Team WilliWaller:

Parental Education CMS for KCL Pediatric Liver Department

Scott Anderson, Layth Mehdi, Victoria Hayes, Artur Ganeev, Edouard Azoulai, Antoine Gosset, Ivan Hristev, Oskar Ljungdell

Introduction:

Our business objective for this project was to develop a content-management system that enables the PLS to deliver education/support to both parents and health professionals in a visual and interactive manner.
Starting with the initial brief given, a group meeting was held to begin breaking down the task into a set of user-requirements (user stories), which were later finalised with the client.

Clients objectives:

- Have an online resource for all the information given to parents when their child is diagnosed with liver disease (Biliary Atresia)
- Present the information in an interactive and visually pleasing manner to allow accessibility for not only parents and professionals, but for children too.
- Direct parents with queries initially to this site as their first port of call for information - alleviate pressure from the nurses through the use of an FAQ page.
- Provide an online drug chart that can be edited and printed by parents as/when they need to.
- (Not the client specific objective necessarily) Put a system in place to allow professionals to edit the website to allow for expansion where necessary.

Outcome:

State applications that form the system
The product developed is an online web application, designed to ensure compatibility with mobiles and tablets. The site integrates a number of different services, with easy navigation through the initial splash-page and navbar.

List of high-level features

- Customizable interactive drug chart that allows users to modify dosages and medications for their child - an option to print this chart is given. Medication details can be added and edited by the admins.
- Announcements can be added, edited and deleted by admin, and are featured on an announcements page with a sliding news bar on the index page to give visitors to the site the latest news from the Paediatric Liver Center.
- Frequently asked questions can be posted by visitors and answered by admin.
- A search bar on every page finds and highlights the inputted word.
- A quiz on each section which keeps track of their score and records which questions are often answered incorrectly - storing this information to be displayed on the dashboard page.
- On the dashboard page the statistics page shows useful information that can be used to determine what parts of the information are not being well understood.
- Each page has a section where visitors can say whether the page was useful or not and this is recorded on statistics.
- A “dark-mode”, which changes the colours of the pages from the default scheme to a darker version - with appropriate “relaxing” colours chosen. Shades of blue, violet, grey and green were chosen to help calm the visitors to the site.
- The client has access to an admin page which has an in-built editor that allows them to edit the content on the page in an intuitive manner, without requiring any high-level technical knowledge.

There were several important features discussed in group meetings that were never implemented. An interactive timeline was put forward to the client as a way of showing the visitors to the site a typical journey of an individual with Biliary

Atresia. The clients advised us against this however as the timeline of each individual varies too much and the addition of a timeline would potentially give people inaccurate expectations.

We also discussed sharing the experience of survivors/parents of individuals with Biliary Atresia, where they could post stories about their past. The clients were fond of this idea, however coordinating meetings with these previous patients would require several steps of administration and could not be done within the deadline.

Design:

The design of the application was broken down into two sections, front-end/back-end.

BACK-END:

- For the back-end of the application we used Flask (a web framework written in Python). In Flask, we used blueprints that allowed us to group specific functionalities within Python packages. This allowed us to produce a well-organised structure within the application, as each function is kept in a separate folder (e.g. announcements, user_login etc.).

- We used SQLAlchemy to make the database and query them directly in Python, which allowed us to design the database schema in a separate Python file (models.py). SQLAlchemy allows us to have different sql databases: a sqlite file for testing and development and a postgresql database for production. Furthermore, it allows us to prevent any sql injection as it automatically escapes sensible characters.
    - Models.py - creates SQL tables for things such as Users and the FAQ.
    - We store generic information about visitors to the site through the use of the "Flask-Track-Usage" library. This is a lightweight library that allows us to store basic metrics about the visitors of our site without using extra cookies.

- Followed Python coding conventions to ensure that the application follows industry standards in terms of both readability and maintainability.
- Flask Track Usage was used to collect user requests to the server in a database table with the purpose of using the data for web analytics that are shown on the admin dashboard.

- We used Docker for deployment as it allows anyone to run our app on any server as long as it supports docker. This avoids bugs between platforms, build issues, different/uncomplete configurations, etc...

FRONT-END:

- The front-end design of the application was handled using Bootstrap (HTML/CSS) framework for the templates - stored in a separate template folder.

- Certain modules required the ability to create/edit/delete pages, with these templates being grouped inside a separate folder within the template folder, this improved code cleanliness and allowed for greater code reuse. Quill was used as an open source editor for editing existing articles.

- We utilised a base template common to all pages, this was done for CSS/JS imports, the navigation bar and the footer.

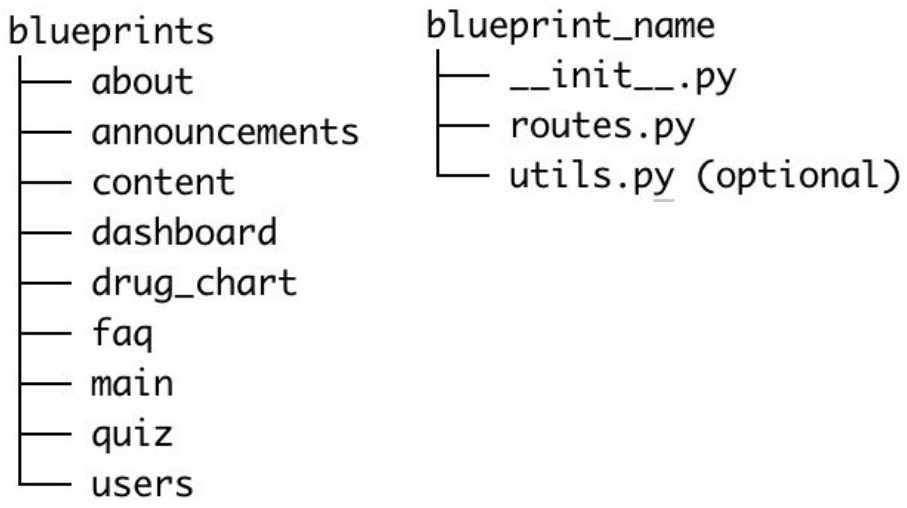
DIAGRAMS:

:

Our flask project has the following structure:

```
flask_root
├── blueprints (under the form of python packages)
├── templates
├── static
│   ├── css
│   ├── js
│   └── resources (images)
├── tests (python package)
│   ├── __init__.py
│   ├── test_back_end.py
│   └── test_front_end.py
├── app.py
├── config.py
└── models.py
```

Each blueprint represents one part of the application. Here are the different blueprint we have with their inner structure:



Finally, flask's template engine allows for inheritance of templates. This allows for reduced code duplication. Here the structure templates follows to inherit one another:

```
splash.html
base.html
├── dashboard_page.html
│   ├── dashboard/edit.html
│   ├── dashboard/helpful_pages.html
│   ├── dashboard/panel.html
│   ├── dashboard/quiz_statistics.html
│   ├── questions/quiz.html
│   ├── users/add_user.html
│   └── users/list_users.html
├── user_page.html
│   ├── 404.html
│   ├── feedback.html
│   ├── users/login.html
│   └── index.html (all of them)
└── form.html
    ├── create.html (all of them)
    ├── edit.html (all of them)
    └── delete.html (all of them)
```

Testing:

We decided to approach testing by only writing tests once the majority of the code was completed as we were aware that, throughout the development process there would be multiple back-end refactors and front-end redesigns. This meant that we would not have to re-write tests as the product was developed. We used Flask-testing, Unittest, Selenium, ChromeDriver and nose2 to conduct our testing.

- **Front-end:** Flask-testing, Selenium and Unittest combined with ChromeDriver to simulate the front-end testing. We also extend the unittest API and utilise the nose2 package in order to run all the tests simultaneously.
- **Back-end:** Unittest and flask's testing library, with nose2 as well.

Front End - simulate a user visiting and interacting with the website in the browser.

- To perform our tests we created test data and using this, we test the login and CRUD functionality of the website.
- We have a 'TestBase' class, which all other classes inherit from, with methods like:
    - **create_app** - which returns an instance of the app. We specify all the app configurations in this method. Such as the database URI or the secret key.
    - **set_Up** - where we set up the chromedriver and put the admin user in the database. This method is called before every test.
    - **tear_Down** - which stops the chromedriver from running
    - **test_server_is_up_and_running** - ensures that the server is up and running.
- The other tests are used so we can simulate a user interacting with it, such as navigating through the site, clicking buttons, entering text into forms and submitting them and testing the CRUD functionality.
    - Clicking buttons/entering text is done by finding the element by id or name.
    - After any form is submitted, we test it's presence in the database by the 'assertEquals' command and counting the number of entries.
    -

Back End - ensures that all operations performed in python are executed correctly and behave as expected.

- View tests that show the correct status codes are produced when urls are loaded from python.
- Database model tests which check that tables contain the correct attributes with addition to being successfully added, updated and removed alongside connection to database..
- Additional tests that check how the app deals with erroneous input such as non existent page links.

All of the testing is automated which saves a huge amount of time towards the end of the development process - all the tests can be run simultaneously and autonomously.

NB: we ran all tests on UNIX systems

Team organisation:

We decided from our initial primer meeting to employ an AGILE approach to our development process. This meant rather than assigning each team member a static role that would remain for the duration of the project, we opted to continuously evaluate our work load as it develops either through weekly meetings or later on with online communication. This allowed each member access to additional support when needed, and also the ability for all of us to learn the multiple new environments we used during the project. Throughout the project, all of us assigned ourselves to the tasks that took advantage of our individual strengths, be it technical or soft skills:

- Anderson
    - Client liaison - Project Task Manager - Front-end page creation - Quiz creation
- Mehdi/Gosset
    - Front-end development - Front-end design - Animation - Information handling - Accessibility - Splash page
- Azoulai
    - Full-stack development - Deployment - Database migration - Content editor - Drug chart Section - Refactoring - Code Quality Management - Worked on all other parts of the website by refactoring and helping other fixing issues/coding features when they had troubles.
- Ganeev
    - Front-end testing - Back-end testing - User creation - User login - Front-end page creation and design- Database Model creation - Quiz section
- Hristev
    - Full-stack development - Web Analytics - CRUD for Content - Database Management - Back-end testing - Web Deployment
- Hayes

    - Front end testing - Quiz section - Quiz statistics - Database Table creation - Announcement Edit (read more) - User login - News scroll
- Ljungdell
    - Front end testing - Announcement section - Search bar - Database Model creation

Other pertinent information:

Throughout the development process we faced numerous difficulties - some were easily fixed due to proper planning and risk assessment before development began. Other problems became apparent mid development and had to be resolved on a rolling basis.

Initially we faced problems with a disconnect between the specifications we were given and the clients requirements - with the client initially asking for an interactive website that would display the information they needed. This problem was addressed during the initial group meetings we had as well as the meeting with a senior member of staff at KCL. We returned to the client with the proposal of providing features which enabled them to extend the site and gain valuable insights - this allowed us to both satisfy the client and deliver a product which more closely aligns with the initial project requirements.

One of these unexpected risks were associated with the implementation of the editor (to edit information present on the index page which shows all the pertinent information regarding liver disease). Because of the way we initially placed the information on the page (hardcoded), we then had to re-do the information through the back-end editor in order to store all the data in a database and have this be modified as and when the client needed. It also brought about issues with the CSS and implementing the "dark-mode" feature for colour-blind visitors to the site. Due to the agile nature of our development approach, we were able to quickly mitigate these problems, with several individuals able to refocus their efforts and help with both the information handling and the different code refactors that were required.

Additionally, another precaution we took due to the nature of our client being part of the NHS was how much data we collect about users and their actions. It was important for us to have some sort of usage data so we can display analytical data about certain parts of the website whilst not storing anything that might violate confidentiality. This has led us away from using third party analytics such as Google Analytics and pushed us towards using a more lightweight alternative in the form of flask_track_usage. All data that is stored does not relate to a specific user or entity, with the addition that our client remains the sole owner of

this information with no access to any other party, unlike if we had utilised Google Analytics.

We decided the inclusion of a user-manual was unnecessary for this project as the screencast demonstration is sufficient for both visitors and administrators of the site.