: 1118008189), a useful guide for common HTML and CSS issues.

This project will also be drawing inspiration from other management information systems. For instance, the *School Information Management System* (SIMS), built by Capita, is an MIS designed for storing student information. This



software is used in over 40 countries, managing over one million students. SIMS is tried and tested, which its design reflects; the UI is simple but very functional, having many colour-coded buttons and context menus for additional functionality. Several features found in SIMS are reflected in PIMS, such as the ability to have multiple teachers or operators, multiple students or patients, the ability to record data, and export features.

GNU Health is an open source patient information database, developed as a



part of the GNU open source project. It features many of the same capabilities as PIMS as well as a couple more advanced features, being designed for use in hospitals and clinics. Being free and open source, the software has become a very popular alterative to traditional patient record software, which can usually be very expensive, for charities, NGOs, and health systems in developing countries. The UI is quite cluttered, but finctional, and features a prominenet sidebar

for ease of navigation. I plan on taking inspiration from this for my redevelopment of PIMS.

Essential features

1. Previous patients, cases and notes can be accessed
2. New patients, cases, and notes can be created and saved.
3. User access and permissions should be restricted based upon their privilege value.

Computational methods

The project can be broken down into the following major components.

- User management

  Creating and editing users as well as user access. An essential part of this is the implementation of user privilege levels - restricting what actions and access users have over records and features.
- Opening patients, cases, and notes

  Retrieving data from the database and displaying it in an organised, neat way.
- Creating patients, cases, and notes

  Accepting data from the user via a form and storing it in the database.
- The triage board

  Displaying and categorising active patients that have been added to automatic triaging into four different levels of severity on the basis of recorded figures.
- Exporting cases & other utilities

  Collecting case data from the database and creating an encrypted PDF file that can be sent to the patient as a record of their treatment. And, other useful features and statistics for the program as required.

Limitations

Compared to a typical application, web apps can face some limitations. An example of this present within PIMS is the pulse oximeter (pulse-ox) import feature. This tool allows users to import pulse-ox readings automatically from supported pulse oximeters using Bluetooth Low Energy (BLE). This works by using the Bleak module for Python, which interacts with the user's device's BLE chip, intercepting the broadcasts from the device and interpreting the raw data. However, while experimental bluetooth low energy capabilities exist on Chromium-based browsers and Firefox, this feature is often disabled by default due to security concerns.

The current approach I am taking for designing the UI - having mobile and desktop users seeing the page from the exact same code - is non-standard and has its limitations. Typically, two versions of a page are designed and are automatically loaded when a user accessing a page, one that is optimised for a desktop or tablet experience, and another that is better tuned for the narrow, small screens of smartphones. However, this introduces unnecessary complexity to the project and would contribute a significant extra workload when designing each page. As such, the chosen design will act as a compromise between the two - a page that is neither optimised for mobile nor desktop, a middle ground where the UI might seem spacious on one device, and cosy (yet clean) on another.

Hardware & software

The PIMS web app is currently hosted on a virtual private server (VPS) rented from Fasthosts, the specifications follow:

Operating system	Ubuntu 22.04 LTS console based
Processors (vCPU cores)	2 x Intel Xeon Gold 5120 @ 2.194GHz
Random access memory	6GB

SSD storage	80GB

The software required to run the web app is listed below:

Name	Function
(APT) Apache2	HTTP/HTTPS web server; uses a web-server gateway interface (WSGI) to direct connections to Flask.
(APT) Certbot	Provides and generates self-signed SSL certificates for Apache2 to allow for HTTPS.
(PIP) Flask	The core framework of the web app.
(PIP) Flask-Login	A Flask module, handles user management and access across pages.
(PIP) Flask-SQLAlchemy	A Flask module, allows the program to link with an SQLite database, and interact with records as objects.
(PIP) Jinja2	A module used by Flask, Jina is a scripting language based on Python that can be embedded into HTML template files. This is then executed when the template is loaded by Flask.
(PIP) PDF Kit	Used to convert the HTML email template into an initial, unencrypted, exportable PDF format.
(PIP) PyPDF2	Takes the file created by PDF Kit and

	encrypts it using a six-digit pin (usually a date of birth) before saving it back to the hard drive.
(PIP) Email	Used to create a sendable email object, handles file attachment and the setting of headers.
(PIP) SMTP Lib *&* SSL	Used to establish a secure connection to the *jamesdev.co.gg* SMTP server for sending exported cases.

For the client side, the web app can be run in any modern browser - mobile or desktop - with the browser needing to be able to support modern HTML5, CSS, JavaScript, and the TLS based HTTPS. The UI design will allow for easy use with a touch screen or physical keyboard and mouse.

Success criteria

Criteria
A responsive UI that fits phone, tablet, and desktop use.
Full compatibility with the Tkinter based desktop app.
Patients, cases and notes can be accessed.
New patients, cases and notes can be added to the database.
Patient cases can be exported via PDF over email.
Ideally, bluetooth pulse oximeter compatibility.
User permission control.