



Face Me – Know Me Interim Report

**TU857
BSc in Computer Science Infrastructure**

**Omair Duadu
C18322011**

Dr. Edina Hatunic-Webster

School of Computer Science
Technological University, Dublin

20/04/2021

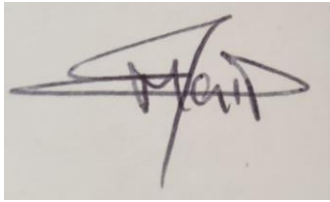
Abstract

This project aims to create an online facial recognition web application that anyone can use to experience what state of the art facial recognition can tell about them. The project consists of two parts: firstly, designing and creating a web application with Express. Secondly, the setup, configuration and maintenance of AWS resources used to host the system. For these gaining a high level of fluency in JSX and AWS resources is essential. The system must not store any user data after being used to protect the user's privacy, and it must incorporate a high level of usability for the user

Declaration

I declare that the work described in this dissertation is, except where otherwise stated, entirely my work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

A handwritten signature in dark ink, appearing to read 'Omair Duadu', is written over a light-colored background. The signature is stylized with a large, sweeping initial 'O' and a long horizontal stroke extending to the right.

Omair Duadu

Date 15/01/2022

Acknowledgements

I would like to thank my supervisor, Dr Edina Hatunic-Webster, for guiding me throughout the project and giving me much needed encouragement and support.

Table of Contents

1. Introduction	10
1.1. Project Background	10
1.2. Project Description	10
1.3. Project Aims and Objectives	11
1.4. Project Scope	11
1.5. Thesis Roadmap	12
2. Literature Review	13
2.1. Introduction	13
2.2. Existing Alternative Solutions to Your Problem	13
2.2.1. BetaFace	13
2.2.2. Facelytics	14
2.2.3. Alternative Solutions Conclusion	14
2.3. Technologies you've researched	14
2.3.1. React (JSX)	14
2.3.2. Python	15
2.3.3. Amazon EC2	15
2.3.4. Amazon S3	15
2.3.5. Amazon Rekognition	16
2.3.6. Node.js	17
2.3.7. Apache	17
2.3.8. Google API	18
2.3.9. Linux	18
2.3.10. Windows	19
2.3.11. HTML, CSS, JavaScript	19
2.3.12. Visual Studio Code	19
2.3.13. NPM	20
2.4. Facial Detection and Facial Recognition	20
2.5. Existing Final Year Projects	20
2.5.1. Face Recognition Application by Patrycjusz Duszek 2018	20
2.5.2. Training Pal by Daniel Tilley, 2018	20
2.6. Conclusions	21
3. System Design	21
3.1. Introduction	21
3.2. Software Methodology	21
3.2.1. Plan-driven Method - Waterfall	22

3.2.2.	<i>Agile Method - Scrum</i>	22
3.2.3.	<i>Planned Approach</i>	23
3.3.	<i>Overview of System</i>	24
3.3.1.	<i>Use Case Diagram</i>	24
3.3.2.	<i>System Resources</i>	24
3.3.3.	<i>Flowchart – System Operation</i>	26
3.4.	<i>UI Wireframes</i>	26
3.4.1.	<i>Demo Page</i>	27
3.4.2.	<i>Home Page</i>	28
3.5.	<i>Conclusions</i>	29
4.	Testing and Evaluation	30
4.1.	<i>Introduction</i>	30
4.2.	<i>Plan for Testing</i>	30
4.3.	<i>Plan for Evaluation</i>	31
4.3.1.	<i>Nielsen's Heuristics</i>	31
4.3.2.	<i>Visibility of System Status</i>	32
4.3.3.	<i>Consistency and Standards</i>	32
4.3.4.	<i>Recognition rather than recall</i>	32
4.4.	<i>Conclusions</i>	32
5.	System Development	32
5.1.	<i>Introduction</i>	32
5.2.	<i>System Development</i>	33
5.2.1	<i>Server</i>	33
5.2.2	<i>Front-end</i>	35
5.3.	<i>Software Development</i>	37
5.3.1	<i>Server Development</i>	37
5.3.1.1.	<i>Package.json</i>	38
5.3.1.2.	<i>Axios</i>	39
5.3.1.3.	<i>Body-parser</i>	40
5.3.1.4.	<i>CORS</i>	40
5.3.1.5.	<i>DOTENV</i>	41
5.3.1.6.	<i>Express</i>	41
5.3.1.7.	<i>Express-fileupload</i>	42
5.3.1.8.	<i>Morgan</i>	42
5.3.1.9.	<i>Multer</i>	42
5.3.1.10.	<i>Nodemon</i>	43

5.3.1.11.	<i>AWS-SDK</i>	43
5.3.1.12.	<i>Paths</i>	43
5.3.2.	<i>Posting</i>	44
5.3.2.1.	<i>Post to Server</i>	44
5.3.2.2.	<i>Uploading to Amazon S3</i>	45
5.3.2.3.	<i>Rekognition Analysis</i>	47
5.3.2.4.	<i>Deleting object from Amazon S3</i>	50
5.4.	<i>Front End</i>	51
5.4.1.	<i>Carousel</i>	51
5.4.2.	<i>Intro to Facial Analysis</i>	52
5.4.2.	<i>How Does Facial Recognition Work</i>	53
5.4.3.	<i>Upload Image</i>	54
5.5.	<i>AWS</i>	57
5.5.1.	<i>AWS IAM</i>	57
5.5.2.	<i>Amazon Simple Storage Service (S3)</i>	57
5.5.3.	<i>Amazon Rekognition</i>	58
5.5.4	<i>AWS SDK</i>	59
6.	<i>Conclusions and Future Work</i>	60
6.1.	<i>Introduction</i>	60
6.2.	<i>Issues and Risks</i>	60
6.3.	<i>Plans and Future Work</i>	60
6.4.	<i>GANTT Chart</i>	61
6.5.	<i>Conclusion</i>	62
	<i>Bibliography</i>	63

Table of Figures

Figure 1: Application Architecture	11
Figure 2: BetaFace Website	13
Figure 3:Facelytics.....	14
Figure 4: Amazon EC2	15
Figure 5: Amazon S3	16
Figure 6: Amazon Rekognition.....	17
Figure 7: NodeJS.....	17
Figure 8: Apache Technology.....	18
Figure 9:Google API.....	18
Figure 10: HTML CSS JavaScript.....	19
Figure 11:Waterfall [19]	22
Figure 12: Agile Method - Scrum [21]	23
Figure 13: UCD of System Functionality	24
Figure 14:System Resources	25
Figure 15: Flowchart of System Operation.....	26
Figure 16: Demo Page	27
Figure 17: Homepage of Application.....	28
Figure 18: Web application URL.....	33
Figure 19: Code snippet of ExpressJS Server	34
Figure 20: Server is listening on port 3000.....	34
Figure 21: Resource folders	34
Figure 22: Folder structure.....	35
Figure 23: Woody Nav Bar	35
Figure 24: Woody Top Carousel.....	36
Figure 25: Woody Team members.....	37
Figure 26: App.js Module require	38
Figure 27: Package.json	39
Figure 28: Npm install modules.....	39
Figure 29: Axios Demonstration.....	40
Figure 30: Body-parser JavaScript library	40
Figure 31: CORS Implementation	40
Figure 32: .env file with config data	41
Figure 33: ExpressJS Implementation for server.....	41
Figure 34: Express-fileupload implement.....	42
Figure 35: Morgan Logger	42
Figure 36: Multer Implementation.....	42
Figure 37: Nodemon Implement	43
Figure 38: AWS-SDK Implement.....	43
Figure 39: App destination paths	44
Figure 40: App.Post-Try-If	45
Figure 41: App.post catch	45
Figure 42: Create a constant avatar.....	45
Figure 43: putObject example.....	46
Figure 44: s3.putObject.....	46
Figure 45: Rekognition example.....	47
Figure 46: Rekognition Parameters for Upload	47

Figure 47: Reckogintion.detectFaces	48
Figure 48: Response data of analysis	49
Figure 49: DeleteObject example	50
Figure 50: S3.deleteObject.....	50
Figure 51:Caroussel	51
Figure 52:Carousel HTML	52
Figure 53: Carousel Javascript.....	52
Figure 54: Intro to Facial Analysis	53
Figure 55: Intro to Facial Analysis counter	53
Figure 56: How does it work?.....	54
Figure 57: Image Upload	54
Figure 58: Image Upload Result	55
Figure 59: formData.....	55
Figure 60: Axios post.....	55
Figure 61: Location to display results.....	56
Figure 62: Error handling client-side	56
Figure 63:Placing result	56
Figure 64: AWS IAM User.....	57
Figure 65: Fyp-img bucket.....	58
Figure 66: Rekognition request.....	58
Figure 67: AWS-SDK.....	59
Figure 68: AWS credentials	59
Figure 69: Attribute Table	61
Figure 70: Gantt Chart	61

1. Introduction

In our daily lives, we are surrounded by many cameras that continue to film and take our pictures from our very own phones to unlock them to our home security, CCTV at work, on the road, shops, and banks. All these cameras inevitable are going to take our data, and no doubt have images of our faces. So, what can they tell about a person from just their face? Many things

This project aims to develop a web application that will be able to detect and distinguish human faces and then output data labels about the faces while building and deploying this on the cloud. The project will examine different fields of technology, such as computer vision science and the cloud.

The web application will take in an image, then upload it to the cloud and run the facial recognition software to examine the image, and then return facial attribute detection in pictures, including gender, age range, emotions, glasses, and facial hair. The application will be accessible to the user on the Internet.

1.1. Project Background

Facial recognition is a way of identifying or confirming an individual's identity using their face. Facial recognition systems can be used to identify people in photos, videos, or in real-time.

Facial recognition is a category of biometric security. Other forms of biometric software include voice recognition, fingerprint recognition, and eye retina or iris recognition. The technology is mainly used for security and law enforcement, though there is increasing interest in other areas of use.

Beyond unlocking phones, facial recognition works by matching the faces of people walking past unique cameras to images of people on a watch list. The watch lists can contain pictures of anyone, including people who are not suspected of any wrongdoing, and the photos can come from anywhere, even from our social media accounts. Facial technology systems can vary, but in general, they tend to operate as follows: detection, analysis, converting the image to data, and finding a match.[1]

This is a field that is expanding rapidly, and as the world is becoming more data-driven, there is no doubt that part of this is the data gained by taking a picture of someone as it is accessible from taking a single image.

1.2. Project Description

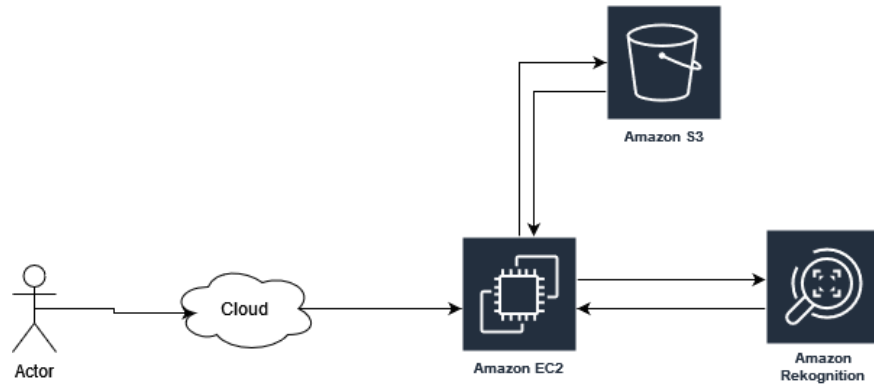


Figure 1: Application Architecture

As shown in the figure above, the user can access the application which is hosted on AWS.

This project is quite complex, and it involves a lot of computer science concepts. Hence there are a few significant challenges. The application will be web-based and hosted on the cloud; therefore, there are many different design choices that can be made. The application will most likely run on React using node.js, and it will have to interact with other AWS APIs and services on the back-end.

On the front end, the user will be presented with a user-friendly UI that will be intuitive and simple to use.

1.3. Project Aims and Objectives

There are many challenges in bringing this project together, especially since it's the first time I have interacted with many of them.

- The first challenge of this project will be to implement the design of the application
- The Express application must work smoothly and efficiently, to not hold back the rest of the system
- Must learn to become familiar with AWS services and APIs
- Integration between my application and AWS services.
- Making the application come live and be accessible on the Internet
- And finally, the infrastructure of the project is a goal to prove and demonstrate the knowledge I gained during my time in college through the many courses I have completed

1.4. Project Scope

The project is primarily about making it available for everyday use to anybody that would find it exciting and relaxed to find out what a computer can discover about them from an image of their face. The inspiration for this is that we see hundreds of cameras all over the place in our everyday lives and wonder what they are truly able to tell about us. With state-of-the-art facial

recognition technology, you can convey someone's age, emotion and many more things with very high accuracy.

Therefore, this project is not aimed at making it possible for governments, big corporations, bad actors and other such entities to analyze people and track them. It is not designed to analyze hundreds or thousands of images at the same time by a single user or client. It is strictly intended for public use

1.5. Thesis Roadmap

Chapter Two Literature review contains all the research that was covered for the designing of the system and building a foundation for software development.

Chapter Three System Design contains all the sketches, use case diagrams, flowcharts, and software methodologies. It is dedicated to creating a base for precise planning of the design, as its name alludes.

Chapter Four Testing and Evaluation is designed to help plan the testing methods and evaluation method, which are very important for a scrum software development approach.

Chapter Five System Development outlines all the elements that consist of the development of the project. It explains all the code and how the system integrates together.

Chapter Six Issues and Future Work is designed to help reflect on the work made by making a comprehensive review of the project. This will illuminate where improvements could be made for future reference in working on the application further.

2. Literature Review

2.1. Introduction

In the first phase of the undertaken research for this project, I have explored the different existing facial recognition applications which are similar to the proposed project. Below are two systems were selected that have identical functionality to the proposed project.

2.2. Existing Alternative Solutions to Your Problem

2.2.1. BetaFace

We offer ready components, such as face recognition SDKs, as well as custom software development services and hosted web services with a focus on image and video analysis and faces and object recognition.

Our technology is used by video and images archives, web advertising and entertainment projects, media content producers, video surveillance and security software solutions, end-user and b2b software developers and others.[2]

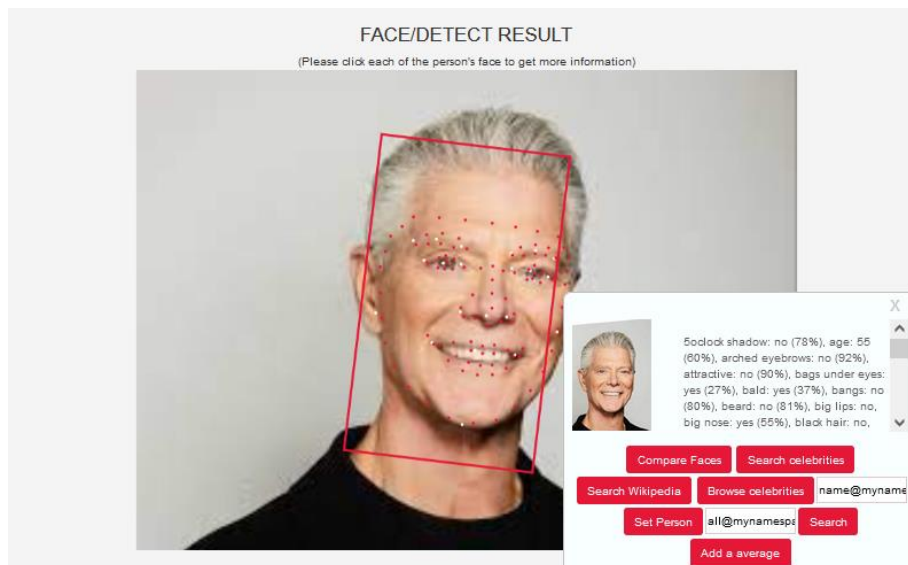


Figure 2: BetaFace Website

Betaface facial recognition suite embraces the whole range of complex operations from fundamental face detection through face recognition (identification, verification or 1:1, 1:N matching) to biometric measurements, face analysis, face and facial features tracking on video, age, gender, ethnicity and emotion recognition, skin, hair and clothes colour detection, hairstyle shape analysis and facial features shape description.[2]

2.2.2. *Facelytics*

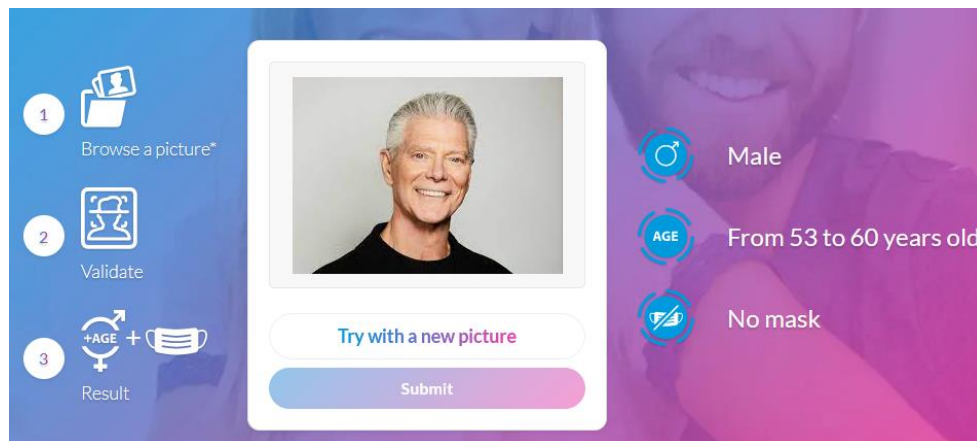


Figure 3:Facelytics

Facelytics is a face recognition solution that is able to detect peoples' morphological criteria such as age and gender by analyzing the video feed in real-time. It relies on any type of camera and can be used directly within your platform or through a cloud-based solution. Thanks to Facelytics, collect data about your visitors, customize content depending on their profile and target your customers.[3]

2.2.3. *Alternative Solutions Conclusion*

The two reviewed applications have many similarities with the proposed system. BetaFace uses facial recognition to offer face recognition SDKs for many different uses in many various fields, from web advertising to video surveillance and security software solutions.[17]Facelytics gathers metrics and data about customers or visitors without bothering them with filling out forms for businesses. However, though these systems are similar to this project, they are not available for use by ordinary users, which is one of the aims. Both BetaFace and Facelytics cater their products suites for use by large organizations.

2.3. *Technologies you've researched*

2.3.1. *React (JSX)*

Billed as a JavaScript library for building user interfaces, React is a Facebook- and Instagram-driven project that provides the "view" part of the MVC (model-view-controller) development paradigm. It was created to enable developers to build large applications with data that changes.[4]

2.3.2. Python

One area where Python shines is web development. Python offers many frameworks from which to choose, including bottle.py, Flask, CherryPy, Pyramid, Django and web2py. These frameworks have been used to power some of the world's most popular sites, such as Spotify, Mozilla, Reddit, the Washington Post and Yelp. The tutorials and articles in this section cover techniques used in the development of Python Web applications and focus on how to program real-world solutions to problems that ordinary people actually want to solve.[5]

2.3.3. Amazon EC2

Amazon Elastic Compute Cloud (Amazon EC2) provides scalable computing capacity in the Amazon Web Services (AWS) Cloud. Using Amazon EC2 eliminates your need to invest in hardware upfront so that you can develop and deploy applications faster. You can use Amazon EC2 to launch as many or as few virtual servers as you need, configure security and networking, and manage storage. Amazon EC2 enables you to scale up or down to handle changes in requirements or spikes in popularity, reducing your need to forecast traffic.[6]

You are using the following Amazon EC2 resources in the Europe (Ireland)

Region:

Instances (running)	0
<u>Dedicated Hosts</u>	0
Elastic IPs	0
Instances	0
Key pairs	0
Load balancers	0

Figure 4: Amazon EC2

2.3.4. Amazon S3

Amazon Simple Storage Service (Amazon S3) is an object storage service that offers industry-leading scalability, data availability, security, and performance. Customers of all sizes and industries can use Amazon S3 to store and protect any amount of data for a range of use cases, such as data lakes, websites, mobile applications, backup and restore, archive, enterprise applications, IoT devices, and big data analytics. Amazon S3 provides management features

so that you can optimize, organize, and configure access to your data to meet your specific business, organizational, and compliance requirements.[7]

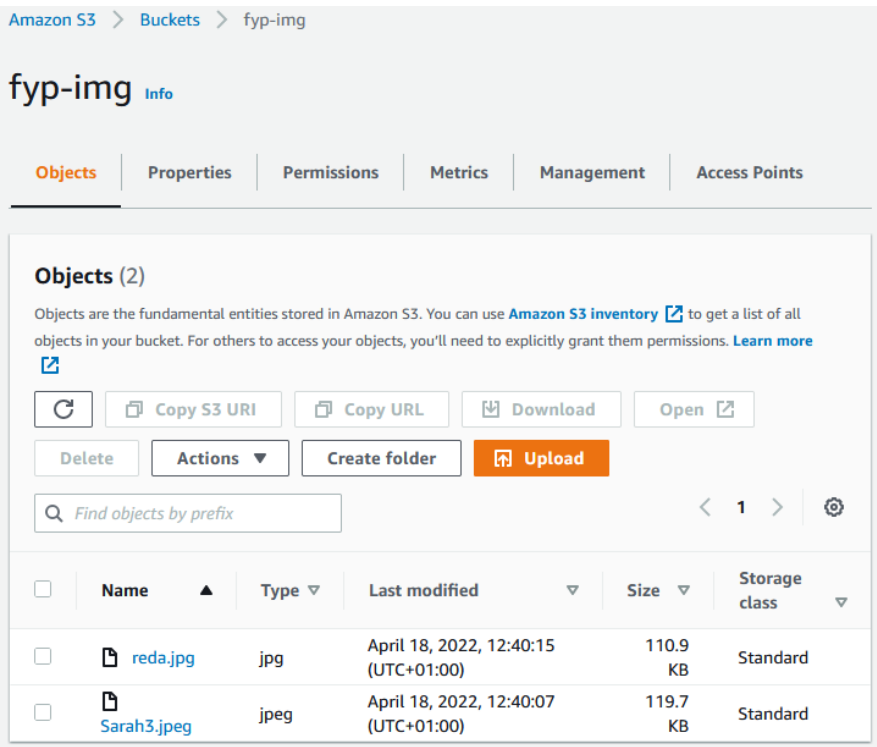
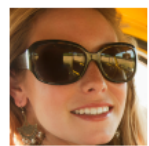


Figure 5: Amazon S3

2.3.5. Amazon Rekognition

Amazon Rekognition makes it easy to add image and video analysis to your applications. You just provide an image or video to the Amazon Rekognition API, and the service can identify objects, people, text, scenes, and activities. It can detect any inappropriate content as well. Amazon Rekognition also provides highly accurate facial analysis, face comparison, and face search capabilities. You can see, analyze, and compare faces for a wide variety of use cases, including user verification, cataloguing, people counting, and public safety.[8]



looks like a face	99.9 %
appears to be female	99.9 %
age range	25 - 35 years old
not smiling	61.1 %
appears to be happy	70 %
wearing glasses	99.8 %

Figure 6: Amazon Rekognition

2.3.6. Node.js

As an asynchronous event-driven JavaScript runtime, Node.js is designed to build scalable network applications. In the following "hello world" example, many connections can be handled concurrently. Upon each link, the call back is fired, but if there is no work to be done, Node.js will sleep.[9]

```
1 // Requiring and importing all the modules
2 const express = require('express');
3 const app = express();
4 const bodyParser = require('body-parser');
5 const fileUpload = require('express-fileupload');
6 const cors = require('cors');
7 const _ = require('lodash');
8 const AWS = require("aws-sdk");
9 const fs = require('fs');
10 const s3 = new AWS.S3();
11 AWS.config.update({region: 'eu-west-1'});
12 const rekognition = new AWS.Rekognition({ region: 'eu-west-1'});
13 const axios = require('axios');
14 const FormData = require('form-data');
15 const multer = require('multer');
16 const morgan = require('morgan');
17 const upload = multer({ dest: './uploads/' });
18 require("dotenv").config();
19
20 module.exports = app;
21
22 const { Rekognition } = require('aws-sdk');
23 const { response } = require('express');
24
25 //add other middleware
26 app.use(cors({
27   origin : "http://localhost:3000",
28   methods : ["GET","POST"],
29   credentials : true
30 }));
```

Figure 7: NodeJS

2.3.7. Apache

Apache is the most widely used web server software. Developed and maintained by Apache Software Foundation, Apache is open-source software available for free. It runs on 67% of all web servers in the world. It is fast, reliable, and secure. It can be highly customized to meet the needs of many different environments by using extensions and modules. Most WordPress hosting providers use Apache as their web server software. However, WordPress can run on other web server software as well.[10]

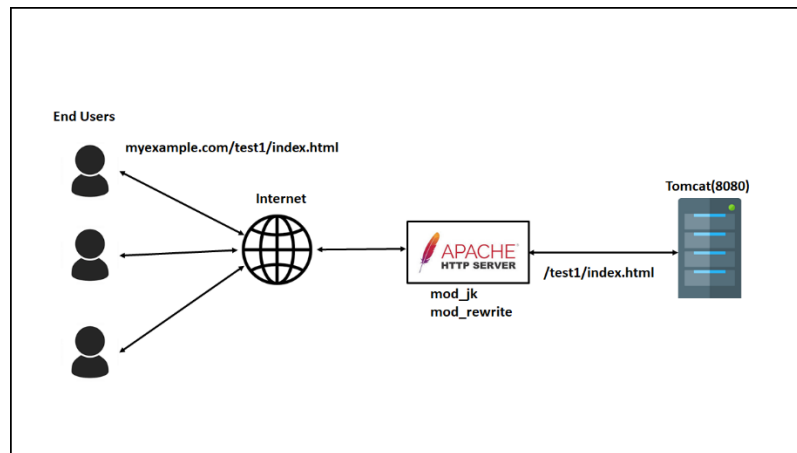


Figure 8: Apache Technology

2.3.8. Google API

Google Cloud APIs are programmatic interfaces to Google Cloud Platform services. They are a vital part of the Google Cloud Platform, allowing you to easily add the power of everything from computing to networking to storage to machine-learning-based data analysis to your applications.[11]



Figure 9:Google API

2.3.9. Linux

Just like Windows, iOS, and Mac OS, Linux is an operating system. In fact, one of the most popular platforms on the planet, Android, is powered by the Linux operating system. An operating system is a software that manages all of the hardware resources associated with your desktop or laptop. To put it simply, the operating system works the communication between your software and your hardware. Without the operating system (OS), the software wouldn't function.[12]

2.3.10. Windows

Microsoft Windows (also referred to as Windows or Win) is a graphical operating system developed and published by Microsoft. It provides a way to store files, run the software, play games, watch videos, and connect to the Internet.

Microsoft Windows was first introduced with version 1.0 on 10 November 1983. Over a dozen versions of Windows were released after that, including the current version, Windows 10.[13]

2.3.11. HTML, CSS, JavaScript

HTML is at the core of every web page, regardless of the complexity of a site or the number of technologies involved. It's an essential skill for any web professional. It's the starting point for anyone learning how to create content for the web.

CSS stands for Cascading Style Sheets. This programming language dictates how the HTML elements of a website should actually appear on the front of the page.

JavaScript is a more complicated language than HTML or CSS, and it wasn't released in beta form until 1995. Nowadays, JavaScript is supported by all modern web browsers and is used on almost every site on the web for more powerful and complex functionality.[14]



Figure 10: HTML CSS JavaScript

2.3.12. Visual Studio Code

Visual Studio Code combines the simplicity of a source code editor with powerful developer tooling, like IntelliSense code completion and debugging.

First and foremost, it is an editor that gets out of your way. The delightfully frictionless edit-build-debug cycle means less time fiddling with your environment and more time executing your ideas.

Visual Studio Code features a lightning-fast source code editor, perfect for day-to-day use. With support for hundreds of languages, VS Code helps you be instantly productive with syntax highlighting, bracket-matching, auto-indentation, box-selection, snippets, and more.[15]

2.3.13. *NPM*

NPM is two things: first and foremost, it is an online repository for the publishing of open-source Node.js projects; second, it is a command-line utility for interacting with the said repository that aids in package installation, version management, and dependency management. Many Node.js libraries and applications are published on NPM, and many more are added every day. These applications can be searched for on <https://www.npmjs.com/>. Once you have a package you want to install, it can be installed with a single command-line command.[16]

2.4. *Facial Detection and Facial Recognition*

Facial recognition is a technology built to identify a person's face from an image or video. This is not a new technology as it has been around for decades. However, as time goes on, it's becoming more integrated into all technology factors in the current data-driven era. Facial recognition can be divided into two parts, detection and recognition. Facial Detection and Recognition are often used interchangeably and used to refer to the same thing; however, there is a significant difference between them.

Simply detection is used to search for the presence of a human's face. It does not care for any attributes or matching it to another look, only the detection of a look. This is very important as many systems rely on this technology to spot if there are any faces in the frame.

Facial recognition allows for facial analysis capabilities by allowing it to understand the attributes those faces have.

2.5. *Existing Final Year Projects*

2.5.1. *Face Recognition Application by Patrycjusz Duszek 2018*

During the project literature review, It is essential to look back at previous accomplishments by alumni as many of the works have many valuable lessons that can be learnt from and appreciated. Patrycjusz Duszek created a facial recognition system that was set up with multiple Raspberry Pi's with cameras attached, acting as a surveillance system. The system can be trained to learn new faces and recognize them if they are in frame. The primary technology focus in this project was the face detection and recognition feature.

There are many benefits to my project's design after reviewing Patrycjusz's system. The project maintains an extreme focus on the face detection aspect, which has encouraged my projects' design to focus more where necessary.

2.5.2. *Training Pal by Daniel Tilley, 2018*

The project designed by Daniel Tilley is genuinely the product of continued refinement and a dedicated iterative process. The project is a health fitness app for logging and setting goals for people who exercise. The initial investment in terms of research is very high, apparently due to the people feedback received from potential users.

Similar to this project, Daniel made his system accessible via a web browser and even created a partially functional mobile app. Due to his research, he ascertained that his user would greatly benefit from the mobile app.

2.6. Conclusions

Many technologies can be used in countless variations for different use cases. However, it is sometimes difficult to pinpoint the exact technologies required. Therefore, much research is necessary for the design process to create the best design in the system architecture stage.

The technologies that will be used in this design are the following:

- | | |
|-------------------------|-----------------------------|
| 1. ExpressJS | - Web Framework |
| 2. Amazon S3 | - Storage |
| 3. Amazon Rekognition | - Facial Rekognition |
| 4. Node.js | - Server |
| 5. NPM | - Open Source Code |
| 6. Visual Studio Code | - Text Editor |
| 7. HTML CSS JavaScript. | - Web programming languages |

3. System Design

3.1. Introduction

This system will allow the users to upload any image to the application, which will be analyzed by facial recognition software and return the data to learn what is revealed about them from a photo of their face. The system will be accessible through the Internet, and there will be a demo image that will have an example image of a person to demonstrate the technique. The system can accept jpeg and png image formats, which will be uploaded to an S3 bucket creating an object. Amazon Rekognition will be run on the newly added S3 object, analyzing any facial data available. The data will be returned and displayed to the user on the home page.

3.2. Software Methodology

The software development methodology is an essential aspect with a tremendous impact on the project. It defines how the development process will lead to the final realization of the final project. Two significant methodologies will be examined to decide the method chosen for this project.

3.2.1. Plan-driven Method - Waterfall

The Waterfall method is a plan-driven method credited to Walter Royce in 1970. It is a suitable method for following a pre-planned strategy where all the steps must be known beforehand. Problems are that the model does not cope well with change, generates a lot of reworks, and leads to unpredictable software quality due to late testing. Therefore, this method does not provide early results and depends on following the plan to get the final product.[18]

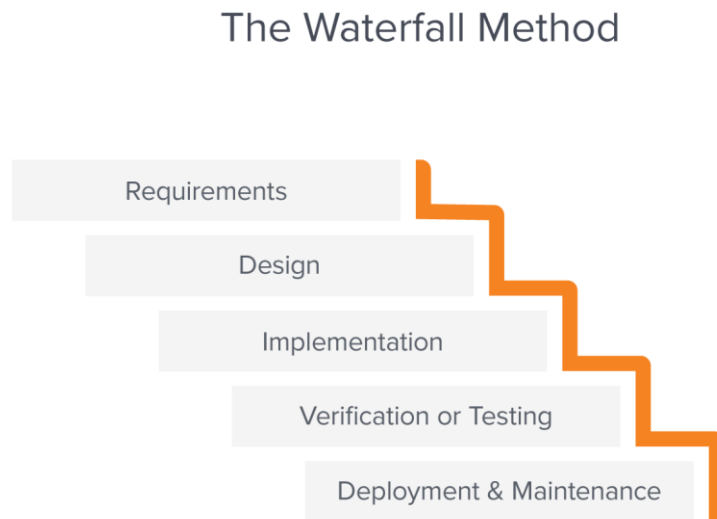


Figure 11:Waterfall [19]

The five main stages of the waterfall method are shown above(Figure 4). The requirement stage is usually contained in a single document, are used to describe each phase of the project, including the costs, assumptions, risks, dependencies, success metrics, and timelines for completion. In the design stage, a higher-level design is created that describes the purpose and scope of the project. Then it's transformed into a physical method using specific hardware and software technologies. Once the design is complete, technical implementation starts, which could be the shortest phase. Testing needs to be done to ensure the product has no errors and all requirements have been completed. Once the software has been deployed in the market or released to customers, the maintenance phase begins.[20]

3.2.2. Agile Method - Scrum

Agile methodologies are based on iterative and incremental development. It is very flexible, allowing changes at any stage of the development process due to generating a lot of feedback early on to improve the product continuously. Therefore, the agile method is highly suited to this project as any feedback during the development can be incorporated early to elevate the project to the highest level possible.

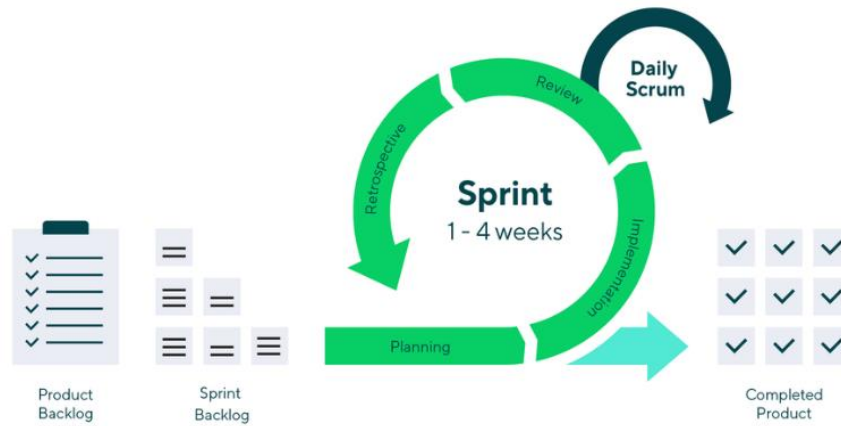


Figure 12: Agile Method - Scrum [21]

Scrum has a long history of proven success and is used in most prominent companies, including Google, Apple, and Facebook. As a framework for project management, it was made in the 1980s first used by companies like Honda and Canon to improve the complex challenge of project management. A decade later, it was introduced into the software development process. A key tenant of the scrum methodology is that no team leader exists, only members who work toward reaching the demands set out by the product owner. The scrum master plays a vital role in bridging the product owner and development team together, being responsible for making sure the unit is on target to hit its goal by the end of the sprint. They run Scrum meetings to diagnose problems the team faces, remove impediments, and make sure the product owner has a reasonable idea of the team's bandwidth.[22]

Scrum breaks workloads into epics and stories, and epics are a large body of work delivered over multiple sprints. Stories are short requirements that are enabler tasks that support the work of completing a feature.[23]

3.2.3. Planned Approach

After considering the suitable methods available, the scrum was selected due to its agile nature and efficient and effective implementation of the division of work. There are two epics in the development of this system, building the React front end application and the setup, configuration, and maintenance of the back-end AWS resources. Each of these epics consists of many stories completed during the sprints. Sprints will be set at weekly intervals for feedback and review.

Epic 1 – ExpressJS application

This will receive priority as more research and learning are still required to complete the interface and communication to the back-end. The stories consist of smaller attainable goals for smooth progression to be achieved every week.

Epic 2 – AWS

This epic is also very critical for the success of the system. With previous module and work experience for the support, this will be approached with care for successful implementation.

3.3. Overview of System

3.3.1. Use Case Diagram

Using case diagrams is an essential tool for outlining and specifying the functionality required of the system. By mapping the processes, the user will navigate through the system logically, and it gives the designer and development team the ability to visualize the system functionality wise. For our system, there is only one role necessary.

Actor – the user will be on the homepage and will have access to all the system's functionality. This includes uploading images and opening the demo page.

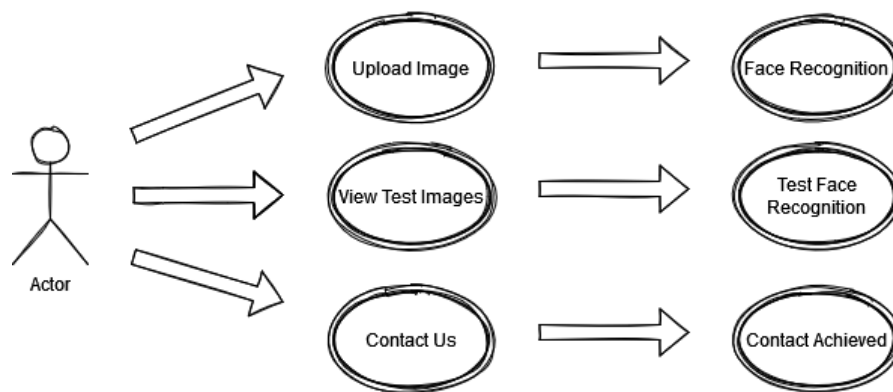


Figure 13: UCD of System Functionality

Figure 6 illustrates the use case diagram of the user. As shown, the user will be able to upload images, resulting in them being processed. After processing, the image data will be returned and displayed on the interface(UI Wireframe below showcasing the interface). The View Test Images functionality allows the user to see what the result will be.

An additional contact us functionality may be added if finished before the schedule. The reference us will allow the user to contact the developer if they have any enquiries, problems, or feedback concerning the system.

3.3.2. System Resources

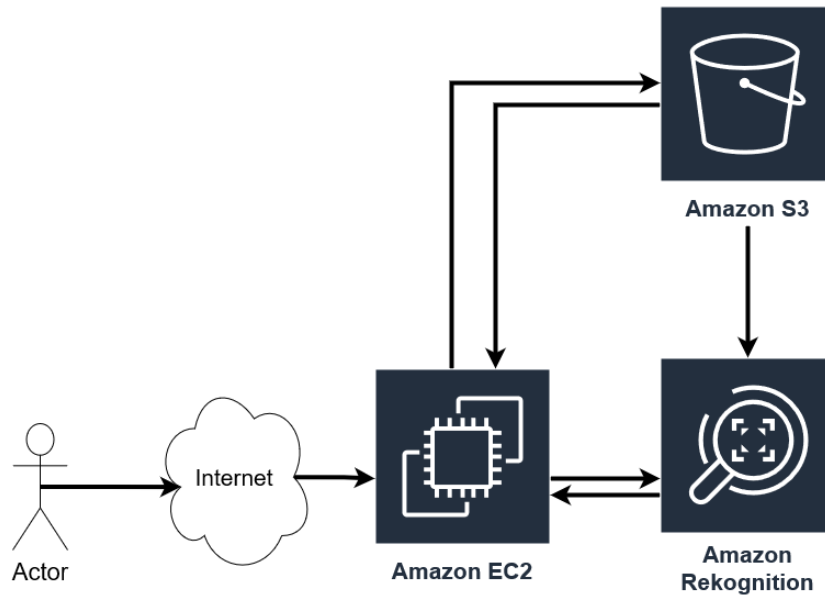


Figure 14: System Resources

Figure 6 illustrates the resource interaction of the system resources. The user visits the web application site hosted on an Amazon EC2 instance. The application(also the instance) can communicate with other AWS resources, which it does through AWS commands. Through this method, it can upload a file to Amazon S3 and create an S3 object. Rekognition will then also be able to access the S3 thing and perform recognition. Data obtained through recognition will be fetched to the application.

3.3.3. Flowchart – System Operation

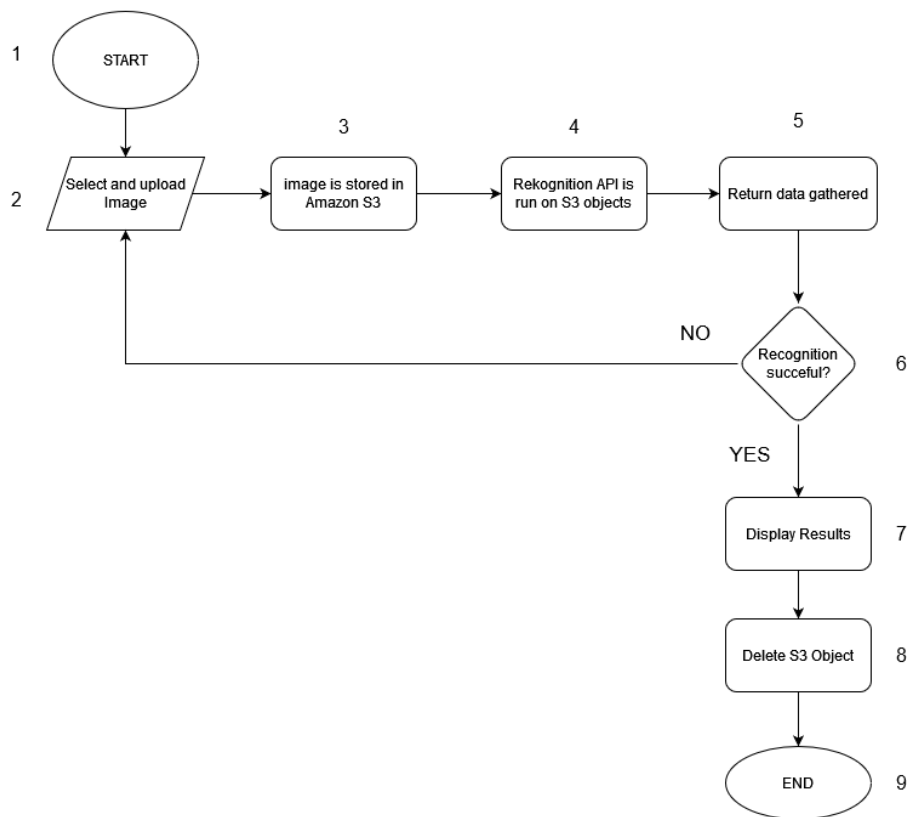


Figure 15: Flowchart of System Operation

Figure 7 above illustrates the systems operations process. (1) Starting the user by opening the web application site. (2) the user selects an image from their filesystem and uploads the image to the site. (3) the system will send the image to an S3 bucket and create an S3 object. (4) Using APIs, the system will feed the S3 object to Amazon Rekognition, which runs facial recognition. (5) The resulting data will be fetched to the application site. (6) The data will be checked for a successful or unsuccessful result. If unsuccessful, error message and return to the second step; if successful, proceed. (7) Display returned results. (8) For data privacy, the image will be S3 object will be deleted. (9) Flow completes and ends.

3.4. UI Wireframes

An interface is the contact point between humans and machines. During the design process of the system interface, drawing wireframes of the design is crucial for the end product. An exceptional advantage is that they are straightforward to make online for a low fidelity prototype. Wireframes can display a user interface's primary and essential features, functions, and content. They typically do not include any styling, colour, or graphics for these reasons. Wireframes are also a helpful tool to test the design assumptions early in the design phase. Once the wireframe is completed with all functionality present, medium-fidelity prototypes are made for more detail, such as colour and graphics. [24]

3.4.1. Demo Page

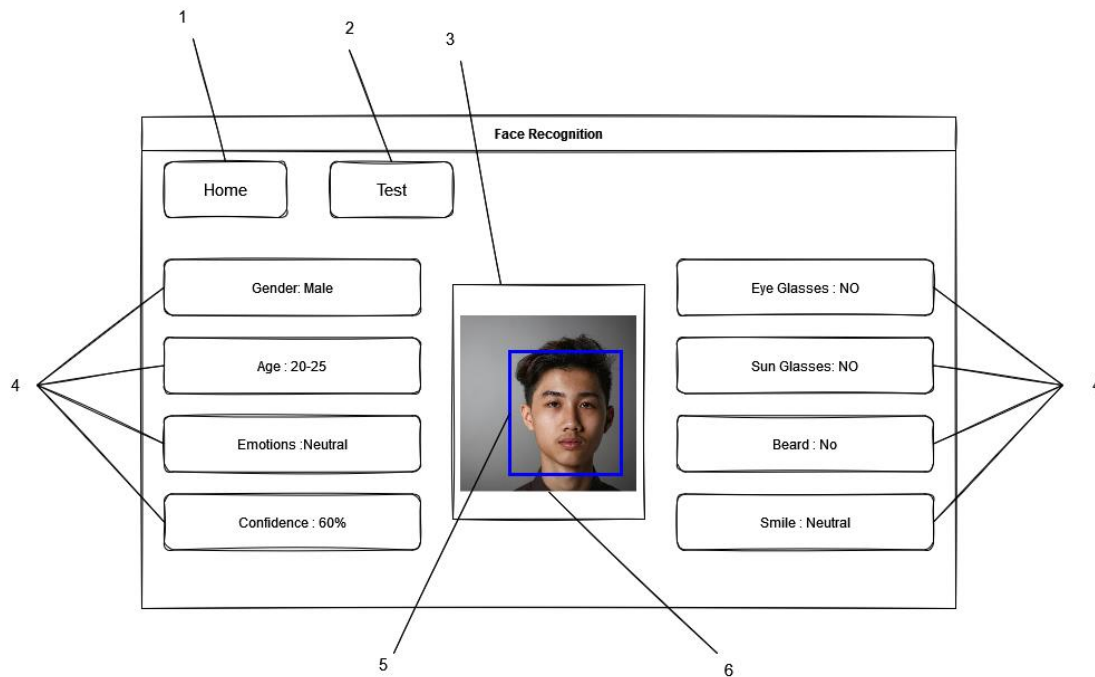


Figure 16: Demo Page

Figure 8 above illustrates the demonstrations page. This page is for demonstrates the result of a successful operation and output. It will display the user's uploaded image in the centre of the screen(3). A blue box will surround the recognized face in the embodiment (5). The attribute boxes to the left and right sides will contain the results(4). The attributes will also have an icon to present each one of them. The user can switch between the working system(1) and the demo system(2).

Wireframe content:

1. Main homepage for the system
2. Test/Demo Page
3. Image box
4. Attributes; Gender, Age, Emotions, Confidence, Eye Glasses, Sun Glasses, Beard, & Smile
5. The blue box surrounding the recognized face
6. Uploaded Image

3.4.2. Home Page

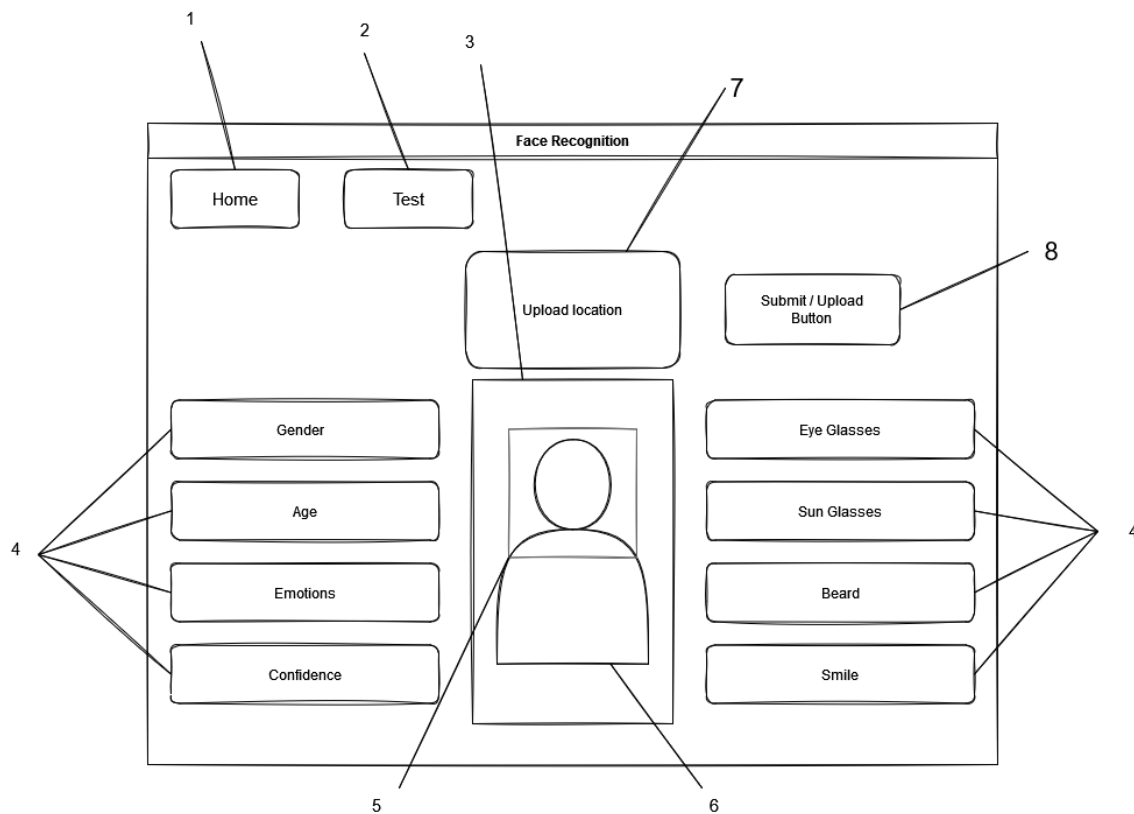


Figure 17: Homepage of Application

The figure above illustrates the homepage. The homepage will contain the main functionality of the system. The Upload functionality will be here in the top middle section with an upload function. Next to it will be the submit button to upload and start the entire operation. When the procedure is successful, it will fill out the attribute boxes(4) and image boxes (3).

Wireframe content:

1. Main homepage for the system
2. Test/Demo Page
3. Image box
4. Attributes; Gender, Age, Emotions, Confidence, Eye Glasses, Sun Glasses, Beard, & Smile
5. The blue box surrounding the recognized face
6. Uploaded Image
7. Upload Location
8. Upload / Submit Button

3.5. *Conclusions*

After researching all the technology completed in the previous chapter, the system has now been refined to what the necessary functionality and features are required for the entire operation. At the beginning of this chapter, different development methodologies were researched to discover the most optimal methods for our use case in this system. We have also divided the system into epics of the scrum ecosystem. This will help clarify and specify the target/goals for each sprint so that casual distractions are avoided. A significant advantage of using scrum is that there will be a lot of feedback early on and throughout the entire sprint. The first epic concerns the front end of the system, and this includes but is not limited to UI/UX, efficient and effective code for front end functionality and features.

Secondly, a use case diagram was created for the user operations; this guaranteed that all functionality and features that the user needs are present so that any design work going forward will benefit from the specifications identified. The UI wireframe and flowchart were iterated multiple times due to reviewing the UCD.

Thirdly, the flowchart was created to logically map the process from start to finish for a successful operation. Also, the flowchart was essential in helping to calculate all the points of failure of the system. Ideally, during development, the system flowchart can be analyzed when faults occur to help spot "weak" points in the system by analyzing every flow/step.

Fourthly, a UI Wireframe was created for the two main pages: the homepage and demo page. This is a necessary procedure as it helps visualize the result and examine if any missing functionality was missing that the user might expect. The wireframes were created and styled to be as intuitive as possible while maintaining visual aesthetics. All these factors combined ensure the best user experience possible while fully displaying the system's functions.

4. Testing and Evaluation

4.1. Introduction

As with any system created, one of the most critical stages is testing, which is even more crucial while handling valuable data. Software development has also always been a highly complex process. Therefore, testing and evaluation throughout the development lifecycle are essential, as leaving mistakes and bugs for later can cause significant disturbances and time delays.

4.2. Plan for Testing

Having a concrete plan for testing the system is the first and most crucial step for correctly testing the system. The purpose behind each test that is performed is to ensure that all project functionality is present and fully working. The Positive Testing method will be used for each test. Seven tests will be completed for full functionality testing.

Test Case	Test Steps	Preconditions	Expected Result	Actual Result
1. Connecting to the System	1. Connect to system URL	1. Server is live and accessible	1. Connection Successful	Pass
2. Uploading and submitting the image to the system	1. Adding file upload section 2. pressing submit	1. Have an image for upload	2. Image reaches the server	Pass
3. Creating an S3 Object	1. Run API for upload	1. Have the image upload on the server	3.S3 object created	Pass
4. Running facial recognition on the upload	1. Run API for Amazon Rekognition with correct details	1. Credential and access to the bucket	4. API run successfully, and data generation	Pass
5. Fetching data generated	1. Fetch request	1. Amazon Rekognition runs	5. Data returned	Pass
6. Analyzing data for success or failure		1.HTTP return code	6. HTTP Code 200	Pass

7. Displaying data on the homepage	1. Fetch handling	1. Functional front end programming	7. Data appears on the homepage	Pass
8. Deleting the S3 object	S3.deleteObject API	1. Credential and access to the bucket	8.Object is deleted	Pass

4.3. Plan for Evaluation

Much more evaluation has to be completed for the design's security, performance, and usability aspect. These are all factors that can be tested during the development stage. They mustn't be overlooked as the system will operate with user face images and the data generated from it, which is very highly sensitive. If a breach of trust occurs during the system's operation, the user can be highly negatively affected.

4.3.1. Nielsen's Heuristics

The system's usability is crucial if it is to be used comfortably by users. An excellent usability evaluation of the project could turn it into a great project. A bad usability score on a project can turn a good project into a bad one if the user is uncomfortable and not inclined to use the system again.

Nielsen's heuristics has seven components, each identifying a particular design aspect. They are; Visibility of system status, Match between system and the real world, User control and freedom, consistency and standards, Error prevention, Minimalist design, Documentation.

Visibility of system status states that the user is aware of what is happening when using the system. Progress bars, loading signs, and pop-ups can make users aware or grab their attention. The uploading feature will contain progress as an indicator

Match between system and real-world – the system should be familiar to the intended user. For example, the system should be in English for English speakers. Uncomplicated and straightforward processes are all that are needed. No advanced long unfamiliar words as well. The simplicity of the system is essential.

User control and freedom – Users can start or cancel operations to maintain control of the system. The system is a web page with only a few clicking possibilities, and procedures can be cancelled by refreshing or closing the tab.

Consistency and standards should be uniform; the same fonts, design, colours, and navigation bar across the system. Things do not appear messy or inconsistent.

Error prevention – the system should consider user faults and, therefore, be designed to be resilient against errors. The system only has five or so buttons on each page for simplicity.

Minimalist design – Similar to matching between system and real-world use, the system should seek to simplify rather than complicate with complicated words.

Documentation – Guides should be available to the user to learn the operation of the application.

The system design achieves all these goals with a high score with its simplistic and modern design.

4.3.2. Visibility of System Status

The system's status must be visible to the user through an appropriate amount of feedback in a simple visual manner to be aware of what is happening. This is achieved by making distinct and separate sections in the website by having big clear headers.

4.3.3. Consistency and Standards

The theme and design of the application remain consistent throughout the application. The font, language, bootstrap classes, and colour scheme are consistent throughout the different pages.

4.3.4. Recognition rather than recall

The different elements in the system are all very visible and remain consistent while maintaining their simplicity, colour and responsiveness.

4.4. Conclusions

The testing phase is crucial as it is a clear signifier of whether the system is ready to be used. If any of the tests mentioned above would have failed, it signifies a significant issue that will hamper the system's completion. However, this is assuredly not all the testing and evaluation that can be made. More in detail, positive functionality tests can be made, and code evaluation will also be on the agenda in the development & testing stages.

The overall evaluation of the project inspires confidence that its users will be left with a satisfactory experience. However, there is always more that could be done to add more error prevention to improve the project's usability.

5. System Development

5.1. Introduction

The development of the design and the finished web application is discussed here. Research sources such as code and reading material for the technologies used in the process are also displayed and explained in this chapter.

The main focus of the design process is to understand and identify the technologies that will be necessary to support the project and ensure all features and elements are integrated successfully.

5.2. System Development

The web application will have two distinct sections, an introduction/educational segment and the facial analysis feature. The academic component was designed to be as modern, user friendly as possible, and intuitive to use for the user to gain a deeper understanding of facial analysis. The facial analysis segment also needs to be intuitive.

The front end features are all kept in mind and need to begin development so that they may be extrapolated. The initial design is never to the published or even late-stage development product level, as more design/feature changes can be implemented later.

The server will be running the key technologies discovered in the research stage for integrated functionality.

5.2.1 Server

Using npm init, a project is initialized, and modules can be added. The first and cornerstone for all of these are ExpressJS, as it will run the server on port 3000. With a live server on the localhost, the web application can be officially accessed from a web browser.

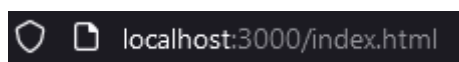


Figure 18: Web application URL

```
// Requiring and importing all the modules
const express = require('express');
const app = express();

// Below is all the listeners to move between the pages in the
app.listen(3000, function(){
  console.log("Server is running on port 3000")
});

app.get("/", function(req, res){
  res.sendFile(__dirname+"/index.html");
});

app.get("/index.html", function(req, res){
  res.sendFile(__dirname+"/index.html");
});
```

Figure 19: Code snippet of ExpressJS Server

```
PS C:\Users\omair\Desktop\Image Rec> npm run watch

> image-rec@1.0.0 watch
> nodemon ./app.js

[nodemon] 2.0.15
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node ./app.js`
Server is running on port 3000
```

Figure 20: Server is listening on port 3000

The terminal displays server status and any issues, requests or posts.

```
app.use(express.static(__dirname + "/public"));
app.use('/img', express.static('img'));
app.use('/css', express.static('css'));
app.use('/js', express.static('js'));
app.use('/lib', express.static('lib'));
app.use('/scss', express.static('scss'));
app.use('/uploads', express.static('uploads'));
app.use('/html', express.static('html'));
app.use('/node_modules', express.static('node_modules'));
```

Figure 21: Resource folders

 css	17/04/2022 4:45 PM	File folder
 html	17/04/2022 4:48 PM	File folder
 img	16/04/2022 6:34 PM	File folder
 js	16/04/2022 6:38 PM	File folder
 lib	14/03/2022 5:09 PM	File folder
 scss	04/04/2022 2:53 PM	File folder
 uploads	17/04/2022 2:47 PM	File folder

Figure 22: Folder structure

The CSS, HTML, javascript, and other resource folders and files, are stored in public, where they can be served to the server and client.

5.2.2 Front-end

The front end of an application is significant as it presents the user with what they will be interacting with. Using the best front-end technologies is essential for user likeability after researching different templates and other resources that can improve the front-end's operability.

After finding an advanced template that uses Bootstrap to provide an interactive website, I decided to adopt this template as it can allow me to make my web application to the next level. This template is free and will enable you to make changes to it. I have made many profound changes to the template to suit the purpose of the application.

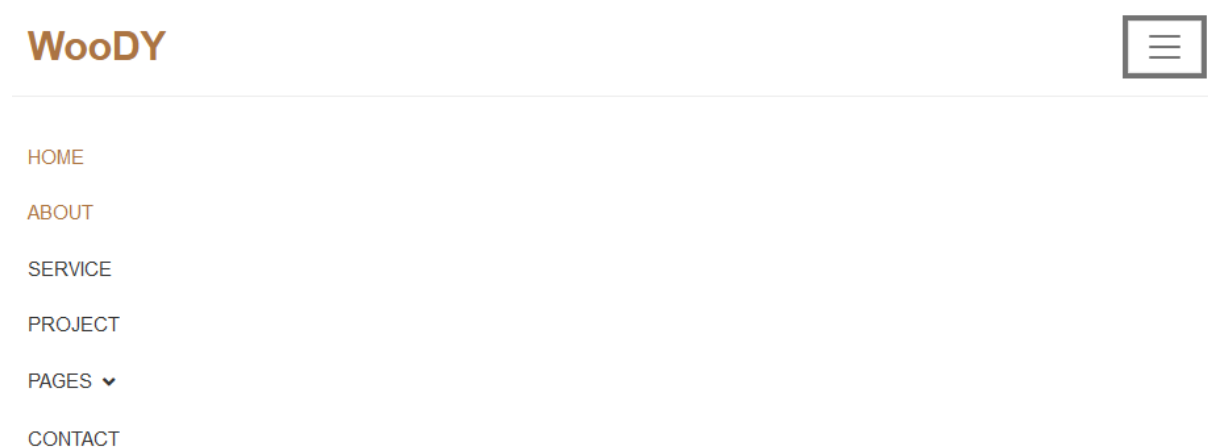
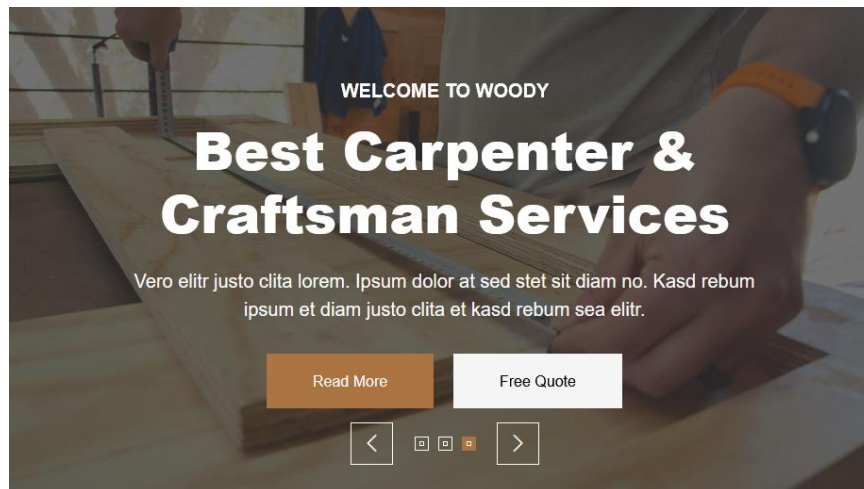


Figure 23: WooDY Nav Bar



Creative Designers

01



Quality Products

02



03



04

Figure 24: WooDY Top Carousel

This is the top section of the website that contains a carousel that displays applications' core purpose action and sets expectations for the website's contents.

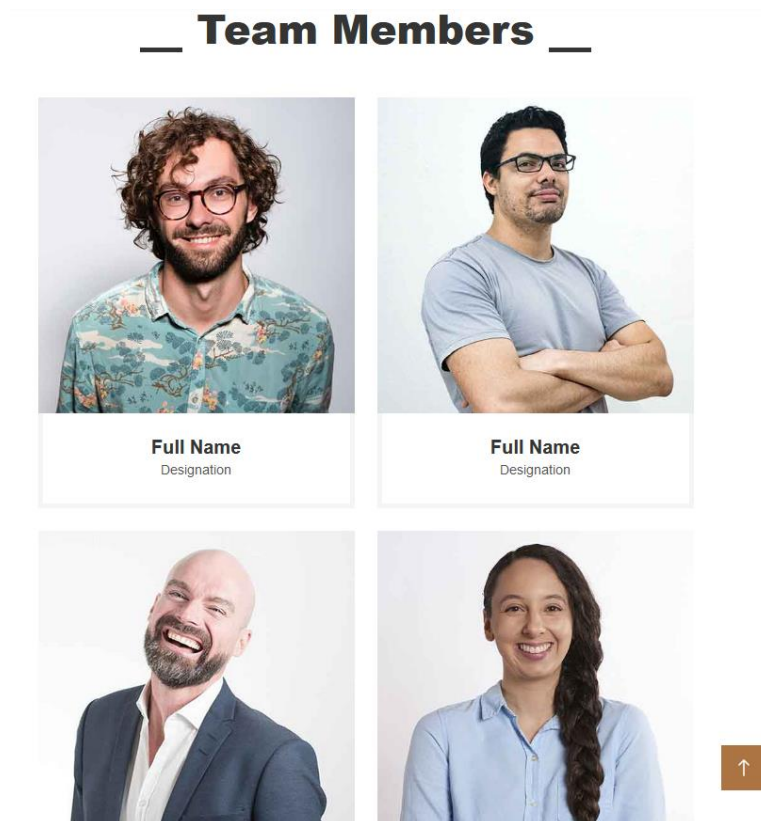


Figure 25: Woody Team members

Example of the contents the template provides. However, the developer sees fit, and all these can be edited, removed, and added upon.

5.3. Software Development

Now that the initial/prototype design of the website is complete, we can start working on the back-end, as it is critical for the main functionality of the web application.

5.3.1 Server Development

The first thing for any NodeJS project is to use `npm init` to start. After adding the module to make the service life, the next step is to manage the file resources orderly. I have placed all of the HTML, CSS, and JavaScript resources in the public folder with other sub-directories to store them, depending on file type.

```
// Requiring and importing all the modules
const express = require('express');
const app = express();
const bodyParser = require('body-parser');
const fileUpload = require('express-fileupload');
const cors = require('cors');
const _ = require('lodash');
const AWS = require("aws-sdk");
const fs = require('fs');
const s3 = new AWS.S3();
AWS.config.update({region: 'eu-west-1'});
const rekognition = new AWS.Rekognition({ region: 'eu-west-1'});
const axios = require('axios');
const FormData = require('form-data');
const multer = require('multer');
const morgan = require('morgan');
const upload = multer({ dest: './uploads/' });
require("dotenv").config();
```

Figure 26: App.js Module require

5.3.1.1. Package.json

The next step was to find all the modules that would be needed. These will be included in the package.json as dependencies. Below is an image of the package.JSON.

```

{} package.json > {} dependencies
1  {
2    "name": "image-rec",
3    "version": "1.0.0",
4    "description": "",
5    "main": "app.js",
6    "directories": {
7      "lib": "lib"
8    },
9    "scripts": {
10     "test": "echo \"Error: no test specified\" && exit 1",
11     "start": "node app.js",
12     "watch": "nodemon ./app.js"
13   },
14   "author": "Omair Duadu",
15   "license": "ISC",
16   "dependencies": {
17     "@aws-sdk/client-rekognition": "^3.58.0",
18     "aws-sdk": "^2.1101.0",
19     "axios": "^0.26.1",
20     "body-parser": "^1.19.2",
21     "cors": "^2.8.5",
22     "dotenv": "^16.0.0",
23     "express": "^4.17.3",
24     "express-fileupload": "^1.3.1",
25     "lodash": "^4.17.21",
26     "morgan": "^1.10.0",
27     "multer": "^1.4.4",
28     "nodemon": "^2.0.15",
29     "vanilla-require": "^1.0.2"
30   }
31 }
32

```

Figure 27: Package.json

```

PS C:\Users\omair\Desktop\Image Rec> npm install axios, body-parser, cors, dotenv, express, express-fileupload, lodash, morgan, multer, nodemon, vanilla-require, aws-sdk --save

```

Figure 28: Npm install modules

This is the command to install all of the necessary modules for NodeJS. Each of them integrates to complete all of the functionality required.

5.3.1.2. Axios

JavaScript library is used to make HTTP requests from node.js. It enables the upload and functionality using Axios.post().

```

postJson = (postData) => {
  axios.post('/scene-setup.json', {
    postData
  })
  .then(function (response) {
    console.log("success!");
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  });
}

```

Figure 29: Axios Demonstration

5.3.1.3. *Body-parser*

This is a NodeJS library used as body parsing middleware, and it is responsible for parsing the incoming request bodies before you can handle them. This makes the incoming data to the server easier to handle.

```

const bodyParser = require('body-parser');
app.use(bodyParser.urlencoded({ extended: true, limit: '5mb' }));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({extended: true}));

```

Figure 30: Body-parser JavaScript library

5.3.1.4. *CORS*

Cross-Origin Resource Sharing is an HTTP header that allows a server to indicate the origins of resources other than its own from which the browser should permit loading. This is used for resources from domains, schemes and ports.

```

const cors = require('cors');

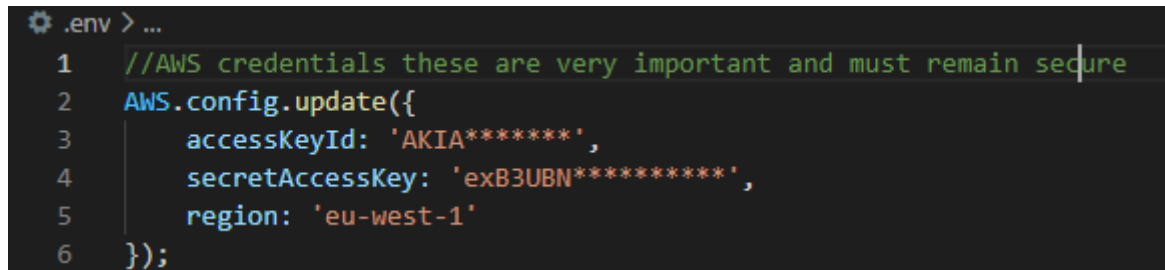
//add middleware
app.use(cors({
  origin : "http://localhost:3000",
  methods : ["GET","POST"],
  credentials : true
}));

```

Figure 31: CORS Implementation

5.3.1.5. DOTENV

This is used for loading lightweight npm packages that automatically load environment variables from a .env file into the process.

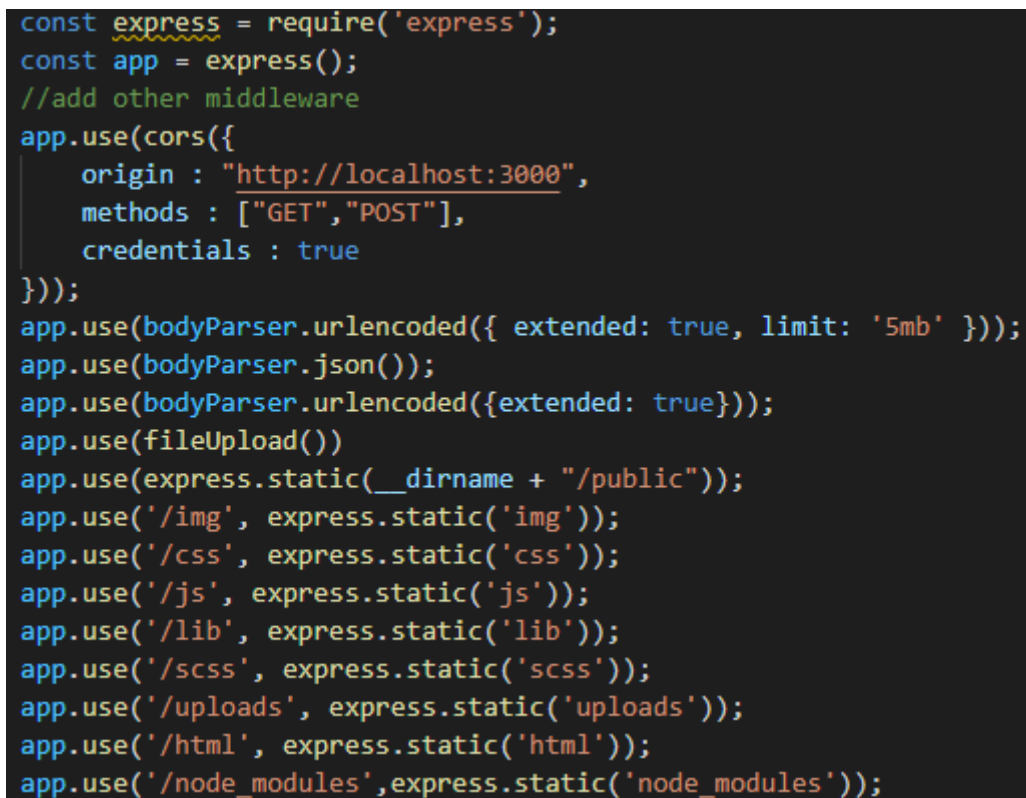
A screenshot of a code editor showing a .env file. The file contains a comment on line 1: '//AWS credentials these are very important and must remain secure'. Lines 2 through 6 contain the AWS.config.update function call with the following parameters: accessKeyId: 'AKIA*****', secretAccessKey: 'exB3UBN*****', and region: 'eu-west-1'. The code is as follows:

```
1 //AWS credentials these are very important and must remain secure
2 AWS.config.update({
3   accessKeyId: 'AKIA*****',
4   secretAccessKey: 'exB3UBN*****',
5   region: 'eu-west-1'
6 });
```

Figure 32: .env file with config data

5.3.1.6. Express

The back-end web application framework for Node.js is designed for building web applications and APIs. It is the standard server framework for Node.js.

A screenshot of a code editor showing the ExpressJS implementation for a server. The code starts with requiring 'express' and creating an app. It then adds several middleware functions: cors (with origin 'http://localhost:3000', methods 'GET' and 'POST', and credentials true), bodyParser.urlencoded (with extended true and limit '5mb'), bodyParser.json, and another bodyParser.urlencoded (with extended true). It then adds fileUpload() and several static middleware functions for different file types and directories: __dirname + '/public', /img, /css, /js, /lib, /scss, /uploads, /html, and /node_modules. The code is as follows:

```
const express = require('express');
const app = express();
//add other middleware
app.use(cors({
  origin : "http://localhost:3000",
  methods : ["GET","POST"],
  credentials : true
}));
app.use(bodyParser.urlencoded({ extended: true, limit: '5mb' }));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({extended: true}));
app.use(fileUpload())
app.use(express.static(__dirname + "/public"));
app.use('/img', express.static('img'));
app.use('/css', express.static('css'));
app.use('/js', express.static('js'));
app.use('/lib', express.static('lib'));
app.use('/scss', express.static('scss'));
app.use('/uploads', express.static('uploads'));
app.use('/html', express.static('html'));
app.use('/node_modules', express.static('node_modules'));
```

Figure 33: ExpressJS Implementation for server

5.3.1.7. *Express-fileupload*

Express-fileupload is a library that provides an easy way for you to handle file upload using the express framework.

```
34 app.use(fileUpload())
```

Figure 34: Express-fileupload implement

5.3.1.8. *Morgan*

Morgan is a library used as a logger. This is important for logging any changes to the servers.

```
morgan(function (tokens, req, res) {  
  return [  
    tokens.method(req, res),  
    tokens.url(req, res),  
    tokens.status(req, res),  
    tokens.res(req, res, 'content-length'), '-',  
    tokens['response-time'](req, res), 'ms'  
  ].join(' ')  
})
```

Figure 35: Morgan Logger

5.3.1.9. *Multer*

Multer is a middleware/library for handling multipart/form-data, primarily used for uploading files. This is important for uploading through Axios as it is used for data handling.

```
const multer = require('multer');  
const morgan = require('morgan');  
const upload = multer({ dest: './uploads/' });
```

Figure 36: Multer Implementation

5.3.1.10. *Nodemon*

Nodemon is used for helping the development of NodeJS applications by automatically restarting the node application whenever changes are made to a file. This is very helpful for the product as it ensures the page is always the newest version.

```
"watch": "nodemon ./app.js"
```

Figure 37: Nodemon Implement

5.3.1.11. *AWS-SDK*

The AWS SDK is for browser-based development allowing developers to access AWS from JavaScript code running directly in the browser.

```
const AWS = require("aws-sdk");  
const fs = require('fs');  
const s3 = new AWS.S3();  
AWS.config.update({region: 'eu-west-1'});  
const rekognition = new AWS.Rekognition({ region: 'eu-west-1'});  
const { Rekognition } = require('aws-sdk');  
const { response } = require('express');
```

Figure 38: AWS-SDK Implement

5.3.1.12. *Paths*

The server includes the destination for all the links to the pages on the website. These are all predefined in their specific folder.

```

// Below is all the listeners to move between the pages in the
app.listen(3000, function(){
  console.log("Server is running on port 3000")
});

app.get("/", function(req, res){
  res.sendFile(__dirname+"/index.html");
});

app.get("/index.html", function(req, res){
  res.sendFile(__dirname+"/index.html");
});

app.get("/about.html", function(req, res){
  res.sendFile(__dirname+"/public/html/about.html");
});

app.get("/service.html", function(req, res){
  res.sendFile(__dirname+"/public/html/service.html");
});

app.get("/project.html", function(req, res){
  res.sendFile(__dirname+"/public/html/project.html");
});

app.get("/contact.html", function(req, res){
  res.sendFile(__dirname+"/public/html/contact.html");
});

```

Figure 39: App destination paths

5.3.2. Posting

The front-end will use Axios to post data to the server in the back-end. This post will enable all the functionality from uploading the file to the back-end, uploading it to Amazon S3, calling Rekognition, and then deleting the s3 object.

5.3.2.1. *Post to Server*

The below code is on the server-side and receives the upload from Axios post's client-side. Inside the post is a try-catch used to receive the files. Inside the try an if statement checks if the request body contains any files. If it doesn't find any files in the request, the if statement will return a status 400 for a bad request and a "no file uploaded" message.

```
//Image upload post and functionality
app.post('/upload', (req, res) => {
  try {
    if(!req.files) {
      res.status(400);
      res.send({
        status: false,
        message: 'No file uploaded'
      });
    }
  }
});
```

Figure 40: App.Post-Try-If

At the end of the try, a catch is used to return a status 500 Internal Server Error, which means something has gone wrong on the website's server and what the error is. The error is also logged on the console. This can help the troubleshooting and identification of the fault.[]

```
} catch (err) {
  res.status(500).send(err);
  console.log(err)
}
```

Figure 41: App.post catch

The error data returned to the user is also displayed below the file upload location. The error logging level is very comprehensive, which is very convenient for the user to understand what may go wrong.

5.3.2.2. Uploading to Amazon S3

```
console.log(req.files.file);

//Use the name of the input field (i.e. "avatar") to retrieve the uploaded file
let avatar = req.files.file;

//Use the mv() method to place the file in upload directory (i.e. "uploads")
//avatar.mv('./uploads/' + avatar.name);
```

Figure 42: Create a constant avatar

In the else statement, the first thing to occur is creating a new constant called avatar which will hold the JSON format data of the file. Avatar is used throughout the rest of the program instead of the req.files.file. If we wanted to gather and store the users' data on the server, it is possible to move it to a folder ./uploads/ using .mv from the fs library. However, we will not include this feature in the system. If we were to implement saving user data, it is essential to disclose this to the user for data privacy regulations such as GDPR.[]

```

/* The following example uploads an object to a versioning-enabled bucket

var params = {
  Body: <Binary String>,
  Bucket: "examplebucket",
  Key: "HappyFace.jpg"
};
s3.putObject(params, function(err, data) {
  if (err) console.log(err, err.stack); // an error occurred
  else     console.log(data);           // successful response
  /*
  data = {
    ETag: "\"6805f2cfc46c0f04559748bb039d69ae\"",
    VersionId: "tpf3zF08nBplQK1XLOefGskR7mGDwcDk"
  }
  */
});

```

Figure 43: putObject example

The above is an example from the AWS documentation where all examples for most APIs can be found. I learned that a few critical pieces of information are necessary from it. The upload must have three parameters: the body that contains the binary string data of the object being uploaded; the bucket name, which is a unique global name; and the Key, which is the object's name.[]

```

//Amazon S3 object upload.
(async() => {
  await
  s3.putObject({
    Body: fs.readFileSync(avatar.name),
    Bucket: "fyp-img",
    Key: avatar.name,
  }).promise();
})();

```

Figure 44: s3.putObject

Using the AWS SDK available for NodeJS, I adapted the API above to the system's needs so that it is simple. This object upload does not need to account for versioning as it is not configured on the bucket as it is required.

5.3.2.3. *Rekognition Analysis*

```
/* This operation detects faces in an image stored in an AWS S3 bucket. */

var params = {
  Image: {
    S3Object: {
      Bucket: "mybucket",
      Name: "myphoto"
    }
  }
};
rekognition.detectFaces(params, function(err, data) {
  if (err) console.log(err, err.stack); // an error occurred
  else    console.log(data);           // successful response
});
```

Figure 45: Rekognition example

This example code above shows how a `rekognition.detectFaces` API call is made. Like `s3.putObject`, you need to define the parameters; however, cleverly, `rekognition` can use buckets of resources. Therefore, in the API parameters, you only need to define the bucket and the name of the object you want to analyze.

```
// Set params for rekognition
const params = {
  Image: {
    S3Object: {
      Bucket: 'fyp-img',
      Name: avatar.name
    },
  },
  Attributes: ['ALL']
};
```

Figure 46: Rekognition Parameters for Upload

In the parameters for the `detectFaces`, I added the `"Attributes: ['ALL']"`, which will return all the analyzed attributes of the face.

```

//Timeout function to give time for the object to be uploaded
setTimeout(() => {
  //analyse face with rekognition
  rekognition.detectFaces(params, function(err, response) {
    if (err) {
      console.log(err, err.stack); // an error occurred
    } else {
      console.log(`Detected faces for: ${avatar.name}`)

      response.FaceDetails.forEach(data => {
        return data
      }) // for response.faceDetails
      //returning the response data from the
      res.send({faceData : response.FaceDetails})
    } // if
    // res.send(data)
  });
}, 2500);

```

Figure 47: Reckogintion.detectFaces

The above API is the final iteration. Running multiple APIs after each other, which rely on the result of each additional development, may cause issues due to network delay, as this happened here. Before the `s3.putObject` was fully complete, the `detectFaces` would be called, resulting in errors. I placed a `setTimeout` that waited for 2.5 seconds before running the second API. This allowed me to avoid network delay errors.

The response of the `detectFaces` API is in JSON and contains all the facial analysis data.


```

Detected faces for: Sarah.jpeg
[
  {
    BoundingBox: {
      Width: 0.4497867226600647,
      Height: 0.28632158041000366,
      Left: 0.2589224874973297,
      Top: 0.306306391954422
    },
    AgeRange: { Low: 4, High: 10 },
    Smile: { Value: true, Confidence: 54.88467788696289 },
    Eyeglasses: { Value: false, Confidence: 96.95269775390625 },
    Sunglasses: { Value: false, Confidence: 98.78929901123047 },
    Gender: { Value: 'Female', Confidence: 99.9991226196289 },
    Beard: { Value: false, Confidence: 91.55838012695312 },
    Mustache: { Value: false, Confidence: 97.06986236572266 },
    EyesOpen: { Value: true, Confidence: 93.85983276367188 },
    MouthOpen: { Value: false, Confidence: 79.83370971679688 },
    Emotions: [
      [Object], [Object],
      [Object], [Object],
      [Object], [Object],
      [Object], [Object]
    ],
    Landmarks: [
      [Object], [Object], [Object],
      [Object], [Object], [Object],
      [Object], [Object], [Object],
      [Object], [Object], [Object],
      [Object], [Object], [Object],
      [Object], [Object], [Object],
      [Object], [Object], [Object],
      [Object], [Object], [Object],
      [Object], [Object], [Object],
      [Object], [Object], [Object]
    ],
    Pose: {
      Roll: -1.146126389503479,
      Yaw: -0.5575604438781738,
      Pitch: 1.5431773662567139
    },
    Quality: { Brightness: 88.06181335449219, Sharpness: 92.22801208496094 },
    Confidence: 99.99982452392578
  }
]

```

Figure 48: Response data of analysis

The response data is sent to the client-side as a FaceDetails object. This makes it very easy to access the contents of the response data on the client-side for displaying to the user.

5.3.2.4. *Deleting object from Amazon S3*

```
/* The following example deletes an object from an S3 bucket. */

var params = {
  Bucket: "examplebucket",
  Key: "objectkey.jpg"
};
s3.deleteObject(params, function(err, data) {
  if (err) console.log(err, err.stack); // an error occurred
  else     console.log(data);           // successful response
  /*
    data = {
    }
  */
});
```

Figure 49: DeleteObject example

The example above demonstrates how to use the `s3.deleteObject` API. It will be used to delete the files after they have been uploaded.

```
setTimeout(() => {
  var paramsDel = {
    Bucket: "fyp-img",
    Key: avatar.name
  };

  s3.deleteObject(paramsDel, function(err, data) {
    if (err) console.log(err, err.stack); // an error occurred
    else     console.log(data);           // successful response
  });
}, 2000);
```

Figure 50: S3.deleteObject

This last API `deleteObject` is also placed in a timeout function to avoid network delays causing errors. The file needs to be deleted after the analysis so that it is not stored, as this would violate GDPR rules.

5.4. *Front End*

The homepage has five sections :

1. Carousel
2. Intro to Facial Analysis
3. How Does Facial Recognition Work
4. Expectations & features
5. Image upload

Each of the sections has its key defining features. Together they make for a complete website that covers all aspects, introducing facial recognition, explaining how it works, sets expectations for the facial analysis and the real facial analysis section.

The entirety of the website uses HTML, CSS, Bootstrap, and certain sections also use Javascript. The use of Bootstrap across the website elevates the design.

5.4.1. *Carousel*

The carousel is used to grab attention and display a preview of the website's contents. JavaScript controls the movement of the carousel.

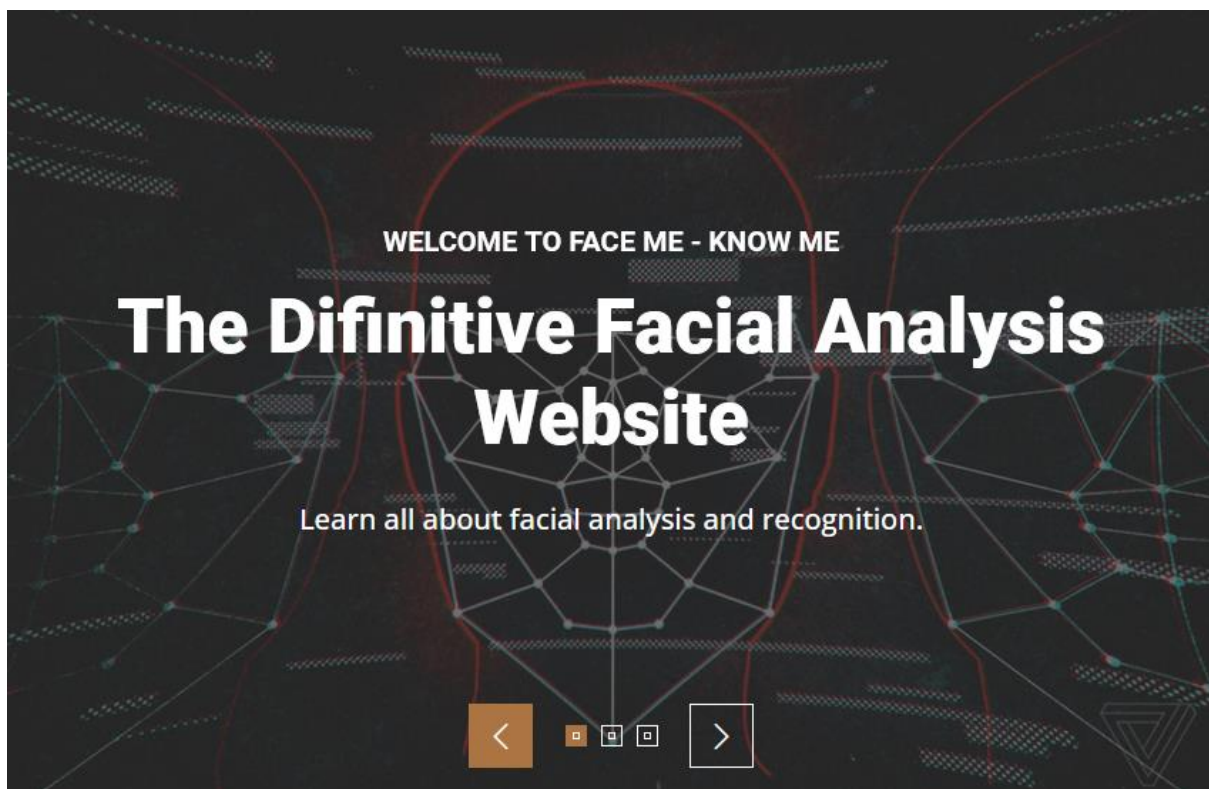


Figure 51:Caroussel

```

<div class="owl-carousel-item position-relative">
  
  <div class="position-absolute top-0 start-0 w-100 h-100 d-flex align-items-center" style="background: rgba(53, 53, 53, .7);">
    <div class="container">
      <div class="row justify-content-center">
        <div class="col-12 col-lg-8 text-center">
          <h5 class="text-white text-uppercase mb-3 animated slideInDown">Welcome To Face Me - Know Me</h5>
          <h1 class="display-3 text-white animated slideInDown mb-4">The Definitive Facial Analysis Website </h1>
          <p class="fs-5 fw-medium text-white mb-4 pb-2">Learn all about facial analysis and recognition.</p>
        </div>
      </div>
    </div>
  </div>
</div>

```

Figure 52: Carousel HTML

```

// Header carousel
$(".header-carousel").owlCarousel({
  autoplay: true,
  smartSpeed: 1500,
  items: 1,
  dots: true,
  loop: true,
  nav : true,
  navText : [
    '<i class="bi bi-chevron-left"></i>',
    '<i class="bi bi-chevron-right"></i>'
  ]
});

```

Figure 53: Carousel Javascript

5.4.2. Intro to Facial Analysis

The Intro to Facial Analysis introduces the topic of facial recognition by giving a backstory and the current situation surrounding facial recognition.

About Facial Analysis

Facial-recognition technology is increasingly common throughout society. We can unlock our phones with our faces, smart doorbells let us know who is outside our home, and sentiment analysis allows potential employers to screen interviewees for desirable traits. In the public sector, facial recognition is now in widespread use—in schools, public housing, public transportation, and other areas.

Despite the widespread use of facial recognition and the concerns it presents for privacy and civil liberties, this technology is only subject to a patchwork of laws and regulations. Certain jurisdictions have imposed bans on its use while others have implemented more targeted interventions. In some cases, laws and regulations written to address other technologies may apply to facial recognition as well.



109

Countries using FRT



626

Million Cameras in China alone

Figure 54: Intro to Facial Analysis

```
// Facts counter
$('[data-toggle="counter-up"]').counterUp({
  delay: 10,
  time: 2000
});
```

Figure 55: Intro to Facial Analysis counter

5.4.2. How Does Facial Recognition Work

In this section, the user gets educated on facial recognition's process to gather data and achieve its purpose. This is a four-step process:

1. Face Detection
2. Face Analysis
3. Converting to Data
4. Finding a Match

It also imparts exciting facts on the use of facial recognition today. The conclusion is that there are too many cameras constantly gathering data. Therefore how do you find out what the current level of technology can analyze from you?

How Does Facial Recognition Work

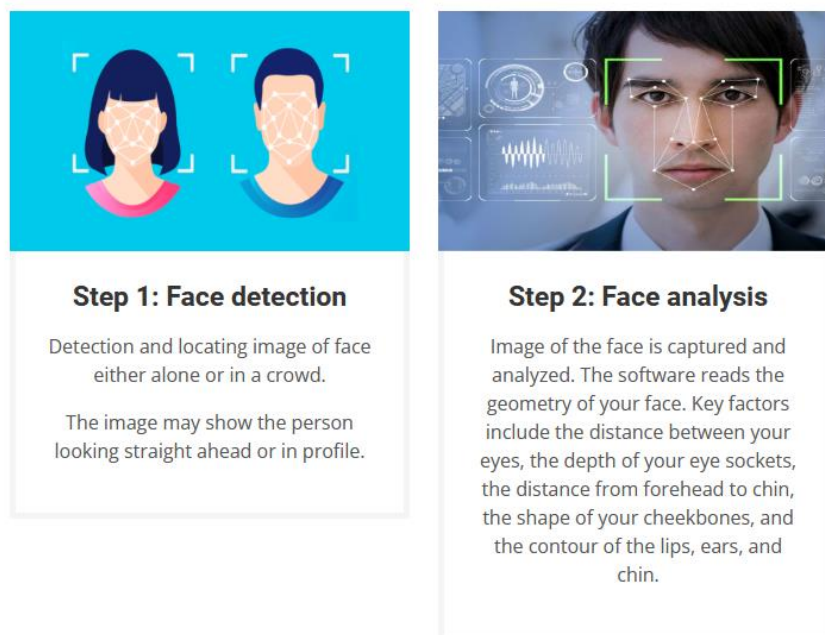


Figure 56: How does it work?

5.4.3. Upload Image

The interface has a title 'Upload Image' followed by a horizontal line. Below it is the text 'Upload Your Image Below for Analysis'. There is a file input area with a 'Browse...' button, the filename 'reda.jpg', and an 'Upload' button. A progress bar at the bottom is filled and labeled '100%'.

Upload Image

Upload Your Image Below for Analysis

Browse... reda.jpg reda.jpg Upload

100%

Figure 57: Image Upload

__ Facial Analysis __

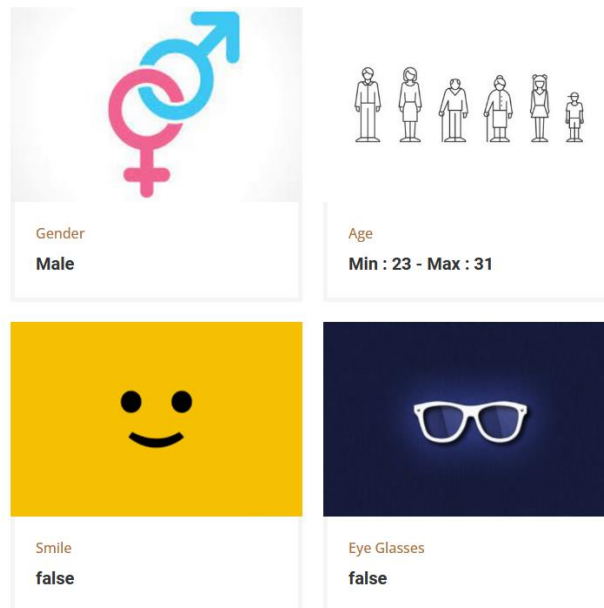


Figure 58: Image Upload Result

This is the core function of the website. The user can upload an image that will be analyzed, and the results returned. The input contents are saved in a constant called `selectedFile` as a file object. A new `formData` var created with the `FormData()`. The `selectedFile` is appended to `formData` under the 'file' label.

The next thing is posting, which will be done with `formData`. The post result is stored in a variable called `res`.

```
//Get Files From Website and package it into FormData format
const selectedFile = document.getElementById("input-file").files[0];
var formData = new FormData();
formData.append("file", selectedFile);
console.log(selectedFile);
```

Figure 59: formData

```
try {
  const res = await axios.post("http://localhost:3000/upload", formData, {
    headers: {
      "Content-Type": "multipart/form-data",
    },
    onUploadProgress,
  });

  console.log(res.data.faceData[0]);
}
```

Figure 60: Axios post

Locations to print the result of the post. The messageElement stores the result of any errors that may occur and prints them out below the file input, and HtmlizeResponse is used for the error handling printout. The rest of the elements identified are where the results will be displayed.

```
//Location to print the results
let messageElement = document.getElementById("message");
let GenderElement = document.getElementById("face_gender");
let ageElement = document.getElementById("face_age");
let smileElement = document.getElementById("face_smile");
let glassesElement = document.getElementById("face_eye_glasses");
let sunGlassesElement = document.getElementById("face_sun_glasses");
let emotionElement = document.getElementById("face_emotion");
```

Figure 61: Location to display results.

```
    } catch (err) {
        messageElement.innerHTML = htmlizeResponse(err);
    }
}

function htmlizeResponse(res) {
    return (
        `

<pre>` +
        JSON.stringify(res, null, 2) +
        "</pre></div>"
    );
}


```

Figure 62: Error handling client-side

```
GenderElement.innerHTML = res.data.faceData[0].Gender.Value;
ageElement.innerHTML = "Min : " + res.data.faceData[0].AgeRange.Low + " - Max : " + res.data.faceData[0].AgeRange.High;
smileElement.innerHTML = res.data.faceData[0].Smile.Value;
glassesElement.innerHTML = res.data.faceData[0].Eyeglasses.Value;
sunGlassesElement.innerHTML = res.data.faceData[0].Sunglasses.Value;
emotionElement.innerHTML = res.data.faceData[0].Emotions[0].Type;
```

Figure 63:Placing result

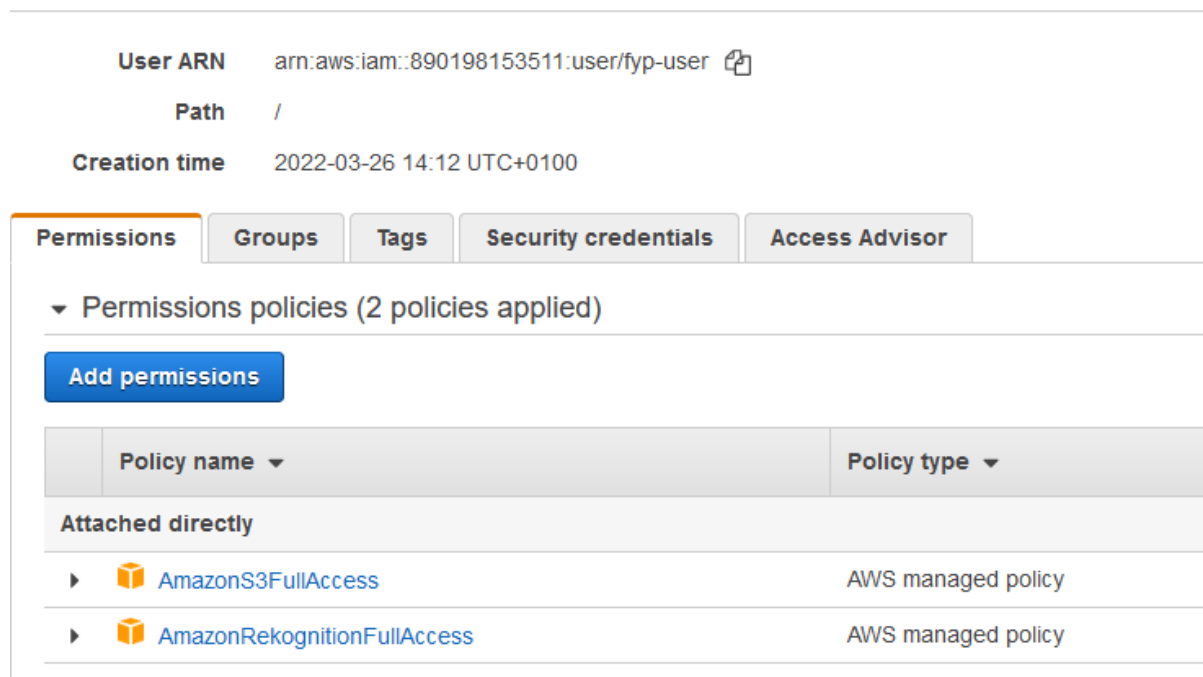
5.5. AWS

Amazon Web Services (AWS) is a secure cloud services platform offering compute power, database storage, content delivery and other functionality to help businesses scale and grow. For this system, we will only require the use of Amazon IAM for identity and access management, Amazon S3 for object storage, and Amazon Rekognition for image analysis.

5.5.1. AWS IAM

To begin, I created a new user with two roles, the first of which allowed access to upload, read, delete objects in an S3 bucket, and the second allowed full access to Amazon Rekognition. These roles are necessary for accessing the services from the AWS CLI and SDK.

Summary



The screenshot displays the AWS IAM console for a user named 'fyp-user'. The 'User ARN' is 'arn:aws:iam::890198153511:user/fyp-user', the 'Path' is '/', and the 'Creation time' is '2022-03-26 14:12 UTC+0100'. The 'Permissions' tab is selected, showing 'Permissions policies (2 policies applied)'. A table lists two policies: 'AmazonS3FullAccess' and 'AmazonRekognitionFullAccess', both are AWS managed policies.

Policy name	Policy type
AmazonS3FullAccess	AWS managed policy
AmazonRekognitionFullAccess	AWS managed policy

Figure 64: AWS IAM User

5.5.2. Amazon Simple Storage Service (S3)

Amazon S3 is an object storage service. It will be used to upload files to access the files so that Amazon S3 can access them. To do this, we must create a bucket with a globally unique. Uploading, accessing and deleting objects in S3 can be done through multiple ways such as AWS console, CLI, and SDK.

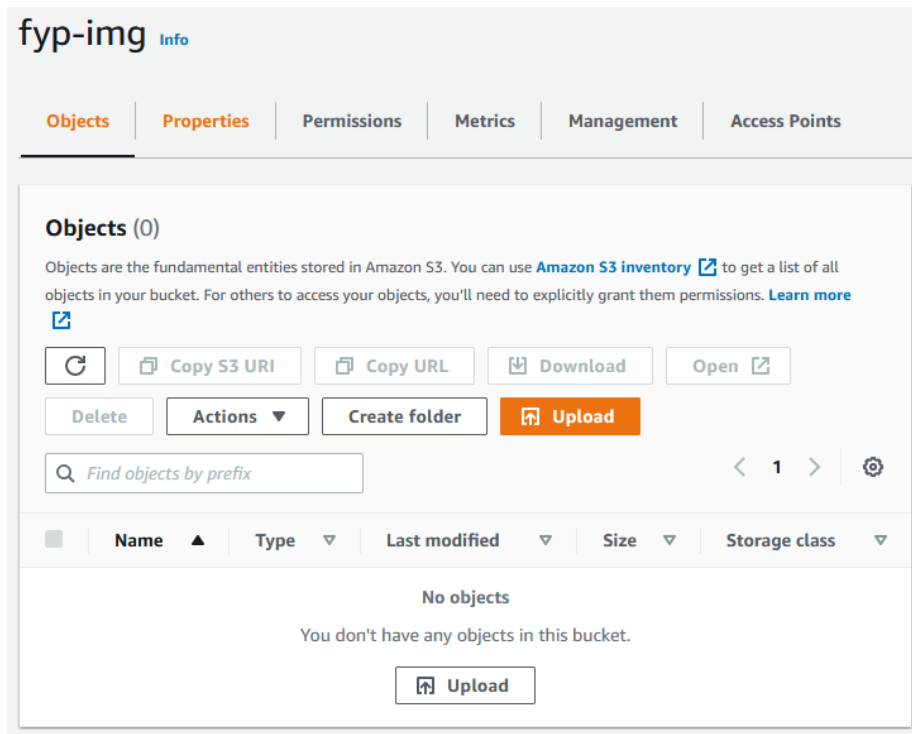


Figure 65: Fyp-img bucket

5.5.3. Amazon Rekognition

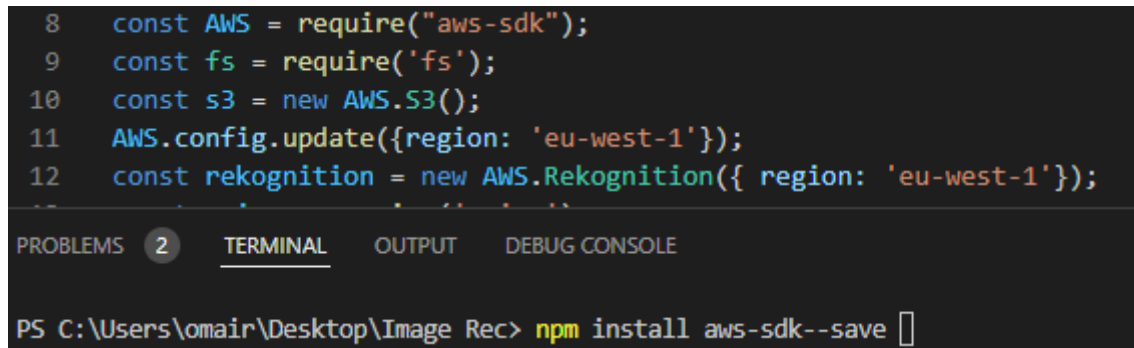
Amazon Rekognition is an image and video recognition service. Rekognition will be accessing the image file from Amazon S3. This is the most optimal method of accessing the files.

```
{
  "Image": {
    "S3Object": {
      "Bucket": "rekognition-
console-sample-images-prod-dub",
      "Name": "drive.jpg"
    }
  },
  "Attributes": [
    "ALL"
  ]
}
```

Figure 66: Rekognition request

5.5.4 AWS SDK

The SDK is used to trigger APIs to interact with the AWS console. To achieve this, though, many steps must be taken. First, the node module must be installed using `npm install aws-sdk --save`. This will add the module to the dependencies in `package.json` and download its resources. Secondly, the module must be required. Thirdly you must define the service used and define them using the required module.



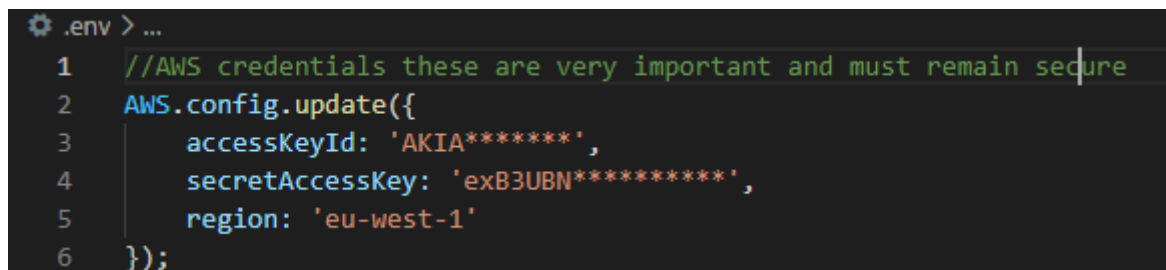
```
8  const AWS = require("aws-sdk");
9  const fs = require('fs');
10 const s3 = new AWS.S3();
11 AWS.config.update({region: 'eu-west-1'});
12 const rekognition = new AWS.Rekognition({ region: 'eu-west-1'});
```

PROBLEMS 2 TERMINAL OUTPUT DEBUG CONSOLE

PS C:\Users\omair\Desktop\Image Rec> npm install aws-sdk--save

Figure 67: AWS-SDK

I now added my AWS credentials to the environment to access the AWS resources with the correct permissions. Now I added my credentials to the `.env` file



```
.env > ...
1  //AWS credentials these are very important and must remain secure
2  AWS.config.update({
3    accessKeyId: 'AKIA*****',
4    secretAccessKey: 'exB3UBN*****',
5    region: 'eu-west-1'
6  });
```

Figure 68: AWS credentials

6. Conclusions and Future Work

6.1. *Introduction*

This chapter concludes the project and focuses on reviewing the issues, challenges and where the project can go in the future. The goal is to reflect on the experiences to be able to highlight what was achieved and learn from the experience on the journey in doing this project.

It is an extensive and complex project that continually gathers data throughout the design process. The project's biggest challenge during development was being able to keep up with the design. Another aspect of being aware is that the possibilities of where the project can be taken are many with more external feedback.

The main point of learning gained from the project planning and design is the difficulty that sometimes appears when attempting to merge all the design choices.

6.2. *Issues and Risks*

Many risks are inherent when developing any application. There are security concerns, performance concerns, usability concerns, and many more problems constantly being evaluated from many perspectives. With any project related to personal privacy, many people are very hesitant, as they should.

6.3. *Plans and Future Work*

As the design work continued to come to completion, many small changes took place across the project, bringing the project closer to perfection. As feedback during the UI wireframes optimization and flowchart kept being formed, I made a list of functions that are already being implemented into the project but also others that would be able to elevate the system to the next level. The core functionality that has been outlined above in chapter three will be the main aim of the design. If there is any additional or left-over time, then the development will continue to reach the other functions.

However, one thing to clarify is that the project goals and requirements were not fully decided and settled on until late December as other features were trying to be incorporated into the design.

Below, Figure 11 contains all the features that could be implemented in the project. Some of these features are more difficult to incorporate than others, such as celebrity matches. It would add much value to the system if this is possible.

Number	Name
1	Online Page
2	Attributes
3	Checking
4	Image Upload
5	Celebrity Match
6	Box detection
7	Aesthetic Homepage
8	Data Deletion
9	Info Page
10	Contact Page

Figure 69: Attribute Table

The valid metric for the project's success will be the monitoring if there is significant traffic to the web application. If there is substantial interest in it, that will prove that there is interest in this field that would be worth pursuing.

6.4. GANTT Chart

Attached below is a GANTT Chart for the Final Year Project. Below is an in-depth breakdown of the time spent with details.

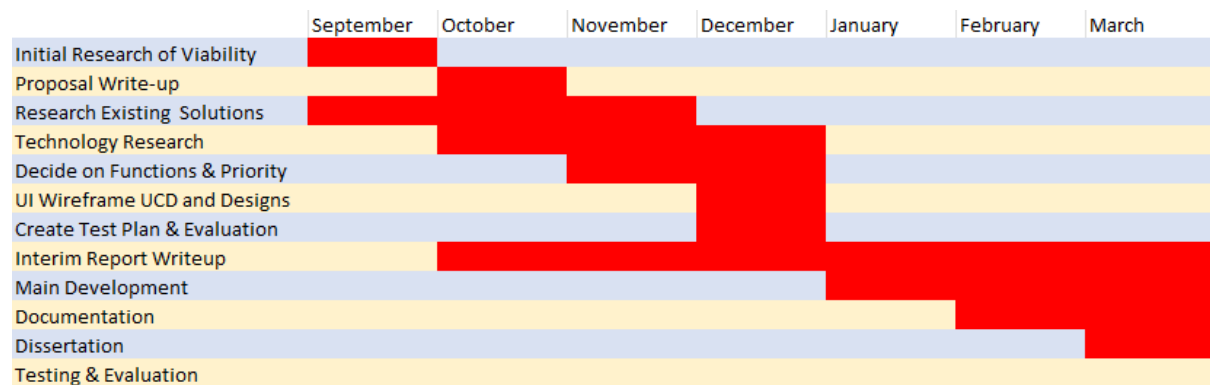


Figure 70: Gantt Chart

Initially, in early September, the project plan was brainstorming, researching, and exploring the possibilities of where the project could lead to. After two weeks of research, I discovered that I had not heard of a web application that was only for users to be able to discern what data a camera can learn about a person from their face. After this lead, the project continued by the search for existing solutions. While many services offered facial recognition software, there were remarkably few that were designed for only an end-user. Finally, after some research, the project idea/direction was decided on.

October: the proposal write-up was due, so the project focus turned to fleshing out the underlying technologies that are to be used in the development of the desired system, although all functionality was not yet decided on. This was important as it set the foundation and furthered the direction of the projection. Interim Report research also began as it helped to further the vision and goals.

November: was significant as the core functionality was finally specified after thorough research of not only technology but also what was available to the public.

December: was a crucial time as most of the design work was completed completing the use case diagram identified the building blocks and main functionality of the system. By identifying the specified functionality, the software development methodology was also decided; agile method scrum. The project was divided into two epics with many stories as the progression would continue with each sprint.

January: will be the month when the Interim Report is completed and finalized. NodeJS programming will also begin; front end development design and functionality will start. I was learning how to further merge the front end and the back-end systems.

February: continued developing the front end and began back-end configurations. Dissertation preparation work needs to be done in

March: back-end development was apace, and most of it was fleshed out and completed. The dissertation was well on the way.

April: Finishing touches were made to the project and dissertation.

6.5. Conclusion

The purpose of this project was to create a web application that can educate and inform people about facial recognition technology its abundance around the world. Not only that, but it also offers the opportunity to witness the information that is being caught by other cameras you are telling others. In regards to this, I believe the project was a success.

Almost all of the expected features were implemented, though various issues and challenges cropped up. The biggest challenge for me was using NodeJS and ExpressJS as it was my first time with these technologies causing me to face setbacks in the schedule.

Thank you very much for spending your time on this report.

Bibliography

- [1] What is Facial Recognition – Definition and Explanation [ONLINE] Available at: <https://www.kaspersky.com/resource-center/definitions/what-is-facial-recognition> [Accessed 13 January 2022]
- [2] Betaface is a face recognition software vendor. [ONLINE] Available at: <https://www.betaface.com/wpa/> [Accessed 13 January 2022]
- [3] Facelytics is a face recognition solution. [ONLINE] Available at: <https://www.facelytics.io/> [Accessed 13 January 2022]
- [4] React: Making faster, smoother UIs for data-driven Web apps [ONLINE] Available at: <https://www.infoworld.com/article/2608181/react--making-faster--smoother-uis-for-data-driven-web-apps.html> [Accessed 13 January 2022]
- [5] Python Web Development Tutorials [ONLINE] Available at: <https://realpython.com/tutorials/web-dev/> [Accessed 13 January 2022]
- [6] What is Amazon EC2 [ONLINE] Available at: <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html> [Accessed 13 January 2022]
- [7] What is Amazon S3? [ONLINE] Available at: <https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html> [Accessed 13 January 2022]
- [8] What Is Amazon Rekognition? [ONLINE] Available at: <https://docs.aws.amazon.com/rekognition/latest/dg/what-is.html> [Accessed 13 January 2022]
- [9] About Node.js by the OpenJS FOundation [ONLINE] Available at: <https://nodejs.org/en/about/> [Accessed 13 January 2022]
- [10] What is Apache? In-Depth Overview of Apache Web Server by Jovan Hernandez [ONLINE] Available at: <https://www.sumologic.com/blog/apache-web-server-introduction/> [Accessed 13 January 2022]
- [11] Google Cloud APIs by the Google team [ONLINE] Available at: <https://cloud.google.com/apis/docs/overview> [Accessed 13 January 2022]
- [12] What is Linux by the Linux foundation [ONLINE] Available at: <https://www.linux.com/what-is-linux/> [Accessed 13 January 2022]
- [13] What is Windows by Computer Hope [ONLINE] Available at: <https://www.computerhope.com/jargon/w/windows.htm> [Accessed 13 January 2022]
- [14] Web Design 101: How HTML, CSS, and JavaScript Work [ONLINE] Available at: <https://blog.hubspot.com/marketing/web-design-html-css-javascript> [Accessed 13 January 2022]

- [15] Article: Why did we build Visual Studio Code? By Visual Studio Code Team [ONLINE] Available at: <https://code.visualstudio.com/docs/editor/whyvscode> [Accessed 13 January 2022]
- [16] Article: What is NPM? by OpenJS Foundation [ONLINE] Available at: <https://nodejs.org/en/knowledge/getting-started/npm/what-is-npm/> [Accessed 13 January 2022]
- [17] Betaface is a face recognition software vendor. BetaFace official page [ONLINE] Available at: <https://www.betaface.com/wpa/> [Accessed 13 January 2022]
- [18] The Waterfall Model in Large-Scale Development by Kai Petersen, Claes Wohlin, Dejan Baca [ONLINE] Available at: https://www.researchgate.net/publication/30498645_The_Waterfall_Model_in_Large-Scale_Development [Accessed 13 January 2022]
- [19] Waterfall Methodology Image [ONLINE] Available at: <https://www.workfront.com/project-management/methodologies/waterfall> [Accessed 13 January 2022]
- [20] The Waterfall Model in Large-Scale Development by Kai Petersen, Claes Wohlin, Dejan Baca [ONLINE] Available at: https://www.researchgate.net/publication/30498645_The_Waterfall_Model_in_Large-Scale_Development [Accessed 13 January 2022]
- [21] Scrum Guide -Guide to Scrum Sprints by Wrike [ONLINE] Available at: <https://www.wrike.com/scrum-guide/scrum-sprints/> [Accessed 13 January 2022]
- [22] How to Build Your First Scrum Team by Marie ProkopetsMarie Prokopets, Co-founder of Nira [ONLINE] Available at: <https://nira.com/scrum-team/> [Accessed 13 January 2022]
- [23] Epics Features and Stories [ONLINE] Available at: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwi-ztyjhdv1AhXUSsAKHY19BVsQFnoECAUQAQ&url=https%3A%2F%2Fwww.gla.ac.uk%2Fmedia%2FMedia_730149_smxx.pdf&usg=AOvVaw25NuzZRS6nVuk-FB4gHhvs [Accessed 13 January 2022]
- [24] Basics Interactive Design: Interface Design: An introduction to visual communication in UI design [ONLINE] Available at: https://books.google.ie/books?hl=en&lr=&id=rV03DQAAQBAJ&oi=fnd&pg=PP1&dq=ui+wireframes+fundamentals+research+paper&ots=qksqoDJR0b&sig=weYGWSHGdoRDWRpZeTgJfvIOEAk&redir_esc=y#v=onepage&q=ui%20wireframes%20fundamentals%20research%20paper&f=false [Accessed 13 January 2022]