A Posture Correction Application For Individuals In Seated Positions

Rayan Meraghni
26138136

26138136@students.lincoln.ac.uk

School of Computer Science

College of Science

University of Lincoln

Submitted in partial fulfilment of the requirements for the
Degree of BSc(Hons) Computer Science

Supervisor: Dr. Helen Harman

May 2023

Acknowledgements

I would like to express my heartfelt gratitude to my parents, my supervisor, and my Cousin for their priceless support throughout this journey. Their constant encouragement, guidance, and belief in me have been instrumental in my success.

Firstly, I am incredibly grateful to my parents for their unconditional love and endless sacrifices. Their wisdom, encouragement, and belief in my abilities have been a constant source of inspiration. They have always been there to provide guidance and support, even during the most challenging times. I am truly fortunate to have them as my role models and mentors.

I would also like to extend my sincere appreciation to my supervisor, Dr. Helen Harman for her invaluable guidance and mentorship. Her expertise, patience, and dedication have played a pivotal role in shaping my skills and pushing me to achieve my best. Her insightful feedback and encouragement have truly been invaluable throughout this journey.

Lastly, I would like to acknowledge Oussama Meraghni, my cousin whom I recently got to know. His assistance and support during the initial stages of my project were immensely valuable. His knowledge, expertise, and willingness to lend a helping hand made a significant impact on the start of my project. I am grateful for his contribution and the meaningful discussions we had that helped shape my ideas.

Abstract.

One of the fields to touch upon human postural problems is computer science, which is still making a contribution by investigating fresh ideas based on cutting-edge ICTs (information and communications technologies). The creation of applications with the right features and the incorporation of new technology might aid in the rollout of effective preventative initiatives to lessen the negative effects of posture issues on people's health.

The suggested project is an Artificial Intelligence (AI) approach that might assist a preventative program to lessen the consequences brought on by the adoption of unfavourable postures. In order to provide an alert if a poor posture is detected, the application built will proceed to detect changes in users posture by employing a Deep Learning (DL) model to distinguish correct postures from incorrect ones. Results of monitoring video taken in real-time would be then accessible on the app interface to track posture habits, and ultimately improve them.

Table of Contents

Introduction 1
1.1 Problem Identification & Motivations 1
Literature Review 3
2.1 Introduction 3
2.2 Postures 3
2.3 Posture Capture Technologies 5
2.4 Models 6
2.5 Notification System 7
2.6 Aims & Objectives 8
2.6.1 Aim 8
2.6.2 Objectives 8
Requirements Analysis 9
3.1 Stakeholders Requirements 9
3.1.1 Business Requirements 9
3.1.2 End-Users Requirements 9
3.2 Functional Requirements 10
3.3 Non-Functional Requirements 10
Design & Methodology 11
4.1 Project Management 11
4.2 Software Development Methodology 12
4.2.1 SCRUM 12
4.2.2 Planning and Sprints 12
4.2.3 Stand-up meetings 14
4.2.4 Sprint Retrospective 15
4.2.5 Backlog refinement 15
4.3 Toolsets and machine environments 16

4.3.1 Hardware..16
4.3.1.1 Jetson Nano..16
4.3.1.2 Raspberry Pi Camera Module v2..17
4.3.2 Programming Languages..17
4.3.2.1 Python..17
4.3.2.2 JavaScript..18
4.3.2.3 HTML and CSS..18
4.3.3 Frameworks and Libraries..18
4.3.3.1 TensorFlow..18
4.3.3.2 ONNX..19
4.3.3.3 OpenCV..19
4.3.3.4 TensorRT..20
4.3.3.5 Django..20
4.3.3.6 Bootstrap..20
4.4 Design..21
4.4.1 Monitoring Device..21
4.4.2 Web Application..25
4.4.2.1 User Flowchart..25
4.4.2.2 Database Entities & Attributes..26
4.4.2.3 Database Entity-Relationship Diagram..27
4.4.2.4 Real-Time Feedback..28
4.5 Testing..29
4.5.1 Unit Testing..29
4.5.2 Performance Testing..29
4.6 Risk Analysis..30
4.6.1 Model Performance..30
4.6.2 Technical Development..31
4.6.3 Jetson Nano Performance..31
Implementation..33
5.1 Overview..33

5.2 Back-end 34
5.2.1 Jetson Nano Setup 35
5.2.2 Model Optimisation 36
5.2.2.1 From TensorFlow-lite to ONNX 36
5.2.2.2 From ONNX to TensorRT 36
5.2.3 Real-time Poor Posture Detection 38
5.2.3.1 Jetson Nano Side 38
5.2.3.2 Django App Side 44
5.3 Front-End 49
5.3.1 User Interface Development 49
5.3.2 Real-time Alert System 51
5.3.3 App Pages 52
Results & Discussion 55
6.1 Results & Discussion 55
Conclusion 58
7.1 Success & Limitations 58
7.1 Future work 59
Appendix 62
8.1 Unit Tests 62

List of Figures

1. A picture of the project Gantt Chart. 11
2. A picture of the Backlog during sprint 2 on Trello board. 16
3. A picture of the body key joints detected by the moveNet. 22
4. A drawing of the monitoring system camera perspective options. 22
5. A drawing of skeletal data used for each camera perspective. . . . 23
6. A picture of the user flow diagram. 26
7. A picture of the database ERD. 28
8. A picture of The web app tree. 34
9. A picture of the monitoring device tree. 34
10. A picture of the Jetson Nano with the Raspberry Pi camera. 35
11. A picture of the moveNet model optimisation process. 38
12. A picture of the python code loading the model. 39
13. A picture of the python code running inference. 40
14. A picture of the python code opening the camera. 43
15. A picture of the python code creating the database. 45
16. A picture of the python code for the SSE. 49
17. A picture of the code to insert the login form. 50
18. A picture of the JavaScript code for alerts. 51
19. A picture of the app’s profile page on a PC. 53

20. A picture of the app’s monitoring page on a phone. 54

List of Tables

1. Sprint 1 & 2. 13
2. Sprint 3 & 4. 14
3. Body joints & thresholds. 24
4. Risk analysis. 32
5. Performance Test. 57

Chapter 1

Introduction

1 Problem Identification & Motivations

In today's digital age, people are increasingly reliant on technology for work, entertainment, and communication. This has led to an increase in sedentary lifestyles and a decrease in physical activity, which can have negative consequences on our health. One of the most common issues that arise from prolonged periods of sitting and using technology is poor posture, which can cause musculoskeletal disorders and other health problems.

Poor posture is characterised by an abnormal alignment of the body that puts undue stress on the muscles, bones, and joints. It can result from a variety of factors, including prolonged sitting, standing, or working in awkward positions, as well as underlying medical conditions. Poor posture can lead to a host of health problems, including neck and back pain, headaches, and decreased lung capacity. Additionally, poor posture can affect one's self-confidence and body image, leading to poor mental health.

Posture recognition technology has emerged as a promising preventive solution to this growing problem. It involves the use of sensors, computer vision, and machine learning algorithms to detect and classify the user's posture in real-time. Posture recognition technology can provide users with immediate feedback on their posture, allowing them to correct it before it causes any harm. This technology can also help users develop good posture habits over time, leading to long-term improvements in their health and well-being.

In this thesis, we present the design and development of a posture recognition app that uses computer vision and machine learning techniques to detect and classify the user's posture in real-time. The app provides users with a visual representation of their body parts and alerts them when they deviate from the ideal posture.

The development process was governed by an Agile methodology (SCRUM), ensuring a systematic and organised workflow. This promoted vigorous testing along with careful planning, as many complex tasks were addressed including Deep Learning model optimisations, web development, database management, and real-time notification system.

Chapter 2

Literature Review

2.1 Introduction

The examination of human posture has received a lot of attention in the literature. The use-cases encompass a variety of application areas, including telemedical applications (Al-Attas et al, 2012), baby sleeping position detection (Khan, 2021), antisocial actions for irregular activity monitoring (Adam et al, 2021).

Most of these applications have in common the same building blocks: Posture capture technologies, machine learning (ML) models for Posture Recognition, and a notification system to alert users. The nature of the project in development falls in this area of studies. To have a clear grasp of what have been achieved already in this field of studies, the literature review will explore postures, posture's capture technologies, models, and notifications generation.

2.2 Postures

Finding from a large population-based study in China revealed that from a total of 595 057 screened students the overall prevalence of incorrect postures in children and adolescents was 65 % (Yang et al., 2020). Other studies investigate the relationship between Low Back Pain (LBP) and sitting behaviour among sedentary office workers. A strong correlation between sitting behaviour and chronic low back pain was discovered, with 75% of participants reporting some degree of either chronic or acute back discomfort (Bontrup et al., 2019). A recognised exacerbating factor for those who have lower back pain (LBP) is prolonged static sitting, which raises the risk of musculoskeletal diseases in the neck, shoulders, arms, and legs (Zemp, et al, 2013). Waongenngarm et al (2015) in their study reported that sitting for one hour causes discomfort in the neck, upper back, low back, and hips and thighs.

These findings collectively support the notion that individuals should pay attention to their sitting habits and strive to adopt and maintain good postures as incorrect posture adoption is at the root of many health problems.

Waongenngarm et al (2015) focused on the identification of the most common adopted postures. This study indicated 3 prominent postures, upright, slump, and leaning forward. After one hour of sitting solely in the low back, postures impacted low back pain. the forward-leaning posture induced the most low back discomfort, followed by the upright and slumped postures, respectively. In comparison to the upright and slumped position, the forward leaning posture was linked to increased iliocostalis lumborum pars thoracis (ICL) and superficial lumbar multifidus (MF) muscle activation. Compared to slumped sitting, the upright position was linked to increased internal oblique (IO), transversus abdominis (TrA), and internal abdominis (ICL) muscular activation.

The slumped posture is the most pleasant for the low back, nonetheless, decreases activation muscle activity, which might negatively contribute to ligament and disc overloading and the onset of LBP. On the other hand, the upright sitting posture appears to be good for the low back. Thus, concluding from the former studies, postures are to be classified to correct and incorrect in order to mitigate the repercussions of prolonged sitting. Upright posture is more suited than others to lessen the negative effects making it the correct posture in our project. Other postures will be deemed incorrect.

Overall, the posture literature review reveals that the only way to lessen the effect of bad posture adoption effect on people health is through an implementation of suitable prevention program as suggested by Bontrup et al (2019).

2.3 Posture Capture Technologies

The dominant technologies that have been used to enable capture of human postures are sensors and computer vision. Sensors fall into two categories; pressure sensors and inclination Sensors. According to Ran et al (2021), pressure sensors are usually implemented by installing them on chair surface and back, whereas inclination detector sensors are wearable and aim to detect the body inclinations postures (Kale et al, 2018). Sensor technologies, as compared to computer vision, offer the benefit of obtaining the data immediately for processing, but have the limitation of being worn or fixed to chairs.

On the other hand, the significant progress of image processing enabled by the enhanced treatment capacity of CPUs and GPUs empowered the application of computer vision. Unlike sensors, it is non-invasive towards the subject. Adam et al (2022) in their study used an HD camera and a Jetson Nano to capture normal and abnormal behaviour in an effort to anticipate human intentions in real-time. The choice of using a Jetson Nano for real-time behaviour analysis highlights the potential of this embedded system for such applications. The Jetson Nano is designed specifically for AI and deep learning tasks, offering a balance between performance and energy efficiency. It provides a dedicated hardware platform that can accelerate the processing of complex neural network models, making it suitable for real-time analysis tasks like posture detection. Utilising the Jetson Nano alongside an HD camera, the study demonstrates the feasibility of capturing and processing high-quality video data for behaviour analysis. This combination allows for accurate and detailed monitoring of human movements and can be particularly useful in posture detection applications.

Paliyawan et al (2014) utilised the Kinect camera to gather a live stream of skeleton data in order to identify office workers who had been seated for an extended period of time. By leveraging the Kinect camera's capabilities, the researchers were able to obtain accurate and detailed skeletal data. This data, combined with appropriate algorithms, enables the detection of specific postures associated with prolonged sitting. The use of visual cues and skeleton tracking from the camera could offer potential for a real-time posture monitoring and intervention strategies to mitigate the negative effects of prolonged sitting.

Piñero-Fuentes et al (2021) contends performance evaluation of a variety of embedded systems to ensure that the neural network method chosen to estimate the users’ joints had hardware that is capable of supporting it. All of the embedded systems that were reviewed were NVIDIA Jetson family. The study shows a low performance on the Jetson Nano, however, a critical observation can be made regarding the lack of consideration given to optimisation techniques prior to deploying the model on the hardware.

While it is important to acknowledge the initial low performance observed on the Jetson Nano, it is equally important to explore potential avenues for optimisation. Optimisation techniques play a crucial role in maximizing the performance of neural network models on embedded systems with limited computational resources like the Jetson Nano. These techniques include model quantisation, model compression, and hardware-specific optimisations, such as TensorRT integration. By optimising the model, it is possible to improve inference speed, reduce memory usage, and enhance overall performance. Furthermore, the Jetson Nano offers the advantage of being small and affordable. the small form factor ensures portability, allowing users to easily carry and use the device in various environments. This portability enhances convenience and flexibility, as users can monitor their posture wherever they go.

2.4 Models

The second building element for posture detection systems is ML models. In the many studies that are available in the literature, various kinds of algorithms have been implemented. Using a Convolutional Neural Network (CNN), Piñero-Fuentes et al (2021) developed a skeletal detection system that estimates the user's posture in real time and provides postural advice for the neck, shoulders, and arms. It was claimed that ergonomics and biomechanics research provided a full description of the suggested angles for those parameters used to measure the proper postures (Chaffin et al, 2006; Woodson et al, 2019).

Other research, like that conducted by Kale et al (2018) examined Support Vector Machine (SVM), K-Nearest Neighbour (KNN), and Artificial Neural Network (ANN) before selecting the latter as the algorithm of choice in an attempt to classify various human postures. Due to the model being designed to operate on an embedded system, the criteria serving for the model selection were the best accuracy with little runtime overhead and memory.

It is arguable that a task of posture classification could prove to be complicated, as it entails the use of a model able of extracting important features in images. Ideally, the perfect choice to tackle such a task would be the CNN, as it is commonly used in similar contexts. As revealed by the results, it is a robust algorithm very accurate at image recognition, that should be conducive to the development of an incorrect posture prevention app.

2.5 Notification System

In the event of an improper posture, an action is due to be undertaken by alerting the user through any form of notification. Khan (2021) in his study developed a baby monitor device connected to the same Wi-Fi as his smartphone app. In essence, the application seeks to give the users more control over their baby. The voice alarm is generated using text-to-speech technology and it says out the dangerous scenario if the youngster adopts a dangerous position. Baby with veiled face is a prime example of the situation that may be the trigger. Every time one or more of these undesirable events take place, the device implements an HTTP server and uses cloud messaging to send notifications to the caregiver's smartphone.

The time span of the specific postures referred to, wasn’t discussed enough in most studies. When preventing an improper posture, the duration of the latter could be accounted for to generate a notification, as a false alert may be triggered if an incorrect posture is captured as the subject is perhaps just moving.

2.6 Aims & Objectives

In the light of an exhaustive literature review delivered, it was clear what the aim and objectives of the project would be, they are as follows:

2.6.1 Aim

- To build an AI application that recognises unfavourable postures susceptible to result in health issues.

2.6.2 Objectives

- To evaluate and compare deep learning models to choose the best-suited one for posture detection.
- To integrate the model chosen to the Jetson Nano board and ensuring that the latter leverages the use of the former.
- To gather datasets and define thresholds for different posture monitoring perspectives.
- To set up a database in order to store all posture-related information, and give feedback to the end user
- To develop a user-friendly application that allows end users to be alerted if one of the three poor posture is detected.
- To test and monitor the app effectiveness and performance in the production environment.

Chapter 3

Requirements Analysis

The sitting posture monitoring app development process abided by stakeholders, functional, and non-functional requirements, which have been extrapolated based on the literature review conducted beforehand. These requirements outlined the features and capabilities of the app, and they consist of the following:

3.1 Stakeholders Requirements

3.1.1 Business Requirements

- An app that prevents office workers from adopting poor postures and promotes good sitting habits.

3.1.2 End-Users Requirements

- Users should be able to create an account on the app.
- Users should be able to login and logout the app.
- Users Should be monitored in front or by side of the camera when sitting.
- Users Should be alerted when an incorrect posture lasts for 10 seconds.
- Users should be able to check their sitting posture statistics.
- Users should be able to photos of their incorrect postures.
- Users Should be able to access a historical video feed containing video's information.
- Users should be able to search their videos' history by date.

3.2 Functional Requirements

The functional requirements describe the app's features and capabilities and they are as follows:

- A friendly interface to interact with the system.
- An authentication system user with sign up, login, and secured password.
- An appropriate ML algorithm to detect postures.
- A mechanism to differentiate between correct and incorrect posture.
- An alert system to notify users when incorrect postures lasts 10 seconds.
- A database to store user information along with historical postures.
- A system to generate useful statistics for the user.

3.3 Non-Functional Requirements

The non-functional requirements, describe the app's performance, and how the system should behave in terms of security, compatibility, scalability, and usability.

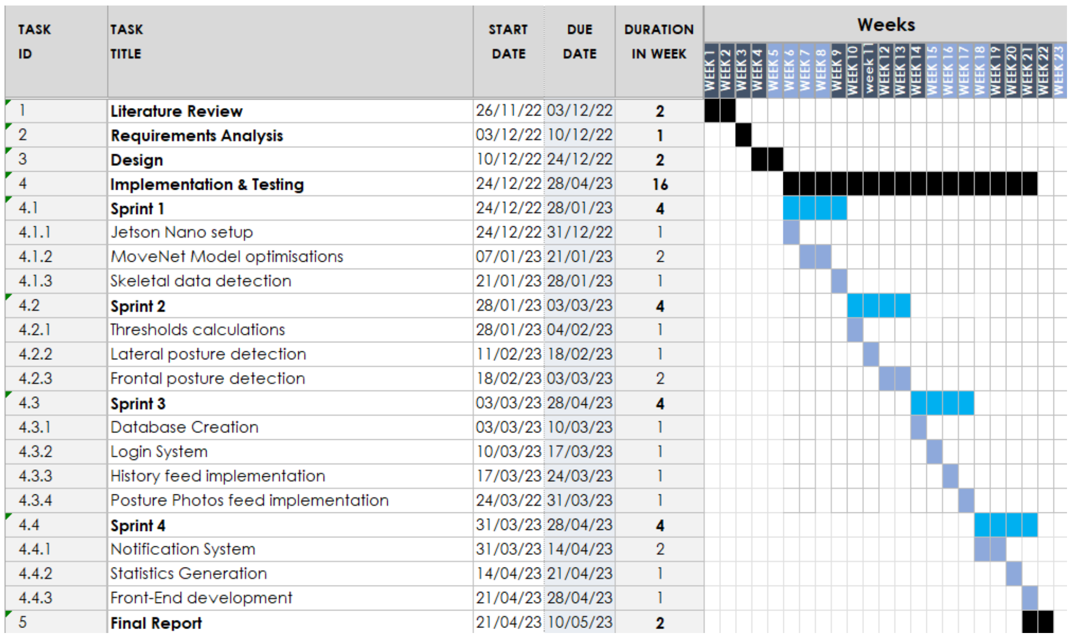
- Performance: The app monitoring video should have the minimum latency time to perform at its best and operate in real-time.
- Compatibility: The app should be compatible with PC and Mobile phone devices.
- Access security: The app should restrict the access to some of its features to authenticated users only. Also, users should not be able to access other users accounts, maintaining data privacy, and security.
- Usability: The app should have a simple and intuitive user interface to enhance the user experience.
- End user Feedback: Users should be able to leave comments or any remarques.

Chapter 4

Design & Methodology

4.1 Project Management

The project was broken down into specific tasks as illustrated in Figure 1. The use of this Gantt allowed for better coordination, and timeline management as tasks could be prioritised to ensure the smooth execution of the project.



TASK ID	TASK TITLE	START DATE	DUE DATE	DURATION IN WEEK	Weeks																						
					WEEK 1	WEEK 2	WEEK 3	WEEK 4	WEEK 5	WEEK 6	WEEK 7	WEEK 8	WEEK 9	WEEK 10	week 11	WEEK 12	WEEK 13	WEEK 14	WEEK 15	WEEK 16	WEEK 17	WEEK 18	WEEK 19	WEEK 20	WEEK 21	WEEK 22	WEEK 23
1	**Literature Review**	26/11/22	03/12/22	**2**	■	■																					
2	**Requirements Analysis**	03/12/22	10/12/22	**1**			■																				
3	**Design**	10/12/22	24/12/22	**2**				■	■																		
4	**Implementation & Testing**	24/12/22	28/04/23	**16**						■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■		
4.1	**Sprint 1**	24/12/22	28/01/23	**4**						🟦	🟦	🟦	🟦														
4.1.1	Jetson Nano setup	24/12/22	31/12/22	1						▨																	
4.1.2	MoveNet Model optimisations	07/01/23	21/01/23	2							▨	▨															
4.1.3	Skeletal data detection	21/01/23	28/01/23	1									▨														
4.2	**Sprint 2**	28/01/23	03/03/23	**4**										🟦	🟦	🟦	🟦										
4.2.1	Thresholds calculations	28/01/23	04/02/23	1										▨													
4.2.2	Lateral posture detection	11/02/23	18/02/23	1											▨												
4.2.3	Frontal posture detection	18/02/23	03/03/23	2												▨	▨										
4.3	**Sprint 3**	03/03/23	28/04/23	**4**														🟦	🟦	🟦	🟦						
4.3.1	Database Creation	03/03/23	10/03/23	1														▨									
4.3.2	Login System	10/03/23	17/03/23	1															▨								
4.3.3	History feed implementation	17/03/23	24/03/23	1																▨							
4.3.4	Posture Photos feed implementation	24/03/22	31/03/23	1																	▨						
4.4	**Sprint 4**	31/03/23	28/04/23	**4**																		🟦	🟦	🟦	🟦		
4.4.1	Notification System	31/03/23	14/04/23	2																		▨	▨				
4.4.2	Statistics Generation	14/04/23	21/04/23	1																				▨			
4.4.3	Front-End development	21/04/23	28/04/23	1																					▨		
5	**Final Report**	21/04/23	10/05/23	**2**																					■	■	

Figure 1: A picture of the project Gantt Chart

4.2 Software Development Methodology

4.2.1 SCRUM

The development of the project's artefact was underpinned by the implementation of a derivative of the SCRUM software development methodology. The motivation behind this choice was the ability SCRUM offers to integrate any new feature, during the development cycle of the app, which could enhance the user experience. This was beneficial in situations where requirements may change or evolve over time, which was the case in this project where changes came to light during the course of the app development. Additionally, SCRUM entails conitinious improvements through regular reflection and adaption. Regular testing is another aspect of this methodology which contributes to identifying issues at an early stage of the development, lessening the risks of major issues or failures later in the project artefact.

On the basis of the individual nature of the project, all workloads whether developing, or testing were individual. Another notable distinction between this derivative and a generic SCRUM methodology, is that stand-up meetings were limited to one a week. They were normally taking place on Wednesdays, where the supervisor played the role of the SCRUM master.

4.2.2 Planning and Sprints

The planning stage involved creating a product backlog, which is a prioritised list of features, functions, and requirements that need to be completed in the project. This product backlog consisted of requirements addressed previously in the requirements analysis section, that were translated into user stories. The user stories were written in a simple and concise language describing the desired app functionalities from a user perspective. User stories were broken down into four sprints as shown on table 1 and table 2. The feasibility of these features were then gauged to establish the period of time it would take to achieve each sprint.

Product Backlog 1/2		
Sprint 1 - (4 weeks)		**Sprint 2 - (4 weeks)**
As a user I should be monitored by an appropriate device.		As a user I should be monitored from a frontal perspective where my back should define the state of my posture.
As a user I should be captured in real-time with minimum delay.		As a user I should be monitored from a frontal perspective where my neck should define the state of my posture.
As a user I should be able to see my body joints on the video while monitored.		As a user I should be monitored from a lateral perspective where my back should define the state of my posture.
As a user I should know my sitting posture state.		As a user I should be monitored from a lateral perspective where my neck should define the state of my posture.
As a user I should be able to start and stop the video.		

Table 1: Sprint 1 & 2.

.

Product Backlog 2/2		
Sprint 3 - (4 weeks)		**Sprint 4 - (4 weeks)**
As a user I should be able to register, login, and logout the app.		As a user I should receive an alert with an audio if I adopt an incorrect posture for 10 seconds.
As a user I should have my password hashed for security.		As a user I should see statistics on my profile page to help me improve my posture over time.
As a user I should be able to add a profile picture.		As a user I should be able to interact with the app on different sized displays such as phones and PCs.
As a user I should be able to check photos of incorrect postures I have adopted throughout a video.		As a user I should find the app interface intuitive and easy to use.
As a user I should be able to access a history feed with details on my postures.		
As a user I should be able to post a feedback on the app.		

Table 2: Sprint 3 & 4.

4.2.3 Stand-up meetings

Normally, the development team holds a daily stand-up meeting to review progress, discuss any obstacles they may have encountered, and plan the day ahead. The stand-up meeting is timeboxed to 15 minutes to keep it focused and concise. Since the project was individual in nature, this aspect of the SCRUM methodology was altered to arrange meetings on a weekly basis instead. Meetings were held with the supervisor, in order to track progress, and get feedback.

4.2.4 Sprint Retrospective

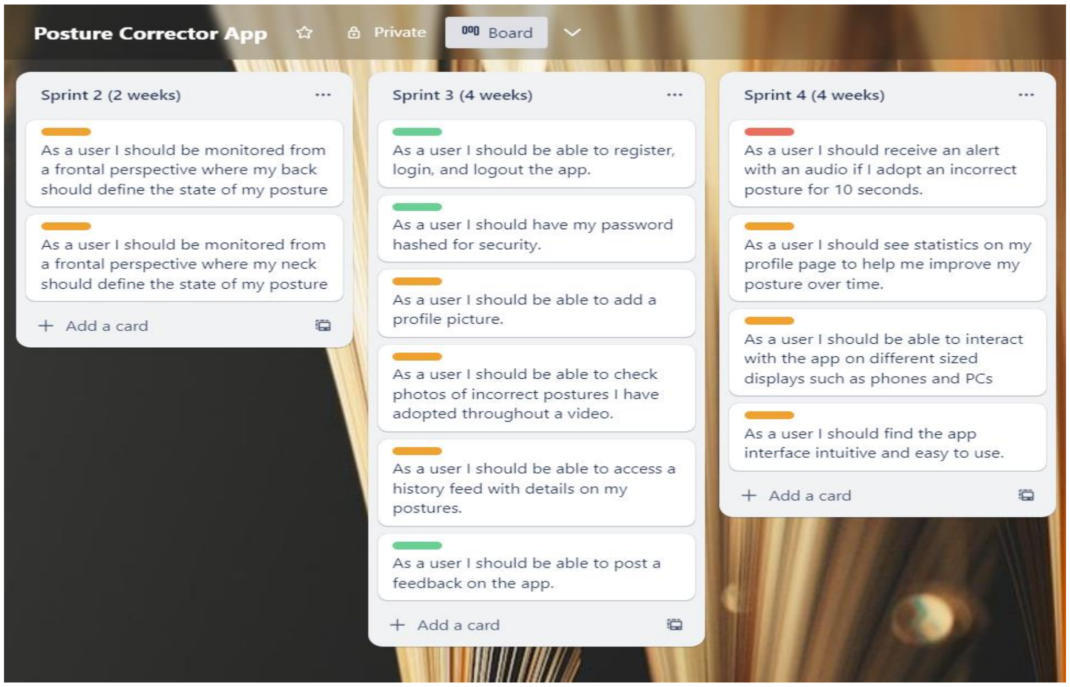
sprint retrospective was to reflect on the sprints and identify areas for improvement. The retrospective included a review of what went well, what didn't, and what changes could be made to improve the next sprint. Overall, the project went as intended, and the sprints' objectives were achieved within the timeframe established.

The only area of improvement identified was in the first sprint when one of the app's features to be delivered relied on training a CNN model for sitting posture recognition. The model performance wasn't ideal and the time was getting tense. Furthermore, the lack of dataset for the model training posed a substantial risk to the progress of the development which jeopardised the project as the app heavily depends on the posture recognition system. Hence, the alternate approach to mitigate the risk was to use a pre-trained model that would serve for the same purpose.

4.2.5 Backlog refinement

Throughout the project, the product backlog was regularly reviewed and refined to ensure that it remains up to date and relevant. Adding new items, removing items that are no longer needed, and updating priorities based on needs were all parts of the backlog refinement.

Figure 2 shows the backlog during the course of sprint 2. Sprint 1 along with other items from sprint 2 are no longer on the board, as they were completed at that point of time. The time left to conclude the sprint ongoing were also updated, and the coloured labels represent an estimation of the complexity of the tasks to be addressed where red, orange, and green refer to high, medium, and low respectively.


Posture Corrector App

Sprint 2 (2 weeks)

As a user I should be monitored from a frontal perspective where my back should define the state of my posture

As a user I should be monitored from a frontal perspective where my neck should define the state of my posture

Sprint 3 (4 weeks)

As a user I should be able to register, login, and logout the app.

As a user I should have my password hashed for security.

As a user I should be able to add a profile picture.

As a user I should be able to check photos of incorrect postures I have adopted throughout a video.

As a user I should be able to access a history feed with details on my postures.

As a user I should be able to post a feedback on the app.

Sprint 4 (4 weeks)

As a user I should receive an alert with an audio if I adopt an incorrect posture for 10 seconds.

As a user I should see statistics on my profile page to help me improve my posture over time.

As a user I should be able to interact with the app on different sized displays such as phones and PCs

As a user I should find the app interface intuitive and easy to use.

Figure 2: A picture of the Backlog during sprint 2 on Trello board

4.3 Toolsets and machine environments

4.3.1 Hardware

4.3.1.1 Jetson Nano

the Jetson Nano is designed for use in embedded systems and AI applications, providing developers with an affordable and accessible platform for developing and deploying deep learning models at the edge. At its core, is powered by a quad-core ARM Cortex-A57 CPU and a 128-core NVIDIA Maxwell GPU (Kurniawan and Kurniawan, 2012). It also features 4GB of LPDDR4 RAM, a Gigabit Ethernet port, four USB 3.0 ports, and a microSD card slot for storage.

One of the key advantages of the Jetson Nano is its low power consumption, making it ideal for use in battery-powered or remote applications. Despite its compact size and low power consumption, the Jetson Nano delivers impressive performance, with the ability to process up to 1 trillion operations per second (TOPS) for AI workloads.

4.3.1.2 Raspberry Pi Camera Module v2

The Raspberry Pi Camera Module v2 (also known as RPi Camera V2 or Raspberry Pi CS 2 Camera) is a small and inexpensive camera that is compatible with the Jetson Nano. The camera module is capable of capturing high-quality still images and high-definition video with a resolution of up to 8 megapixels and 1080p, respectively.

It features a 1/4-inch 8-megapixel sensor, a fixed focus lens, and supports various features such as automatic white balance, exposure control, and image stabilisation. The camera module can also capture images and videos with a wide range of frame rates, ranging from 15 to 90 frames per second, making it suitable for a sitting posture detection app intended to work in real-time.

4.3.2 Programming Languages

4.3.2.1 Python

Python is versatile as it can be used for a wide range of applications, from web development and machine learning to scientific computing and data analysis. it also provides a lot of built-in functionality, such as support for regular expressions and file handling, which can simplify many programming tasks.

Python has been used for both the web application back-end and computer vision portions of the app. Python's ease of use and flexibility made it an attractive choice for both of these tasks. Additionally, Python has several libraries for computer vision, machine learning and web development which could simplify the development of the app. The libraries used in order to build the app will be discussed in the frameworks and libraries section of this chapter.

4.3.2.2 JavaScript

One of the main benefits of using JavaScript in web development is its ability to enhance the user experience by allowing for dynamic content and real-time interactions without the need for page reloading. The use of JavaScript enables the creation of modal pop-ups and audio alerts to notify the user of any posture alerts, which makes the user experience more engaging and intuitive.

Moreover, JavaScript offers a wide range of libraries and frameworks that simplify the development process by providing pre-written code for common functionalities. For instance, the use of Bootstrap can help to streamline the creation of responsive layouts and dynamic content, reducing the amount of time and effort required to build a web application.

4.3.2.3 HTML and CSS

HTML and CSS are two essential technologies used in web development. HTML stands for Hypertext Markup Language, which is a markup language used for creating web pages and applications. HTML is used for structuring content on the web and defining elements such as headings, paragraphs, images, links, and forms.

On the other hand, CSS stands for Cascading Style Sheets, which is a style sheet language used for describing the presentation of a document written in HTML. CSS is used for defining the visual appearance of a web page, including layout, colours, fonts, and other design elements.

4.3.3 Frameworks and Libraries

4.3.3.1 TensorFlow

TensorFlow is an open-source software library for dataflow and differentiable programming across a range of tasks. It is a popular library for building and training machine learning models, particularly deep neural networks. The library is designed to work efficiently with large-scale neural networks and to support distributed computing. TensorFlow offers a variety of APIs for different levels of abstraction, from low-level APIs for building custom models to high-level APIs for quickly building and training common types of models. The library also includes pre-trained models and tools for visualisation, data preparation, and model deployment.

4.3.3.2 ONNX

ONNX (Open Neural Network Exchange) is an open standard for representing deep learning models. It provides a format for exchanging models between different deep learning frameworks, allowing developers to create models using their preferred framework and then deploy them on a variety of hardware platforms. ONNX is supported by a wide range of popular frameworks, including TensorFlow.

The use of ONNX in the app will be favourable to export the deep learning model chosen using a particular framework, and then deploy it on the Jetson Nano. ONNX can help make the app more flexible and compatible with a wider range of hardware and software environments.

4.3.3.3 OpenCV

OpenCV offers a range of functionalities such as object detection, image and video processing, feature extraction, and more. This library provides a set of pre-built algorithms and tools that help developers to build applications faster and more efficiently.

OpenCV could be used to perform posture detection by analysing the user's body position through a camera feed. The app could use OpenCV to detect the user's posture and alert them if they are sitting in an unhealthy position for an extended period.

4.3.3.4 TensorRT

TensorRT is a high-performance deep learning inference engine developed by NVIDIA that optimises and deploys trained neural networks for production environments. It is designed to achieve low-latency and high-throughput inference for a wide range of deep learning applications, such as image recognition. TensorRT is optimised for NVIDIA GPUs such as the Jetson Nano, making it an ideal choice for applications that require real-time processing of large amounts of data.

The main goal of TensorRT is to provide a way to optimise trained neural networks for deployment on production hardware. It does this by taking a trained model, usually in a format such as TensorFlow or ONNX, and optimising it for deployment on an NVIDIA GPU. The optimisation process involves several steps, including layer fusion, precision calibration, and kernel auto-tuning. These steps are designed to reduce inference latency and memory usage while increasing throughput and maintaining accuracy.

4.3.3.5 Django

Django is a high-level web framework for building web applications in Python. It follows the model-view-controller (MVC) architectural pattern, which separates the application logic, user interface, and data models into separate components. it provides many features out-of-the-box that are required to build web applications such as authentication, database support, and URL routing. It also provides an object-relational mapping (ORM) system that makes it easy to interact with databases using Python code, without needing to write SQL queries directly.

4.3.3.6 Bootstrap

Bootstrap provides a set of pre-designed CSS styles, JavaScript plugins, and HTML templates that can be used to create responsive and mobile-first web pages and web applications. Bootstrap allows developers to build web pages quickly and easily using a grid system, responsive typography, and pre-built UI components such as buttons, forms, and navigation bars. This reduces the amount of time and effort required to design and code a website from scratch, while also ensuring that the website looks consistent and professional across different devices and screen sizes.

4.4 Design

The posture sitting app was structured as a combination of a Jetson Nano device and a Django web application. It was developed to be accessible on both PCs, for those who would be sitting at a desktop, and on phones, for others engaged in different activities while seated.

4.4.1 Monitoring Device

The Jetson Nano served as a monitoring device to track the user's posture while seated. The device is equipped with a camera that captures images of the user and a pre-trained CNN model called "single pose moveNet" from TensorFlow that is used to detect and monitor the user's posture in real-time. This pre-trained model detects the user's body key joints.

The MoveNet Single Pose model works by processing input images through a series of layers. The input image is first passed through a series of convolutional and pooling layers to extract low-level features such as edges and corners. This is followed by a set of intermediate convolutional and pooling layers that progressively learn more complex features, such as body part edges and joints. The intermediate layers also include residual connections, which help to improve gradient flow during training and enable the network to learn deeper and more complex representations. The output of the intermediate layers is then passed through a set of fully connected layers to produce the final pose estimation. The output is a set of confidence scores and coordinates for each body part, which indicate the likelihood that a given body part is present and the position of that body part in the image. The body part predicted by the model form a human skeleton that could be seen in Figure 3.

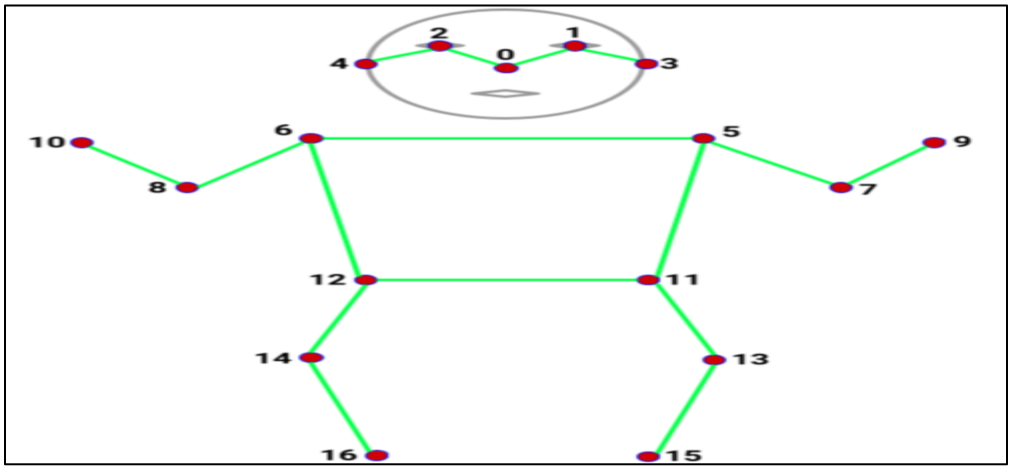


Figure 3: A picture of the body key joints detected by the moveNet

The app was designed to operate from three camera perspectives of choice. This gives users more flexibility and allow them to be monitored from the angle that suits them the most. Camera perspectives worked towards throughout the project are lateral right, lateral left and frontal as shown in Figure 4.

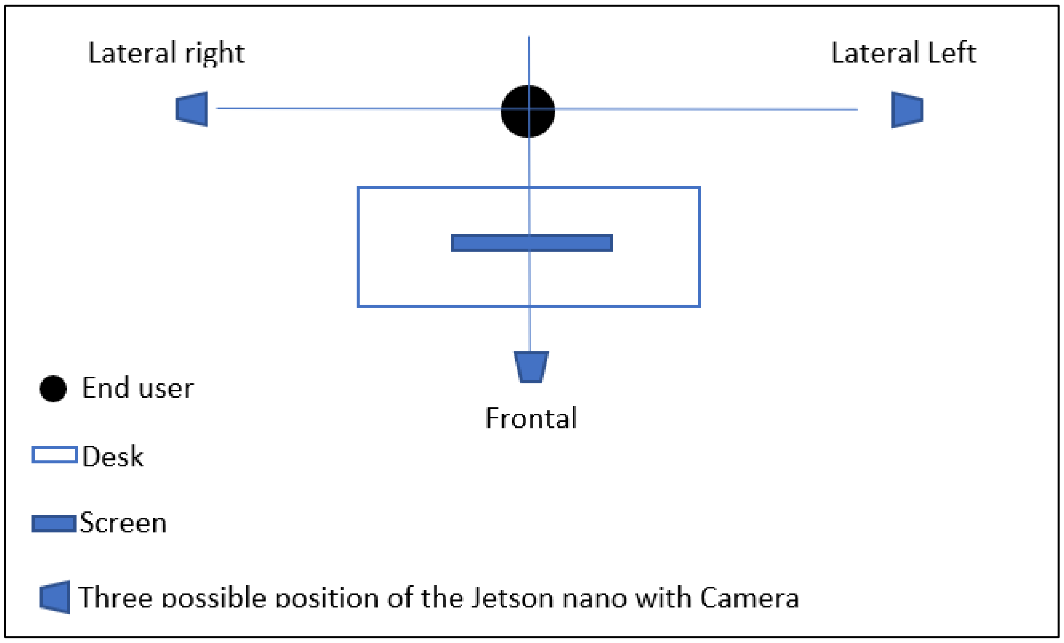


Figure 4: A drawing of the monitoring system camera perspective options

Users skeletal data were capitalised on to determine users posture based on the perspective of the camera. The moveNet model detection on these angles can be seen in Figure 5.

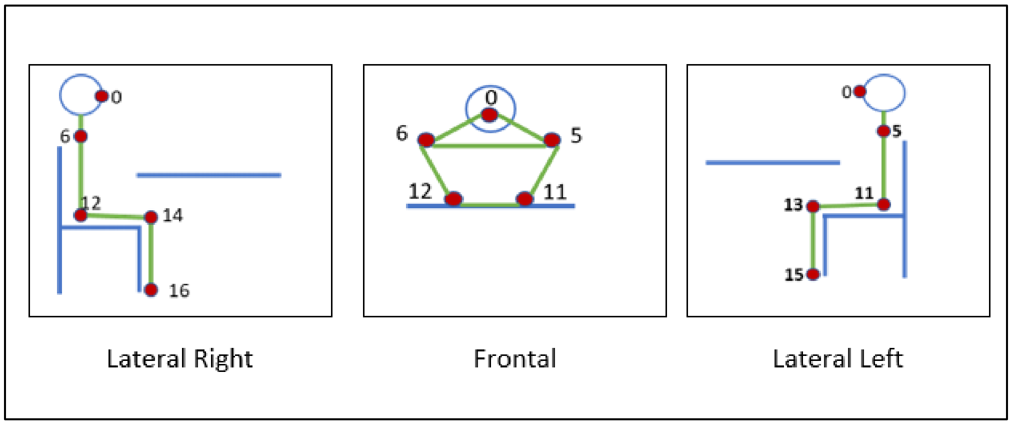


Figure 5: A drawing of skeletal data used for each camera perspective.

The use of the body joints predicted by the moveNet model differed depending on the camera perspective, whereas results of detections were based on the neck, and back postures for all cases. Table 3 shows thresholds and body parts used for each camera perspective. It is noteworthy that thresholds were established using a dataset collected from Google Image. These images included different types of postures that were processed with the moveNet model to ultimately set the thresholds.

Camera Perspective	**Body Parts**	**Neck Posture**	**Back Posture**
Lateral Right	-Nose (0) -Right shoulder (6) -Right hip (12) -Right Knee (14)	DNS < 8 pixels: -Bent Forward	RHA < 90° : -Forward
		DNS > 8 pixels: -Upright	90° < RHA < 115° : -Upright
			RHA > 115° : -Reclined
Lateral Left	-Nose (0) -Left shoulder (5) -Light hip (11) -Left Knee (13)	DNS < 8 pixels: -Bent Forward	LHA < 90° : -Forward
		DNS > 8 pixels: -Upright	90° < LHA < 115° : -Upright
			LHA > 115° : -Reclined
Frontal	-Nose (0) -Right shoulder (6) -Left shoulder (5) -Right hip (12) -left hip (11)	35° < RSA 35° < LSA -Upright	DSH < DNH: -Upright
		35° > RSA 35° > LSA -Bent	DSH > DNH -Forward

Table 3: Body joints & thresholds.

Table acronyms meaning:

- **DNS:** Distance between Nose and Shoulder.
- **DSH:** Distance between Shoulder and Hips.

- **DNH:** Distance between Nose and Hips.
- **RHA:** Right Hip Angle at joint 12.
- **LHA:** Left Hip Angle at joint 11.
- **RSA:** Right Shoulder Angle at joint 6.
- **LSA:** Left Shoulder Angle at joint 5.

4.4.2 Web Application

The Django web application serves as the user interface for the app. It provides a platform for users to interact with the app, view their posture history, and monitor their posture in real-time. The web application is designed using Django's Model-View-Template (MVT) architecture. The user interface is designed to be intuitive and easy to use, providing users with a clear view of their posture statistics, posture history, as well as photos of incorrect postures captured.

The posture data captured by the Jetson Nano is sent to the Django app server using a post request. On the app server side, The data are stored in a PostgreSQL database, in order to enable users to retrieve their posture history and view their progress over time.

4.4.2.1 User Flowchart

The app user flowchart is the visual representation of the steps the user takes to navigate through the app. As shown in Figure 6, It maps out the various screens, actions, and decisions that the user may encounter while navigating through the system, from the initial entry point to the desired outcome. Producing the user flowchart helped ensure that the system was designed with the user's needs and goals in mind, leading to a better user experience.

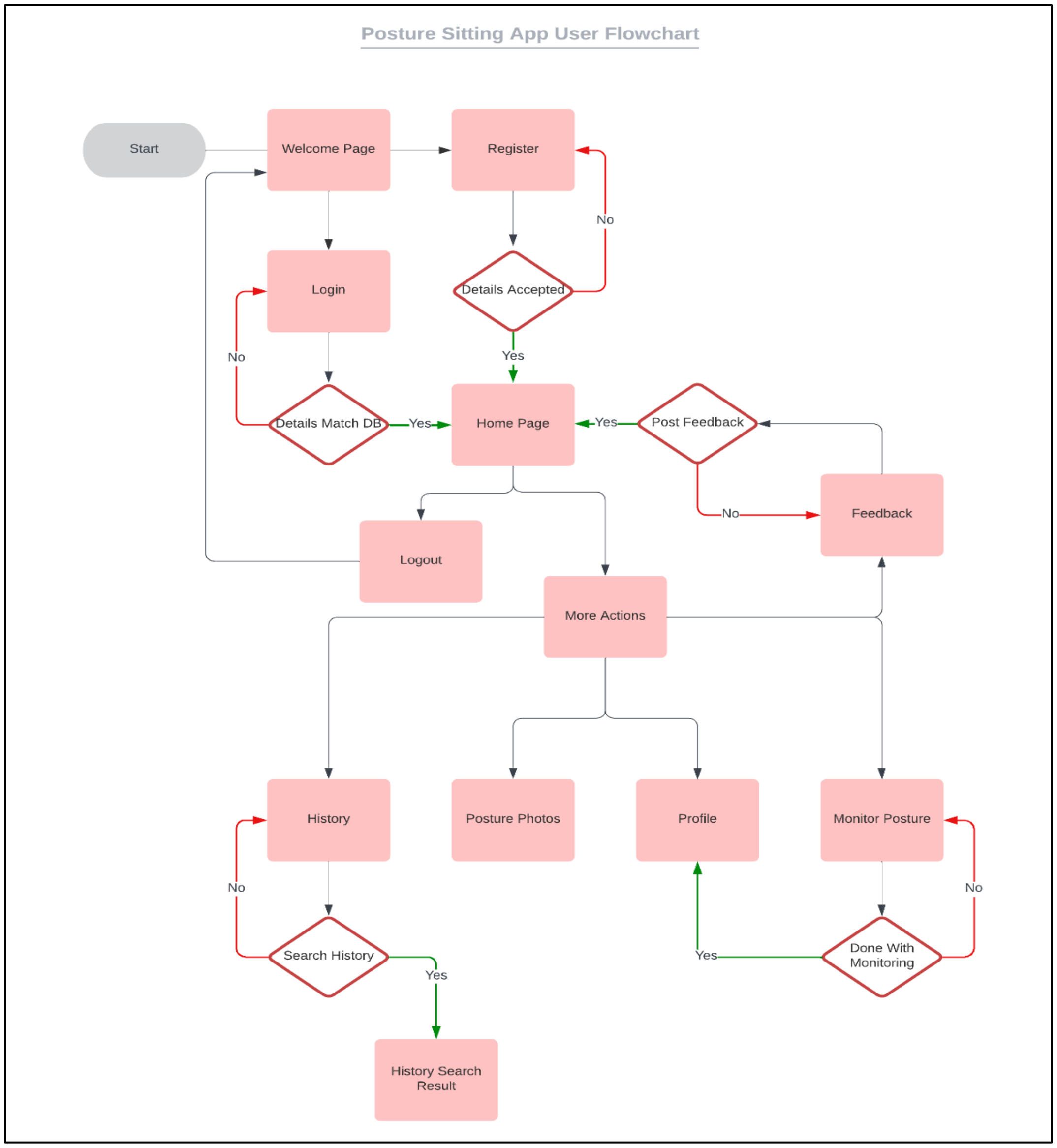


Figure 6: A picture of the user flow diagram

4.4.2.2 Database Entities & Attributes

After a meticulous analysis of the requirements, it was evident that the database structure would consist of a User entity for authentication, which was linked to other entities such as Notifications, Videos, Feedback, and Poor Postures using foreign keys. These entities have attributes to store all the relevant information about the user's notifications, videos, feedback, and poor postures.

- **User:** This entity represents the users of the app. It has attributes such as first name, last name, email, profile picture, date created, is active, and is admin.
- **Notifications:** This entity is related to the User entity through a foreign key. It has attributes such as back alert and neck alert, which are integers that represent the number of alerts received by the user for bad posture.
- **Videos:** This entity is also related to the User entity through a foreign key. It has attributes such as start time, end time, total time, total alerts, incorrect postures, and posture score. The incorrect postures attribute is an array field that stores the type of incorrect posture detected in the video.
- **Feedback:** This entity represents the feedback given by users of the app. It has attributes such as opinion, which is the text of the feedback, and date created, which is the date and time the feedback was created.
- **Poor Postures:** This entity represents the instances of poor posture detected by the app. It has attributes such as posture photo, which is the image of the user with bad posture, and date created, which is the date and time the poor posture was detected.

4.4.2.3 Database Entity-Relationship Diagram

The Entity-Relationship Diagram (ERD), is a visual representation of a The app database's structure that served the organisation and clarification of the relationships between the entities mentioned formerly. The ERD allowed for better understanding in the database development process, and provided a clear way to visualise the various entities, their attributes, and the relationships between them. As shown in Figure 7, entities are represented by rectangles, and relationships between them are represented by lines.

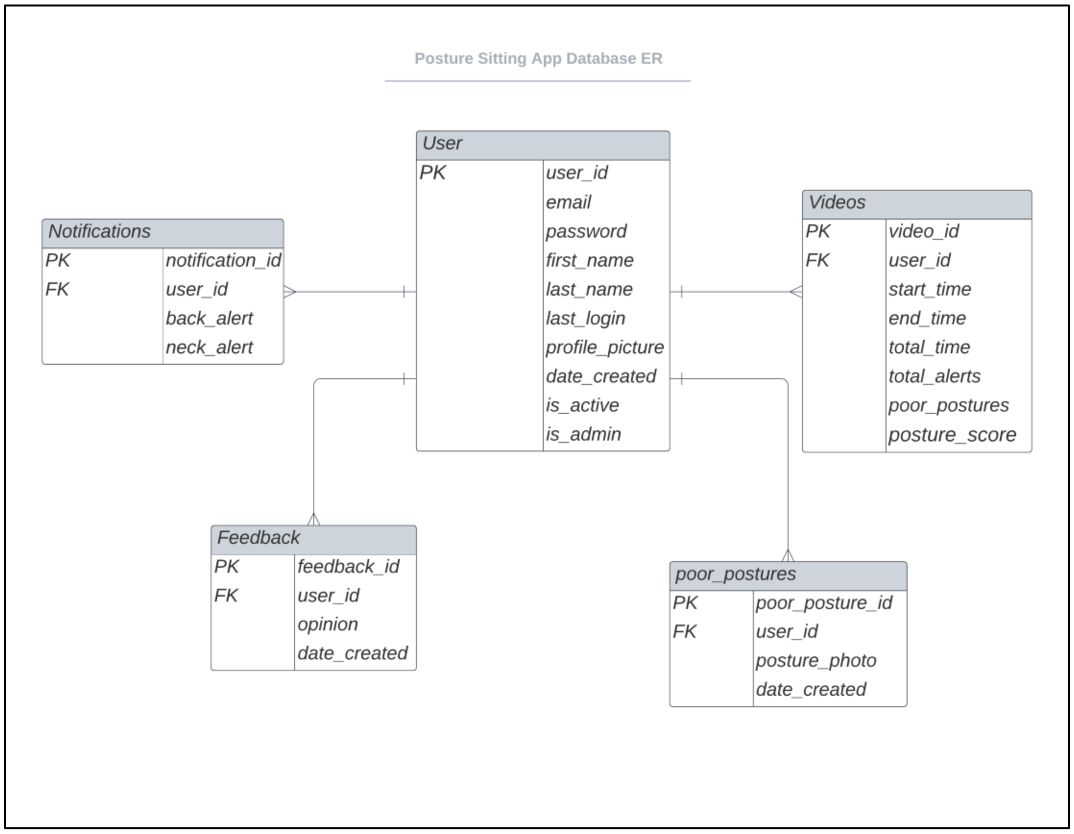


Figure 7: A picture of the database ERD

4.4.2.4 Real-Time Feedback

In the app, Server-Sent Events (SSE) were used to update the user interface in real-time upon database updates. This was to allow users to monitor their posture in real-time and receive immediate feedback if they are sitting in a poor posture for a period of 10 seconds.

SSE is a technology that allows a web page to receive automatic updates from a server via a persistent connection. In other words, the server can push new data to the client whenever it is available, rather than the client needing to constantly poll the server for updates. Hence, SSE was used to push updates to the user's browser whenever there is a change in the posture data stored in the PostgreSQL database. SSE connections are established between users' browser and the server, when users monitor their posture on the app, the client-side JavaScript code then updates the user interface to show the alert in the form of a modal that plays an audio to draw the users attention.

4.5 Testing

4.5.1 Unit Testing

Unit testing is a critical part of software development that involved testing individual units or components of the sitting posture app in isolation from the rest of the system to ensure that they function as intended. Unit testing was a part of the development cycle as it took place during each sprint. That was to ensure that each new feature or code change was tested thoroughly before being integrated into the app.

To effectively implement unit testing in the project, it was essential to define test cases and test scenarios that cover all possible use cases and edge cases. As the development was SCRUM driven, unit tests were written before writing the actual code, which ensured that the code was testable, well-structured, and to meet the requirements introduced by test cases. The execution of these tests contributed immeasurably to detect any defects or issues in the code, and tackle them to drive the sprint to completion within the timeframe established. Units tests will be included in the appendix.

4.5.2 Performance Testing

Performance testing was imperative to evaluate the performance of the pre-trained CNN model in its initial form. These tests helped immensely to verify that the model could process data within an acceptable timeframe and would perform reliably under heavy loads. It is noteworthy that performance was a condition that had to be fulfilled as the app requirements dictated a real-time functioning of the product.

Performance testing were integrated into the development process, specifically throughout Sprint 1. The reason for this was the integration of the model to the monitoring device and the optimisations that took place to make it run optimally during the sprint. These tests proved to be invaluable to detect performance issues to be addressed swiftly and efficiently, and to comply with the project timeline.

The performance testing process involved running a set of tests on the CNN model to evaluate its performance on the Jetson Nano. Tests were designed to simulate real-world usage scenarios and used to identify any bottlenecks or performance issues. The results of these tests were then analysed to identify any areas for optimisation.

After a meticulous testing of the performance, optimisation techniques have been applied in order to satisfy the project requirements and move forward with the development process, these techniques are discussed in the implementation section.

4.6 Risk Analysis

4.6.1 Model Performance

Training a new CNN model to detect postures requires a significant amount of labelled data, and expertise in deep learning. Risks associated with this task include the availability and quality of training data, potential overfitting or underfitting of the model, and the time and resources required for training and fine-tuning the model.

A mitigation that have been applied to this risk that occurred through the course of the project was using a pre-trained CNN model instead. The model was not trained to recognise incorrect postures but could detect key joints of the body that were exploited to serve the same purpose; detecting incorrect postures.

4.6.2 Technical Development

One technical development risk associated with the application is the complexity of its development. The app comprises multiple interconnected components, including the Django web app, the Jetson Nano monitoring device, and the integration between them. The complexity arises from the need to ensure seamless communication, data synchronisation, and real-time processing between these components. Furthermore, developing a comprehensive and user-friendly web app with authentication, data storage, and real-time updates requires careful planning and implementation. Challenges may arise in designing the user interface, handling user authentication securely, managing the database, and handling concurrent requests effectively. Similarly, the development of the Jetson Nano monitoring device involves challenges in implementing computer vision techniques, and integrating the pre-trained CNN model. Ensuring real-time posture detection, minimising latency, and handling video input from the integrated camera require expertise in computer vision and deep learning.

To mitigate the risk of complexity, it was crucial to conduct thorough planning, design robust architectures, and follow best practices for software development. Regular testing, debugging, and iterative development helped identify and resolve issues early on. Forums were consulted, and libraries were used to facilitate the development of the app.

4.6.3 Jetson Nano Performance

The performance of the Jetson Nano, being the monitoring device, is also a potential risk as its computational capabilities and resource limitations may impact the real-time posture detection process. There is a risk of the Jetson Nano experiencing latency or slower processing times when analysing the video input and making posture predictions. This can result in delays in detecting poor posture and triggering alerts to the web app.

It was important to consider the Jetson Nano's performance specifications, and mitigate this risk by optimising the inference process, and conducting thorough testing to ensure its efficient and reliable operation within the app. Monitoring and addressing performance constraints was crucial to mitigate this risk and provide a smooth user experience. Table 4 summarises the risks analysis.

Potential Risks	**Impact**	**Probability**	**Mitigation Strategy**
Trained posture recognition Model Performance	Failure of the monitoring system.	High	**-**Data Augmentation techniques. -Use of pre-trained models
Jetson Nano performance	Latency of the video may impact the user experience	High	- Hardware Upgrade -Model Optimisation Techniques to run inference.
Technical development complexity	The project may take longer to complete	Medium	-Use of libraries -Consulting forums

Table 4: Risk analysis.

Chapter 5

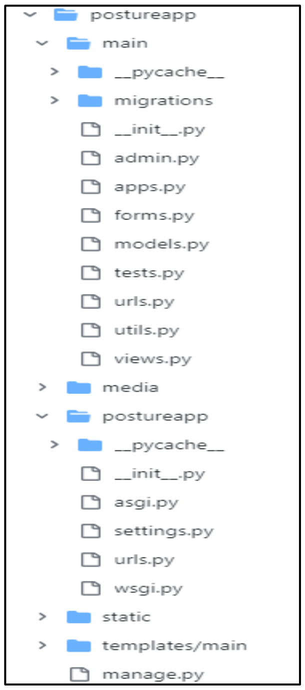
Implementation

5.1 Overview

The application is composed of two primary components: a Django web app and a monitoring device, namely the Jetson Nano. These components work together to provide the functionality of the app. The web app serves as the user interface, allowing users to interact with the system, access their accounts, view posture statistics, and receive alerts. It also enables users to provide feedback, and access historical data.

On the other hand, the monitoring device, the Jetson Nano, plays a crucial role in posture detection. It utilises computer vision techniques and a pre-trained CNN model to analyse the user's sitting posture in real-time. The Jetson Nano captures video input from its integrated camera and processes it using the deep learning model. If poor posture is detected, the device triggers an alert to the web app, which then notifies the user.

The combination of the web app and the Jetson Nano forms a seamless and comprehensive system for monitoring and improving sitting posture. Both structures of the Django app and Jetson Nano are represented in Figure 8 and Figure 9 respectively.


```
postureapp
  main
    __pycache__
    migrations
    __init__.py
    admin.py
    apps.py
    forms.py
    models.py
    tests.py
    urls.py
    utils.py
    views.py
  media
  postureapp
    __pycache__
    __init__.py
    asgi.py
    settings.py
    urls.py
    wsgi.py
  static
  templates/main
  manage.py
```

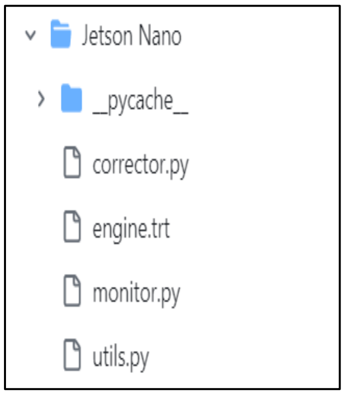

```
Jetson Nano
  _pycache_
  corrector.py
  engine.trt
  monitor.py
  utils.py
```

Figure 9: A picture of the monitoring device tree

Figure 8: A picture of The web app tree

5.2 Back-end

5.2.1 Jetson Nano Setup

Setting up the Jetson Nano was a relatively straightforward process. With the right software and packages installed, this embedded system is a robust hardware for building computer vision applications that can run in real-time when used appropriately.

The Jetson Nano had to be connected to a monitor, keyboard, and mouse using HDMI and USB cables, then a microSD card was inserted with the Jetson Nano image pre-installed. This image was downloaded from the NVIDIA website, where the instalment of software such as SD memory card formatter, as well as Etcher was required to format the SD card properly.

In order to boot the Jetson Nano, it was connected to a power source using a power supply. The initial setup was then completed by following the on-screen prompts of the Jetson Nano which included setting the time zone, language, and keyboard layout. Following that, necessary libraries and packages were installed including TensorFlow, and OpenCV, among other packages. The Raspberry Pi camera was then connected to the Jetson Nano as shown in Figure 10, and the hardware was ready to host a computer vision script to monitor the user's sitting posture.

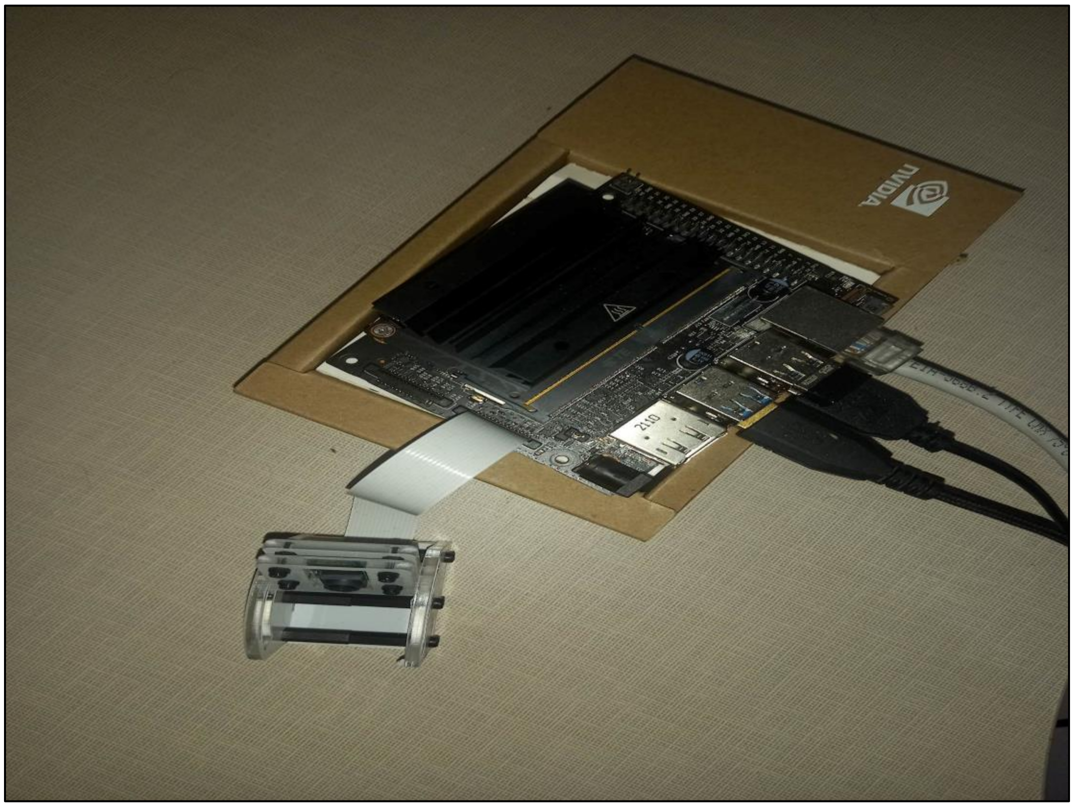


Figure 10: A picture of the Jetson Nano with the Raspberry Pi camera

5.2.2 Model Optimisation

5.2.2.1 From TensorFlow-lite to ONNX

To begin the process of optimising the MoveNet model from TensorFlow-lite to ONNX, the first step was to clone the TensorFlow-onnx repository on GitHub with the following command in the Jetson Nano command prompt:

'git clone https://github.com/onnx/tensorflow-onnx'

Next, other required libraries were installed. These included protobuf, onnx and tf2onnx, where protobuf is a protocol buffer library used for data serialisation, and tf2onnx is a Python package used to convert TensorFlow models to ONNX format. These libraries were installed using pip, the package installer for Python.

The MoveNet model was then converted from TensorFlow-lite to ONNX format using the following command:

'python -m tf2onnx.convert --opset 13 --tflite tflite--file --output model.onnx'

The 'opset' parameter refers to the version of the ONNX operator set that is used, and 13 is the compatible version with the libraries installed formerly. The 'tflite--file' parameter specifies the path to the TensorFlow-lite model file, whereas the '--output' parameter specifies the name of the output ONNX file.

After running the conversion command, the MoveNet model was converted to the ONNX format, ready to be optimised further prior to deployment as at this stage, the model was still not adequately performant to be integrated to the app.

5.2.2.2 From ONNX to TensorRT

Converting the ONNX model to a TensorRT engine was necessary to deploy the model on the Jetson Nano. This process entailed the use of TensorRT, a high-performance deep learning inference library developed by NVIDIA, which enables efficient deployment of trained models on NVIDIA GPUs. This conversion had to be performed on the Jetson Nano as the optimised TensorRT engine is not portable between different architectures. The engine generated was tailored to run on the Jetson Nano's GPU, leading to faster and more efficient inference.

Before converting the ONNX model to a TensorRT engine, several libraries were installed on the Jetson Nano. These libraries included CUDA and cuDNN, which are required for GPU acceleration. Additionally, the ONNX runtime and the TensorRT runtime were required, as well as the Python package 'pycuda', which provided a Python interface to CUDA.

The NVIDIA 'trtexec' tool was used to convert the optimised ONNX model to a TensorRT engine. This tool is a command-line utility for creating and optimising TensorRT engines from ONNX models. The command-line inputs for 'trtexec' included the ONNX model file, the desired output engine file, and the batch size. Furthermore, the precision mode of the TensorRT engine could optionally be chosen with the --fp16 or --int8 flags. Following the command typed in the command prompt to achieve this:

'sudo trtexec –onnx=<onnx_model_path> --saveEngine=<trt_engine_path>'

onnx_model_path is a placeholder for the onnx model optimised earlier path, whereas trt_engine_path will be the location of the TensorRT engine generated. This operation created a serialised TensorRT engine file that was subsequently used to run inference.

The diagram in Figure 11 illustrates the process undertaken to fully optimise the pre-trained CNN model and leverage its use on the Jetson Nano.

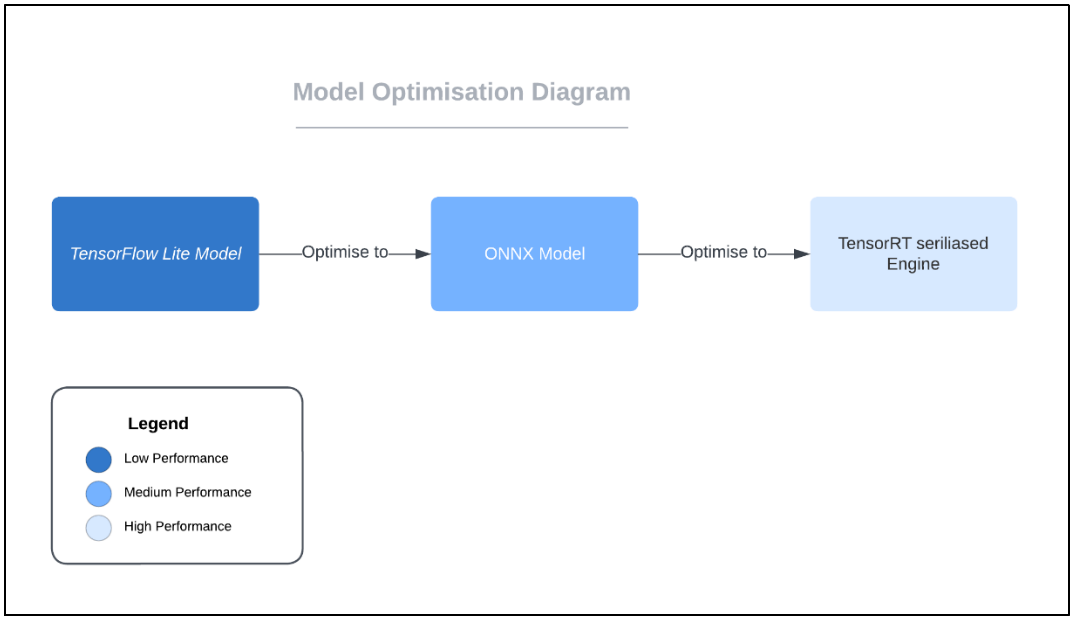


Figure 11: A picture of the moveNet model optimisation process

5.2.3 Real-time Poor Posture Detection

5.2.3.1 Jetson Nano Side

Now that the 'MoveNet' model is optimised and runs optimally on the Jetson Nano board, the app directory for the monitoring system was split into 3 files, and a folder to store poor postures photos and send them to the Django app where they will be stored in the database. The code was written in an object-oriented way to help in organising and structuring it in a logical modular way. This promotes reusability which can save time and effort in developing new features or applications.

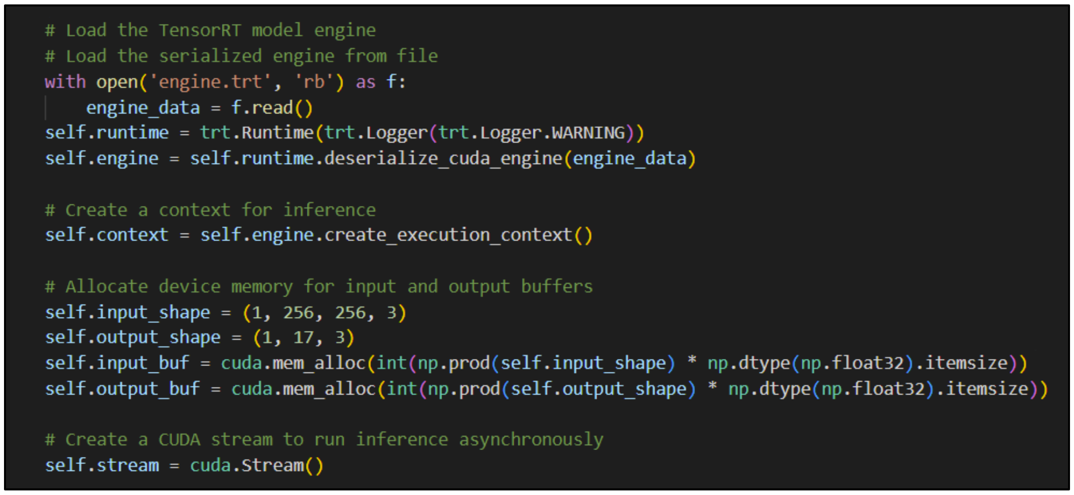
- **Connecting the Jetson Nano to the Django app:**

A class was created with a constructor that takes in several parameters such as url, email, password, and camera position. The email and password parameters are used to authenticate the user to the Django App Server from the Jetson Nano through a post request. On the other hand, The url parameter stores the app host url and the port attribute was set to '5000'; the port on which the Django app is listening. These parameters are in place to send all relevant users posture information to the app's database during the video.

Both of the jetson nano and the django app were connected to the same network in order for the app to run. This was since the Jetson Nano sends post requests to the app to process and receive the results of the computer vision script. If the Django app and Jetson Nano are not on the same network, the post requests may not be able to reach the app, resulting in communication errors and a failure to perform the posture detection process. Therefore, to ensure the app functions properly, the Django app and Jetson Nano must be connected to the same network, allowing for the successful exchange of post requests and responses.

- **Loading Model:**

The constructor loads the pre-trained TensorRT model engine that has been optimised from the file 'engine.trt' and creates a context for inference. As illustrated in Figure 12, device memory is allocated for input as well as output buffers, and a CUDA stream is created to run inference asynchronously.



```python
# Load the TensorRT model engine
# Load the serialized engine from file
with open('engine.trt', 'rb') as f:
    engine_data = f.read()
self.runtime = trt.Runtime(trt.Logger(trt.Logger.WARNING))
self.engine = self.runtime.deserialize_cuda_engine(engine_data)

# Create a context for inference
self.context = self.engine.create_execution_context()

# Allocate device memory for input and output buffers
self.input_shape = (1, 256, 256, 3)
self.output_shape = (1, 17, 3)
self.input_buf = cuda.mem_alloc(int(np.prod(self.input_shape) * np.dtype(np.float32).itemsize))
self.output_buf = cuda.mem_alloc(int(np.prod(self.output_shape) * np.dtype(np.float32).itemsize))

# Create a CUDA stream to run inference asynchronously
self.stream = cuda.Stream()
```

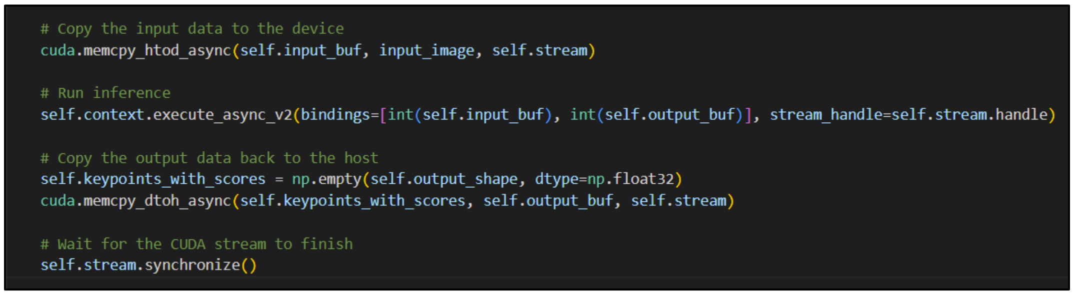
Figure 12: A picture of the python code loading the model

Thereafter, attributes are initialised to store various data related to posture detection, such as incorrect postures, and body part coordinates. The latter is a dictionary that was used to identify the coordinates of the user's 17 body joints.

The detect method makes predictions on the input image to detect 17 body joints and update the coordinates of each key point in the parts coordinates dictionary. The method takes an input image as an argument in the form of an array.

First, the input data is copied to the device asynchronously using the cuda.memcpy_htod_async method. Then, the inference is run using the execute_async_v2 method of the context object, which executes the model asynchronously and takes two arguments, the input and output buffers of the model.

Next, the output data is copied back from the device to the host using the cuda.memcpy_dtoh_async method. The synchronise method is then called on the stream object to wait for the CUDA stream to finish as shown in Figure 13.



```
# Copy the input data to the device
cuda.memcpy_htod_async(self.input_buf, input_image, self.stream)

# Run inference
self.context.execute_async_v2(bindings=[int(self.input_buf), int(self.output_buf)], stream_handle=self.stream.handle)

# Copy the output data back to the host
self.keypoints_with_scores = np.empty(self.output_shape, dtype=np.float32)
cuda.memcpy_dtoh_async(self.keypoints_with_scores, self.output_buf, self.stream)

# Wait for the CUDA stream to finish
self.stream.synchronize()
```

Figure 13: A picture of the python code running inference

After that, as the video frames represent a grid, the parts_coordinates dictionary is updated with the user's latest cartesian coordinates detected for each body part. This detection process is undertaken for each frame of the video followed by other operations to establish the user's sitting posture.

The subject's posture is technically estimated by a monitoring method, which performs the surveillance in segments of 10 seconds and notifies the user if the subject happens to be in an improper posture for the entire duration. This method takes the camera position into consideration (lateral right, frontal, or lateral left) and calls the appropriate methods to check for poor posture.

- **Neck Posture Detection:**

For a frontal camera perspective, the angle between the subject's nose and shoulders is calculated, based on the nose, left shoulder, and right shoulder cartesian coordinates. Likewise, the angles between the nose and the right shoulder and the nose and the left shoulder is established. If both angles are greater than 35 degrees the neck posture is defined as upright and the sitting posture deemed as correct. However, an angle less than 35 degrees would result in an incorrect posture with a neck bent.

On the other hand, the lateral camera perspective is purely reliant on comparing y-coordinates of the subject’s nose and shoulders. The distance between the nose, and left shoulder as well as the nose and right shoulders must both be over 8 pixels to be considered as a correct posture where the user neck is upright. Otherwise, if these distances are less than that threshold, the neck posture is concluded as incorrect and its actual state as bent. This applies to both lateral right perspective and lateral left as nothing technically change.

- **Back Posture Detection:**

In order to determine whether a user is sat correctly from a frontal camera view based on the back position on the grid, the distance of the hip and shoulder are inferred along with the distance of the shoulder and hip. If the nose and hip segment turns out to be greater than the shoulder and hip segment, the back posture is considered as correct and upright. The opposite scenario would conversely define the posture as incorrect with a back leaning forward. Note that these segments are calculated by accounting for the body joints detected by the CNN model, the knowledge of the cartesian coordinates of the hips, shoulders, and nose are leveraged to serve this purpose.

As for the lateral camera perspective, the approach taken differs since angles can be easily calculated. Similarly to the previous solutions, the cartesian coordinates are exploited to calculate the hip angle. In this case, the body parts used are the shoulders, hips, and knees. The hip angle is deducted and the user's posture state would depend on the result yielded. If the hip angle falls into the range of 90-115 degrees, the user is sat upright whereas angle less than that range would convey the user being sat leaning forward. Another potential sitting posture, that is solely inferable from this camera perspective, is the reclined sitting posture which is established when the hip angle is greater than the threshold range.

- **User Alerts:**

As the back and neck monitoring is ongoing, the results of the postures are stored in arrays to inform the user about the incorrect postures adopted throughout the video. The posture detection methods defined formerly contribute to keep track of the user's posture in real-time and allow the implementation of a method timer that calls other methods to alert the user. These posture detection methods returns either 1 or 0 referring to a correct and incorrect posture respectively. This is where the timer method comes into play, it checks the result of these methods continuously where nothing would happen if the posture methods return 1 for a correct posture.

In contrast if the result indicates an incorrect posture by returning 0, a counter would be incremented in the class. this counter is compared to the number of frames generated in 10 seconds, so that if both number are equal, an alert would be triggered to remind the user to sit correctly. If the counter don't reach the number of frames, and the user ease back into a correct posture, the counter would be reset. This system has been implemented so that the alerts won't spam the user and to offer some flexibility. The timer works for both of the neck and back, and triggers alerts accordingly by other methods that makes post requests to the Django app.

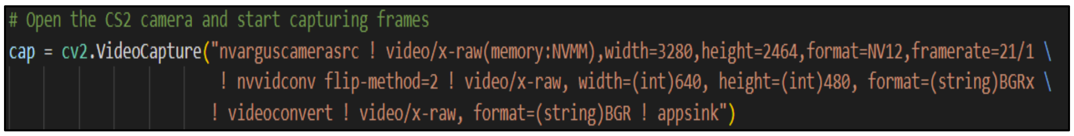
Furthermore, the timer uses another method to save the last frame of the incorrect posture held for 10 seconds in the incorrect posture folder within the monitoring device repository. Many other relevant information are stored in variables throughout the monitoring video, namely the number of alerts, the start, and end time. These are pushed into the app's database through another method that carry out a post request, intended to run as the user stops the program. The database is therefore populated with posture relevant data, including posture photos if any incorrect posture was adopted.

- **Starting Monitoring Device:**

In order for the Monitoring device to run, the user must boot the Jetson Nano and execute the monitor.py script from a terminal.

The program prompts the user to authenticate by entering their email address, and their password. The password is hidden by using the getpass library for security purposes. The user is then requested to opt for a camera angle to be monitored from; this could either be Frontal, Lateral Right, or Lateral Left depending on the user's preference and the position of the camera relative to their position. Once a valid monitoring option is chosen, the program launches popping up a screen that shows the user, their skeleton, as well as their neck and back posture status.

For the Raspberry Pi camera to be recognised by the Jetson Nano, certain parameters have to be passed as a string to the OpenCV function used to open the camera as shown in Figure 14.



```
# Open the CS2 camera and start capturing frames
cap = cv2.VideoCapture("nvarguscamerasrc ! video/x-raw(memory:NVMM),width=3280,height=2464,format=NV12,framerate=21/1 \
                        ! nvvidconv flip-method=2 ! video/x-raw, width=(int)640, height=(int)480, format=(string)BGRx \
                       ! videoconvert ! video/x-raw, format=(string)BGR ! appsink")
```

Figure 14: A picture of the python code opening the camera

The video frames are pre-processed by resizing and converting them to a float32 data type. An instance of the class defined earlier is then created, allowing all the operations described in the class to come into effect in a real-time scenario.

As users are done monitoring themselves, they can end the program by pressing the **‘q’** key on the Jetson Nano. All the data generated would be transferred to the Django app where users can see their statistics, video history, and incorrect posture photos if any were detected throughout the video.

5.2.3.2 Django App Side

In order to create a Django app to serve as a user interface (UI) for the posture correction application, several steps were taken. The first step was to create a Django project (postureapp), which is a collection of settings and configurations for a specific website. This was done using the following command:

‘django-admin startapp <projectname>’

Once the project was created, a new app was added to it (main). This was done using the following command:

‘django-admin startapp <appname>’

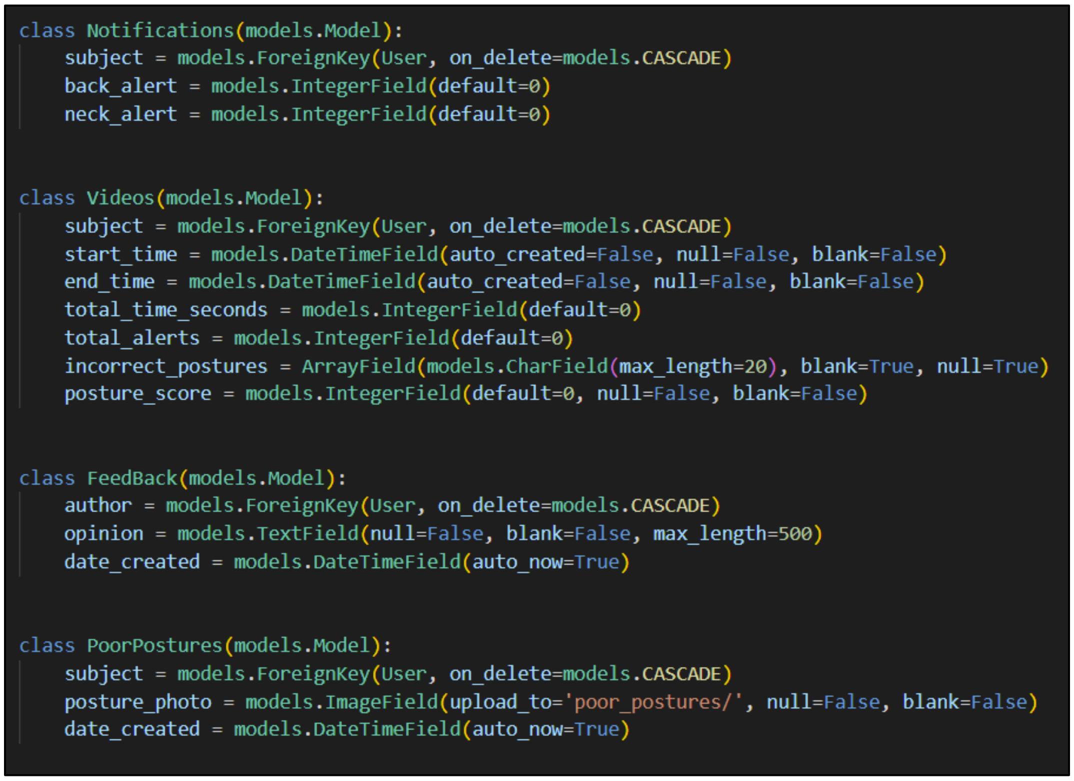
This app contains the models, views, and templates required to implement the functionality of the posture correction application. Automatically, a settings.py file is generated where many configurations were done including setting the PC used to develop as a host for the server, and accepting a high number of post requests so that the Jetson Nano could be synchronised to the App and exploited.

Other configurations were setting and creating media, static, and template directories where the path of these folders had to be specified within the app. The media directory is to store the app and user’s photos, whereas the template and static directories are for HTML files and front-end styling languages respectively.

As for the Database configuration, a Django user with all privileges granted was created and connected to the app to access the PostgreSQL database, and make migrations. Django Channels were installed to enable real-time alerts that will be discussed in the real-time alerts section.

- **Models:**

The next step was to create the database tables which entailed defining the models for the posture app. In essence, models are Python classes that define the database schema for the app, they were created based on the ER diagram Figure 7) shown in the design section. Figure 15 shows how these models included classes for storing user information, such as email and password, as well as posture-related data, such as images and incorrect postures.



```python
class Notifications(models.Model):
    subject = models.ForeignKey(User, on_delete=models.CASCADE)
    back_alert = models.IntegerField(default=0)
    neck_alert = models.IntegerField(default=0)


class Videos(models.Model):
    subject = models.ForeignKey(User, on_delete=models.CASCADE)
    start_time = models.DateTimeField(auto_created=False, null=False, blank=False)
    end_time = models.DateTimeField(auto_created=False, null=False, blank=False)
    total_time_seconds = models.IntegerField(default=0)
    total_alerts = models.IntegerField(default=0)
    incorrect_postures = ArrayField(models.CharField(max_length=20), blank=True, null=True)
    posture_score = models.IntegerField(default=0, null=False, blank=False)


class FeedBack(models.Model):
    author = models.ForeignKey(User, on_delete=models.CASCADE)
    opinion = models.TextField(null=False, blank=False, max_length=500)
    date_created = models.DateTimeField(auto_now=True)


class PoorPostures(models.Model):
    subject = models.ForeignKey(User, on_delete=models.CASCADE)
    posture_photo = models.ImageField(upload_to='poor_postures/', null=False, blank=False)
    date_created = models.DateTimeField(auto_now=True)
```

Figure 15: A picture of the python code creating the database

Once the database models were defined, A migration was then performed to create the database based on the model classes designed. This migration enabled Django to create the database tables in PostgreSQL behind the scene. The commands to proceed were the 2 following:

'python manage.py makemigrations'

'python manage.py migrate'

These models were then accessed and queried on views.py either by the use of Django's built-in functions, or manually using customised methods and attributes.

- **Views:**

Views are essentially created to handle HTTP requests and render templates (HTML files), they are Python functions that take a request as input and return a response. In this app, views were incorporated for authenticating users, capturing camera input, processing images, and displaying results. These functions were also used to restrict some pages access strictly to the authenticated user for security measures. This was achieved using the Python decorators from Django library 'login required'**.**

Furthermore, views functions were implemented to accomplish other goals such as logout the user, querying the database for user's information, and inserting new data to the database in some scenarios. As a case in point, a user creating a new account would post data to the app front-end, which would be handled by the views from a back-end perspective. The data provided is inserted as a new row in the User Table in the app database, by the view function created to handle this task.

Post requests from the Jetson Nano were received in views endpoints with 'csrf exempt' decorators, where csrf stands for Cross-Site Request Forgery. Using these decorators make the views functions exempt from Django's middleware protection, enabling the Jetson Nano to populate the database if the user’s credentials matched the database record, when starting the monitoring script.

Creating views implied connecting them to URLs using the URLconf (URL configuration) system. This system maps URLs to view functions or classes that handle the HTTP requests and generate the responses. The URLconf is defined in the urls.py file of the Django App. It contains a set of URL patterns that are matched against the incoming request's URL. When a match is found, the corresponding view function or class is called to handle the request and generate the response.

- **Posture Statistics:**

The utils.py file of the Django app contains utility functions used in order to generate user statistics based on their database record. The statistics are generated by querying the database for posture scores (1), total time and alerts for each video. These data are then used to calculate the total good posture time (2), The percentage of posture improvement (3), and the average posture score (4). The total days spent on the app, posture latest score, and best score are also inferred by the use of function imported from utils.py. The calculations done to generate posture statistics are shown below, where poor posture time is 10 seconds; the time taken to generate alerts.

$$Posture\ score = 100 - \left(\frac{alerts\ number \times poor\ posture\ time}{total\ time} \times 100\right) \tag{1}$$

$$Good\ posture\ time = \text{total time} - \text{alerts number} \times \text{poor posture time} \tag{2}$$

$$Posture\ improvement = \left(\frac{last\ score - first\ score}{first\ score}\right) \times 100 \tag{3}$$

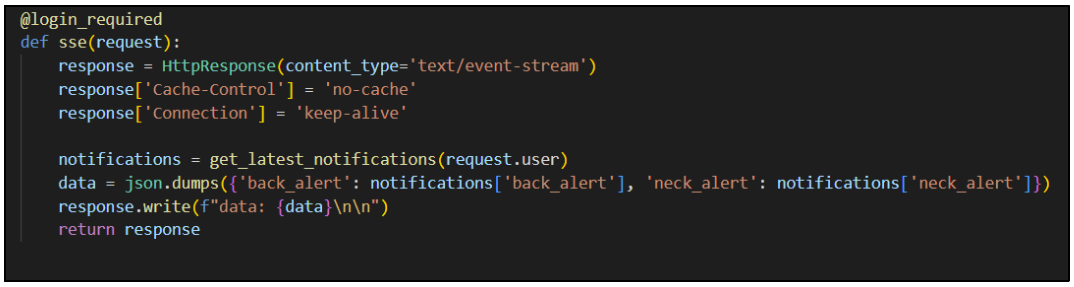
$$Posture\ Average = \frac{\sum posture\ score}{posture\ scores\ number} \tag{4}$$

- **SSE for Real-Time Alert System:**

In order to put the alert system in effect, the connection between the server and client is kept open for the server to push data changes to the latter. Once the user is authenticated, the function creates a new HttpResponse object and sets the content type to **'**text/event-stream'. This is because the function is designed to send real-time updates to the user's browser, so the response needs to be in a specific format that allows for server-sent events. The response headers are then set to 'no-cache' and 'keep-alive' to ensure that the connection stays open and that the data sent to the user is not cached.

Next, the function calls the 'get_latest_notifications' function, passing in the user object as an argument. This function retrieves the latest notifications for the user, specifically any alerts related to poor back or neck posture. The alerts are then packaged into a JSON object and sent back to the user's browser as a server-sent event. The JSON data includes the back_alert and neck_alert keys, each containing a Boolean value indicating whether there is currently an alert for poor posture in that area.

The function returns the HttpResponse object to the browser, which will keep the connection open to allow for future updates to be sent in real-time. Overall, this function allows the user to receive real-time updates about their posture while they are using the posture sitting app. The Notification model in the database is queried for changes of its fields 'back_alert' and 'neck_alert'. The fields are integer fields that increment by 1, if the camera on the Jetson Nano side captures the back or neck sustaining a poor posture for 10 seconds. The signal to increment these fields is then given through a Post request from the jetson Nano to the Django app endpoint. The 'sse' functions is shown in Figure 16.



```
@login_required
def sse(request):
    response = HttpResponse(content_type='text/event-stream')
    response['Cache-Control'] = 'no-cache'
    response['Connection'] = 'keep-alive'

    notifications = get_latest_notifications(request.user)
    data = json.dumps({'back_alert': notifications['back_alert'], 'neck_alert': notifications['neck_alert']})
    response.write(f"data: {data}\n\n")
    return response
```

Figure 16: A picture of the python code for the SSE

The system alert section under the front-end will tackle how JavaScript was implemented to render the alert on the user interface in real-time.

5.3 Front-End

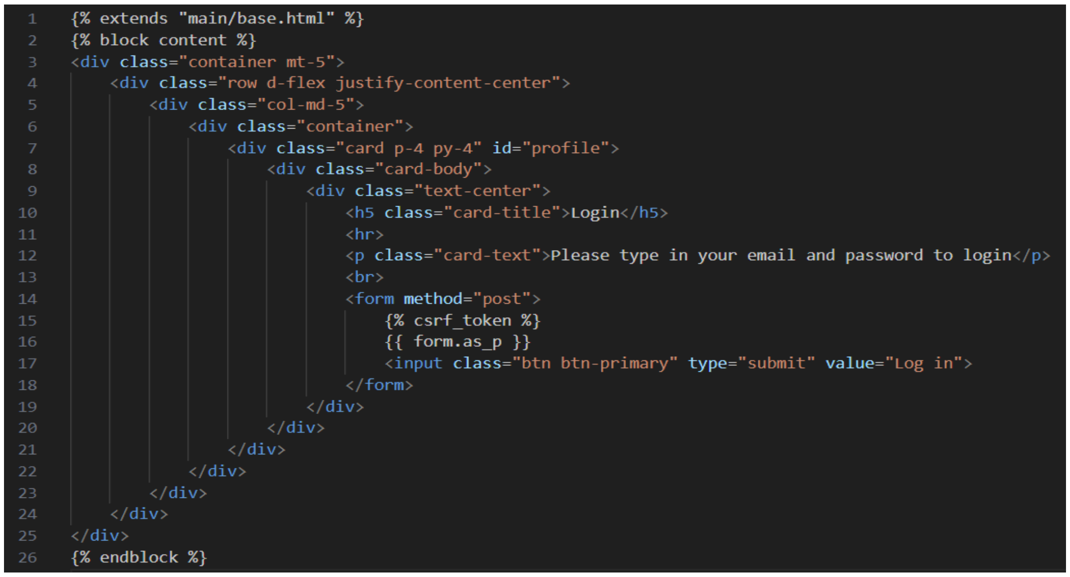
5.3.1 User Interface Development

The posture sitting app's user interface contains multiple distinct pages and features; it was created with bootstrap so that the app could be viewed whether on a phone or a computer, giving the user more flexibility and making the software more accessible. HTML was used to structure pages and CSS to style them.

HTML templates included forms for user authentication and profile picture upload, as well as displays for posture correction results. Some of these forms such as the login and registration form were written in python, and inserted in html using Jinja2. Python forms are classes used to handle user input and generate HTML forms. It is a functionality that Django provides as a convenient way to define and validate user input, which was used to develop the front-end of the app. These Forms were defined in the forms.py file which typically resides in the same application (main) as the views using the forms.

Jinja2 is a templating engine used in Django that allows the creation of reusable templates with placeholders to be filled in with specific data at runtime. The app took advantage of Jinja2 inheritance in Django templates by allowing the creation of a base template that can be extended by child templates. This means that the child templates can inherit the layout and structure of the base template while only overriding specific sections.

Moreover, Jinja2 allowed the connection of the back-end to the front-end of the app by the passing of variables from views to templates using context dictionaries. A context dictionary contains key-value pairs where the keys are variable names and the values are the data to be displayed (Python data). These variables can then be accessed in the template using the variable syntax, which is typically wrapped in double curly braces (e.g., {{ variable_name }}). This is illustrated in Figure 17.



```
{% extends "main/base.html" %}
{% block content %}
<div class="container mt-5">
    <div class="row d-flex justify-content-center">
        <div class="col-md-5">
            <div class="container">
                <div class="card p-4 py-4" id="profile">
                    <div class="card-body">
                        <div class="text-center">
                            <h5 class="card-title">Login</h5>
                            <hr>
                            <p class="card-text">Please type in your email and password to login</p>
                            <br>
                            <form method="post">
                                {% csrf_token %}
                                {{ form.as_p }}
                                <input class="btn btn-primary" type="submit" value="Log in">
                            </form>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
{% endblock %}
```

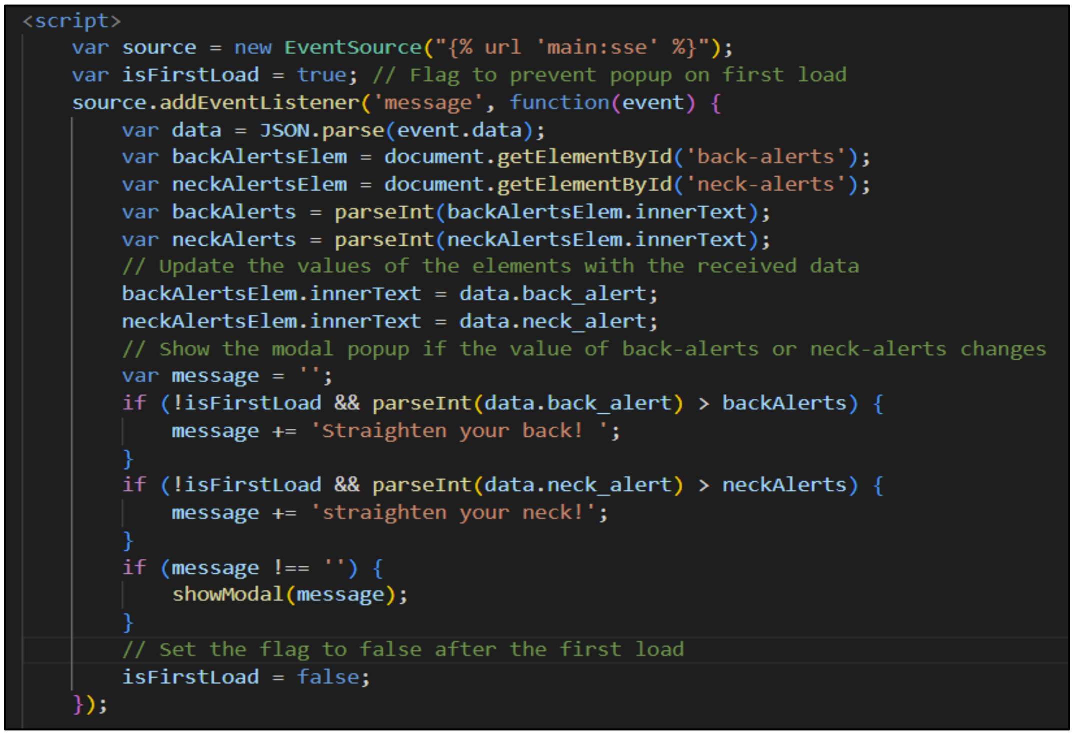
Figure 17: A picture of the code to insert the login form

The classes within the div tags are from bootstrap and allow the use of pre-coded elements, in an attempt to leverage time and quickly build the user interface.

On the other hand, CSS code defined the font style and size for different elements of the App. For instance, it sets the font-family of the body element to "Arial, sans-serif" and sets the font size for h2, h3, and p elements to different values. CSS was also used to position and align elements on the app, and define responsive design rules to ensure that the web app looks appealing on different screen sizes. Elements, such as blockquote, cite, and ul (unordered list) were specifically styled, to make them stand out and enhance the user experience. Interactive elements were created on the app such as the cta-button that has a hover effect that changes the background colour on mouseover, making it more interactive and engaging for users.

5.3.2 Real-time Alert System

The JavaScript code listens to a Server-Sent Event (SSE) stream that is set up in the Django view **'sse'** using the EventSource object (Figure 16).The SSE stream sends data in the form of JSON messages that contain the total number of back and neck alerts. When the SSE stream receives a message, the JavaScript code updates the values of the back-alerts and neck-alerts elements with the received data. It also checks if the new value of back or neck alerts is greater than the previous value and displays a modal popup on the UI with a message if it is. The modal displayed plays an audio notification and disappears after three seconds. The implementation of this feature is shown in Figure 18.



```
<script>
    var source = new EventSource("{% url 'main:sse' %}");
    var isFirstLoad = true; // Flag to prevent popup on first load
    source.addEventListener('message', function(event) {
        var data = JSON.parse(event.data);
        var backAlertsElem = document.getElementById('back-alerts');
        var neckAlertsElem = document.getElementById('neck-alerts');
        var backAlerts = parseInt(backAlertsElem.innerText);
        var neckAlerts = parseInt(neckAlertsElem.innerText);
        // Update the values of the elements with the received data
        backAlertsElem.innerText = data.back_alert;
        neckAlertsElem.innerText = data.neck_alert;
        // Show the modal popup if the value of back-alerts or neck-alerts changes
        var message = '';
        if (!isFirstLoad && parseInt(data.back_alert) > backAlerts) {
            message += 'Straighten your back! ';
        }
        if (!isFirstLoad && parseInt(data.neck_alert) > neckAlerts) {
            message += 'straighten your neck!';
        }
        if (message !== '') {
            showModal(message);
        }
        // Set the flag to false after the first load
        isFirstLoad = false;
    });
```

Figure 18: A picture of the JavaScript code for alerts

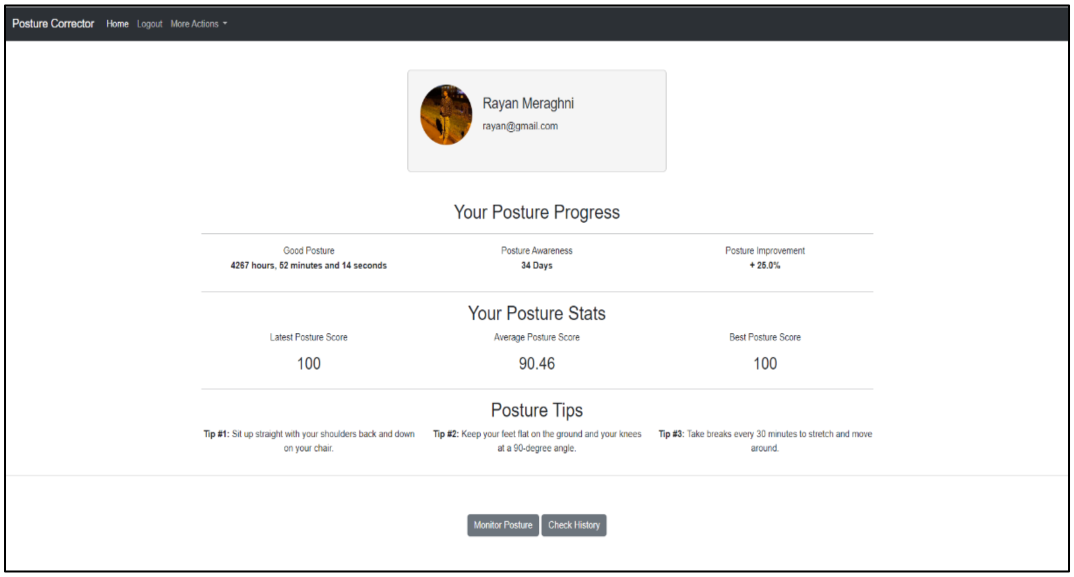
Together, the HTML and JavaScript code create a dynamic interface that allows the user to view their total number of back and neck alerts in real-time, and also provides alerts for any new alerts received. This improves the user experience by providing instant feedback on their posture and encouraging them to correct their posture before any discomfort or pain occurs.

5.3.3 App Pages

The welcome page is likely the first page that users will encounter, providing an introduction to the app and its features. Moreover, it has a feedback sections where users with an account can leave their thoughts and reflections on the app.

The sign up page and login page are important for user authentication, allowing users to create an account and access the app's features. Once registered or signed in, the navbar provides a navigation menu that allows users to easily access different features within the app.

The profile page (Figure 19) is where users can view their personal information, such as their name, email address, profile photo, and password. In addition, this page presents statistics of the users posture namely, the good time spent in a good posture, latest posture score, best posture score, and others.



Rayan Meraghni
rayan@gmail.com

Your Posture Progress

Good Posture	Posture Awareness	Posture Improvement
4267 hours, 52 minutes and 14 seconds	**34 Days**	**+ 25.0%**

Your Posture Stats

Latest Posture Score	Average Posture Score	Best Posture Score
100	90.46	100

Posture Tips

Tip #1: Sit up straight with your shoulders back and down on your chair.

Tip #2: Keep your feet flat on the ground and your knees at a 90-degree angle.

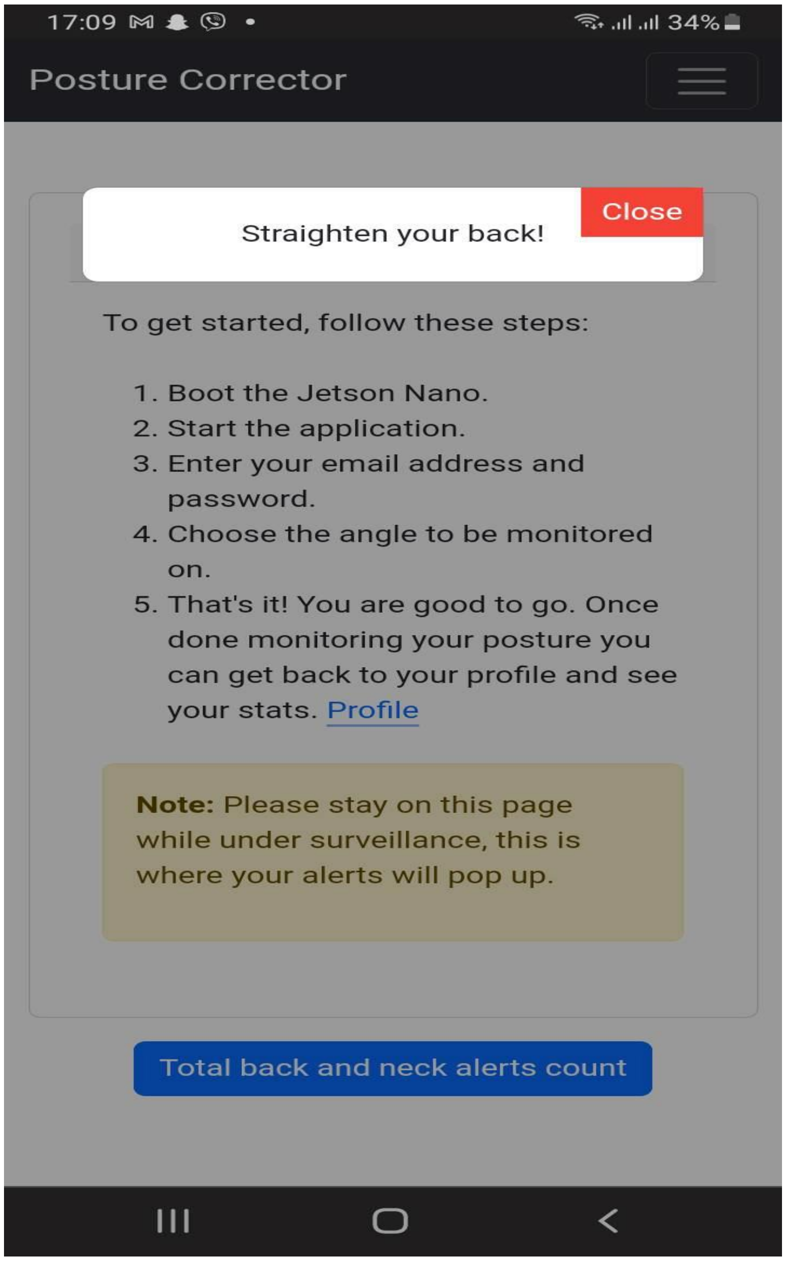
Tip #3: Take breaks every 30 minutes to stretch and move around.

Figure 19: A picture of the app's profile page on a PC

The feedback page allows users to provide feedback on the app, which can help to improve its functionality and user experience, while The posture photos page is where they can view images of themselves in poor postures, which would help them to identify areas for improvement.

The history page is where users can view their past posture monitoring sessions, including data on their posture score and the amount of time spent in different postures. User can also search for their video's history using a search bar that query the database history by date.

The monitor posture page is likely one of the key features of the app, allowing users to actively monitor their posture in real-time and receive alerts when they are in a poor postures for 10 seconds of time. Figure 20 illustrates the monitoring page of the app on a phone screen with the modal's alert for the back.


17:09 34%

Posture Corrector

Straighten your back!

Close

To get started, follow these steps:

1. Boot the Jetson Nano.
2. Start the application.
3. Enter your email address and password.
4. Choose the angle to be monitored on.
5. That's it! You are good to go. Once done monitoring your posture you can get back to your profile and see your stats. Profile

Note: Please stay on this page while under surveillance, this is where your alerts will pop up.

Total back and neck alerts count

Figure 20: A picture of the app's monitoring page on a phone

Chapter 6

Results & Discussion

6.1 Results & Discussion

The original aim and objectives initially set were met by developing an AI application that recognises unfavourable sitting postures. The compliance with the SCRUM methodology adopted allowed for a smooth and well-structured development that facilitated the implementation of an effective solution to the problem tackled in this project.

In order to enhance the user experience, the app was tested during each sprint throughout the course of the development by means of pre-defined unit tests to assess its functionality and user experience. The results indicated a correct implementation, as the app provided an intuitive interface and effectively alerted users when incorrect postures were detected. These tests are included in the appendix. Furthermore, the application's compliance with the original requirements specification was assessed. The authentication system, including sign-up, login, and password retrieval mechanisms, provided the necessary security measures for user accounts. The database effectively stored user information and historical posture data, enabling seamless retrieval and analysis. The functional requirements were successfully met, as the application incorporated the ML algorithm to differentiate between correct and incorrect postures, effectively alerting users when incorrect postures persisted for a defined duration. The intuitive interface allowed users to access and interpret their sitting posture statistics easily.

In terms of non-functional requirements, the application demonstrated minimal latency, ensuring real-time operation. It was compatible with both PC and mobile devices, enhancing its accessibility and usability. The access control mechanism restricted certain features to authenticated users, safeguarding user privacy and data. The simple and intuitive user interface positively contributed to the overall user experience, making the application easy to navigate and understand. Lastly, the provision of a feedback mechanism allowed users to share their thoughts and suggestions, facilitating continuous improvement.

Performance measures were also conducted to evaluate the app's efficiency and real-time capabilities. Latency tests demonstrated that the app operated within an acceptable range, ensuring timely posture detection and alert notifications. The integration of the moveNet model through the ONNX, and TensorRT optimisations played a crucial role in achieving the desired performance on the Jetson Nano, meeting the project's objective of real-time posture monitoring. Table 5 shows testing results of performance throughout the first Sprint of development. These measures have been inferred using python libraries (psutil, time) at each stage of the optimisation process.

Test Case	**Memory Usage (Megabytes)**	**Video Latency (seconds)**	**Inference time per Frame (seconds)**
moveNet model performance on Jetson Nano in TensorFlow-lite format.	328.11 MB	8.00 s	0.45 s
moveNet model performance on Jetson Nano in ONNX format.	98.06 MB	6.00 s	0.36 s
moveNet model performance on Jetson Nano in TensorRT format.	156.86 MB	0.5 s	0.043 s

Table 5: Performance Test

Chapter 7

Conclusion

7.1 Success & Limitations

The project has been successful in achieving its aims and objectives. The app is user-friendly and provides a convenient solution to monitor and improve posture, potentially reducing the risk of health issues arising for those likely to sustain poor sitting postures.

The app is functional, with features such as authentication, posture detection, alert system, and database integration, all of which meet the initial requirements specified. Furthermore, the app accessibility makes it go beyond its initial purpose as it could be of benefit to a various range of people besides office workers. Users aren’t constrained to use the app on computers as they can use it on phones, meaning that they can take advantage of its monitoring system to keep good posture habits in different environments, while seated.

Nevertheless, the system currently only supports four specific postures (upright, forward, reclined, and neck bent) and does not cover other types of postures, which limits its effectiveness for some users. From the frontal camera perspective, the reclined posture couldn’t be detected as in this scenario, no precise threshold could be found to distinguish this posture from an upright posture. Additionally, these postures could be solely identified from specific camera angles, meaning that failure to comply would likely result in an incorrect posture feedback. The thresholds established were concluded from a dataset collected on internet, whereas ideally, it would be preferable to set these thresholds based on criteria specified by experts who specialise in ergonomics.

7.2 Future work

Regarding future work, there are areas of improvement and expansion that can be pursued to enhance the functionality and usability of the posture monitoring app. Enabling SSH (Secure Shell) access to the Jetson Nano board through the app interface, would allow users to control the board remotely without having to physically access it. This would provide greater flexibility and convenience for users, as they could start and stop the posture monitoring process remotely from the app, rather than having to manually start it on the Jetson Nano.

Another potential area for future work is to expand the posture recognition capabilities of the app beyond the three postures currently supported. Specifically, the app could be expanded to recognise other postures from various camera angles using the same body parts detected by the current model. For instance, the app could be trained to recognise when a user is standing with poor posture, or when they are holding their phone in a other ways that strain their neck.

Finally, a more ambitious future project would be to train a CNN model from scratch as initially intended using a larger dataset of images to identify various postures and body positions from different angles and in complex environment. This would require a significant amount of data and computing power, but it would provide a more robust and comprehensive posture monitoring solution that could be applied in various contexts, including healthcare, and ergonomics.

References

Al-Attas, R., Yassine, A. and Shirmohammadi, S. (2012). Tele-medical applications in home-based health care. In: 2012 IEEE international conference on multimedia and expo workshops. July. IEEE. 441-446. Available from: https://ieeexplore.ieee.org/abstract/document/6266424 [accessed 01 January 2023]

Khan, T. (2021). An intelligent baby monitor with automatic sleeping posture detection and notification. AI, 2(2), 290-306.

Adam, M.I., Ramachandran, P. and Alex, Z.C. (2021). November. Human Irregularity Detection Based on Posture and Behavioral Analysis. In 2021 Innovations in Power and Advanced Computing Technologies (i-PACT) 1-6. IEEE. Available from: https://ieeexplore.ieee.org/abstract/document/9697052 [accessed 15 January 2023]

Yang, L., Lu, X., Yan, B. and Huang, Y. (2020). Prevalence of incorrect posture among children and adolescents: Finding from a large population-based study in China. Iscience, 23(5), p.101043. Available from: https://www.sciencedirect.com/science/article/pii/S2589004220302285 [accessed 17 January 2023].

Bontrup, C., Taylor, W.R., Fliesser, M., Visscher, R., Green, T., Wippert, P.M. and Zemp, R. (2019). Low back pain and its relationship with sitting behaviour among sedentary office workers. Applied ergonomics, 81, p.102894. Available from: https://www.sciencedirect.com/science/article/pii/S0003687019301279 [accessed 20 January 2023].

Zemp, R., Taylor, W.R. and Lorenzetti, S. (2013). In vivo spinal posture during upright and reclined sitting in an office chair. BioMed research international, 2013. Available from: https://www.hindawi.com/journals/bmri/2013/916045/ [accessed 21 January 2023].

Waongenngarm, P., Rajaratnam, B.S. and Janwantanakul, P. (2015). Perceived body discomfort and trunk muscle activity in three prolonged sitting postures. Journal of physical therapy science, 27(7), 2183-2187. Available from: https://www.jstage.jst.go.jp/article/jpts/27/7/27_jpts-2015-165/_article/-char/ja/ [accessed 23 January 2023].

Ran, X., Wang, C., Xiao, Y., Gao, X., Zhu, Z. and Chen, B. (2021). A portable sitting posture monitoring system based on a pressure sensor array and machine learning. Sensors and Actuators A: Physical, 331, p.112900.

Paliyawan, P., Nukoolkit, C. and Mongkolnam, P. (2014). Prolonged sitting detection for office workers syndrome prevention using kinect. In 2014 11th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON). May. 1-6. IEEE. Available from: https://ieeexplore.ieee.org/abstract/document/6839785 [accessed 25 January 2023].

Piñero-Fuentes, E., Canas-Moreno, S., Rios-Navarro, A., Domínguez-Morales, M., Sevillano, J.L. and Linares-Barranco, A. (2021). A deep-learning based posture detection system for preventing telework-related musculoskeletal disorders. Sensors, 21(15), p.5236.

Chaffin, D.B., Andersson, G.B. and Martin, B.J. (2006). Occupational biomechanics. John wiley & sons.

Woodson, W.E.; Tillman, B.; Tillman, P. (1992). Human Factors Design Handbook: Information and Guidelines for the Design of Systems, Facilities, Equipment, and Products for Human Use. New York, NY, USA: McGraw-Hill.

Kale, H., Mandke, P., Mahajan, H. and Deshpande, V. (2018). Human posture recognition using artificial neural networks. In 2018 IEEE 8th International Advance Computing Conference. December. IACC. 272-278. IEEE. Available from: https://ieeexplore.ieee.org/abstract/document/8692143 [accessed 25 January 2023].

Kurniawan, A. and Kurniawan, A. (2021). Introduction to nvidia jetson nano. *IoT Projects with NVIDIA Jetson Nano: AI-Enabled Internet of Things Projects for Beginners*.

Appendix

8.1 Unit Tests

Sprint 1 - Unit Testing			
Test Case	**Test Execution**	**Expected Result**	**Pass/Fail**
The app should be able to run on the jetson nano using the CNN model optimised in performance tests.	Loading The CNN model in a script and running inference on the Jetson Nano.	The CNN model should run without any issues on the Jetson Nano.	Pass
The app should let the user start a video and end it.	Booting Jetson Nano and starting the app through the terminal then clicking the key ‘q’ to end the video.	A screen showing the camera perspective should pop up and be destroyed as the key ‘q’ is pressed.	Pass
The app should give the user 3 camera perspectives to be monitored from.	Starting the app and trying all possible camera perspective options.	The app should prompt to choose 3 options: - frontal camera - lateral right camera - lateral left camera	Pass
The app should show body joints on the screen and draw	Starting the app choosing all camera perspectives and checking the screen	A screen should pop up showing the body key joints within the	Pass

connections between them.	popup for body joints and segments.	frame and draw connections.	

Sprint 2 - Unit Testing			
Test Case	**Test Execution**	**Expected Result**	**Pass/Fail**
The app should detect the users back posture based on their hip angle. This should be from a lateral right and left camera perspective.	Starting the app and choosing lateral right camera perspective. Placing camera on the right side, and sitting on a chair, at 90 degrees of the camera by side. Moving my back to check the posture detected by the app. Same process followed for the left camera perspective.	90° < RHA < 115° -Back Upright	Pass
		RHA < 90° : -Back Forward	
		RHA > 115° : -Back Reclined	
The app should detect the users neck posture based on The distance between the shoulders and nose. This should be from a lateral right and left camera perspective.	Starting the app and choosing lateral right camera perspective. Placing camera on the right side, and sitting on a chair, at 90 degrees of the camera by side. Moving my neck to check the posture	Distance > 8 pixels -Neck Upright	Pass

	detected by the app. Same process followed for the left camera perspective.	Distance < 8 pixels -Neck Bent	
The app should detect the users back posture based on the distance between the nose and shoulders compared to the distance between shoulders and hips . This should be from a Frontal camera perspective.	Starting the app and choosing lateral right camera perspective. Placing camera on the right side, and sitting on a chair, at 90 degrees of the camera by side. Moving my back to check the posture detected by the app.	DSH < DNH: -Back Upright	Pass
		DSH > DNH: -Back Forward	
The app should detect the users neck posture based on the interior angles of the right and left shoulders. This should be from a Frontal camera perspective.	Starting the app and choosing lateral right camera perspective. Placing camera on the right side, and sitting on a chair, at 90 degrees of the camera by side. Moving my neck to check the posture detected by the app.	35° < RSA 35° < LSA -Neck Upright	Pass
		35° > RSA 35° > LSA -Neck Forward	

Sprint 3 - Unit Testing			
Test Case	**Test Execution**	**Expected Result**	**Pass/Fail**
The app should let the user create an account	Creating an account	Account creation	Pass
The app should let the user logout the account created	Logging out.	Logging out	Pass
The app should let the user login his/her existing account.	Login into an existing account	Accessing account	Pass
The app should let users upload a profile photo.	Creating an account with a profile photo, and checking the profile page to see if it was uploaded.	Finding the profile photo uploaded	Pass
The app should display incorrect posture photos captured throughout a video	Starting a video and sustaining an incorrect posture for 10 seconds. Checking if the incorrect photos page displays the incorrect posture	Finding the incorrect posture photo.	Pass

The app should display all posture data captured throughout a monitoring video.	Starting a video and doing different activities while being monitored. Stopping the video and checking the history page.	Finding all posture relevant data.	Pass
The app should let the user post a feedback	Posting a feedback on the app with an existing account, and checking the welcome page to see if it was posted	Finding feedback on the welcome page.	Pass

Sprint 4 - Unit Testing			
Test Case	**Test Execution**	**Expected Result**	**Pass/Fail**
The app should trigger a real-time notification alert in the form of a modal that lasts for 3 seconds	Starting a monitoring video and sustaining an incorrect posture for 10 seconds, while being connected on the app	Alert modal coming up on the screen of the monitoring page in real-time	Pass
The app should play a notification audio in real-time upon the alert modal display.	Starting a monitoring video and sustaining an incorrect posture for 10 seconds, while being connected on the app	Alert modal playing an audio coming up on the screen of the monitoring page in real-time	Pass
The app should be accessible on phones	Connecting to the app on phone and checking if all features work	Functioning of all features	Pass
The app should be well-structured, intuitive, and visually appealing on phones.	Connecting to the app with a phone and navigating through all pages of the app.	Visually appealing interface with clear instruction.	Pass
The app should be well-structured, intuitive, and visually appealing on PCs.	Connecting to the app with a PC and navigating through all pages of the app.	Visually appealing interface with clear instruction.	Pass

The app should generate different statistics on the profile page	Starting a monitoring video as sat on a chair. Stopping the video and repeating the process five times to check for the posture statistics consistency. Navigating to profile page and checking statistics	Consistent statistics on the profile page.	Pass