Project Plan

Problem

Currently consultants can only qualitatively analyse cardiac perfusion, even though theoretical research has proven that it is possible to automate quantitatively analysis of these scans. Our solution gathers valuable quantitative perfusion analysis without the need of being a perfusion expert. This empowers cardiovascular consultants all over the world with a new technique aiding in diagnosis and treatment plans.

State of the Art

One other solution, performs analysis inline in the scanner, scanner-specific, quantitative values are given with the image straight away. However this requires, custom hardware, only Siemen Scanners. And analysis cannot be done retrospectively. This is not very flexible. Our approach is a cloud based platform for anyone to use. Where any CMR DICOM files can be analysed for perfusion quantitatively. The only hardware required is the consultant's computer.

Aims

Creation of a cloud web based platform which will integrate quantitative perfusion analysis algorithms previously developed within the group. The platform will provide an efficient method of uploading DICOM files and then viewing and exploring the quantitative perfusion results in any modern web browser.

Create a demonstration of the platform to be shown at the Society of Cardiovascular Magnetic Resonance (SCMR) conference talk, specifically regarding the future of quantitative perfusion analysis to gather potential users.

Work Plan

The cloud based platform will be coded in python/flask using a secure mySQL database to host the files. Custom HTML/CSS/JavaScript will be written for the project to present the content in an intuitive view. A walkthrough demonstration of the platform will be recorded which will be shown at the SCMR talk. The timeline for the project is as follows:

- **Jan 2018** - Implementation of algorithms in cloud based platform and preview demo created for SCMR talk.
- **Feb 2018** - Early alpha build of platform up and running, with internal testing at St Thomas'
- **March 2018** - Selected groups of consultants will test the next Beta iteration of the platform
- **April 2018** - Polished core product with possibility of adding user/log in access.

Deliverables

The final outcome will be a web based platform that consultants all over the country will be able to use and share their data easily between different NHS trusts, with plans to expand across the border. This will be the first nationwide system for consultants to analyse their CMR perfusion DICOM files quantitatively.

Evaluation

The success of the project will be whether the web platform carries out the intended function from start to finish without any bugs. Another measure of success is whether consultants end up using it as a tool for aiding the diagnostic plan. The platform should also be created in a modular fashion so future features such as new algorithms, and a user login system can be easily implemented. We expect the platform to be used for a small pilot multicentre study starting in April 2018.

Acknowledgments

I would like to thank my supervisor Dr Amedeo Chiribiri for his invaluable guidance and knowledge on CMR perfusion. I would like to especially thank Cian Scannell, PhD student, who was working alongside me, writing the python algorithms for motion correction/quantification and was always there to help if I encountered problems. Finally, I would like to mention Dr Oleg Aslanidi for his helpful advice when choosing projects.

Abstract

Cardiovascular magnetic resonance (CMR) perfusion is rapidly becoming the favoured technique to detect and quantify myocardial ischaemia, due to its non-invasive and non-ionising nature. Until recently, it was only possible to assess CMR perfusion qualitatively; however, advancements in the automation of motion correction have made it possible to quantitatively analyse CMR perfusion data in an accurate and reproducible manner. This has opened up the possibility of widespread adoption of CMR perfusion as a frontline diagnostic tool in both tertiary and non-specialised hospitals, with quantitative analysis able to predict very early signs of myocardial ischaemia before clinical symptoms develop. CoreCMR is an online cloud platform built upon Python-Flask architecture providing a robust full-automated method of standardising quantification of CMR perfusion. CoreCMR makes use of a custom-built motion correction and quantification (MCAQ) python engine developed at King's College London; it provides research fellows and consultant cardiologists with the most update method of analysing CMR perfusion images. CoreCMR will be made widely accessible across clinical research facilities and hospitals across the UK, in an attempt to standardise and increase the popularity of CMR perfusion in the diagnosis of myocardial ischaemia.

Table of Contents

Table of Figures

1 Introduction

1.1 Background

Cardiovascular disease (CVD) is the leading cause of death worldwide, accounting for 26% of all deaths in the United Kingdom (UK) in 2016 (1). This places a tremendous burden on the UK economy, with an estimated total cost of £19 billion each year. Coronary heart disease (CHD) is the most common form of CVD, where a build-up of fatty substance known as atheroma leads to a narrowing of coronary arteries. This restricts the supply of oxygen and nutrients from the blood to the heart, resulting in a condition known as myocardial ischaemia (2). Patients with this condition display symptoms of pain and discomfort within the heart, classified as angina.

If left untreated CHD increases the risk of a blockage within the coronary arteries, which could lead to localised cell death of causing a myocardial infarction (heart attack) (3). As a result, treatment of CHD is incredibly time dependent. For effective treatment, it is vital that there are efficient and quick techniques to image CHD. Currently a range of different techniques of imaging CHD are employed, including Positron Emission Tomography (PET), Computed Tomography (CT) and Magnetic Resonance Imaging (MRI) (4).

MRI is sets itself apart from the other techniques as it is the only modality to use non-ionising radiation, of magnetic fields and radio waves, to produce structural and functional images (5). This non-invasive procedure has progressed considerably since

it's invention in 1973 by Lauterbaur & Mansfield (6), and a specialised branch known as Cardiac Magnetic Resonance (CMR) focuses on imaging the cardiovascular system. One CMR technique in particular assesses perfusion which is the flow of blood normalised by tissue mass.

CMR perfusion is carried out by injecting a patient intravenously with a bolus of Gadolinium-based contrast agent (7). Immediately after which two series of images are taken in three short-axis slices (basal, mid and apical), one while the patient is under pharmacological stress and the other during rest. These images are all saved in the digital imaging and communication in medicine (DICOM) format.

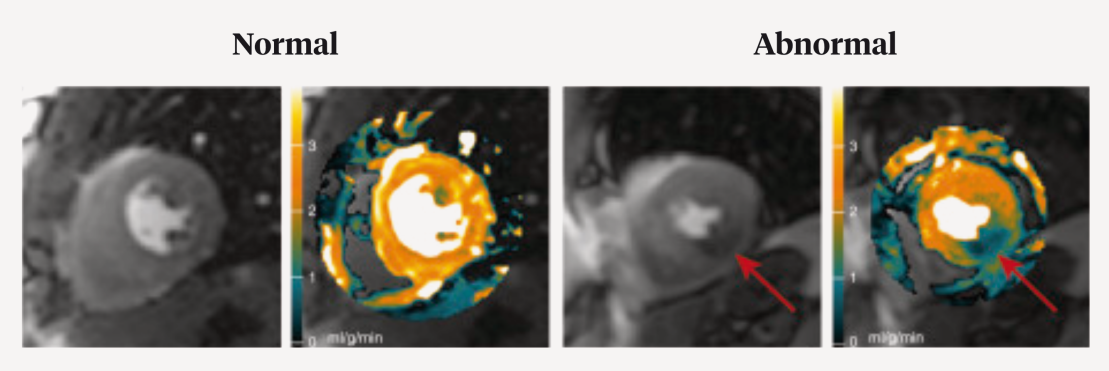

Figure 1 *Normal and abnormal CMR mid slices and perfusion maps during stress adapted from (8)*

As shown in Figure 1, the brighter areas in the CMR slices correspond to higher concentrations of gadolinium. In healthy patients, the distribution of gadolinium within coronary arteries should be evenly distributed. However, in patients with CHD, due to the narrowing of the arteries, it is possible to see a hypo intense perfusion defect (9). This demonstrates how CMR perfusion is a vital diagnostic

technique, as it can be used to identify any obvious signs of a coronary occlusion before noticeable clinical symptoms appear.

Current clinical analysis of CMR perfusion image series is done by eye, looking at the temporal and spatial distribution of the contrast agent. Although this has been proven to provide a good level of diagnostic accuracy (10), the complex qualitative nature of this assessment depends on the competence and training of the assessor as these images regularly suffer from dark-rim MRI artefacts which can be confused for perfusion defects (11). The extensive training required and risks of misinterpretation are hindering the wide-spread adoption of CMR perfusion. As such, CMR perfusion is predominately assessed qualitatively only in experienced tertiary centres.

If these problems can be overcome, CMR perfusion has the potential for widespread adoption, especially in local hospitals, bringing with it a host of benefits over other imaging techniques from PET perfusion and to a lesser extent single-photon emission computed tomography (SPECT) and CT perfusion. CMR perfusion has a greater spatial resolution than these other techniques, providing opportunity for a more accurate assessment and diagnosis (12). As mentioned previously, CMR perfusion is the only modality which uses non-ionising radiation and is therefore safer for the patient. This comes with the added bonus of being able to monitor patients more closely, as additional scans can be taken without any increased risk to patient welfare, except for those with claustrophobia. The current gold standard for diagnosing CHD is coronary angiography, however it has a severe limitation being an invasive procedure requiring the use of catheters (13).

These overwhelming benefits of CMR perfusion motivate the need to provide a better, cheaper, diagnostic tool to analyse these scans; a quantitative method. Techniques for quantitative evaluation were proposed over 20 years ago (14). Quantification of a CMR perfusion image series only requires the value of the arterial input function (AIF) and the concentration of contrast agent present in the myocardial tissue. However, this method has one significant downfall. Accurate quantification takes place on a pixel-wise level, requiring a patient to remain completely still during scanning. CMR scans on average take 60s (15), much longer than a patient can hold their breath. Current clinical practice tries to reduce movement as much as possible by asking a patient to hold their breath initially and then breathe shallowly. However even this slight movement within the scans causes the quantification method to become ineffective (16).

Over the last few years new methods have been developed to accurately correct for this motion autonomously (17), creating the possibility of efficient quantification analysis in a clinical setting. This brings an immediate benefit for patients during scanning as they can now relax and breathe normally. The scans, now assessed by an objective algorithm rather than by eye, mitigate the risk of dark rim MRI artefacts producing false positives for CHD (18). This simplifies and speeds up analysis of CMR perfusion, with a quantitative method no longer requiring extensive training. Quantification also has the added advantage of detecting a disease with a global reduction in flow, e.g. a multi-vessel obstructive microvascular disease (19). A global reduction in flow is otherwise undetectable qualitatively, as visually the contrast agent would be spatially distributed in a uniform manner.

As a result of these breakthroughs a few early fully automated quantification solutions have recently been developed. One solution is a series of programs which can be installed on MRI scanners, giving them the additional ability to quantitatively process CMR perfusion images immediately after scanning (20). Less than 15 scanners currently have this technology worldwide, and it is limited to the Siemen's brand. Another drawback of this method is that CMR perfusion scans can only be processed on the fly and not retrospectively.

An alternative solution solves this issue and is a fully automated universal framework capable of quantifying large-scale data sets of CMR perfusion image series on generic desktop computers (8). Hsu et al. recorded competitive and reproducible results, creating myocardial blood flow (MBF) maps from CMR perfusion data in less than 50 seconds. However, the computer used for testing was still a high-performance system with an Intel Core i7-6950X 3.0-GHz CPU using a 64-bit Microsoft Windows 10 operation system. It remains unclear whether results will be fast and consistent on other underpowered systems such as laptops, as it is currently not available for use in the public domain.

Current quantification solutions are still hampered by their research focus, with no standardised platform, which would be needed for a wider clinical and research adoption. Local software solutions are limited to either the brand of MRI scanner or to the speed of the machine that are running it. Local software solutions also require users to install and download packages, not providing an easy way for users to share data with each other.

An ideal solution would be a cloud based platform hosted on heavy duty servers, with a user interface accessible from the world wide web. As processing would be carried out server side rather than client side, this would provide the fastest CMR quantification solution to users regardless of their device, be it a fully rigged desktop computer or an outdated laptop. A cloud platform comes with the added advantage of easily allowing users to securely share CMR perfusion data with each other, if they wish to, without having to download any propriety software. For example, consultants may wish to seek another opinion on a particular patient, and this gives them an easy opportunity to anonymise the data and ask for advice.

1.2 Objectives

The main objective of this project is to create a minimum viable product (MVP) that will demonstrate the potential of a cloud based CMR perfusion analysis system. Even though this may be an MVP, it was designed from the outset to be a well-rounded platform taking into account all aspects of web development from an intuitive user experience (UX) frontend to efficient and secure processes in the backend. It will initially be used by research fellows and consultant cardiologists within King's College London with plans to expand in the future upon validation. To ensure that the final MVP platform meets these standards, a few important proof points, assessing security, accessibility and scalability were set out at the start of this project.

Accessibility

CoreCMR must be accessible. The platform needs to be lightweight, with an optimised footprint for users to access it on a web browser from any device. The system will be used by a variety of research fellows and consultants and therefore needs to be intuitive and easy to navigate, catering for everyones needs. The user interface of the platform will be carefully planned with User Experience (UX) mockups created in wireframe and design software such as Affinity Designer.

Security

CoreCMR must be secure. It is recommended that CoreCMR has a login functionality ensuring that only registered users can access the platform and patient DICOM information. Upon registration, users will have to accept the privacy policy and terms of CoreCMR, to be in line with the EU's General Data Protection Regulation (GDPR) act, which will come into action 25 May 2018. Every time patient data is processed it should be given a unique secure ID of some kind, for a user to easily and securely share data with authorised colleagues. All processes should be SSL-secure ready for deployment onto a cloud server such as Heroku.

Scalability

CoreCMR must be scalable. CoreCMR will eventually handle and process large volumes of data, submitted by multiple users. It is proposed that the 'CoreCMR' solution will be based upon the Flask architecture, a flexible and scalable micro-framework with support for running python scripts in the background where the

motion correction and quantification calculations will take place. This MVP of CoreCMR should serve as a base for more features and processing modules to be added in the future; with code well commented and neatly structured in a modular fashion.

2 Theory

2.1 Framework

CoreCMR is built upon the Flask micro-framework. Flask was created by Admin Ronacher in 2010; based upon the Werkzeug toolkit and Jinja2 template engine (21). It is labelled as a micro rather than a full framework as it is a lightweight package containing the bare minimum needed for web development, rather than providing a full package. This allows for full flexibility during development, not needing to rely on particular inbuilt modules, having very inclusive support for third party extensions. This is perfect for CoreCMR which makes uses of a host of additional open source libraries. There are other newer and potentially faster micro-frameworks available, such as Quart, however they are less stable and are not as well documented as Flask (22).

2.2 Motion Correction & Quantification Engine

CoreCMR makes uses of a robust custom motion correction and quantification (MCAQ) engine written in python and developed in parallel at King's College London. It was designed to be robust, suitable for analysing CMR perfusion in a clinical setting.

Traditional approaches of motion correction use non-rigid image registration techniques applied retrospectively which involves choosing a reference image for each frame. However, these methods fail in a CMR perfusion context due to their

inability to account for dynamic contrast enhancement. Hamy et al (23), found a way around this by separating contrast enhancement from dynamic contrast-enhanced MRI bowel and prostate image series, using robust principal component analysis (RPCA). RPCA has been previously used in video surveillance to detect the foreground of images (24). In a similar manner Scannell at King's College London found that this method could be effectively applied to a full image series of myocardial perfusion data as shown in Figure 2, identifying the bulk motion caused by respiration (25).

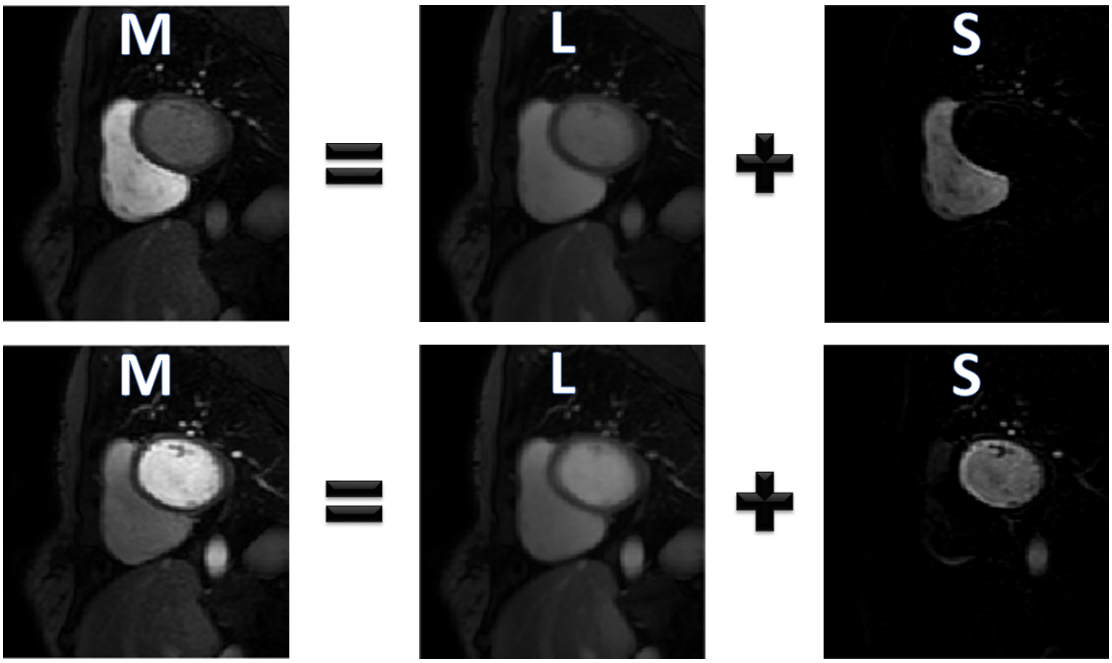

Figure 2 *Demonstration of RPCA in action in myocardial perfusion data. Original data matrix, M, is split into low-rank, L, and sparse, S, components (21)*

RPCA can then be combined with a traditional PCA approach to eradicate the motion between frames, creating a synthetic motionless series of DICOM images. The MCAQ engine was written to save this series as a downloadable DICOM series

for each slice (mid, apical and basal), as well as outputting a instant 'cine-loop' preview video for each slice to display on the CoreCMR results webpage. This instant preview makes use of the 'ffmpeg' open source library to convert the DICOMs into a continuous looping mp4 video file (26).

The motionless series of DICOM images are then forwarded to the quantification part of the MCAQ engine. Quantitative analysis makes use of the indicator-dilution theory (27), which models the myocardium with a singular input and output, assuming that there is a linear relationship between the concentration of Gadolinium contrast ($C(t)$ in mmol $\cdot$ ml^{-1}) and the signal intensity (SI) (28). This ultimately gives the law of conservation of indicator mass, with MBF in units of ml $\cdot$ min$^{-1}g^{-1}$.

$$\frac{dC_{myo}}{dt}(t) = \text{MBF} \cdot \big(C_{in}(t) - C_{out}(t)\big)$$

The time that a particle (i.e Gadolinium contrast) spends inside the system is known as the transit time, however the transit time is not a fixed value as there are many different paths that a particle can take. To fix this a probability distribution of transit times $h(t)$ is used, giving rise to the residue function, R_F:

$$R_F(t) = \text{MBF} \cdot R(t) = \text{MBF} \cdot (1 - \int_0^t h(\tau)d\tau)$$

Initially, the transit time is equal to zero, therefore R must be equal to one, and is a positive, decreasing function. This allows MBF to be quantified through the calculation of $R_F(t)$,

$$\text{MBF} = R_F(0) = \max\{R_F(t)\}$$

The MCAQ engine's motion correction is to such a high standard that is possible to quantify individual voxels of the CMR perfusion image series, making full use of the high spatial resolution of MRI. This translates in the MCAQ engine as a downloadable CMR perfusion map for each slice, generated in .jpeg or DICOM format with a preview on the results page of CoreCMR.

3 Method

3.1 Preparation

In September 2017, the original CoreCMR.org website was an old DICOM database owned by King's College London accessible from the internet. The new CoreCMR cloud platform was planned to replace this database inheriting the name. However, it would take months until an MVP would be finished ready to replace the outdated original. Therefore, a new CoreCMR landing page was coded within a week, to act as a placeholder showing a tiny glimpse of what was to come. For a screenshot please refer to Figure 3. It had an integrated email sign up form to garner and establish any possible external interest. Another benefit of this landing page was that it boosts CoreCMR.org's Search Engine Optimisation (SEO); the online visibility of a website in a web search engine (e.g. Google, Bing) unpaid 'organic' results (29). This raises CoreCMR.org's credibility and reputation, rather than being an empty unused space.

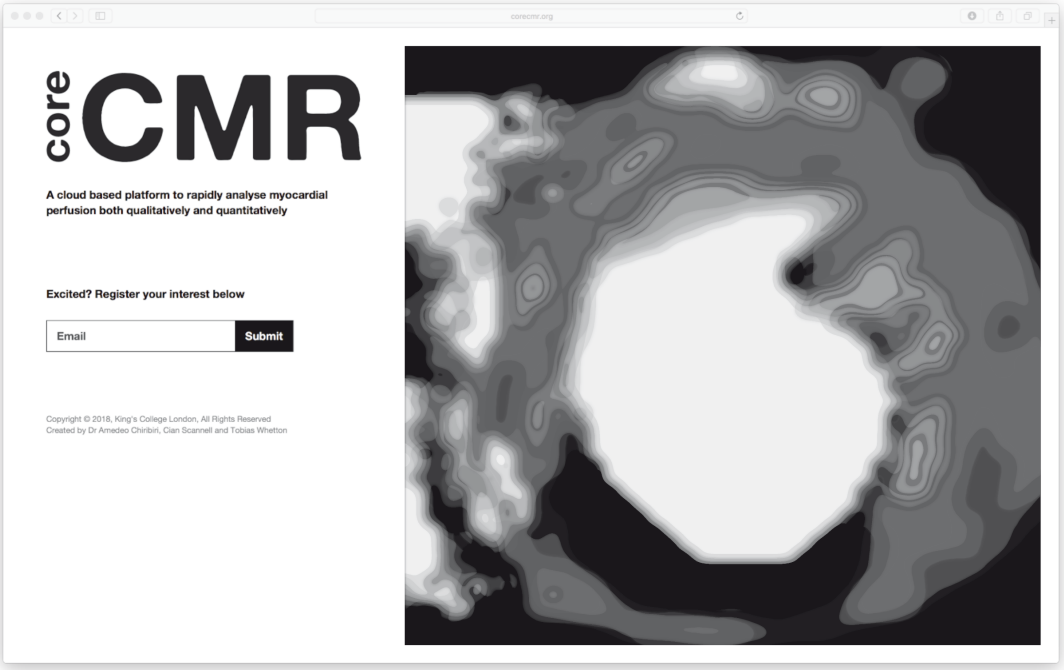

Figure 3 *A screenshot of the original CoreCMR landing page*

3.2 Setup

The first step was to create a folder titled CoreCMR locally, serving as the root directory of the project, or more commonly known as the repository. A service known as Git was then employed, for version control to monitor and track different changes (30). To take full advantage of this, the local repository was cloned into a private repository in the cloud hosted by the 'Github' service. During the course of development this allowed CoreCMR and then MCAQ engine to be updated in parallel.

Another tool 'virtualenv', was installed allowing the creation of a bundle of software inside the repository, isolating it from the wider local system. This is known as forming a virtual environment and it makes it easy to specify which Python binary to use and which dependencies are needed. CoreCMR was chosen to be developed in the newer Python 3 (released in 2008) over Python 2 (released in 2000) due to its cleaner codebase and guaranteed support in the future (31). A Python 3 virtual environment was then setup inside the repository, under the name *env/* using 'Pip', a system for easy install and management of software packages written in Python (32). Once the virtual environment was activated, Flask was then installed inside again using the Pip service.

The first python script written was *run.py*. This contains the Python-Flask instructions on how to start and configure a local development server, for continuous testing of the platform during development. Flask has a useful inbuilt debugging tool, and this was activated by adding a `debug=True` argument to the `app.run` function within *run.py*, as shown below:

```
app.run(debug=True)
```

Due to the complex nature of CoreCMR, the application was broken down and organised into a logical group of inter-connected modules, known as packages. These packages were housed in the appropriately named *app/* folder in the repository. This is modular approach of splitting the code into more manageable segments is good practice; increasing the potential for scalability, and encouraging code reuse and

efficiency. A ‘skeleton’ package structure was created, where each file has very few lines of code to set out its future role and purpose; but won’t be ’filled in’ until further down the pipeline. The hierarchy of the repository can be found below:

```
run.py
app/
  __init__.py
  views.py
  config.py
  models.py
  forms.py
  external/
  static/
  assets/
  templates/
requirements.txt
```

Within the *app/* path, Flask required a script to initialise the CoreCMR application bringing all the components together, and this was called *__init.py__*. The interactions of all of the different modules, would be defined later in a separate script called *views.py*, and the models that the application would use would be defined in another appropriately named script *models.py*. In *models.py* empty classes, such as `User`, were promptly written for definition later on.

All of the configuration information of CoreCMR was planned to be located in *config.py*, the only script not included in version control. Here a `Config` class was defined containing a secret key to encrypt sessions and cookies. In order to send

information from the user to the backend of the system, forms would be used and they would be classified in *forms.py*; with empty classes for login and registration created in preparation.

Moving on from the scripts, folders were added to the application structure. */external/* would be the path for the python scripts for the motion correction and quantification modules. */static/* would be the public directory for all of the stylesheets, javascript, fonts and images. */assets/* would be the location of pre-processed scripts. And finally, */templates/* is where templates for the website will be located.

Moving back up the hierarchy, *requirements.txt* a text file is written to list all of the required external open source packages for the CoreCMR. It serves as an easy way to monitor and install what versions the virtual environment is using. Please refer to the appendix for an exhaustive list of all the additional packages required.

The last stage of the setup process was to create a *README.md* file, in the markdown (.md) text format. This contains information about the CoreCMR repository, and a step-by-step installation guide to help any future developers working on CoreCMR to quickly get accustomed to the inner workings of the platform.

README.md

CoreCMR

About

This is the private repository for CoreCMR, cloud based quantitative analysis of myocardial perfusion CMR. Built using Flask, SQLite, Numpy, Scikit, SASS and pure HTML/CSS/JS.

The platform is currently being developed by Tobias Whetton (BEng student) and Cian Scannell (PhD student) at King's College London; under the supervision of Dr Amedeo Chiribiri, Consultant Cardiologist at King's College London.

First-time Set Up

Please follow the following steps to set up the installation locally on your computer:

1. Clone the repository using `git clone https://github.com/cianmscannell/CoreCMR`
2. Load into repository using `cd CoreCMR`
3. Install `virtualenv` using `pip install virtualenv`
4. Set up the virtual environment `virtualenv env`
5. Load into the virtual environment using `source env/bin/activate`
6. Install the flask library in this virtual environment with `pip install flask`
7. Install the requirements `pip install -r requirements.txt`
8. Install `ffmpeg` using `brew install ffmpeg`

Database Setup

1. In terminal, tell flask how to import python app `export FLASK_APP=run.py` , if using windows use `set` instead of `export`
2. In terminal, initialise the database `flask db init`
3. With every database change, `flask db upgrade`

Running CoreCMR

1. Load into the CoreCMR folder using, `cd CoreCMR`
2. Enter the virtual environment with `source env/bin/activate`
3. Run CoreCMR `./run.py` on OS X/Linux/Cygwin or `$ flask\Scripts\python run.py` on Windows
4. Visit `http://127.0.0.1:5000/` in your browser

Figure 4 *A screenshot of the Readme.md file*

3.3 Design

Once the 'skeleton' application structure of CoreCMR was finished, it was now important to consider the uniform resource identifier (URI) structure. In other words, how the function of CoreCMR would be segmented across different web pages. URIs are the same as uniform resource locators (URL), except they define the endpoint or path (e.g. `/login`) whereas URLs define the whole address (e.g. `www.corecmr.org/login`) (33). Mock-ups of this URI structure were hand drawn (see Figure 5) from expert advice from the eventual users of the platform; researchers and consultant cardiologists within the Guy's and St Thomas' NHS Trust Foundation. It was decided for simplicity and ease of use that there were three key pages; 'Login/Registration', 'Home' and 'Results'.

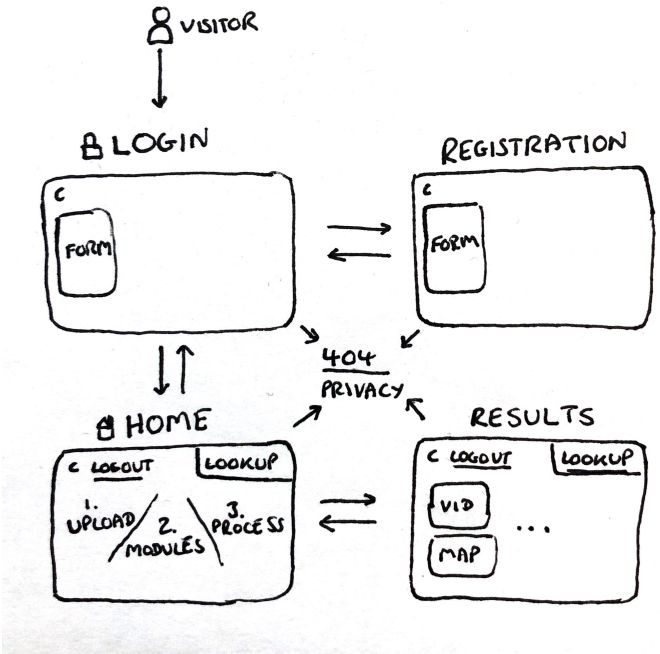

Figure 5 *A photograph of an early sketch of the CoreCMR structure*

1. **Login/Registration** for visitors to sign in or create a CoreCMR user account, accepting the terms and privacy policy.
2. **Home** where users could upload and process their DICOMs, as well as search for previously processed data.
3. **Results**, where users could preview/download their motion corrected DICOMs and CMR perfusion maps.

Other aspects of the website's structure were also considered in the layout, including privacy policy and 404 error page. Before starting to code and implement this URI structure, the user experience (UX) was mocked up to visually see how everything fits together. It was important that this step was carried out early, as it is a lot easier to change the design before it is being committed to code, reducing the risk of any problems further down the line.

Affinity Designer, a professional UX concept software was used to create a minimalist but functional design (34), as seen in Figure 6. When designing these pages, it was important to consider that the main users would be consultant cardiologists and research fellows on a variety of devices from laptops to mobiles. This meant that the designs had to be responsive, meaning that they would adapt to different screen sizes.

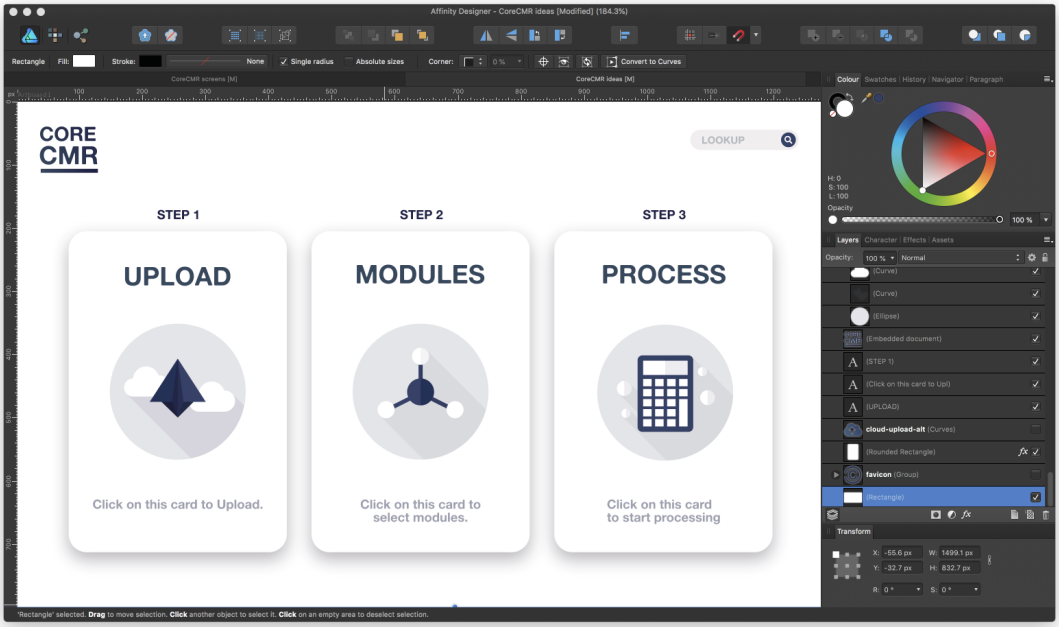

Figure 6 *Using Affinity Designer to create the UX design mock-ups*

A lot of thought was put into how the system could be segmented over the three pages, keeping as many elements as possible in the same locations. From a developer's perspective, this encouraged greater code reuse and task efficiency. And from a user's perspective this reduces the learning curve of understanding how to interact with the platform.

These mock-ups were not just plans, an added benefit was that any part of them could be exported into scalable vector graphics (SVG), an XML-based image format built for the web (35). As a result, the CoreCMR logo and associated icons, which were designed in house were exported for use later in templates.

3.4 Templates

Now the UX mock-ups had to be recreated and coded in Hypertext Markup Language (HTML). The first part of this process was to write a series of HTML templates in *app/templates/*. Conveniently Flask has an inbuilt templating engine called Jinja2, which allows for a HTML inheritance structure, as well as for more powerful functions such as for-loops. When a user requests a specific URL, these HTML templates are processed on the backend and then served to the user as a single HTML file. The template structure for CoreCMR was implemented as follows:

```
base.html
login.html
register.html
index.html
results.html
privacy.html
404.html
blocks/
  lookup.html
svg/
```

First the parent template, *base.html* was written to define the generalised structure mapped out in the UX mockups. Within this template, Jinja2's `{% block %}` function was used to set out the locations which can be populated by the remaining child *.html* templates. These child templates were written to inherit from the parent template by adding `{% extends "base.html" %}` in their first line.

The *.html* files in the template hierarchy were all given self-explanatory names except perhaps for the *404.html* page. A HTTP 404 error occurs when the server cannot find a page requested by the user (36). The *404.html* template was used to inform the user of this and gives them the option to return to the homepage. A *blocks/* path was added for single elements that are repeated across a selection of templates. In CoreCMR system there was only one such use case and this was the 'lookup ID search bar' that would be implemented so users could find previously processed data. To avoid having to repeatedly edit the same html code in two different locations, this *lookup.html* block was created.

The final element in the *templates/* directory is the *svg/* path. Instead of using `<img>` html tags to display SVG files exported from Affinity Designer, it was far more convenient to display them natively using Jinja2's function `{% include 'svg/logo.svg' %}`. This gives a developer greater control over the images, with the ability to change their colour for example using HTML code (35).

To ensure all of the templates were engineered to be responsive, an external lightweight grid system framework 'Gridlex' was used, to quickly form columns and rows of content that adapt to the device of the user (37). It is incredibly simple but versatile system that works by wrapping HTML containers in `.col` classes within a flexible box `.grid` class.

To style the templates, the syntactically awesome stylesheet (SASS) language were used (38). But unlike the templates, only one SASS stylesheet was written, called *base.scss*. As the CoreCMR website was planned to have a consistent style, a lot of

elements could inherit the same stylings from each other, leaving no need for individual stylesheets for each template.

3.5 Routes

Routes are the different URLs that an application implements, and they are defined in Flask with a series of 'view functions'. Each function is mapped to a independent URL, and when visited by a user, carries out specific tasks. One such task is to render a single HTML file from the series of templates created previously. All of these 'view functions' were written in *views.py*.

The simplest and most generic view function, was the definition of the user homepage, i.e. when a user visits the root (/) or index (/index) url. This definition uses the inbuilt Flask function `render_template()` to substitute all of `{{ ... }}` blocks with real values, collating the 'family tree' of HTML templates into one single master *index.html* file with 'Home' as its title.

```
@app.route('/')
@app.route('/index')
def index():
  return render_template('index.html', title='Home')
```

The most important view function was the definition of `'/process'` route, `@app.route('/process', methods=['GET', 'POST'])`. This is where the uploaded DICOMs undergo processing using the MCAQ engine. This function was

written to check whether the uploaded files were indeed DICOMs, and then organised them before inputting them in the MCAQ engine located in *app/external/*. Processing the CMR perfusion image series take some time, and so a loading screen was written in *index.html* for users to understand that the application is still working and hasn't crashed. The loading screen consisted of a looping notched circle with processing messages underneath, with a screenshot provided in Figure 7.

***Figure 7** A snippet of the loading screen*

The MCAQ engine returned cine-loop motion corrected videos of mid, apical and basal slices, with corresponding perfusion maps and they were displayed together on the '/results' URL path, defined by another view function. A .zip file containing all of the input/output data of the MCAQ engine was also generated for users to download for further analysis.

Notice how the `'/process'` has an additional `methods` argument. This was defined so files uploaded by the user would be 'sent' to the view function via the `POST` request method. A corresponding HTML form was then written in *index.html* template to

serve as the input, with the input field for the file upload shown in the code block below.

```
<input type="file" id="file" name="file" multiple=""
accept=".dcm" oninvalid="alert('Please upload DICOM files
before processing');" required/>
```

Throughout the development of CoreCMR, thought was put in to ensure that the platform was as smart and fool-proof as possible. This input field is a great example as it will only accept DICOM files to be uploaded and alerts a user if they forgetting to upload the DICOMs before clicking to begin processing.

3.6 Security

As the platform will be handling sensitive patient data, it is of utmost importance that it is all secure, only accessible to registered users. To add login functionality a library known as 'Flask-Login' was setup in combination with a more advanced form extension 'WTForms' and a database manager 'SQLite' to store usernames and passwords (39) (40). All the passwords were hashed to ensure that were securely encrypted.

Once this security framework was setup, `@login_required` was added into all of the view functions apart from login, registration and privacy policy pages. This route

catches guest users denying them access to any possible sensitive content, redirecting them to the login screen and prompting them to register.

```
@app.route('/')
@app.route('/index')
@login_required
def index():
  return render_template('index.html', title='Home')
```

An issue with the `'/process'` view function was that it would overwrite the previous results every time it was called. At this point, there was no method of saving previous uploaded files, output videos and perfusion maps. Another problem was that any user would be able to see the last uploaded DICOM files under the '/results' URL. New variables, `result_id` and `result_path`, were defined to solve this.

```
while True:
  result_id = ''.join(random.choices('ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890',k=6))
  result_path = os.path.join(app.config['RESULTS_PATH'], result_id)
  if os.path.exists(result_path):
     result_id = ''.join(random.choices('ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890',k=6))
     result_path = os.path.join(app.config['RESULTS_PATH'], result_id)
  os.makedirs(result_path)
  break
```

Python 3 has an inbuilt psuedo-random function, `random.choices()`, and this was used to create and assign a new secure six-character alphanumeric ID code to DICOMs every time they were processed. This `result_id`, was then combined with location information to create a `result_path` variable, which would define the path

to a folder titled with the ID number within *app/static/results*. So, for example, an ID #23A9SB might be generated and this would create a path *app/static/results/23A9SB* to store all of the uploaded files and processed data.

This ID number acts as a simple and secure way of managing the different DICOM upload data, with 2.17 trillion combinations. It also serves as easy method for consultants to share data with each other and look up previously processed dataset instantly. To enable this function the *lookup.html* block was finally coded with Jinja2, making use of a secure form to check whether a result ID is valid or not.

3.7 Testing

Before pushing live to a development server, a few elements were changed. It is important to consider that multiple users will log on and process DICOMs at the same time, this phenomenon is known as threading (41). Fortunately, Flask has a simple way to achieve this by adding the additional keyword argument `threaded=True` in `run.py`:

```
app.run(debug=True, threaded=True)
```

The first deployment for testing was done on a Mac Mini server within the confines of St Thomas' Hospital's LAN. Installation of CoreCMR on the Mac Mini was simple, only needing to pull the CoreCMR repository from Github and downloading

all the packages from *requirements.txt*. The setup tutorial in *README.md* file came in use here, ensuring that no installation steps were forgotten.

As this particular Mac Mini server was hosting other services used by clinical research fellows, it is important that CoreCMR would not interfere with these tasks. And so additional arguments were added to the `app.run` function, changing the port from the default `5000` to `1234`, and editing `host='0.0.0.0'`. This allows the server to be publicly accessible through the host's IP address on the Local Area Network (LAN), through the 1234 port, as follows, *10.0.1.134:1234/*.

```
app.run(debug=True, host='0.0.0.0', threaded=True, port=1234)
```

The creation of a favicon was the finishing touch to the platform, with the two C characters in the CoreCMR logo modified in Affinity Designer and exported into a 32x32 png format. This was inserted into the *base.html* template, with the following code,

```
<link rel="shortcut icon" type="image/png" href="{{
url_for('static', filename='img/favicon.png') }}"/>
```

4 Results

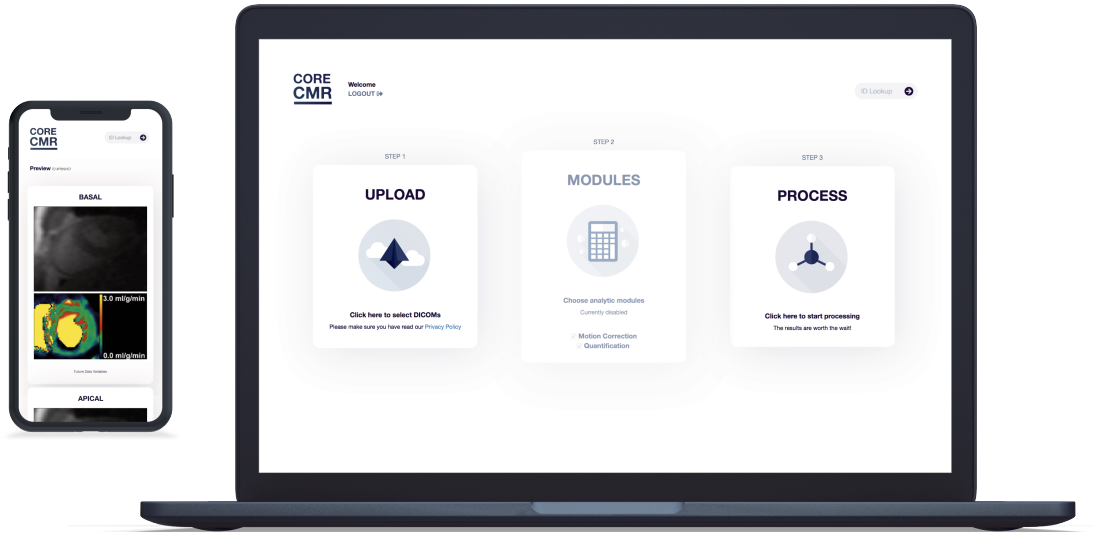

Figure 8 *The finished CoreCMR cloud platform running on mobile and laptop devices*

This MVP of CoreCMR is a resounding success, already in use by clinical research fellows on the local network, on devices from laptops to mobiles (as shown in Figure 8). It is able to handle multiple users logging on and quantitatively analysing CMR perfusion images in a quick and efficient manner; with processing times for motion correction and quantification of CMR perfusion image series averaging at a competitive 45 seconds (for each slice). Users have also found it easy to securely share processed DICOM data between each other, with the unique alphanumeric ID code. A brief walkthrough of the platform is provided with screenshots below.

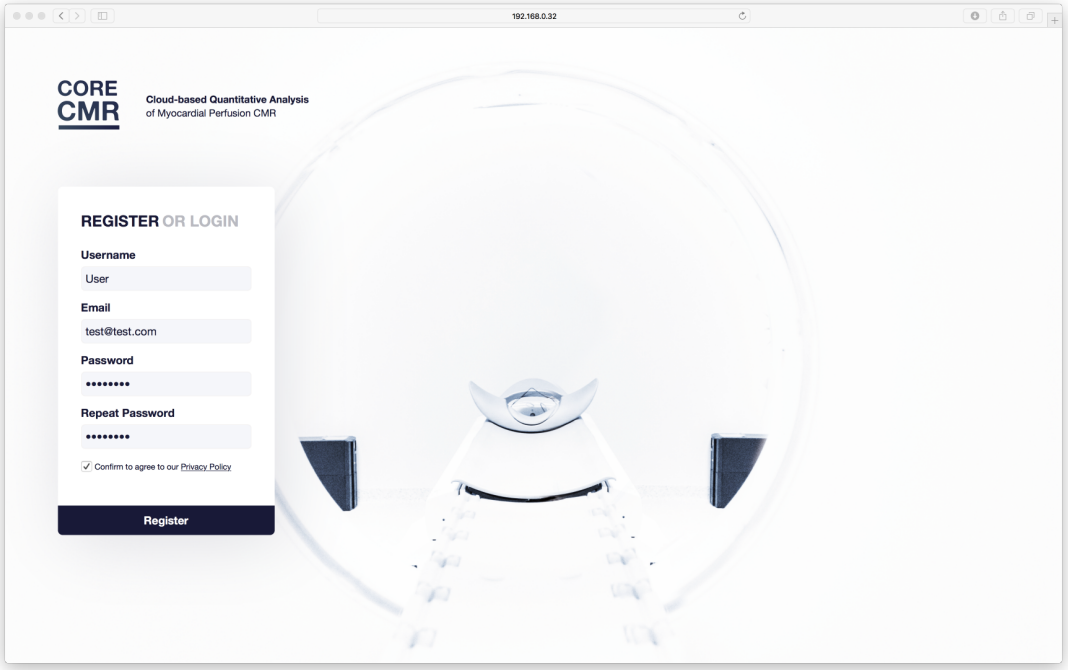

Figure 9 *A screenshot of the CoreCMR registration page*

Upon navigating to CoreCMR, visitors are prompted to register and log on before they are able to use the platform, as shown in Figure 9. Here they must accept the privacy policy and terms and conditions, fulfilling the requirement for GDPR. A link is provided to a separate privacy policy page, with a long document detailing how data is used by the CoreCMR platform.

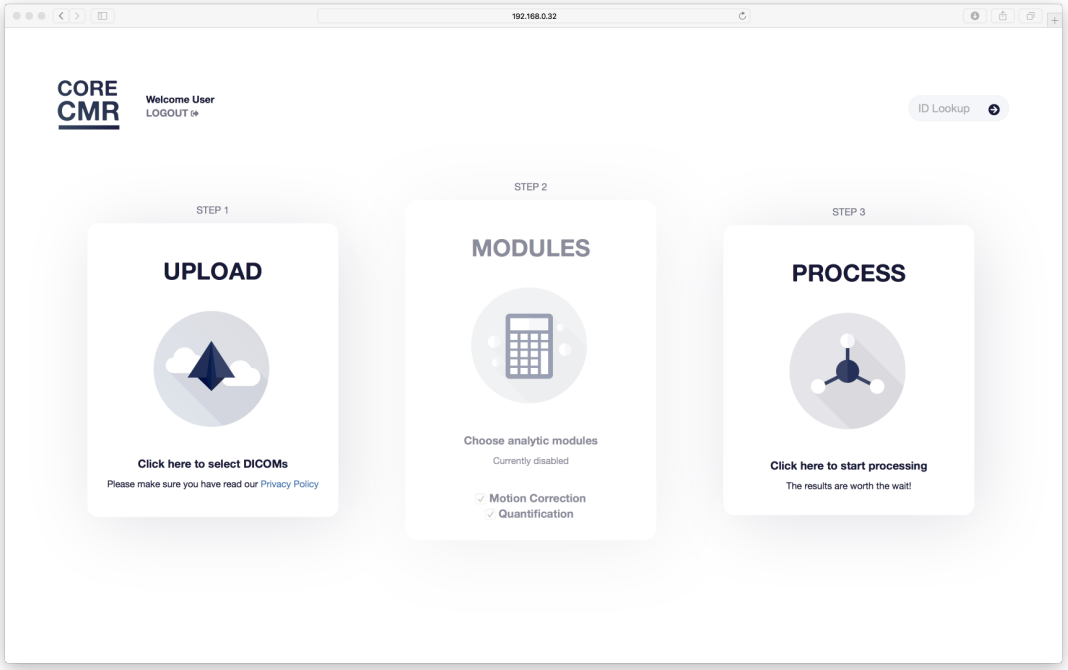

Figure 10 *A screenshot of the CoreCMR homepage*

Upon logging into the platform users are directed to the homepage, where a three-step guide walks them through how they can upload their own files, as demonstrated in Figure 10. Once uploaded, users can click on the process card, upon which a loading screen will appear until their results are ready. Returning users can lookup pre-processed CMR perfusion data by entering the unique six-character alphanumeric ID code in the top right-hand lookup bar.

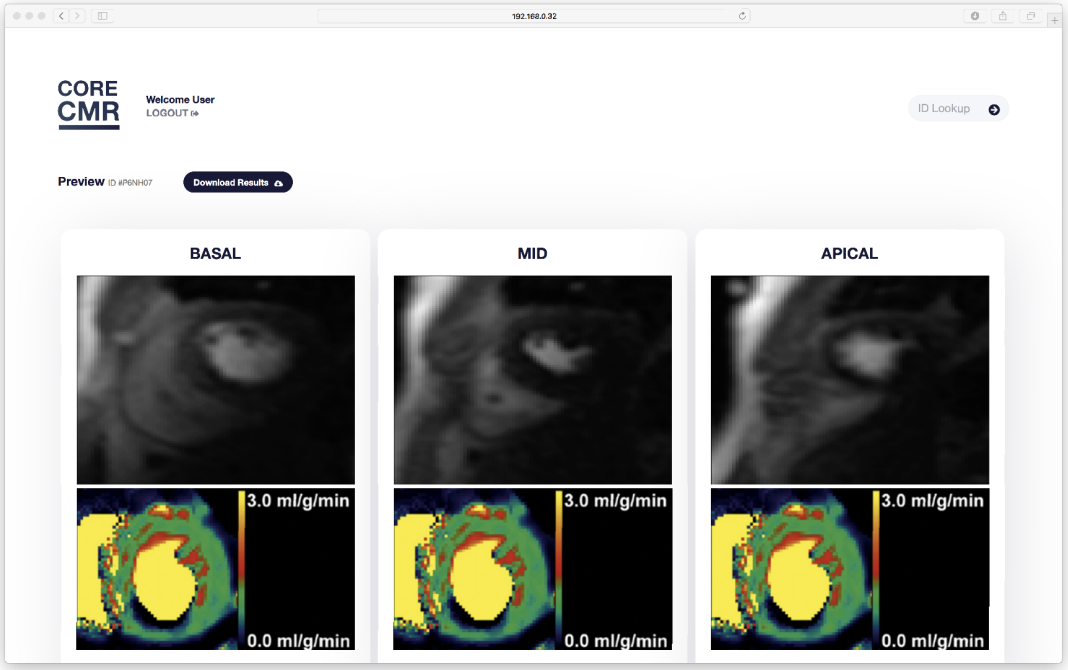

Figure 11 *A screenshot of the CoreCMR results page*

Once processed the CMR perfusion image series are automatically segmented into the apical, mid and basal slices with cine-loop videos displayed for an instant preview of motion corrected images along with quantitative perfusion maps. All of this information is available to download as a .zip file using the blue 'Download Results' button preview provided, as shown in the screenshot in Figure 11.

5 Discussion

5.1 Outcome

The final CoreCMR platform fulfilled the specifications as outlined in the project plan. All the aims were met or surpassed in accordance with the timeline, apart from a few minor points. The project plan outlines how a demonstration of CoreCMR would be shown at the Society of Cardiovascular Magnetic Resonance (SCMR) conference talks. It was decided that a CoreCMR demo would not be presented at this year's SCMR conference, as in hindsight CoreCMR was too premature, and was at risk of exposing the idea before it was fully developed.

Due to the limited amount of time available, the MVP of CoreCMR was not tested by consultants as much as hoped, with it only being finished in April. However, considering that no knowledge of Python 3 and SQLite languages was known at the start of the project it is impressive that a fully working platform was finished on time, including user and log in access. Git served as a great method to visualise the progress, as it recorded the updates to the repository, with a screenshot of this timeline in Figure 12. The Flask debugging tool, was incredibly helpful in getting used to the Python language, identifying problems within the code. The platform did not deter much from the original mockups, which shows that the rigid planning structure was successful and effective.

Figure 12 *The commit timeline since 27/01/2018 for the CoreCMR Github repository*

CoreCMR offers some great immediate benefits to patients, because of its inbuilt motion correction engine, patients no longer need to hold their breath and can breathe normally within the scanner, making them more comfortable. It also offers immediate benefits for consultants and researchers, allowing them to quickly and accurately see motion corrected CMR perfusion images with associated perfusion maps, without needing to be an expert on the topic or develop their own code. Even though there are some great immediate benefits, there are still some limitations of the platform in its current state.

5.2 Performance

The current CoreCMR MVP server can only handle low lever traffic with no more than 10 concurrent users, as it is using the built-in Flask server. However, if the Mac Mini was upgraded to a performance web server, such as UWSGI or Gunicorn, it would be possible to get up to at least 100 concurrent users with the code in its current state (42). For live production on CoreCMR.org, a full Web Server Gateway Interface (WSGI) should be employed such as Google App Engine, or Heroku, perhaps in combination with Amazon S3 or Azure. Outsourcing the processing tasks of the MCAQ engine to a professional cloud server, in theory will drastically cut down processing time and CoreCMR has the potential to be the world leader in speed compared to other solutions on the market.

The inbuilt task queue for threading in Flask is sufficient for the CoreCMR testing environment in the St Thomas Hospital LAN. However, in a live production of CoreCMR, it might be beneficial to use an open source distributed task queue, such as Celery (43). This will increase the real-time operation and speeding up the processing of DICOMs if there are a large number of active users at the same time. Celery also supports asynchronous tasks, so in the future DICOMs can be processed in the background platform and the user can visit other pages of the CoreCMR or submit more DICOMs to be processed while waiting.

Aside from this, the backend of CoreCMR has an incredibly modular structure built with future scalability in mind. It has been made easy for new processing engines using alternative quantitative algorithms to be included in the platform by adding their packages in the *external/* path. With some engines already in development and plans to include them as soon as ready, ensuring that clinicians are kept up to date with the latest research in CMR perfusion. The database system chosen for CoreCMR was SQLite due it being an incredibly fast and efficient database. A huge advantage is that it is self-contained in one single file, *app.db* making it extremely portable and reliable. This was all that was needed for the MVP as it is only being used to store and manage user data. However, this self-containment comes with a downside, SQLite could not be used to store the data processed by users. As SQLite only allows one single write operation to take place at any given time, so it would be an inefficient method with a limited throughput. This is one reason why the SQLAlchemy library is being used; an object relational model (ORM) built to bridge the gap between the object-oriented nature of Python and the schema of a database

(44). The main benefit of SQLAlchemy is that it easily allows a developer to switch the database backend of CoreCMR in the future if needs be.

In the end, it was decided that the most effective way of storing data would be to create a new folder under *app/static/results/* named with the associated ID code every time `'process'` is requested. This is beneficial for testing and using CoreCMR on a small scale, as developers and research fellows who have access to the CoreCMR backend can easily drag and drop files they need onto their computer. However, a drawback is that once CoreCMR starts being used by a greater population, the number of folders might eventually become unmanageable. A more robust and scalable method would have been to save all data both user and results data in a PostgreSQL database, however this is a lot more complex and is overkill at these early stages. Nevertheless, inactive classes for DICOMs and results data were added in *models.py* in case PostgreSQL will be used in the future.

5.3 Security

The platform was built to be very secure, requiring users to register and log on before use, with all passwords securely hashed on the *app.db* SQLite database. Having said that once a client has logged in, currently there is no protection against automated brute force attacks. CoreCMR was only built with security in mind of the confines of the St Thomas Hospital private LAN. It was assumed that only authorised research fellows and consultants would have access to the network and so could immediately register and log on to use the system.

A malicious user could in theory try and data-mine previously processed DICOM files, and could loop through all 2.17 trillion combinations of the ID codes associated with previously processed data. Although extremely unlikely, CoreCMR could be protected against this by employing the reCAPTCHA service (45), to verify a user's identity, preventing attacks from a bot network. An alternative solution to this would be to lock users out of their accounts if they look up an invalid ID code five times in a row.

Another interesting point to note about the ID codes is that the six-character alphanumeric code generated is not truly random, but pseudo-random. The `random.choice` function uses a mathematical algorithm to produce the random numbers and therefore in theory could be predicted (46). A better means of producing random ID's would have been to use an unpredictable physical means to generate the numbers, such as atmospheric noise. This true random technique is employed for hashing user passwords. However, assuming that logged in users aren't malicious, the current method of ID generation is satisfactory.

One more measure taken to enhance security was to use 'WTForms' over standard HTML web forms. 'WTForms' sanitisises inputs, preventing any possible structured query language (SQL) injection-style attacks. This was especially important for logins. For enhanced security, an admin user role could have been defined with an onboard queue added in the sign-up process. This extra step in the signup process means that admins would need to accept a user's registration before they could sign on to the platform.

In the first official release of the CoreCMR platform onto the internet, a step should be taken to secure the connection between the user and the server. Let's Encrypt, a free Certificate Authority service, could be used to setup and autonomously maintain a Transport Layer Security (TLS) encryption, providing a secure 256-bit HTTPS encryption channel between the user and the server.

5.4 Functionality

Current user functionality is simple but limited. Users can only log on, use the platform and log off. If more time was available, profile pages could have been added, allowing users to track all of the scans that they have processed in one single location. A profile page could also serve as a good way for users to get in touch with each other, with the possibility of integrating contact links for emails. A settings page could also be added so that users could reset their password, and change their username. The reason why a reset password function was not included in the MVP was that it requires an email address and external client such as Sendgrid to automate emails (47). As data is not tied to a user's account currently, they can always register with another email address or request for their old account to be deleted on the database so they can re-register with a new password.

The Jinja2 templating engine, is great for the current implementation of CoreCMR where only static content (images and videos) is shown to the user; with Jinja2 being very fast. However, Jinja2 can be limiting. In the future CoreCMR might want to be used in a more interactive manner, manipulating DICOM files live on the platform, and at this point it might be best to port the frontend into the React framework (48).

Porting to React would allow for a more powerful interface with additional support for native Windows, Mac or mobile applications, using React Native. Therefore, users would have greater flexibility, with the possibility of downloading a CoreCMR application on their device, to view and even processing DICOMs offline.

Templates were styled with SASS and not raw CSS, as SASS gave a lot more flexibility and power over the styling of CoreCMR with the ability to create variables within the SASS document for colours, macros and offering shortcuts for adding child styles (33).

5.5 Browser Standards and Error Handling

One issue with web development is that browser standards are not all the same. The most popular browsers, Safari, Chrome and Firefox all interpret code in a different manner from each other (49). As a result, UX elements may appear in slightly different locations from each other, and certain JavaScript/CSS events are not supported. CoreCMR was consistently tested in different browsers throughout the development process, however sometimes the UX appearance may be affected but the functionality remains the same so these differences are ignored. One instance of this is the JavaScript `onclick` event called when the user clicks the process DICOM button. This initialises the loading screen flawlessly on the Chrome browser, however this feature does not seem to be officially supported on Safari; only functioning half of the time.

The loading screen on the whole was designed well, however it could be considered a bit simple. It might have been better to create another separate *loading.html* template. This would firstly resolve the previous issue of using a JavaScript `onclick` event, and then during processing information could be passed to the user with the addition of a progress bar. If used in combination with a task queue the loading bar could in fact just be a separate block in the corner of the web page like *lookup.html* allowing the user to navigate to other parts of the site during processing.

As the MVP version of CoreCMR is only being tested internally on a secure local server, error handling was not a the first priority, however steps were still taken to reduce any possible data leaks and errors. *config.html* was stored in the local environment and was excluded out of version control using `git ignore`. This was done because if version control would ever be comprised so would all of the sessions and cookies. A HTTP 404 error page, *404.html* was written, so if users visit a URL that doesn't exist, CoreCMR will inform of this and give them the option to return to the homepage. 404 is the only likely error that will occur on this early build of CoreCMR. It is unlikely that other HTTP errors such as 500 internal server error and 403 forbidden error will be encountered; so, for simplicity's sake they were ignored.

Even though no errors were found during extensive final testing, this does not mean that none might occur in the future. There is still a very tiny risk that a secure application programming interface (API) key could be leaked to a random person. To avoid this, especially if CoreCMR is deployed on a publicly available server, a functioning error handler should be written; when the application throws an

exception, a JavaScript Object Notation (JSON) doc is returned instead of a backtrace returning code including potentially sensitive information.

5.6 Conclusion

CoreCMR is an outstanding proof of concept, demonstrating how a competitive and standardised solution to CMR perfusion analysis can be achieved in the form of a cloud-based platform. However, it is still in its infancy, with room and need to expand its tools and features. The positive feedback from research fellows within the confines of St Thomas' Hospital show great potential, but a true testament to its success will be whether it will be adopted by other research hospitals across the UK.

6 Future Work

CoreCMR will open up a host of opportunities regarding future work. As the platform becomes more popular, it will provide an ever-expanding database of CMR perfusion images. This rich source of CMR perfusion image data, which will be very useful for any new research on myocardial ischaemia. It has already been discussed that the CoreCMR platform could potentially integrate additional machine learning capabilities that will improve its quantitative data outputs, using a neural network.

Currently only motion correction and quantitative data modules developed by King's College London are installed on the CoreCMR system. However, there are possibilities to collaborate with other research teams; allowing them to install their own custom modules. This could provide an objective way to compare and contrast the different methods of motion correction and quantification of CMR perfusion images; hopefully find a golden standard. On top of this other processing modules such as 4D flow CMR could be added onto the platform, giving rise to even more complex analysis.

CoreCMR sets an example, that it is now possible for analysis of DICOM images to move from local computers to cloud servers. Similar platforms could be created for other imaging modalities, such as PET and CT, serving as an easy way to share and process any DICOM images. In in the far future, a delocalised platform almost like a social network could be created, for consultants to anonymise, process and analyse data – sharing and discussing it with other colleagues in the profession all around the UK.

8 References

1. Dicks E. BHF Cardiovascular Disease Statistics - UK Factsheet. 2018;: 1–11.

2. Barth J, Schumacher M, Herrmann-Lingen C. Depression as a risk factor for mortality in patients with coronary heart disease: a meta-analysis. *Psychosomatic medicine*. [Online] Psychosomatic Medicine; 2004;66(6): 802–813. Available from: doi:10.1097/01.psy.0000146332.53619.b2

3. Spaan J, Kolyva C, van den Wijngaard J, Wee ter R, van Horssen P, Piek J, et al. Coronary structure and perfusion in health and disease. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*. [Online] The Royal Society; 2008;366(1878): 3137–3153. Available from: doi:10.1098/rsta.2008.0075

4. Newell MC, Schwartz RS, Lesser JR. Noninvasive Coronary Artery Imaging with CT and MRI. In: Vlodaver Z, Wilson RF, Garry DJ (eds.) *Coronary Heart Disease*. [Online] Boston, MA: Springer US; 2011. pp. 83–95. Available from: doi:10.1007/978-1-4614-1475-9_5

5. Barkhausen J. Ischemic Heart Disease, MRI. *Encyclopedia of Diagnostic Imaging*. [Online] Berlin, Heidelberg: Springer, Berlin, Heidelberg; 2008. pp. 1016–1020. Available from: doi:10.1007/978-3-540-35280-8_1327

6. Mansfield P. *Mansfield, Peter: A Personal View of My Involvement in the Development of NMR and the Conception and Development of MRI*. [Online] Chichester, UK: John Wiley & Sons, Ltd; 2007. 1 p. Available from: doi:10.1002/9780470034590.emrhp0119

7. Arai AE, Hsu L-Y. Myocardial Perfusion Using First-Pass Gadolinium-Enhanced Cardiac Magnetic Resonance. *Cardiovascular Magnetic Resonance Imaging*. [Online] Totowa, NJ: Humana Press; 2008. pp. 313–329. Available from: doi:10.1007/978-1-59745-306-6_15

8. Hsu L-Y, Jacobs M, Benovoy M, Ta AD, Conn HM, Winkler S, et al. Diagnostic Performance of Fully Automated Pixel-Wise Quantitative Myocardial Perfusion Imaging by Cardiovascular Magnetic Resonance. *JACC: Cardiovascular Imaging*. [Online] 2018. Available from: doi:10.1016/j.jcmg.2018.01.005

9. Coelho-Filho OR, Rickers C, Kwong RY, Jerosch-Herold M. MR Myocardial Perfusion Imaging. *Radiology*. [Online] Radiological Society of North America, Inc; 2013;266(3): 701–715. Available from: doi:10.1148/radiol.12110918

10. Barkhausen JR, Hunold P, Jochims M, Debatin JRF. Imaging of myocardial perfusion with magnetic resonance. *Journal of Magnetic Resonance Imaging*. [Online] Wiley-Blackwell; 2004;19(6): 750–757. Available from: doi:10.1002/jmri.20073

11. Greenwood JP. *Clinical evaluation of Magnetic Resonance imaging in Coronary heart disease*. [Online] 2012. Available from: doi:10.1186/ISRCTN77246133

12. Jerosch-Herold M. Quantification of myocardial perfusion by cardiovascular magnetic resonance. *Journal of Cardiovascular Magnetic Resonance*. [Online] BioMed Central; 2010;12(1): 57. Available from: doi:10.1186/1532-429X-12-57

13. van der Wall EE. Myocardial perfusion imaging in coronary artery disease: SPECT, PET or CMR? *Netherlands Heart Journal*. [Online] Bohn Stafleu van Loghum; 2012;20(7-8): 297–298. Available from: doi:10.1007/s12471-012-0300-z

14. Wilke N, Simm C, Zhang J, Ellermann J, Ya X, Merkle H, et al. Contrast-enhanced first pass myocardial perfusion imaging: Correlation between myocardial blood flow in dogs at rest and during hyperemia. *Magnetic Resonance in Medicine*. [Online] Wiley-Blackwell; 1993;29(4): 485–497. Available from: doi:10.1002/mrm.1910290410

15. Breeuwer M, Quist M, Spreeuwers L, Paetsch I, Al-Saadi N, Nagel E. Towards automatic quantitative analysis of cardiac MR perfusion images. *International Congress Series*. [Online] 2001;1230: 967–973. Available from: doi:10.1016/S0531-5131(01)00163-7

16. Kramer CM, Chandrashekhar Y, Narula J. CMR-Based Quantitative Myocardial Perfusion. *JACC: Cardiovascular Imaging*. [Online] 2012;5(2): 237–238. Available from: doi:10.1016/j.jcmg.2012.01.004

17. Milles J, van der Geest RJ, Jerosch-Herold M, Reiber JHC, Lelieveldt BPF. Fully Automated Motion Correction in First-Pass Myocardial Perfusion MR Image Sequences. *IEEE Transactions on Medical Imaging*. [Online] 2008;27(11): 1611–1621. Available from: doi:10.1109/TMI.2008.928918

18. Hazel RD, Reichek N, Wang Y. Quantitative evaluation of the dark rim artifact in cardiac perfusion images. *Journal of Cardiovascular Magnetic Resonance*. [Online] BioMed Central; 2009;11(Suppl 1): P243. Available from: doi:10.1186/1532-429X-11-S1-P243

19. Liu A, Wijesurendra R, M Liu J, C Forfar J, M Channon K, Neubauer S, et al. 113 Novel diagnostic criterion for microvascular ischaemia using cmr perfusion imaging: validation against invasive index of microvascular resistance. *Heart*. [Online] BMJ Publishing Group Ltd and British Cardiovascular Society; 2017;103(Suppl 5): A85–A86. Available from: doi:10.1136/heartjnl-2017-311726.112

20. Kellman P, Hansen MS, Nielles-Vallespin S, Nickander J, Themudo R, Ugander M, et al. Myocardial perfusion cardiovascular magnetic resonance: optimized dual sequence and reconstruction for quantification. *Journal of Cardiovascular Magnetic Resonance*. [Online] Journal of Cardiovascular Magnetic Resonance; 2017;: 1–14. Available from: doi:10.1186/s12968-017-0355-5

21. Ronacher A. *Flask Quickstart Documentation*. [Online] flask.pocoo.org. Available from: http://flask.pocoo.org/docs/0.12/quickstart/ [Accessed: 25 April 2018]

22. Jones P. *Quart*. [Online] gitlab.com. Available from: https://gitlab.com/pgjones/quart [Accessed: 26 April 2018]

23. Hamy V, Dikaios N, Punwani S, Melbourne A, Latifoltojar A, Makanyanga J, et al. Respiratory motion correction in dynamic MRI using robust data decomposition registration – Application to DCE-MRI. *Medical Image Analysis*. [Online] 2014;18(2): 301–313. Available from: doi:10.1016/j.media.2013.10.016

24. Sobral A, Bouwmans T, ZahZah E-H. Double-constrained RPCA based on saliency maps for foreground detection in automated maritime surveillance. IEEE; 2015. pp. 1–6. Available from: doi:10.1109/AVSS.2015.7301753

25. Scannell CM. Robust Motion Correction of Free-Breathing Myocardial Perfusion MRI Data. 2017;: 1–38.

26. Bellard F. *FFMPEG*. [Online] ffmpeg.org. Available from: https://www.ffmpeg.org [Accessed: 26 April 2018]

27. Zierler KL. Theoretical Basis of Indicator-Dilution Methods For Measuring Flow and Volume. *Circulation Research*. [Online] American Heart Association, Inc; 1962;10(3): 393–407. Available from: doi:10.1161/01.RES.10.3.393

28. Kellman P, Arai AE. Imaging Sequences for First Pass Perfusion - A Review. *Journal of Cardiovascular Magnetic Resonance*. [Online] 2007;9(3): 525–537. Available from: doi:10.1080/10976640601187604

29. Samuel S. Search engine optimisation to improve your visibility online. *In Practice*. [Online] British Medical Journal Publishing Group; 2013;35(6): 346–349. Available from: doi:10.1136/inp.f2703

30. Willmore FT, Jankowski E, Colina C, editors. Developing with Git and Github. *Introduction to Scientific and Technical Computing*. [Online] Taylor & Francis Group, 6000 Broken Sound Parkway NW, Suite 300, Boca Raton, FL 33487-2742: CRC Press; 2016. pp. 39–53. Available from: doi:10.1201/9781315382395-4

31. Hill C. Introduction. *Learning Scientific Programming with Python*. [Online] Cambridge: Cambridge University Press; 2016. pp. 1–7. Available from: doi:10.1017/CBO9781139871754.001

32. Python Software Foundation. *Pip (package manager)*. [Online] pypi.org. Available from: https://pypi.org/project/pip/ [Accessed: 26 April 2018]

33. Masinter L. *The 'data' URL scheme*. [Online] RFC Editor, 1998 Aug. Available from: doi:10.17487/rfc2397

34. Affinity Designer - Professional graphic design software. Available from: https://affinity.serif.com/en-gb/designer/

35. Neumann A. Scalable Vector Graphics (SVG). *Encyclopedia of GIS*. [Online] Cham: Springer, Cham; 2017. pp. 1831–1840. Available from: doi:10.1007/978-3-319-17885-1_1159

36. Koehler W. Web page change and persistence?A four-year longitudinal study. *Journal of the American Society for Information Science and Technology*. [Online] Wiley-Blackwell; 2002;53(2): 162–171. Available from: doi:10.1002/asi.10018

37. G L. *Gridlex*. [Online] gridlex.devlint.fr. Available from: http://gridlex.devlint.fr [Accessed: 26 April 2018]

38. Catlin H, Weizenbaum N, Eppstein C. *SASS*. [Online] sass-lang.com. Available from: https://sass-lang.com [Accessed: 26 April 2018]

39. Team W. *WTForms*. [Online] readthedocs.io. Available from: http://wtforms.readthedocs.io/ [Accessed: 26 April 2018]

40. Hipp DR. *SQLite*. [Online] sqlite.org. Available from: http://www.sqlite.org/ [Accessed: 26 April 2018]

41. Goerzen J. Threading. *Foundations of Python Network Programming*. [Online] Berkeley, CA: Apress; 2004. pp. 443–468. Available from: doi:10.1007/978-1-4302-0752-8_21

42. Chesneau B. *Gunicorn*. [Online] docs.gunicorn.org. Available from: http://docs.gunicorn.org/ [Accessed: 26 April 2018]

43. Solem A. *Celery: Distributed Task Queue*. [Online] celeryproject.org. Available from: http://www.celeryproject.org [Accessed: 26 April 2018]

44. Bayer M. *SQLalchemy: Object Relational Model*. [Online] Available from: http://docs.sqlalchemy.org/en/latest/ [Accessed: 26 April 2018]

45. *reCAPTCHA*. [Online] google.com. Available from: https://www.google.com/recaptcha [Accessed: 26 April 2018]

46. Konyagin S, Shparlinski I. Prediction of Pseudo-Random Number Generators. *Character Sums with Exponential Functions and their Applications*. [Online] Cambridge: Cambridge University Press; 2009. pp. 91–98. Available from: doi:10.1017/CBO9780511542930.012

47. Saldana I, Jenkins T, Lopez J. *Sendgrid*. [Online] sendgrid.com. Available from: http://sendgrid.com [Accessed: 26 April 2018]

48. React - A JavaScript library for building user interfaces. Available from: https://reactjs.org

49. Deveria A. *Browser Compatability Table*. [Online] caniuse.com. Available from: http://caniuse.com [Accessed: 26 April 2018]

9 Appendix

9.1 Demonstration & Further Information

The local server at St Thomas' Hospital is now permanently up and running on the wired Local Area Network and can be accessed by visiting *10.0.1.134:1234/* using any web browser, although Google Chrome is recommended.

If you do not have access to the server to view and test CoreCMR, or if you would like access to the repository and the full code behind the platform, please email tobiaswhetton@hotmail.com for assistance. Thank you.

9.2 Requirements.txt

requirements.txt

```
alembic==0.9.7
blinker==1.4
certifi==2018.1.18
chardet==3.0.4
click==6.7
cssmin==0.2.0
cycler==0.10.0
Cython==0.26.1
decorator==4.1.2
dominate==2.3.1
Flask==0.12.2
Flask-Assets==0.12
Flask-Login==0.4.0
Flask-Mail==0.9.1
Flask-Migrate==2.1.1
Flask-SQLAlchemy==2.2
Flask-WTF==0.14.2
idna==2.6
itsdangerous==0.24
Jinja2==2.9.6
Mako==1.0.7
MarkupSafe==1.0
matplotlib==2.0.2
networkx==2.0
nibabel==2.2.0
numpy==1.13.3
olefile==0.44
parameterized==0.6.1
Pillow==4.3.0
pydicom==1.0.2
pyelastix==1.1
pymaxflow==1.2.4
pyparsing==2.2.0
python-dateutil==2.6.1
python-editor==1.0.3
python-http-client==3.0.0
pytz==2017.2
PyWavelets==0.5.2
psycopg2==2.7.3.2
requests==2.18.4
sass==2.3
scikit-image==0.13.1
scikit-learn==0.18.1
scipy==0.19.0
SimpleITK==1.0.1
simplejson==3.13.2
six==1.11.0
SQLAlchemy==1.2.2
ujson==1.35
urllib3==1.22
visitor==0.1.3
webassets==0.12.1
Werkzeug==0.14.1
WTForms==2.1
WTForms-SQLAlchemy==0.1
```

Figure 13 *The comprehensive list of additional packages used by CoreCMR, located in requirements.txt*

GRAYSCALE PERFUSION MAP