Technical Manual

Project Name: Passport Application System

Student Name: Stephen McDonagh

Supervisor: Alistair Sutherland

Module Code: CA400

Contents

0 Abstract

The idea for my final year project was to implement a facial detection system. I have built this around a passport application system, which allows users to enter the appropriate information as well as an Image which is checked to contain one single face. Through the user interface the user will be updated on the application progress and if they have successfully completed the application. Within the application is a chatbot for users to ask passport related questions if they have any. Upon completing the application an email is sent out telling them their application is complete and their order number for their passport.

1 Introduction

1.1 Motivation

The motivation for my final year project was purely to learn how to build a web application from frontend to backend. After completing my third-year project, I did not feel like I had learned enough from doing a game which is why I chose a completely different area to work on this year. I believe this was a wise decision as I have now gotten exposure to developing a full stack application.

I had previously never worked on any frontend technologies, so I felt this was a good time to get my first experience with this type of development. After speaking to members of my team during my internship, I was recommended to investigate using Angular or VUE. So, this was something I needed to research before committing to a framework.

1.2 Research

After deciding I was going to do a facial detection application, I needed to research the best possible solutions. Something that seemed to keep popping up was OpenCV. This would be a perfect solution for facial detection and is also not limited to a single programming language. From my research I discovered that a solution could be made in Java, c++ and Python. As Java would be my programming language I'm strongest with, this seemed like an obvious choice. This left me then to research exactly which classifier would be the best choice for my application. OpenCV has LBP Cascade Classifiers and Haar Cascade Classifiers.

For my frontend framework, I spent a lot of time deciding what would be the best choice for someone with no frontend experience. VUE is relatively new, so this made me lean towards using Angular. Angular is one of the leading industry frontend frameworks which is well documented and there is plenty of help on websites like tutorialsport and stackOverflow.

To choose my Backend framework didn't take long to research. I had no experience with any, but Springboot seemed to be the best choice. The extensive documentation also helped at the beginning in the best practices etc.

2 Technology Used

2.1 Frontend Technology

As previously mentioned, I chose Angular as my framework for frontend. Specifically, I am using Anglular6. Angular is an extremely powerful frontend framework, and the Angular CLI has many great libraries and functions which can be imported into your project. Within Angular I used the following languages:

- HTML
- CSS
- Typescript
- Bootstrap

Using all these technologies I was able to create a fully functioning frontend for my final year project. I also used Atom as my editor for my frontend. The frontend is running locally on port 4200

2.2 Backend Technology

As Java is my programming language of choice, I used Springboot as my backend framework. Under Spring I could have developed my whole application from frontend to backend, but I wanted to take this opportunity to learn something new which is why I chose to use Angular instead. Under Springboot I used the following technologies:

- Java8
- Maven- As my build tool
- SQLite3- As my database
- REST- GET & POST requests from frontend to backend.
- OpenCV- For facial Detection
- DialogFlow (API.AI)- For my chatbot Service
- Google SMTP- email confirmations\

2.3 Testing Technology

Having a thoroughly tested project from frontend to backend was a goal of mine. Therefore, from the beginning of the project I was thinking about how I should test my project and the different types of testing I should include. Testing is an extremely important part of the development process and it is extremely important as a developer to know how to test correctly. I am using the following technologies to test my Passport Application System:

- Junit
- Mock MVC- mocking database
- Protractor – Similar to selenium, this is used for UI testing.
- JMeter- This was used to stress test my backend.
- Postman – testing API calls

3 Problems Encountered & Solutions

3.1 Connection of Frontend to Backend

One of the early problems I had was trying to understand how I send and receive information from my Angular frontend to my Springboot backend and vice versa. To fix this I set up the proxy (pictured below) which rewrites requests from my frontend which is on port 4200 to my backend with is on port 8080.

```
{
  "/server": {
    "target": "http://localhost:8080",
    "secure":false,
    "changeOrigin": true,
    "logLevel": "debug",
    "pathRewrite":{
      "^/server": ""
    }
  }
}
```

On the command line this is how it Looks when I send a post request of a completed application to be stored in my SQL database.

```
[HPM] Rewriting path from "/server/api/v1/applications" to "/api/v1/applications"
[HPM] POST /server/api/v1/applications ~> http://localhost:8080
```

3.2 Database Choice

In previous years, I had used Microsoft SQL server Database, so I decided this year I would research a different type. There is plenty of benefits to SQLite, such as better performance and reduced complexity. After some research I felt like this would be a good choice to use with my final year project. Below is a SQL script for creating the same database structure as the one in my application

```
CREATE TABLE
    application
    (
        id BIGINT NOT NULL,
        name VARCHAR,
        dob VARCHAR,
        email VARCHAR,
        number VARCHAR,
        pps_number VARCHAR,
        application_date varchar,
        PRIMARY KEY (id)
    );
```

3.3 Sending Image from Frontend

Previously I had the users needing to press a button to submit their image to be checked as ‘passport quality’. I did not like this functionality and wanted to make the process more user friendly. To fix this I needed an async method which will send the image off as soon as it is submitted. I believe this to be a much better method for the user. I used the following code to help me resolve this issue

```
async OnFileSelected(event){
  this.fileUploaded = <File>event.target.files[0];
  const fd = new FormData();
  fd.append('file', this.fileUploaded);
  this.http.post('server/api/v1/detect-face', fd)
  .subscribe(
    data => {
      this.imageResponse = JSON.stringify(data);
      if(this.imageResponse == "true"){
        this.imageMessage ="Your Image Has been accepted";
      }
      else{
        this.imageMessage ="Please Enter A new Image";
      }
    },
    err => console.error(err)
  );
}
```

3.4 REST API

Building a Restful application is completely new to me. Previously I had understood the general concept but had never had the chance to learn about it in detail. Plenty of research was needed before I could get into implementing my own REST API'S.

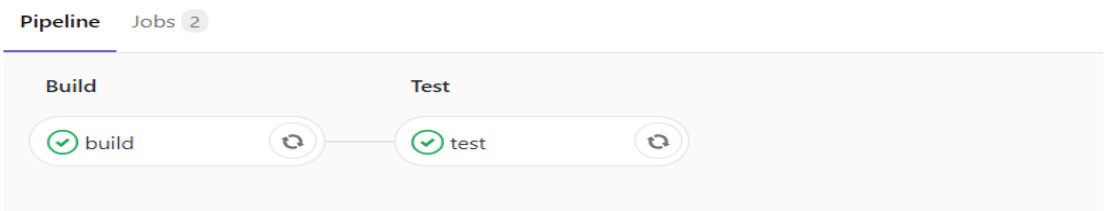
Luckily the Springboot documentation is very helpful and I was able to understand much better how I should go about implement my own endpoints. Below I have included a screenshot for my endpoint api/v1/applications

```
@RestController
@RequestMapping("/api/v1/applications") //Endpoint for this class
public class ApplicationsController {

    @Autowired
    private ApplicationRepository applicationRepository; //Using jpa repository

    //Get request for all the applications within Database
    //Then shown on frontend
    @GetMapping
    public List<Application> list() { return applicationRepository.findAll(); }

    //Post request application to database
    @PostMapping
    @ResponseStatus(HttpStatus.OK)
    public void create(@RequestBody Application application){ applicationRepository.save(application); }

    //Get request for a single database entry done by id
    @GetMapping("/{id}")
    public Application get(@PathVariable("id") long id) { return applicationRepository.getOne(id); }
}
```

3.5 Gitlab Pipeline/ Automating Testing

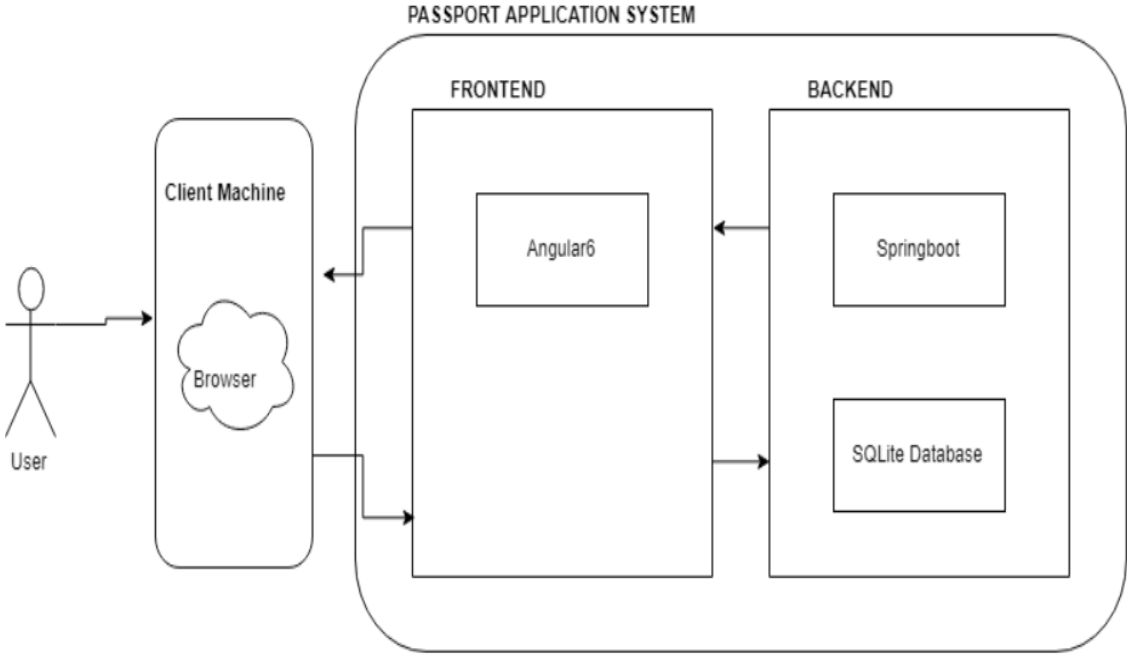
During my Internship I learned about Jenkins and setting up pipelines which automatically builds, tests and deploys applications. During my time I didn't get a chance to work with this, but I felt that setting up my own pipeline would give me some experience as to how CI/CD works in industry.

I began by researching how it works and quickly decided that it would make more sense for me to set up my pipeline on Gitlab rather than on Jenkins. Below you can see my most recent pipeline and the jobs within it. The first job Builds my Springboot project and when that is finished it moves onto the testing stage which runs my unit and integration tests



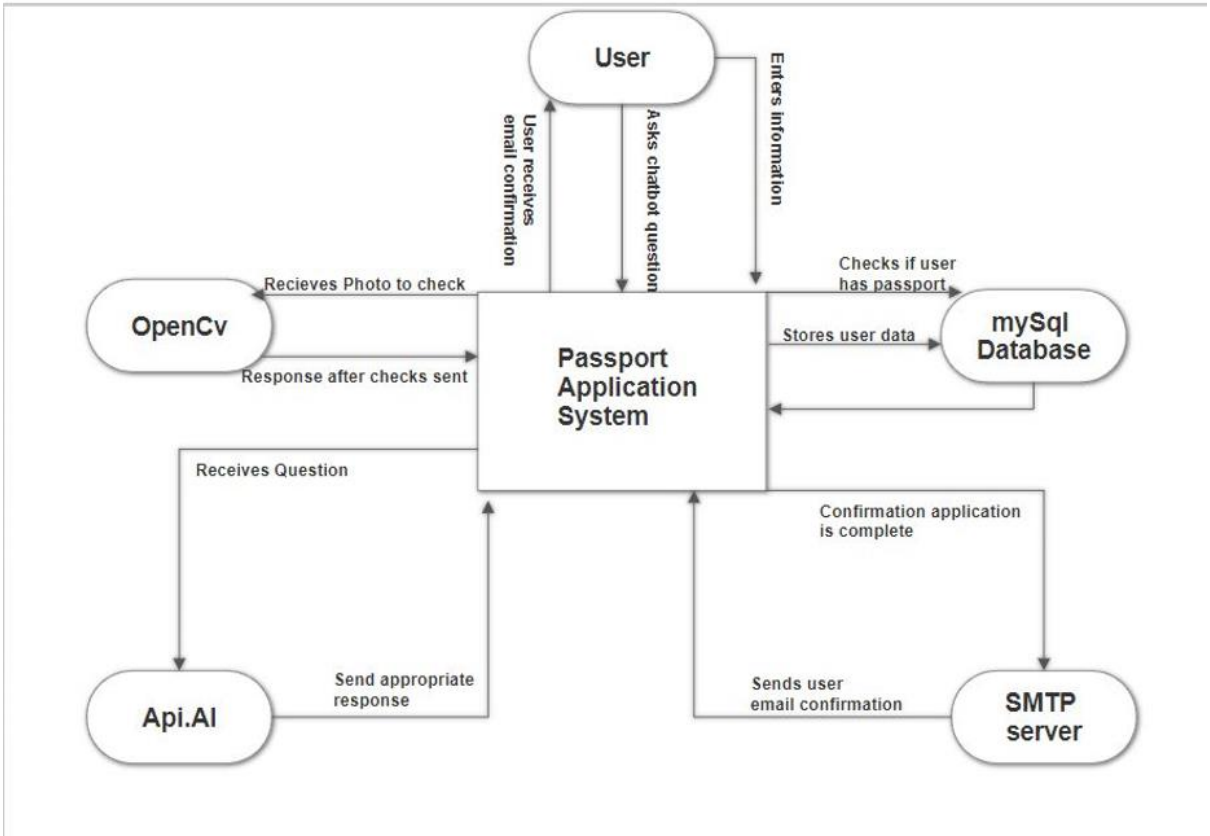
4 Architecture Design

This diagram below shows at a high level how the major components interact with each other.

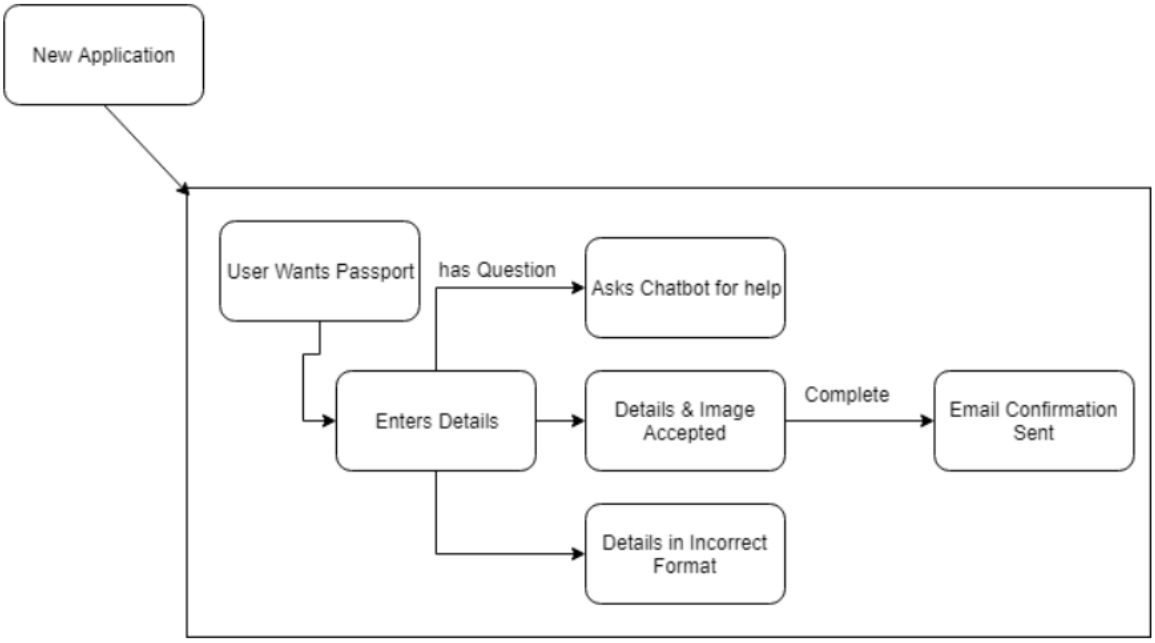


5 Design

5.1 Context Diagram

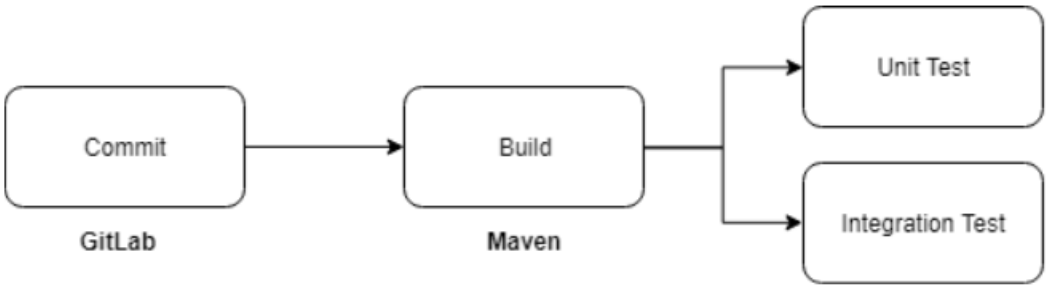


5.2 Data Flow Diagram



5.3 Gitlab Pipeline Diagram

This is the design of my Gitlab pipeline which automatically builds & tests my application after a commit



6 Implementation

6.1 Facial Detection

To implement my facial detection service, I am using OpenCV. OpenCV can be used for all kinds of object detection but it also works well for faces.

To set this up, I first needed a facial detection controller which will be my endpoint of my Springboot application. This Is where the image the user inputs will be sent to. What this endpoint is expecting to be sent is a multipart file.

```
@RestController
public class FacialDetectionController {

    @Autowired
    FaceDetectionService faceDetectionService;

    //Returns true if One face is detected
    //Requesting multipart file
    @PostMapping("api/v1/detect-face")
    @ResponseBody
    public boolean detectFaceImage(@RequestParam("file") MultipartFile file) {
        try{
            if(isValidImage(file)){
                System.out.println("CORRECT IMAGE TYPE");
                System.out.println("Successfully Uploaded: "+ file.getOriginalFilename());

                return faceDetectionService.detectFace(file);
            }
            else{
                System.out.println("ERROR");
                return false;
            }
        }
        catch(Exception e){
            System.out.println(e);
            return false;
        }
    }
```

Once this has been checked to be the correct file (JPG), then the facial detection service can be called on this image. What this service does is loads the LPB classifier that I am using, and then checks to see if any faces can be found within the image that has been uploaded. This service can be seen below

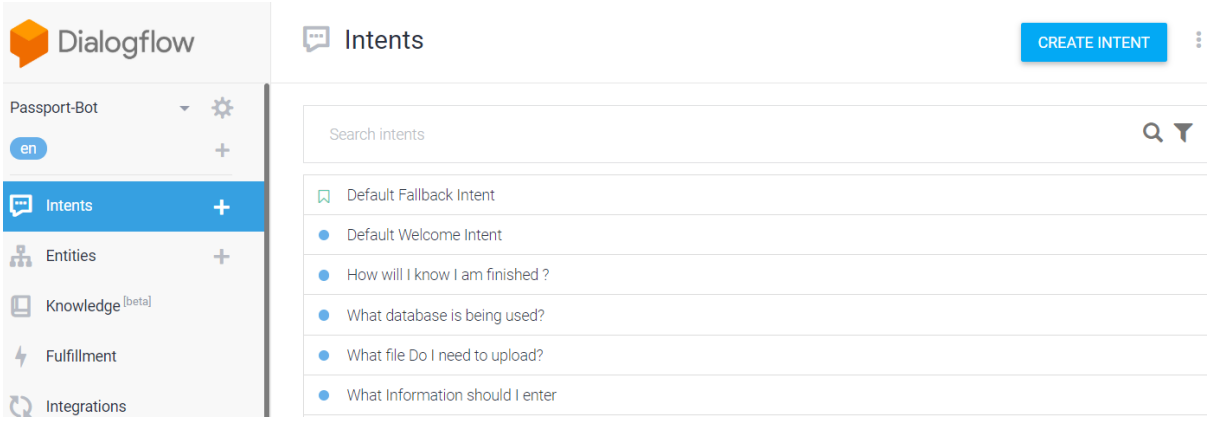
```
public boolean detectFace(MultipartFile file) throws IOException {

    //Create an arraylist of faces detected
    faces = new ArrayList<>();

    MatOfRect faceDetections = new MatOfRect();
    CascadeClassifier faceDetector = new CascadeClassifier();
    //Cascade classifier being loaded locally
    faceDetector.load( filename: "C:\\Users\\mcdons59\\DCU\\4thYearProject-CA400\\2019-ca400-mcdons59\\src\\Applic

    faceImage = Imgcodecs.imdecode(new MatOfByte(file.getBytes()), Imgcodecs.CV_LOAD_IMAGE_UNCHANGED);
    faceDetector.detectMultiScale(faceImage, faceDetections);

    System.out.println(String.format("Detected %s faces", faceDetections.toArray().length));

    //Draws rectangle around faces: Not seen by user.
    for (Rect rect : faceDetections.toArray()) {
        faces.add(new Face(rect.x, rect.y, rect.width, rect.height,  ratio: 0));
    }

    //Checking if number of faces detected = 1
    if (faceDetections.toArray().length == 1) {
        return true;
    } else {
        return false;
    }
}
```

6.2 Chatbot

I really liked the idea of implementing a chatbot into my web application. I thought this would be a nice feature for the users. There is plenty of options to choose from when implementing chatbots, but it seemed like DialogFlow was a great option.

DialogFlow allows you to train the chatbot how you want. So, for my application I have trained it to answer passport related questions. It makes it very easy to customize your chatbot for your application.

Based of some of the feedback from the user testing, I was able to increase its performance in questions that I had not previously thought of.



6.3 Email Service

For my email service I wanted to create something like what most businesses already had, which is an email confirmation after successful completion of an application. I believe it is extremely important to confirm with the user when they have done something correct.

Due to GDPR I was unable to use users actual email addresses to show of this functionality. This is something I discussed with my supervisor and I was warned not to use users actual email addresses to display this functionality.

After the user finishes, an email is sent telling them that they successfully completed their passport application and the order number of their passport. The code for my email service is as follows

```
@RequestMapping("api/v1/email")
@ResponseBody
String home() {
    try{
        sendEmail();
        return "Email Sent!";
    }
    catch(Exception e){
        return "Error in sending email: "+e;
    }
}

//Using Google SMTP server we are able to send emails
//Mime Message helps with
public void sendEmail() throws MessagingException {
    MimeMessage message = mailSender.createMimeMessage();
    MimeMessageHelper mailHelper = new MimeMessageHelper(message);

    int usersOrderNumber = generateRandomOrderNumber();

    mailHelper.setTo("mcdonagh1996@gmail.com");
    mailHelper.setText("Thank you for your passport order. The order has been received. Your order Number is " + usersOrderNumber);
    mailHelper.setSubject("Passport Order Confirmation");
    mailSender.send(message);

}
```

6.4 Input Validation

To stop users making mistakes, I implemented some input validation for my frontend. What this means is that if the user enters their date of birth or any of the other inputs in the incorrect format, they UI will tell them that something is wrong.

It should not be up to the user to know exactly what they are doing. They should be informed if what they have done is correct or not.

I implemented this in Angular, and it is made extremely easy using regex patterns. The user is automatically updated with either a correct or incorrect response.

```
pattern="^([0-2][0-9]|(3)[0-1])(-)(((0)[0-9])|((1)[0-2]))(-)\d{4}$"
```

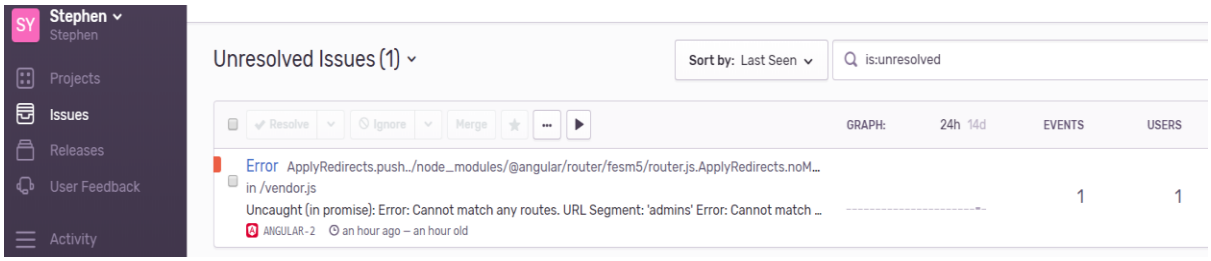
If the pattern is incorrect, this message is displayed

```
<div id="DobIncorrectFormat" *ngIf="applicationForm.controls['dob'].errors && applicationForm.controls['dob'].errors.pattern">
   Please enter a valid date of birth (DD-MM-YYYY)
</div>
```

6.5 Logging & Debugging

Throughout the development phase I found debugging my frontend difficult. This is because it is not possible to write to log files within Angular. To get around this problem I came across Sentry. This allowed me to set up a console where all the errors would be thrown.

I found this extremely useful and setting it up within my application was not too difficult. Below you will see an image of the Sentry dashboard, as well as an unresolved error that was in my application.



For my backend I also implemented logging within Springboot. By making some changes to properties, I was able to implement logging into my application. Below is the changes made to my properties file.

```
logging.level.root=INFO
logging.file=SpringBoot.log

# Logging pattern for the console
logging.pattern.console= "%d{yyyy-MM-dd HH:mm:ss} - %msg%n"

# Logging pattern for file
logging.pattern.file= "%d{yyyy-MM-dd HH:mm:ss} [%thread] %-5level %logger{36} - %msg%n"
```

6.6 Project Architecture

Before starting this project, I had an important choice to make regarding the overall architecture. Within Springboot you can have frontend code and have the whole application running under one server. From research I believe this is also like how Django works. I had two reasons for not choosing to do my project this way:

1. I felt like this was a good opportunity to learn a new framework such as Angular, by choosing to have my project running under Springboot would stop be from getting this experience.
2. From my research it is important to have projects as decoupled as possible. An example of why this is important is that if I made a small CSS change I now only need to restart my angular server rather than shutting down my whole application. I felt like this was a huge positive and was a factor in my decision process.

7 Future Work

Regarding future work, there is still plenty left for me to learn with both Angular and Springboot. As mentioned in my blogs I would like to investigate securing my web application from front to backend. Unfortunately, this was not in the scope of my final year project, but I have already seen possible solutions using Auth0. This is something I would like to research in the future.

After completing my Accessibility testing, the results concluded that my user interface could be more user friendly. My choice of colours for the UI would need to be better if work was to continue with this project.

I have an interest in deploying my application and it is an area I would like to learn more about. Going forward I would like to investigate Docker & deployment to AWS.

8 Conclusion

Overall, I am extremely happy with my final year project and the work I am submitting. My goal when starting this project was to learn as much as possible about web application development and I believe I have accomplished that.

I have learned a lot from frontend development, backend development, implementing facial detection into a web application & testing. These are all skills that will be useful to me when starting to work in industry.

9 References

Development

1. https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#boot-documentation
2. https://angular.io/docs
3. https://www.baeldung.com/spring-boot-start
4. https://dzone.com/articles/building-your-first-spring-boot-web-application-ex
5. https://opencv-java-tutorials.readthedocs.io/en/latest/
6. https://www.baeldung.com/spring-email
7. https://angular.io/guide/http

Testing

1. https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-testing.html
2. https://www.baeldung.com/spring-boot-testing
3. https://www.protractortest.org
4. https://www.tutorialspoint.com/spring_boot/spring_boot_rest_controller_unit_test.htm
5. https://www.tutorialspoint.com/jmeter/