UKRAINIAN CATHOLIC UNIVERSITY

BACHELOR THESIS

Development of a visualization system for neural network results on medical images

Author:
Andrii BOROVETS

Supervisor:
Oles DOBOSEVYCH

A thesis submitted in fulfillment of the requirements
for the degree of Bachelor of Science

in the

Department of Computer Sciences
Faculty of Applied Sciences

Lviv 2020

Declaration of Authorship

I, Andrii BOROVETS, declare that this thesis titled, "Development of a visualization system for neural network results on medical images" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Bachelor of Science

Development of a visualization system for neural network results on medical images

by Andrii BOROVETS

Abstract

Heart stenosis corresponds to a frequent problem with the heart arteries when the valve is getting too narrow and does not open properly. The flaps of a valve may become thicker or bound together, and as a result, preventing the valve from fully opening. This problem can be detected and treated, if diagnosed in time. To determine if patients' conditions are critical or not – may require a significant amount of time of the doctor.

Diagnosis timing is one of the most critical factors with this particular disease, frequently determining the outcome for the patient. Considering the current general medical service practice, there is no apparent way to create a prioritized patients list, so that patients with more critical or urgent conditions could be treated first.

In this project I worked on a web application that would accept the MPR images from a user(doctor) and feed these images to a Neural network algorithm that would predict the state of the patient and prioritize the list of patients for the doctor. The app also has a very intuitive and self-explanatory probability chart to facilitate doctor's review of each patient's case.

Altogether, the app is expected to address the problem of making urgent stenosis cases apparent to the doctor.

Acknowledgements

I am grateful to Oles Dobosevych for all the help with the thesis and overall help during my student years.

Also I would like to say thanks to Bohdan Petryshak and Mariia Dobko for giving me an opportunity to take part in this project.

Contents

List of Figures

List of Abbreviations

CT	**C**omputed **T**omography
DOM	**D**ocument **O**bject **M**odel
JSON	**J**ava **S**cript **O**bject **N**otation
HTML	**H**yper **T**ext **M**arkup **L**anguage
CSS	**C**ascading **S**tyle **S**heets
JS	**J**ava**S**cript
TS	**T**ype**S**cript
XML	**E**xtensible **M**arkup **L**anguage
JSX	**J**ava**S**cript **X**ML
TSX	**T**ype**S**cript **X**ML
API	**A**pplication **P**rogramming **I**nterface
JWT	**J**SON **W**eb **T**oken
CNN	**C**onvolutional **N**eural **N**etwork
MPR	**M**ultiplanar **R**econstruction
ES6	**E**CMA**S**cript **6**

Chapter 1

Introduction

1.1 Problem statement

The project was started by a group of students at ML UCU laboratory in 2019, and while I was looking for my bachelor thesis topic, I learned that the project team was looking for a front end developer. As a coincidence, I was doing a business analysis of a much smaller but similar project in 2018 within the Business thinking course, and while working half time as a front end developer at a small startup in Lviv, the project team's offer was exactly what I was looking for.

The Idea of the project is to help doctors and people with Atherosclerosis - (sometimes called "hardening" or "clogging" of the arteries) is the buildup of cholesterol and fatty deposits (called plaques) on the inner walls of the arteries. These plaques can restrict blood flow to the heart muscle by physically clogging the artery or by causing abnormal artery tone and function.

Every third death in the world is caused by cardiovascular diseases. The most common reason for these fatalities is coronary heart disease (impaired blood flow to the heart muscle). One of the most common methods to detect coronary insufficiency is a CT Coronarography. (more info – Cleveland Clinic Website)

The problem with the procedure of CT Coronarography is that it generates large sets of images (1000 – 1500 per one patient), and each doctor usually has to handle a dozen of patients a day. Both of these make it not feasible for a doctor to analyze all the data in a reasonable time, even if doctors would spend most of their working day just reviewing CT images. A side effect of the problem with this practice is that there is a chance that patients with critical conditions may be at the end of the queue, and by the time when the doctor will check their CT Coronarography images it might be too late for treatment. The rational solution is to deploy an algorithm to prioritize the patients, so the doctor would start to analyze the CT images for patients that are more likely to have critical conditions, it is - the ones that require immediate actions.

1.2 Solution that is being explored within the project

There is a group of people that are developing a neural network algorithm to solve this problem. Basic idea – the algorithm can be trained, based on the pre-analysed sets of images and past patient's data, to analyse large sets of patient's CT images and make predictions of the level of stenosis on each important area of the coronary tree for the current patient. This way the doctor would be provided with predictions of the general patient's state of coronary artery, and can immediately focus on treating the patient with the likelihood of coronary disease.

The UI of the project has to provide interface for all the user interactions with the algorithm, plus some service functions, including:

- in general, simple and easy to use interface for a multi-user application.
- the interface to handle authorization and authentication, so that each doctor gets access to their own profile and see the list of the patients that the particular doctor is working with
- browse and sort the list of all the patients in the doctor's database
- in-detail view of for the patient's record
- instantly indicate the predicted state for the patients
- upload of the CT Coronarography images of the patient
- a quick look at the images that correspond to some problems with the coronary arteries, so that the doctor can quickly verify that the patient is really in critical condition.
- as there is a general policy that a doctor cannot publish patient's data, there would be a way to identify the patient by patient MRN (Medical record number)
- exceptions handling and more

Chapter 2

Technologies

2.1 React

React is a modern and sophisticated JavaScript library to build user interfaces for web applications. It is developed and actively maintained by Facebook. React application code is made of fragments called components. Each component can be rendered in the DOM. The React components can be controlled by 'props', which can be passed to a component while rendering the page, which helps to reuse components by passing different data/styles to them.

There are two types of React components:

- the functional components - look like the default JavaScript functions that return the JSX,
- the class components – are declared using ES6.

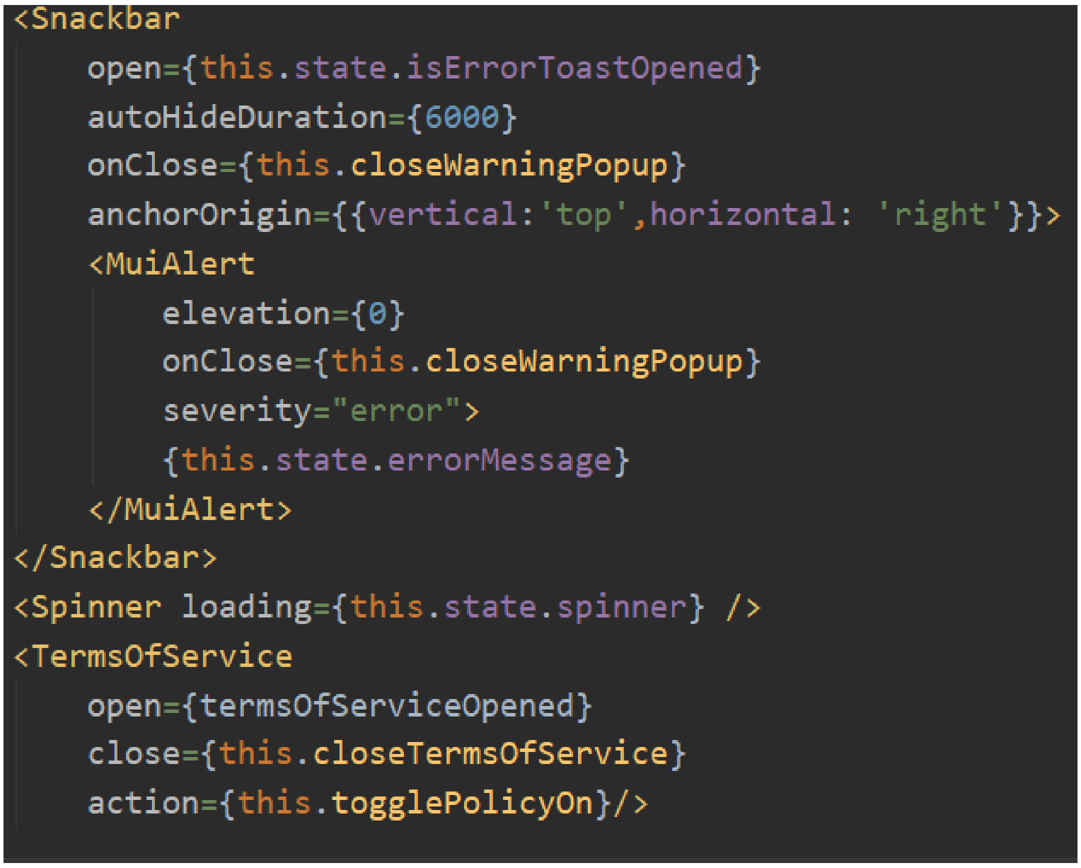
The main difference between class-based components and functional components is that the class components have state, which allows a developer to save different variables within a component and dynamically re-render the component when the state variable is changed. With the new React releases, the class base components are becoming out of date and replaced with React hooks, which allow to have a state in functional components. Furthermore, the class components have a lifecycle methods, these methods helped to perform some actions when the component was about to be rendered, after the component was rendered, when component will unmount, when it is just updated. Here is the list of some of the most used lifecycle methods:

- componentDidMount - this method will execute after everything is rendered
- componentWillMount - this method will execute before the rendering, can be used to prepare some visual data
- componentDidUpdate - this method is executed every time the state of this class component is changed
- componentWillUnmount - this method will be called right before the component will be removed from the DOM.

All these methods got replaced with useEffect hook. Functional components are easier to use, test, and are preferred from the application's performance consideration. So I was trying to follow best practices in React while working on this project.

2.2 JSX

JSX – JavaScript XML is a JavaScript syntax extension that comes from React library. It allows to use html like code/tags to form the UI with full access to the javascript features. Below is an example of JSX code:

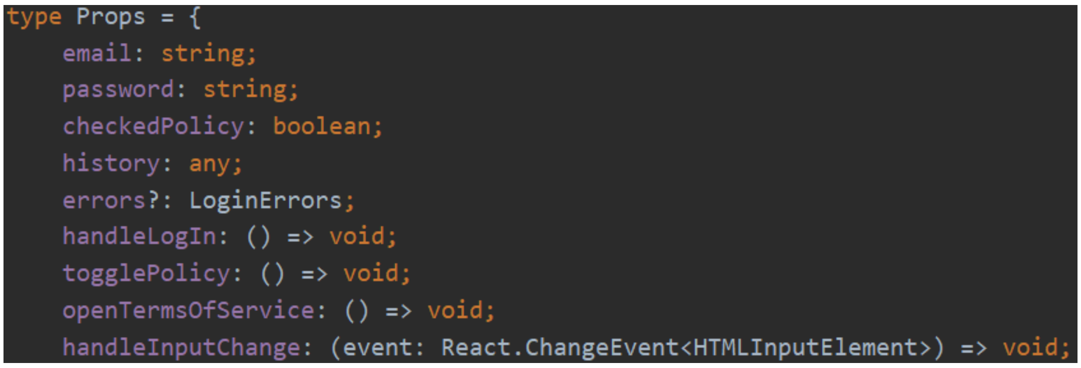


```
<Snackbar
    open={this.state.isErrorToastOpened}
    autoHideDuration={6000}
    onClose={this.closeWarningPopup}
    anchorOrigin={{vertical:'top',horizontal: 'right'}}>
    <MuiAlert
        elevation={0}
        onClose={this.closeWarningPopup}
        severity="error">
        {this.state.errorMessage}
    </MuiAlert>
</Snackbar>
<Spinner loading={this.state.spinner} />
<TermsOfService
    open={termsOfServiceOpened}
    close={this.closeTermsOfService}
    action={this.togglePolicyOn}/>
```

FIGURE 2.1: JSX syntax example

2.3 TypeScript

TypeScript - is a JavaScript extension by Microsoft, that brings types to JavaScript and makes it more controllable. It saves lots of development time and simplifies shared code development. It allows to define which props should be passed to a component and controls that the type of the props matches the desired type. This way it makes it impossible to pass invalid prop and thus saves application's debugging time and streamlines the development process. Below is an example of the props types defined:

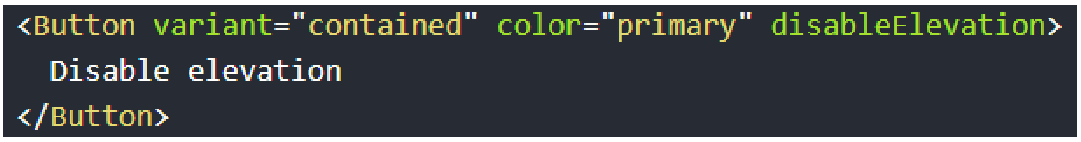


```
type Props = {
    email: string;
    password: string;
    checkedPolicy: boolean;
    history: any;
    errors?: LoginErrors;
    handleLogIn: () => void;
    togglePolicy: () => void;
    openTermsOfService: () => void;
    handleInputChange: (event: React.ChangeEvent<HTMLInputElement>) => void;
```

FIGURE 2.2: example of TypeScript syntax.

2.4 Material UI

Meterial UI - is a very popular, large library that contains build and styled React components. It makes the development process much faster and simpler. I used this library for the popups/modals, alert toasts that display errors from requests, buttons for nice styling and hover effects. Material UI is used as follows: library contains the overridden default components like buttons, paragraphs, etc and they accept props to manipulate what is rendered in the result. For example, below is a code for an application UI button:



```
<Button variant="contained" color="primary" disableElevation>
  Disable elevation
</Button>
```

FIGURE 2.3: example of material UI button.

What is different here from the regular button, the Material UI button tag is capitalized, which means that it is a component that was not an original HTML tag, plus, the props that this component can accept. Styling components from Material UI are a bit different then regular styling components. Material UI offers its own way for styling components, while the main idea is very similar to a plain css, it has the look of a json, where the keys are the class names of your choice and values are a css like structures.

2.5 D3

D3 - For the charts I have used the d3 js library, which produces dynamic and interactive data visualizations.

2.6 React Router DOM

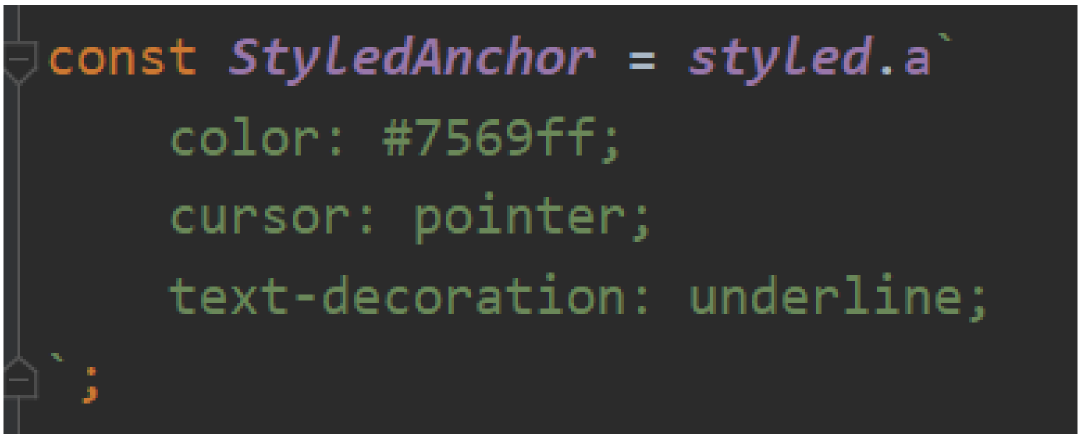
React Router DOM - This is a library that handles the changes in the path, so when the user is redirected from '../login' to '../patients' this library handles that redirect and loads the corresponding component.

2.7 React Context

React Context – this is a state management tool that allows to wrap a group of components and share a state between them. As there are lots of variables that are required among all the components. React Context allows to avoid continuous prop passing from the parent to a child component.

2.8 Styled componetns

For styling I have used styled components from '@emotion/styled' library -which provides a way to create components with styles attached to it. It allows you to create an element like <a></a> with attached CSS styles. Below is an example of a styled component:



```
const StyledAnchor = styled.a`
    color: #7569ff;
    cursor: pointer;
    text-decoration: underline;
`;
```

FIGURE 2.4: example of emotion/styled component.

2.9 Local storage

Another useful feature that I have used is called local storage. The Local storage is a kind of JSON data that is supported by the web browser, and allows you to store the key-value pairs. What makes it special is that after you save something to the local storage and refresh the page - your data will be still saved in the local storage. Even if you will restart the browser - the Local storage will still retain everything you have saved. This feature is very useful for the case when you don't want your user to authenticate every time he/she tries to load the page. You can just save the

JWT token to the Local storage after the first time a user has logged in. When the user will load the page again, you will have to check if the token is still valid and if it is - you can safely retrieve the data for the user's session.

2.10 Why React

React uses Virtual DOM – in-memory data structure, which takes the real DOM and creates a tree structure that is called the virtual DOM. One of the most important advantages of the virtual DOM over the regular one is the way it is updated. In the regular DOM everything is updated and re-rendered, while in the virtual DOM only the components that were actually changed are being updated.

In addition, React code is readable and is easy to maintain. Because of its JSX syntax it looks really similar to a plain HTML.

React uses one-direction data flow, which means that changes to a child component can't affect the parent component, so only desired components will be updated.

Another time-saving feature Facebook introduced with React is the ability to reuse anytime code components from any interface level.

Chapter 3

Workflow

3.1 Git workflow

During the whole project I was following the following industry-standard code workflow:

- for each new feature there was a separate GIT branch – called task/name-of-the-feature.
- for each bug that was found on the page – there was a bugfix/name-of-bug branch.
- after finishing the feature and making sure that there are no bugs (or fixing bug), I would open a PR (Pull request) to the master branch. This way the master branch is always ready for a release or deployment.

Ideally, there should be another branch that would be called develop that would contain all the features that are prepared for a release. The develop branch should be tested properly and after all tests will pass, then it should be merged to master. This was omitted, as I was working by myself and I was also responsible for testing.

By following this GIT workflow, one will always have an opportunity to revert the app to the state where it was working properly or to the past working stage. It also is a good technique for a case when multiple developers are working on the project, as it will avoid some conflicts that can occur otherwise. It also provides an easy to use structure to the project changes and development, with clear naming for all the branches - particularly essential for larger or distributed teams..

3.2 Project structure

For the project itself I was following Atomic Design Methodology – where each component tree is divided into 3 folders:

1. Atoms – are the simplest components, they are the smallest things and can't be decomposed more without losing their meaning. In terms of the frontend, the atoms are elements like inputs, labels, headings, buttons, etc. They are the basic elements that can't be decomposed into smaller bits.
2. Molecules – are the combinations of atoms, a simple group of elements like a label of an input or a form component, etc.
3. Organisms – are components that contain certain logic and can consist of a combination of atoms, molecules or other organisms. For example - a group of components like navigation panel, sidebar, footer, etc.

4. Pages – are the sets of the organisms, wrapped to represent the whole page layout. There are many different methodologies to structure the project, but I selected Atomic Design Methodology as it provides an excellent structure to store all project elements, is very easy to read, navigate and understand.

3.3 Project's folder architecture

- node modules – all the external libraries
- public – contains index.html, 404.html, favicon.
- Src – contains react components, api files, css files , etc.
  - Api – contains APIs for general use.
  - Features – contains all the pages.
    - home
      - components – contains atoms, molecules, organisms required for the home page.
      - Pages – uses components to build layout for the home page.
    - login
      - api – contains the function that calls the login endpoint.
      - components - contains atoms, molecules, organisms required for the login page.
      - pages - uses components to build layout for the login page.
      - helper.ts – contains the functions that help to validate a user's input.
      - interfaces.ts – contains interfaces used in the login page.
    - Patient - contains required components/api files for the patient's details page that contains patient's details.
    - passwordReset – contains required components/api files for the password reset page.
    - patients – contains required components/api files for the patients list page that contains patients list.
    - patientChart - contains required components/api files for the patient's chart page.
    - register - contains required components/api files for the register page.
  - UI
    - components – contains atoms, molecules, organisms that are shared between multiple pages like inputs, headers, spinners, warning messages, etc.
    - Images – contains static images (logo).
- App.css – contains global styles.
- App.tsx – contains page routing.
- Index.tsx – contains a logic for rendering the App component into an index.html element with root id using the ReactDOM library.

3.4 Work progression

3.4.1 Application home page

The first page that was developed was the home page. It is a simple page that has a logo and 2 buttons – login and register that would redirect to a respective application page.

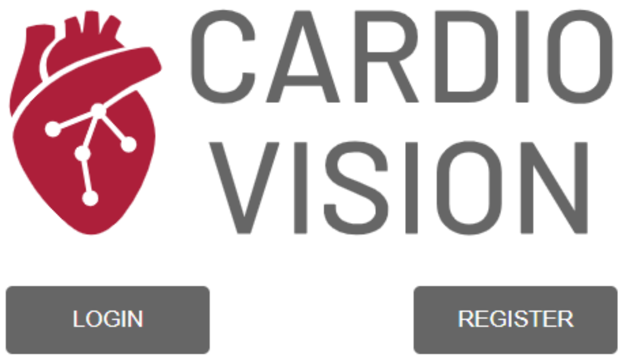


FIGURE 3.1: Home page.

3.4.2 The login page

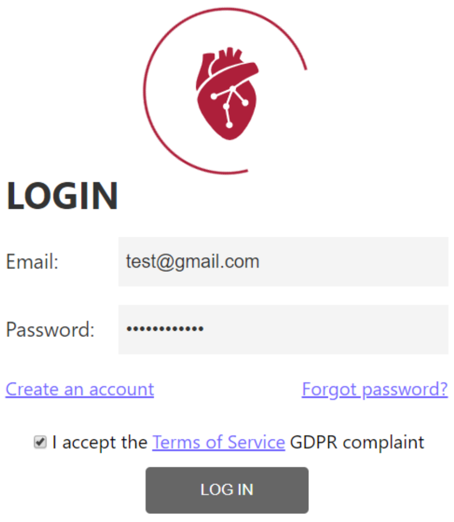


FIGURE 3.2: Login page.

The login page contains 2 inputs for email and the password, 2 redirect links (to register page, if a user does not have an account(and to the forgot password page). It also has a check mark to accept the terms of service (if not checked – user cannot proceed) and the login button which triggers the login request.

The login request simply takes the user's email and password and if the user exists, returns a token, that is stored in the local storage and can be used for further requests. Also, if the user will go to the login page and the local storage will have the key value populated - then the key will be checked for validity and if valid, the user will be redirected right to the patients page.

On the patients page the user will have an option to logout - which will clear the storage. The UI has a simple check if the email/password field is empty or not. If empty - an error message under the respective input box will be shown indicating that this field is required.

If the login request will not end with a success status (200 code), then the toast error message will appear in the top right corner of the page with the error details. If the request fails (if the backend server is not reachable), then the same error toast will appear indicating that something went wrong.

The login page also has a small logo or the project which is also a redirect link to the home page. While the login request is being sent and the UI is waiting for the response, the spinner is displayed around the heart logo, providing lively feedback for the user.

The login page also has a term of service – a simple popup that displays the service disclaimer. After user is logged in and the token is stored in the session, user is redirected to the patients list page.

3.4.3 Register page

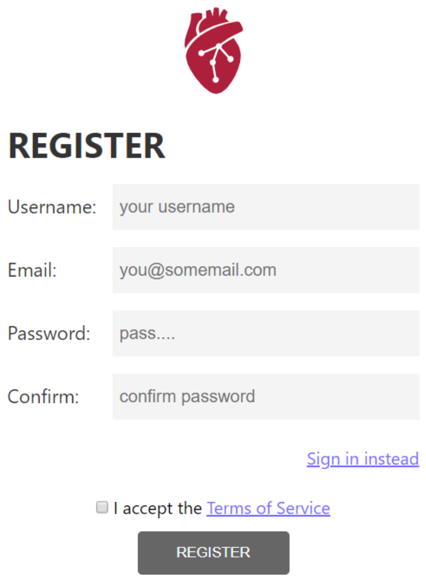
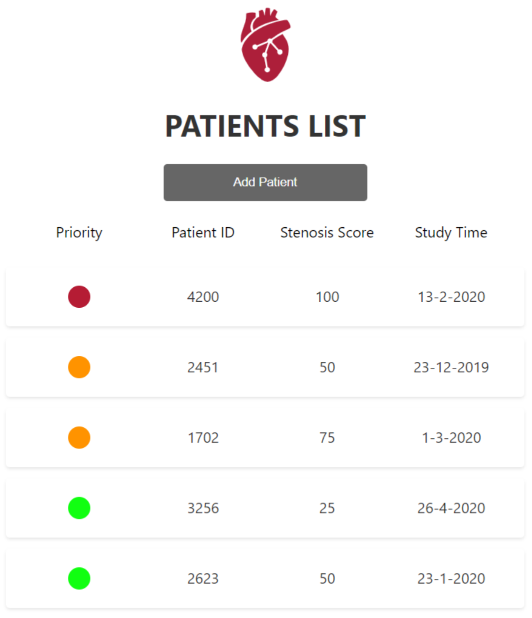


FIGURE 3.3: Register page.

The Register page has the same layout and similar design as the login page. The only differences are the 2 new input fields: username and confirm password fields. The redirect link – takes the user to the login page (in case a user has accidentally gone to the register page). It also has the same toasts messages if something goes wrong with the register request. The spinner is also turned on if request is pending and is located around the heart logo. After the user creates an account a redirect to the patients list page is triggered.

3.4.4 Patients list page



PATIENTS LIST

Priority	Patient ID	Stenosis Score	Study Time
🔴	4200	100	13-2-2020
🟠	2451	50	23-12-2019
🟠	1702	75	1-3-2020
🟢	3256	25	26-4-2020
🟢	2623	50	23-1-2020

FIGURE 3.4: Patient list page.

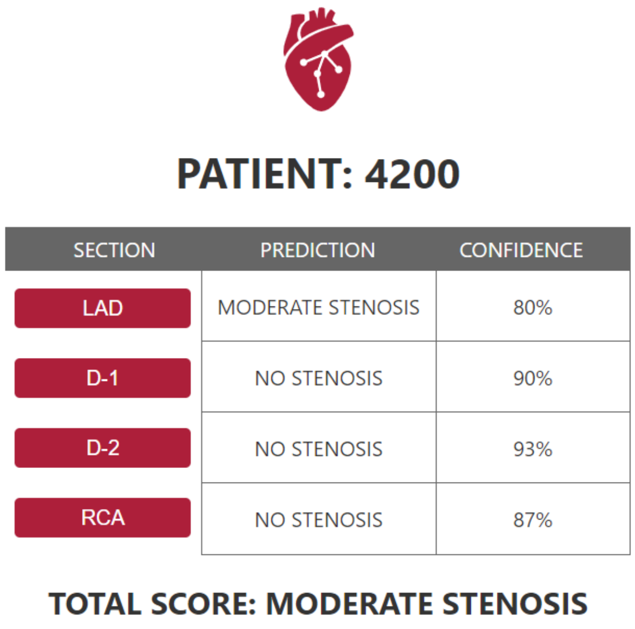
Patients list page is a main application page that displays the list of all patients. The patients list is sorted by priority (critical, avatage, non-critical). To get the patients api, you will have to have the authorization token in the local storage. If the token is missing or has expired, the user will be redirected to the login page.

The list has 4 columns (priority, patients id, Stenosis Score, study time). There is no patient's name as doctors are not allowed to share patient's details for legal reasons. The priority is displayed with a simple colored circle (green for non-critical patients, orange for average priority patients and blinking red for critical patients). All patients have a hover effect as each patient's line in the list is also a redirect link to the patient page which contains all the patient's details.

There is also an option to add new patient on the patients page. By clicking add patient button - the popup will open. It will have a few fields to fill in (stenosis score, study time) and there is a box where a doctor can upload images from CT Coronarography. After the doctor clicks the add button, the patient's data is transferred to the backend which is feeding the neural network algorithm with the data and returns a prediction which is then used to order the newly created patient into the list and to create a detailed graph for each patient.

If the patient was already given some treatment or if the patient was added by accident, there is also an option to delete the patient.

3.4.5 Patient's details page



PATIENT: 4200

SECTION	PREDICTION	CONFIDENCE
LAD	MODERATE STENOSIS	80%
D-1	NO STENOSIS	90%
D-2	NO STENOSIS	93%
RCA	NO STENOSIS	87%

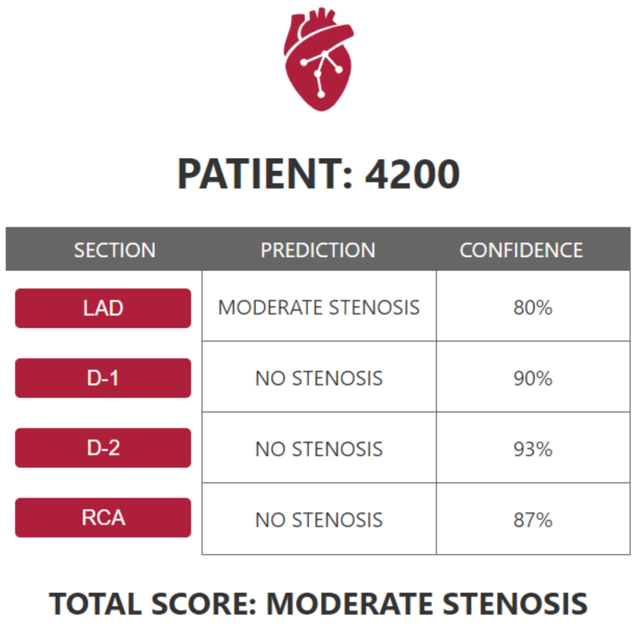
TOTAL SCORE: MODERATE STENOSIS

FIGURE 3.5: Patient's details page.

Patient's details page is a simple table that displays the patient's details. The table has 3 columns (section - different sections of a coronary tree, prediction - over each section if it has stenosis or not, confidence - the confidence of the prediction being correct).The page also has patient's ID and total score of patient's condition.

When the doctor is redirected from patients list page to patent details page - the url on the patient page is formatted as follows: '<hostname>:3000/patient/4200'. The number at the end of the url is the patient's id. This is a method I use to handle the data transfer to another page; On the patients list page, when a user clicks on the patient's line, the patient's id gets into the url and on the patient page I am parsing the url to retrieve the patient's id so I can call the api to get patient's data. Each line in the section column is clickable and is a redirect link to the patient's graph page.

3.4.6 Patient's graph page



PATIENT: 4200

SECTION	PREDICTION	CONFIDENCE
LAD	MODERATE STENOSIS	80%
D-1	NO STENOSIS	90%
D-2	NO STENOSIS	93%
RCA	NO STENOSIS	87%

TOTAL SCORE: MODERATE STENOSIS

FIGURE 3.6: Patient's details page.

Patient's graph page contains a probability graph that displays probabilities for various stenosis levels and is linked to CT images (the images that were used to feed the Neural network algorithm), The graph is interactive, a user can hover over the graph to look at the corresponding image of the arteries in different angles with the specific area of the photo highlighted.

3.5 Component map

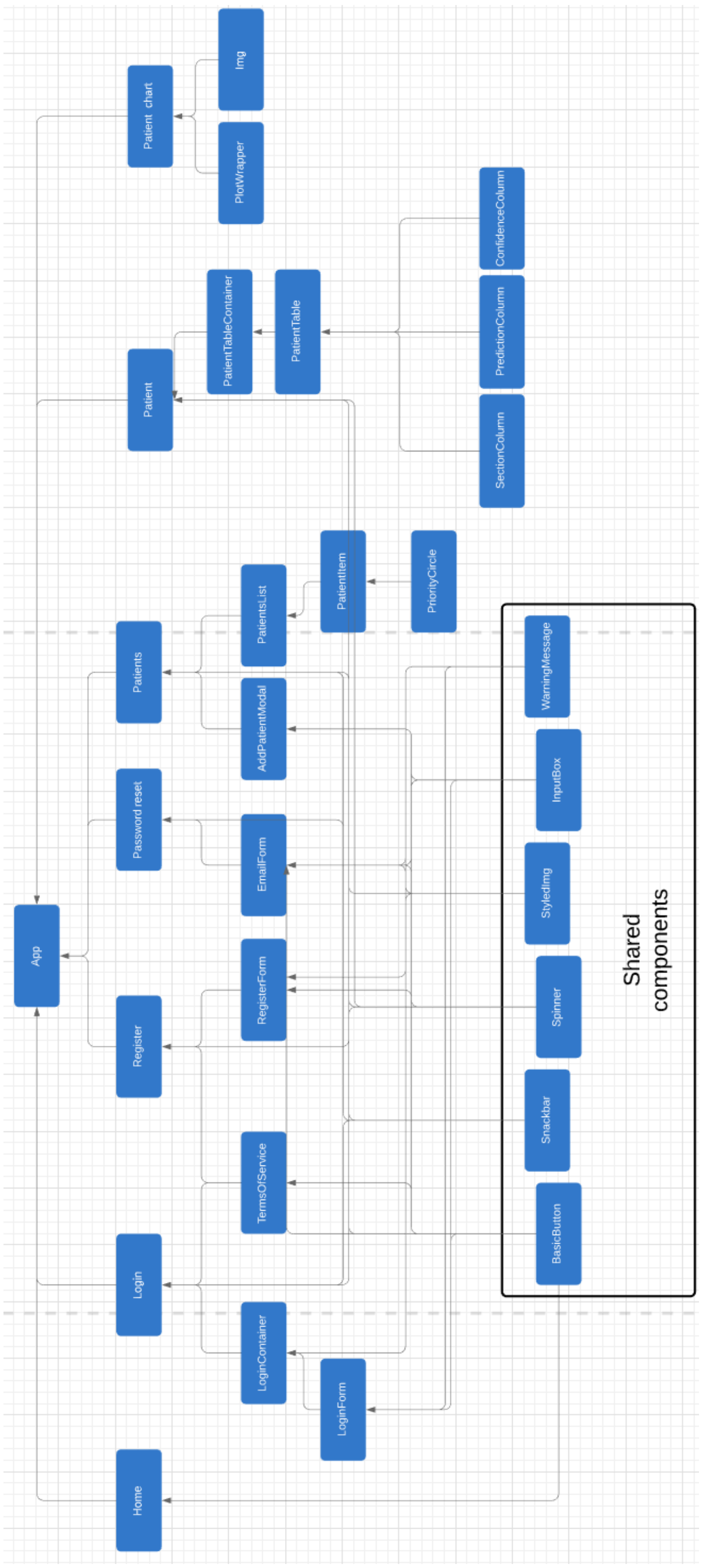
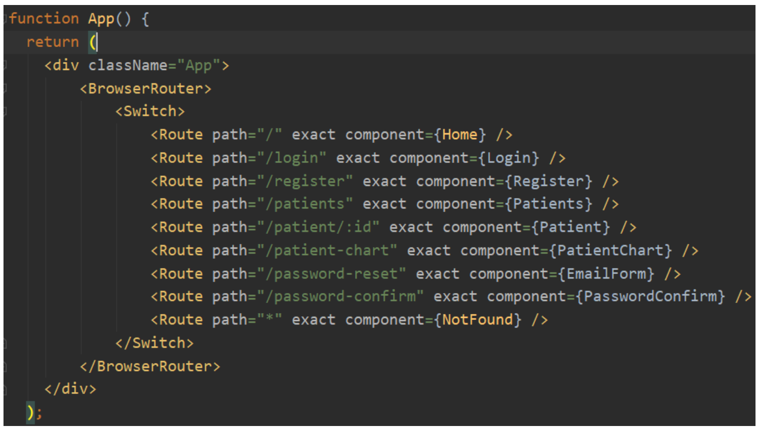


FIGURE 3.7: component map.

The app component serves as a wrapper for all the other page components. In the App component I am using the React DOM router library - that depending on the app path will open a corresponding component. For example, the app is running on the port 3000, so if the user will manually type in the browser url path the 'http://<host>:3000/login' the login component will be rendered. The App.tsx contains a BrowserRouter wrapper from React DOM router library that has a switch in it with cases though the different path variables and different components. Below is the APP component:



```
function App() {
  return (
    <div className="App">
        <BrowserRouter>
            <Switch>
                <Route path="/" exact component={Home} />
                <Route path="/login" exact component={Login} />
                <Route path="/register" exact component={Register} />
                <Route path="/patients" exact component={Patients} />
                <Route path="/patient/:id" exact component={Patient} />
                <Route path="/patient-chart" exact component={PatientChart} />
                <Route path="/password-reset" exact component={EmailForm} />
                <Route path="/password-confirm" exact component={PasswordConfirm} />
                <Route path="*" exact component={NotFound} />
            </Switch>
        </BrowserRouter>
    </div>
  );
```

FIGURE 3.8: App.tsx content.

The second row of the component map (Image 11) contains all the components with encapsulated logic. For example, the register component will contain:

- all the api calls
- all the processing and checks for errors
- all the methods for handling the input change
- all the variables and methods.

So the user email variable and the method for handling the input change is stored in the register component and it is passed to the register form component where the value and method is passed to the input field.

All logic level components also have a spinner in them and the Snackbar components (which are shared components, located in the UI folder). The spinner is activated for the time while the request is being sent and the app is waiting for the response. The Snackbar component is a wrapper for a toast warning message (when something went wrong with the call or the backend service is not working). Taken from the material UI library, it shows the message for a short period of time, with the simplistic and nice design and fade in/out effect.

The next row of the component map contains components that are responsible for styles and position of other components.

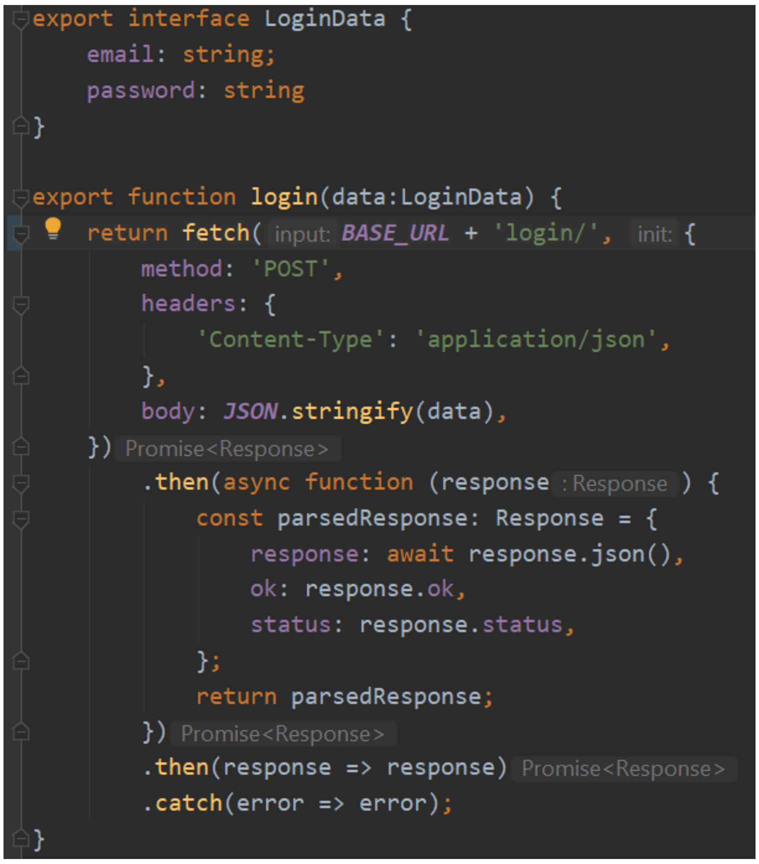
The styled components that are almost in each component and that are containing some unique styles that cannot be shared among other components are not displayed on the component map for the sake of map readability.

The styledImg component is the image component that has certain styles attached to it, for the most part I was using this component for the heart logo that is also a redirect link to the home page.

3.6 API calls and structure

3.6.1 Login API

Login API is a POST API method, requires no authentication, it takes two required fields - a user email address and a password.



```
export interface LoginData {
    email: string;
    password: string
}

export function login(data:LoginData) {
    return fetch( input: BASE_URL + 'login/',  init: {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify(data),
    }) Promise<Response>
        .then(async function (response : Response ) {
            const parsedResponse: Response = {
                response: await response.json(),
                ok: response.ok,
                status: response.status,
            };
            return parsedResponse;
        }) Promise<Response>
        .then(response => response) Promise<Response>
        .catch(error => error);
}
```

FIGURE 3.9: Login api function.

Login function is taking as an argument an object with the structure that is defined in the LoginData interface (Image 13). To make a request I am using a fetch api from JS, which provides access and handling functionality for requests and responses. The ‘BASE$_U$*RL′isavariablethatcontainsthebaseurltothebackendserveranditisusedinallAPIcalls.*
therequest, isformedwithheaders, andbody.Inthesecondparttheloginfunction − aresponseisformed.

For all the requests I am forming the response object with the following key/values
:

- response - contains the response from the api call (for the login api the response is an object with the key of key and value of the authentication token that can be used for further requests)
- ok - boolean value which indicates if the request was successful or not

- status - in case of request failure, contains detailed information on the failure.

The next part of the login function is the error handling, which simply returns en error that was catched and which is used on the frontend to show the user an error message.

3.6.2 Register API

Register API is a POST API. It requires no authentication, it takes four arguments (user email, username, user password and a confirm password). The response is the same as for the login API.

3.6.3 Patients API

Patients APIs (POST API), require authentication key from the login API, in the body it takes an object with the following keys:

- page - the page number of the patients list.
- size - the number of patients per page.
- NumOfPages - the total amount of pages.

This API has a pagination feature, to handle a large amount of patients. For example, instead of loading the whole list of patients which potentially can be a large amount of data - this method allows to load a small portion of data at a time. The API returns an array of patients where each patient has the following structure:

- id - identification number
- priority - there are 3 levels (significant stenosis, moderate stenosis, non significant stenosis)
- study time - a date when the patient's images were taken
- stenosis score - a score number, assigned by the the doctor

3.6.4 Add Patient API

Add Patient API is a POST API that requires authentication key from login/register API. This API takes an array of images, study time of the patient and stenosis score. Once the API is called the images are processed on the backend and the patient list is then updated.

3.6.5 Remove Patient API

Remove Patient API is a DELETE API that requires authentication key from login/register API. This API is taking one parameter - id of the patient. Once the API is called, the patients list is updated.

3.6.6 Patient's Details API

Patient's Details API is a GET API that requires the authentication key from login/register API. This API is taking one parameter - id of the patient. The API returns details of the patient - an array of objects with the following structure:

- section - section of the heart arteries
- prediction - a priority (non-significant stenosis, moderate stenosis, significant stenosis).
- confidence - a percentage of confidence of the prediction

3.6.7 Patient chart API

Patient chart API is a GET API that requires the authentication key from login/register API. This API is taking 2 parameters - patient's id and section of heart artery. The API returns an array of objects where each object has the following structure:

- mprIndex - number of the MPR image on x axis of the graph
- img - an MPR image.
- prediction probabilities - array with length of 3, with the probabilities for each class, for example: [0.8, 0.15, 0,05] which would mean that probability of non significant stenosis is 0.8 or 80 percents, moderate stenosis with probability of 0.15 or 15 percents and non-significant stenosis with probability of 0.05 or 5 percents.

Chapter 4

Data

4.1 Data that we collect from users (doctors)

- email - for authentication and for the case when the user will forget the password
- study time - requested from the doctor when a patient is created. This is the date when the patient has taken the procedure (when the images of the patient's heart were taken).
- Coronary Computed Tomography Angiography scans or extracted Multiplanar Reconstruction (MPR) images - usually it is a set of 50 images of one of the heart arteries. Each image is an image of the artery that is scanned across its length and is cut in half, and each image has a representation of the artery cut in half in different angles. The difference of angle is 7 degrees in order to not miss out the stenosis. Below is an example of how it looks:

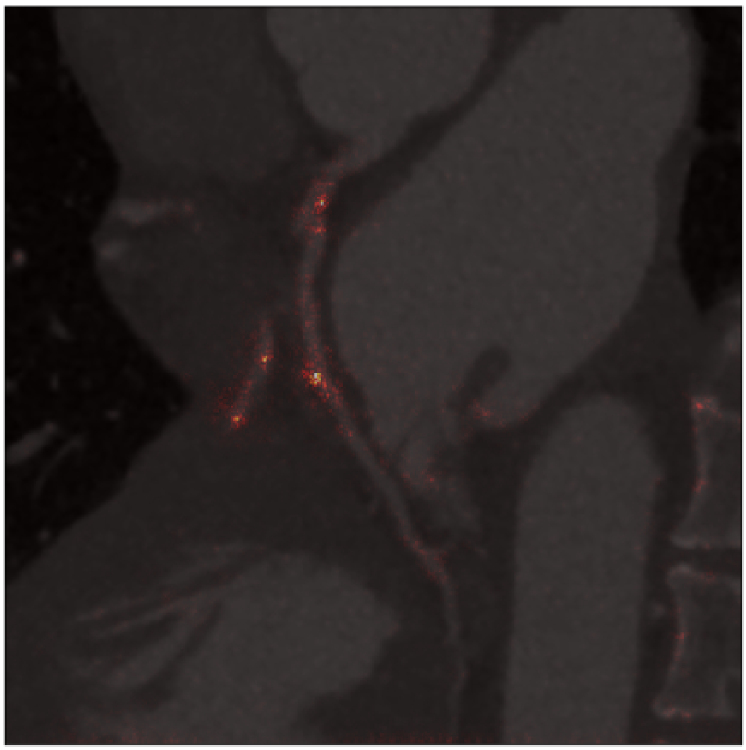


FIGURE 4.1: MPR image.

4.2 Training data

For training of the algorithm that predicts stenosis, a curved MPR images of the coronary artery were used with stenosis levels annotated by professional radiologists from a reputable source - Future Medical Imaging Group (FMIG) in Australia. Current dataset consists of about 160 thousand MPR images which were extracted from CCTA scans of 828 unique patients.

4.3 Model output data

The Model accepts 50 MPR images. For each image there is a prediction of probability for all 3 possible classes: non-significant stenosis, moderate stenosis, significant stenosis. Below is an illustration of how the output would look like for the patient that is probably healthy: [0.9, 0.07, 0.03] - this would mean that probability that this branch has a non-significant stenosis is equal to 0.9 or 90 percents, moderate stenosis with probability of 0.07 or 7percents, and significant stenosis with probability of 0.03 or 3percents

Chapter 5

Background information

This section contains some information about the work on the Neural network algorithm that was done by a group of people from UCU.

MPR - the process of using the data from axial CT images to create a more anatomical representation of the coronary artery by tracking the whole specific artery branch, along the CT volume, and generating its two-dimensional image.

The first step was to convert all the CCTA (Coronary Computed Tomography Angiography) into sets of MPR images for each artery branch there is (this was done by a radiologist assistant). As initially the images were marked-up by the scanner and these marks can affect the predicted result of the neural network algorithm, the first challenge for the team was to remove the mark-up and meta information from the image.

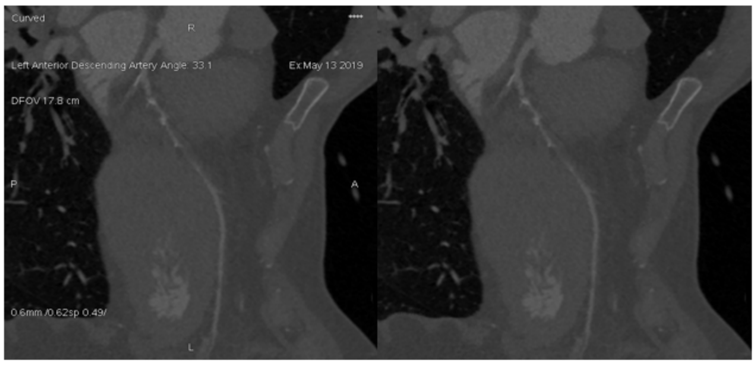


FIGURE 5.1: The left image shows row image with the text, the right image - after the text is removed.

After the images are cleaned from mark-up they are passed to the ShuffleNet V2 - which is an extremely efficient convolutional neural network architecture that is designed for the mobile devices with limited computing power. This neural network produced probability predictions for each class: non-significant stenosis, moderate stenosis, significant stenosis. Where non-significant stenosis - is the stenosis level from 1 to 50 percents, moderate stenosis - is the stenosis from 50 to 69percents, and significant stenosis - is the stenosis where more than 50 percents of stenosis is present and immediate doctor's action is required.

Arteries	Arteries	Sections	Sections
LAD	824	LAD	822
-	-	D-1	729
-	-	D-2	356
-	-	D-3	68
LCX	722	LCX	639
-	-	PLV-LCX	15
-	-	PDA-LCX	17
RCA	721	RCA	91
-	-	OM	6
-	-	OM-1	81
-	-	OM-2	281
-	-	OM-3	75
-	-	PLV-RCA	609
-	-	PDA-RCA	71

The table above displays a collected dataset statistics. Number of cases containing certain arteries and branches for all the 828 patients.

For each branch there are 50 MPR images that represent it from the different angles, this is done to avoid missing a problem case, as it can be located anywhere along the vessel. So there are 50 images with an angle difference of around 7 degrees.

For each branch there are 50 MPR images that represent it from the different angles, this is done to avoid missing a problem case, as it can be located anywhere along the vessel. So there are 50 images with an angle difference of around 7 degrees.

There were a lot of challenges during the development process that the team had to deal with. One of the problems in the dataset is that on some MPR images there was displayed more than one artery branch. if one of the extra branches on the MPR image will have the stenosis, this might greatly impact the way that the model will output the predictions for that branch. Another problem is with the stent that can be installed in the heart artery. Stent is a small tube that is placed in the artery to make the artery wider to increase the patient's blood flow. The stent in the image would affect the algorithm, rendering incorrect output predictions of the model. Also, there is a chance that the image may show an myocardial bridging - which is a natural and usually harmless condition when the coronary artery goes through the heart muscle instead of lying on its surface. This condition may also affect the way the model is outputting the predictions.

Chapter 6

Results

The resulting web application is a fully functional service that can be used for prioritizing the list of patients using neural network algorithms . The user interface is really easy to use and requires no walkthrough to start using the application.

User interface part of the application (my focus in the project)

There are a number of interesting features implemented in the app. The app has a basic multi-user functionality, such as setting up an account, restoring passwords and navigating through the personalized database. It allows a user (MD) to create a list of patients by uploading the MPR images and describing some minor details about the patient. There is a full functionality of the patient's list management - the list may be sorted by the state of the patient (patients with significant stenosis and with longer study time will be at the top of the list and there is also a visual that attracts the eye of the doctor to that patient). The user can view the patient's details that would show the state of each section of the heart arteries and tell the predictions made on them, as well as the confidence of each prediction.

Separately, it is worth mentioning the implementation of the stenosis predictions and MPR image review functionality. From the patient's details view, the user can opt to review each section of the heart of the patient being assisted with the graph, where each point on the x axis has an corresponding MPR image at a certain angle, at y axis there is a confidence level. The graph has 3 lines to assist the analysis: green - which displays the confidence of the non-significant stenosis score, yellow - average stenosis score, red - significant stenosis score. The graph is interactive and easy to use, with the instant display of the changing MPR image corresponding to the hover point on the graph. During the hovering - the MPR image will change to a corresponding index on the graph. Implemented by a smooth integration of UI and backend code, this allows the user to quickly find the areas where the system predicted high stenosis score and check the data to confirm it or refute.

Reusability and adaptability is yet another benefit of following the best practices of the React application - the components from the app can be easily modified to fit into any kind of the application that contains tables and graphs. Each component can be easily changed and styled and can be reused by passing different props to it.

The final web application is responsive - quite readable and usable even on various display sizes, including small screens on the go - tablets or phones.

From the perspective of the neural network algorithm logic the team has connected with the doctor from Australia, who provided general guidance for the problem and application's requirements. The final app's algorithm can detect the stenosis score in curved MPR images. The team has collected a large training data set which consists of 160 thousand MPR images Which were extracted from CCTA scans of 828 unique patients. The proposed method is showing better results than previous attempts [1] with the accuracy of 80 percents for multiclass classification of stenosis level. The new dataset that was created from CT scans of 828 patients is one of the reasons that the algorithm is performing with better results then its predecessor. The other improvement is that the used method omits centerline extraction and does not require any handcrafted features. Furthermore, we obtain explainable results and display features which are making an impact on the network's decisions.

Accuracy of researched method vs RNN-based[1]:

- Segment-level: 0.81 / 0.80(RNN)
- Artery-level: 0.81 / 0.76(RNN)
- Patient-level: 0.80 / 0.75(RNN)

F1 score of researched method vs RNN-based[1]:

- Segment-level: 0.81 / 0.75(RNN)
- Artery-level: 0.82 / 0.77(RNN)
- Patient-level: 0.80 / 0.75(RNN)

6.1 Post Scriptum

There were many challenges that were arising in the process of developing the neural network that would make the predictions based on the MPR images. The team of UCU students did a great amount of work and I was honored to help them with the UI part. If you are interested in more details about the development process and issues that were arising during it, please check their work here - pdf report

Should you be interested to get more information on the UI part of the project or technologies used in the web application, please contact me at borovetsandriy@gmail.com

Bibliography

[1] Zreik et al. A recurrent CNN for automatic detection and classification of coronary artery plaque and stenosis in coronary CT angiography. IEEE Transactions on Medical Imaging, 38(7):1588– 1598, 2019.

[2] React documentation - https://reactjs.org/

[3] TypeScript documentation - https://www.typescriptlang.org/

[4] General information - https://www.wikipedia.org/

[5] D3 documentation - https://d3js.org/

[6] Heart problems/anatomy - https://www.heart.org/

[7] Lucid chart builder - https://app.lucidchart.com/

[8] Material UI - https://material-ui.com/

[9] Emotion/styled - https://emotion.sh/docs/styled

[10] React DOM router - https://reacttraining.com/react-router/web/guides/quick-start

[11] General frontend tips - https://www.w3schools.com/

[12] ShuffleNet v2 - https://arxiv.org/abs/1807.11164

[13] CNN-CASS: CNN for Classification of Coronary Artery Stenosis Score in MPR Images - https://arxiv.org/abs/1807.11164

[14] Node js - https://nodejs.org/en/