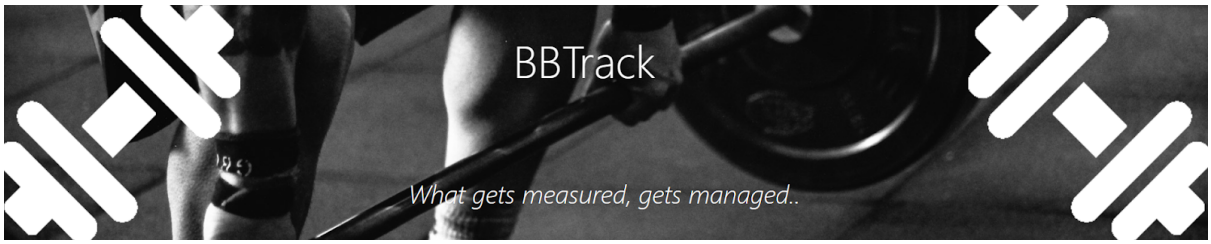
Technical Manual for Barbell Track



Date	06/05/2021
Jordan Voss	17327513
Nathan Ndombasi	13517227

Table of Contents

1. Motivation

As a member of the DCU powerlifting club, and other powerlifting clubs outside of DCU, we felt that there was not enough data available to athletes and coaches to be able to optimise training and get the best results from it. Our project will help with personal trainers and clients alike involved in strength sports such as Powerlifting/Olympic Lifting, field sports such as Rugby or GAA, and will also be useful for those who train using the gym as a hobby. We feel it will be useful especially for a coach or personal trainer who wants to understand more about the gym sessions their client has, as well as seeing what factors influence the session of the individual, e.g. Sleep, Nutrition, Hydration, Mood. Our Project is an app for tracking many factors that come into play when we decide to take the leap into the world of the gym, in particular weights. From bar speed, velocity and path, to other factors outside of the gym such as quality of sleep, nutrition and hydration on the day of/before training day.

The term known commonly as RPE (Rate of Perceived Exhaustion) which essentially rates the difficulty/10 of the set just completed by an athlete (How many more reps could I have completed? etc.). Many sports, but, primarily in Powerlifting and Olympic Lifting, use these techniques in order to keep training at a certain intensity for the individual for the quickest and most efficient ways of gaining strength. This RPE is very individual, so, starting off with a default speed and velocity we will determine an RPE of a set for the individual. As time goes on however the app will take input from the individual, as a set may move as slow as an RPE 9/10 (Implying the athlete thinks they could only have completed 1 more rep), but in terms of how it felt for the athlete, they may have felt there was more 'in the tank' so perhaps it was an RPE 8 for them. By examining the trends in their lifts, and their own input on how the set felt, we can use this date to extract some customised RPE - speed mappings. By default an average athlete will have a bar speed of 0.7m/s when a set is an RPE 8, however thanks to inputs from the user, we know that generally the speed of the bar when they lift at RPE 8 is actually 0.6m/s. This allows for an individualised training approach which is becoming more and more necessary in the industry as there is no 'one size fits all' in terms of programming or RPE.

The path of the bar is another factor that we will take into account when considering RPE and form checking. Perhaps if an athlete's weight shifts forward and they begin to ascend while squatting, we can see as we draw the bar path for the individual to see where they are going wrong, as the bar in a squat should move in a straight line (however this differs from lift to lift) We can tell the athlete, perhaps this was an RPE 9 today, although you hit this weight 2 weeks ago at RPE 8, but here are some potential reasons for that. This can be extremely useful for both veterans of the sports and novices or people only pursuing them as a hobby.

2. Research

When researching the project we realised that there were not many applications like this available to athletes and coaches in the powerlifting industry. There were a few that tracked the speed of the bar, but none that calculated RPE from that statistic and created custom values, and even fewer that would track the body parts and give information based on the body movement of the user.

We decided to use python as our main language to implement the app, as it seemed to be the most efficient way of creating an app like this. We also used Flask to make the app creation process more seamless. We used a postgresql database and hosted on heroku. Which we found out at the latter stages of the project, using the free version of heroku would require more space as the project size is quite large with model files and more. We also used Amazon S3 Buckets to store our videos as it appeared to be the best possible way to store videos to avoid large project size and using the heroku ephemeral filesystem, the videos would be deleted after every reload or update of the app. Using s3 we can store the videos easily and keep them upon update or reload.

With further research we narrowed down our options of ways to implement both the body and barbell tracking. We settled on using a piece of coloured paper on the side of the barbell for tracking the bar. Using this, and knowing the size of the bar we could determine the pixels per millimetre and time taken to move x amount of pixels to determine the speed of the barbell.

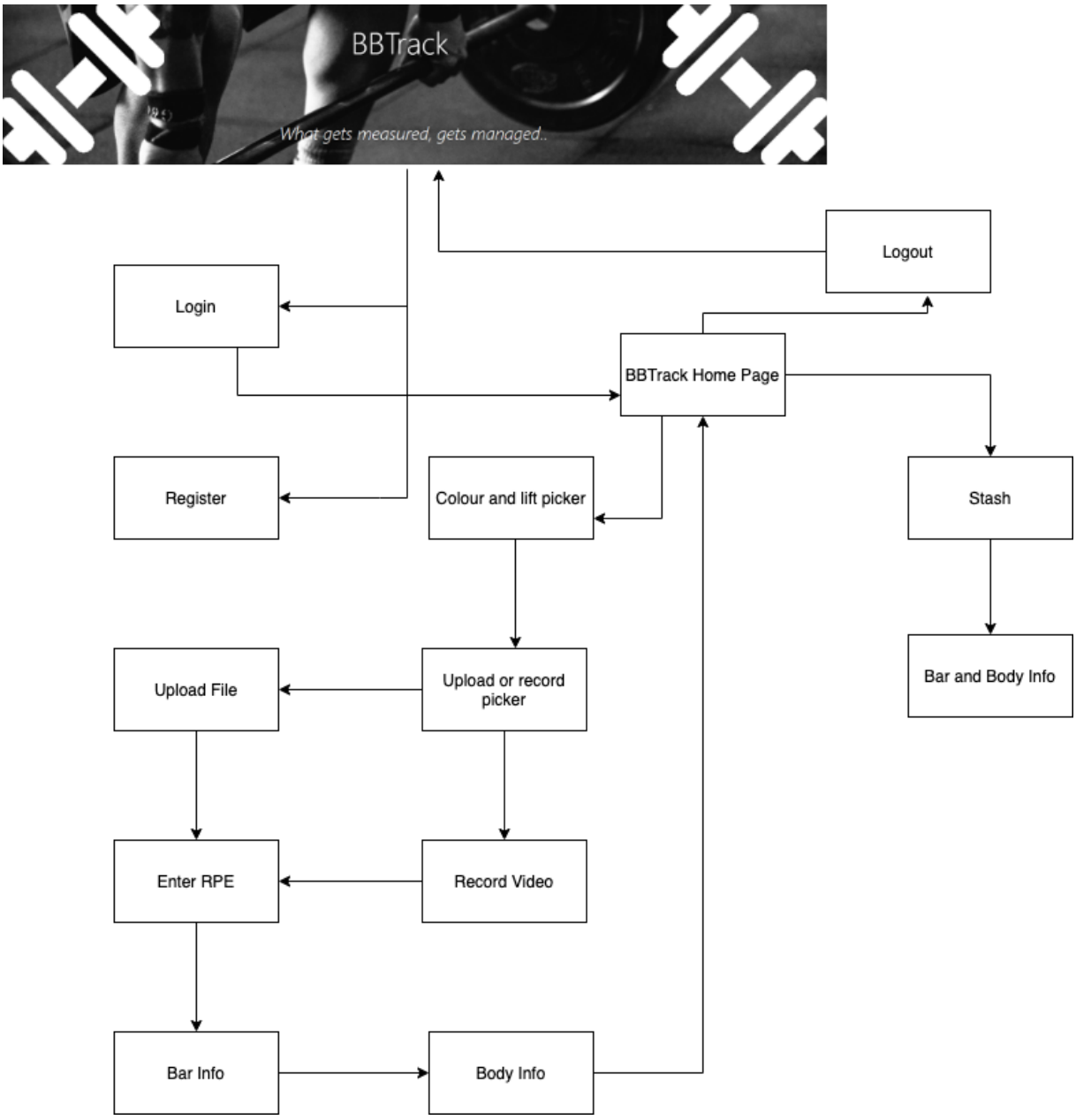
We also Found the best way of tracking the body parts, using the same technology, OpenCV as the barbell tracking we could read the dataset into our network, and process from there. Creating a list of points and from there analysing the position and angles between points.

In hindsight, we would have used another service such as amazon's elastic beanstalk to host the app, as it is much more scalable.

To implement the custom RPE feature, we needed to research more about regression as we needed to calculate the line of best fit between the speed and RPE values. Upon further research we decided to use Matplotlib, a plotting library used in Python, so that we can visualise the line of best fit. For the regression, we used sklearn, a tool that is used in predictive analytics. This was the most concise way of calculating the linear regression. For us to implement the linear regression, we must first create a polynomial feature. We did this by creating a pipeline combing the PolyNomialFeatures and LinearRegression. This allowed us to make a model and fit the list we recently created (p) into it. We predict and receive the line of best fit with the model.predict.

3. Design

3.1. High Level Application Design



This diagram demonstrates the high-level design of the app. When the users go onto the app, they are shown the home page with the options to login or create an account. They can then go to the home page and have the options to go to the lift and colour picker page, stash page or logout.

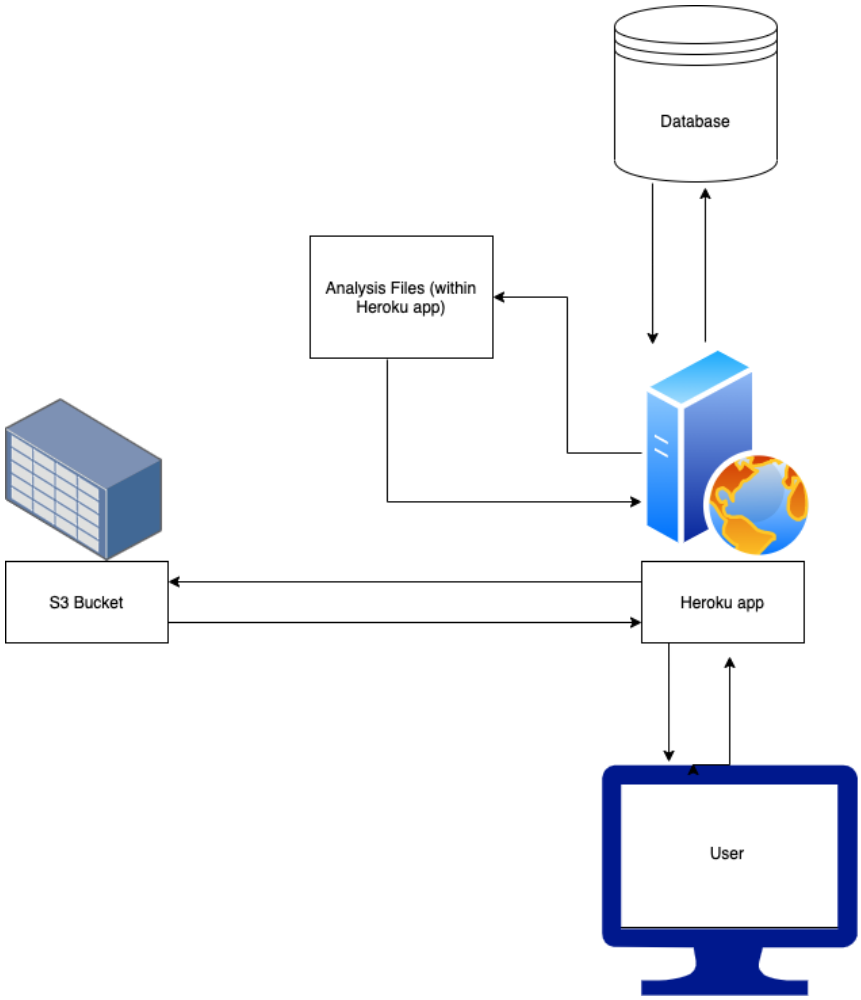
From the lift and colour picker page they then choose whether to upload or record their lift. If they upload, they can upload the file, otherwise they can record a video on the app. One they have uploaded or recorded they can see the page showing the bar tracking. They can give their own estimation as to what RPE they found the set to be and they go to the Bar info page. Here they can see their own customised rpe chart. As well as the video showing the bar speed and path, with the rpe and speed shown on the screen too. Once they are finished with looking at the bar information they can select to go to the body tracking page. Here they are shown the video with the body

tracking information to the side, detailing how they have scored and ways to improve if they didn't score for a section.

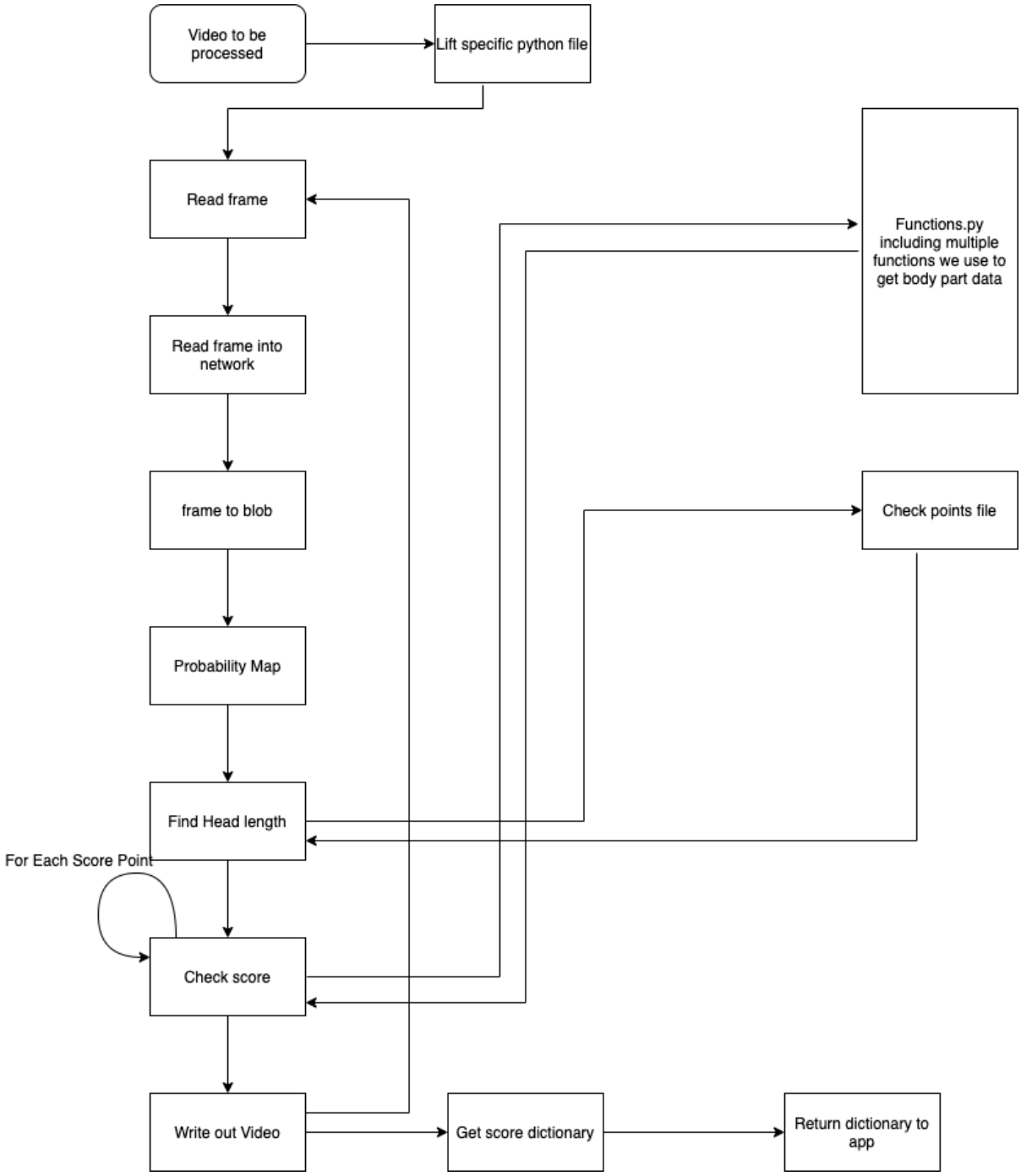
If the user chooses to go to the stash page from the home page they will be able to see a list of all of the videos they have had processed on the app. When they click a video they will be brought to a screen showing them the two videos, bar tracking and body tracking, as well as all of the information they received when they recorded their video the last time.

If they choose to log out, they will be brought back to the original home page.

3.2. High Level Architecture of all Components

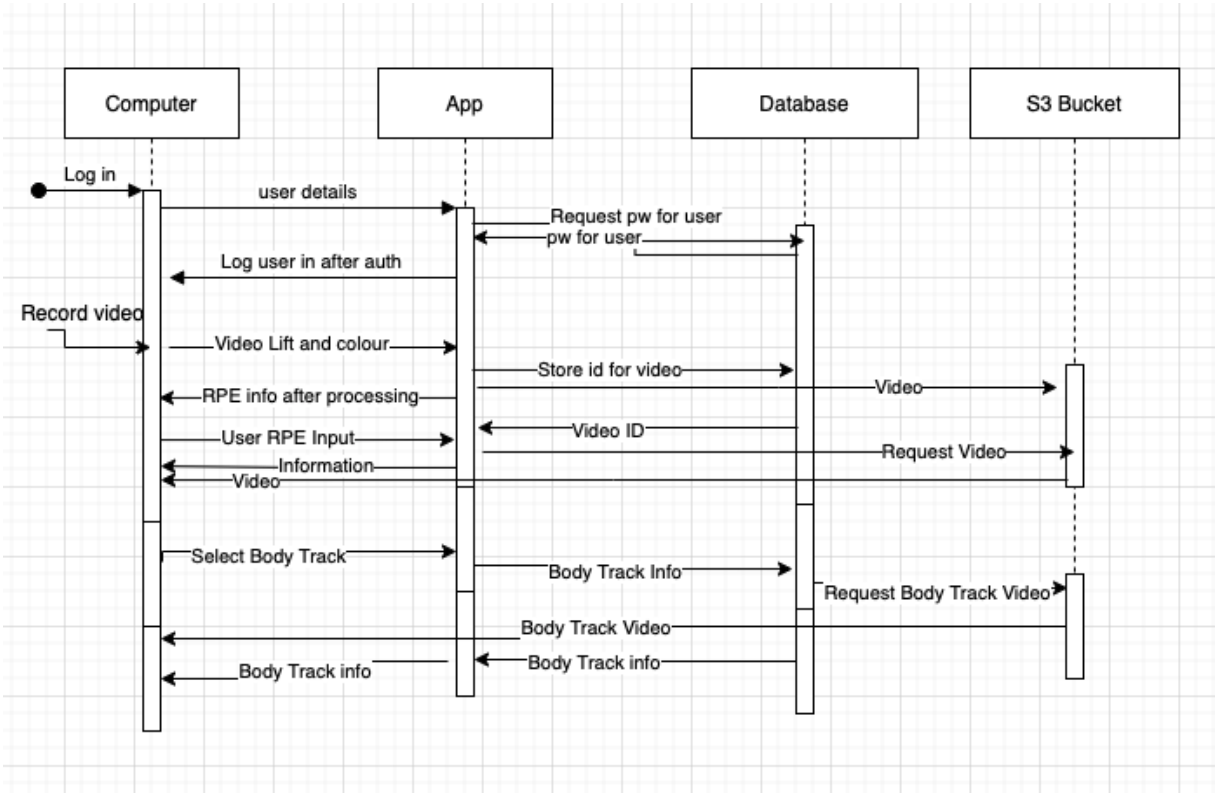


3.3. Body Tracking Analysis Architecture



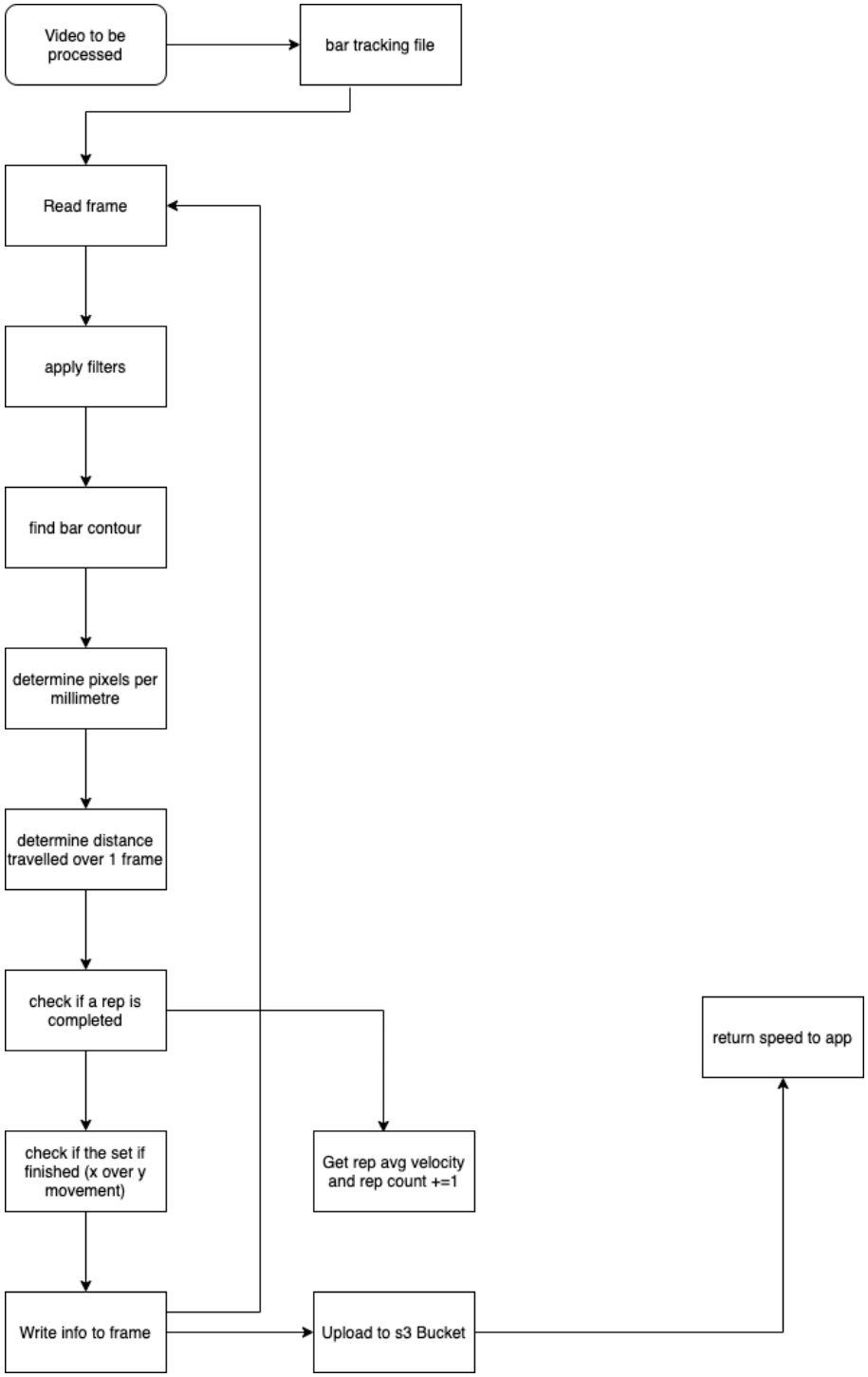
We start with the app sending a video url to the python file specific to whatever lift we are executing. We read the frame, then read it into the network. After this we turn the frame into a blob and create our list of points for the frame using the probability map. We then find the head and forearm lengths as we will use these to predict other body parts if necessary using the check points file. Then for each scoring criteria, we check if the user is passing or failing that portion in this frame. Once we have done this, using our functions file, for each scorable element, we write the frame out to the video file and repeat this process until all frames are captured when we then create the score dictionary and return it to the rest of the app.

3.4. Log in and Record a Video Sequence Diagram



Here we can see the process of logging in and recording a video and afterwards seeing the information. The user logs in and sends the user details to the app. These details are cross referenced with the user's password to authenticate them. They are then shown the logged in screen. When the user records the video they send the lift colour and video to the the app where the id for the video is stored and the processed video is sent to the s3 bucket.The bucket then sends back the video along with the information, RPE etc. to the user. They can then input their RPE, and see more detail concerning their rpe chart, speed and bar path. When the user selects body track, we store the body track information in the database and request the video from the S3 Bucket. All the information is then sent to the users computer and shown on their screen.

3.5. Bar Tracking Analysis Architecture



We start by reading in the video to the bar tracking file, we read the frame and apply the necessary filters. We then find the bar contours given the colours specified by the user. We can now determine the pxpmm (Pixels per Millimetre) and determine the distance travelled over one frame. We can now check if a rep is completed and if it is we can get the average velocity for the rep and add one to the rep count.

Then we check if the set is finished and write the frame to the output video. Then we either repeat for the next frame or if we are finished we upload the video to the s3 bucket and return the bar speed to the app.

4. Implementation

To implement the app we used Flask to create a python flask app, with different routes and templates to portray the information we needed on them.

4.1. User Management

To manage the users we needed to create secure registering and logging in for the app. We did this using flask_login to create sessions for the users and remember them if they selected to do so. We used our user model to create database tables containing user ids, usernames, and hashed passwords, so they weren't visible to anyone.

4.2. Video Recording

We needed to use javascript to record the video and send over the blob via a POST request to the flask app. We added event listeners for each button to start, stop and submit the video and had 2 video elements on the page, one showing the video and another recording, only one of which would be visible to the user at a time. We used ajax to send the blob to the flask app.

4.3. Barbell Tracking

In order to track the barbell ,the users could navigate to the capture video pages, which would allow them to select what lift they would be doing and what colour the piece of paper on the bar would be. From there the information is passed on via HTTP requests to the upload or record section where the user can upload their own video or record one from the app.

When the user has uploaded or recorded their video we upload it to our s3 bucket under username/original, and run the barbell tracking on this video. This is done by filtering the image using multiple masks and colour changes to make the edges of the barbell and piece of paper to stand out, then finding the contours where the colour is within certain ranges (depending on what colour the user has selected) from there we determine the center of the barbell and the pixels per millimetre ratio to gauge the speed. From here we keep track of where the center of the barbell is for the last 10 frames and call our function to determine the velocity in the y direction. We get the average velocity for each rep and append them to a list of average velocities, the last of which will be used to determine the RPE of the set. We also check for more x movement over y, which would indicate that the set has been completed and to stop tracking reps, as well as a threshold that the bar must move in order to start counting the reps. We use the check_for_reps function to determine if a rep has been completed. Here we check for points of inflection and once we hit the second point of inflection, we know that a rep has been completed. I.e. the user has gone down and up (or up then down) with the bar. After each frame we write out to the file encoded to support the mp4 format.

The user, once the barbell tracking has been completed, is shown a screen with the video of the barbell tracking. They are also provided with a text box in which they can enter the RPE that they perceived the set to be. This will help us when we go onto RPE customisation, as RPE is a subjective measurement, we can take input from the users to change their personal PRE values to a speed or 1.3m/s will no longer correlate to an RPE 7 but to an RPE 8 etc.

Once they input their RPE we customise their values if necessary, and show them the same video, but with their custom RPE chart below. There is also a button that will bring them to the body tracking portion of the app.

4.4. Body Tracking

When the user clicks to see the body tracking portion of their lift, they are brought to a screen showing a video with all of the major key points of the body drawn on. And to the right of this there is information detailing the things that went right or wrong in the lift.

In order to achieve the body tracking portion of the app, we first needed to know what lift we would be tracking, as we would look for different things for each lift. This helped us to save time, and without the time constraints we would have liked to implement some exercise recognition so it would no longer be necessary for the user to specify what lift they would be recording, making the user experience better flowing and easier for the users.

When we take in the video from our s3 bucket. There are some things we need to do before we can begin to process the video. First, we read the datasets into our network and use the OpenCV blobFromImage method to turn the frame we will process into a BLOB (Binary Large OBject) for processing. We then use the same masks and filters on the image as in the barbell tracking, as we will need to find the barbell again, incase we cannot find some points (like the wrists which would be covered by the bar in some circumstances)

We then create a probability map for each of the key points we need. These were put into a list in a particular order so we could reference them using this dictionary:

```
POSE_NAMES = {
    "HEAD": 0,
    "NECK": 1,
    "RSHOULDER": 2,
    "RELBOW": 3,
    "RHAND": 4,
    "LSHOULDER": 5,
    "LELBOW": 6,
    "LHAND": 7,
    "RHIP": 8,
    "RKNEE": 9,
    "RANKLE": 10,
    "LHIP": 11,
    "LKNEE": 12,
    "LANKLE": 13,
    "CHEST": 14
    }
```

We could very easily find the point of any body part using this.

The probability map only put the items into the list if they were above the probability threshold that we set, otherwise the element of the list was put in as None. We created a function show_heatmap to show a heatmap portraying the point we wanted and the point of highest probability that this body part was located. We used this for our accuracy testing when comparing our videos with the barbell blocking some parts, and videos without the barbell getting the PCK (Percentage of Correct Keypoints).

Now that we have found most if not all of the points, we needed to make some assumptions so we could track the body. Sometimes we might not have enough key points to be able to complete the processing accurately enough to give real, genuine feedback to the user. We knew however,

that the human body is built with certain ratios. For example the length of the forearm is equal to the length of the foot. Or the length of the foot is 85% of the length of the head. Using these assumptions we could get foot size, which would be relevant to ensure the bar stayed over the mid foot of the athlete, and many other parts. We also assumed that if a certain body part couldn't be tracked if it was one of the shoulders, elbows, wrists, ankles, knees, or hips, that it would be roughly equal to the same body part on the other side. As we were recording these videos from a side angle, it was common that we could get only the left side body parts. So if this was the case for any part that occurred on both sides of the body, then we assumed it was equal to the other and just directly behind.

The same is true if we couldn't find hands or ankles to get the distance between them for the bar position. If we had one knee, this meant we could find the second knee, as well as the two ankles. We did this by determining the shin length, which was equal to the head length * 2. And knowing that the ankles were directly below the knees, we could give an estimation as to where the ankles would be. For the hands, if we couldn't find any, we would use the barbell point as the mid point between them

Now that we were able to get all of the points we needed, we could begin to process the video. Each lift would be scored out of 3, for each element we looked for that was performed correctly, the athlete would get a point, obviously aiming to get all 3/3!

The elements we scored on were:

Squat:

Bar Positioned over Midfoot - we want to make sure the bar stayed roughly over the mid foot (within a threshold) throughout the lift. This provides better leverages to move the weight, and helps to decrease risk of injury.

Knees and Hips extending at the same rate - In order for the athletes centre of gravity to remain in the right place when holding a bar on their back, it's important that the knees don't extend quicker than the hips, or vice versa, this could cause the athlete to fall forward or backward, which can be very dangerous when using heavier weight.

The Athlete performs the squat with correct depth - in order to get the most out of the lift, the athlete needs to make sure their hips go below their knees. This helps to strengthen the muscles in all stages of contraction and provides stability when using heavier weight.

Bench Press:

The Athlete's hips stay on the bench - we want to make sure the athletes hips don't come off the bench when performing the lift. Failure to keep them on the bench can result in lack of control and dropping the bar onto their body.

The Athlete's feet don't leave the floor - In order to keep tension in the body, and control of the weight, the athlete must keep their feet on the floor, failure to do so may again cause the bar to fall on top of them.

The bar doesn't go past the chest - If the athlete goes too low with the bar, they won't be able to get it back up, and will have the weight lying on their chest, for the people benching high numbers, this is not ideal.

Deadlift:

Bar Positioned over Midfoot - we want to make sure the bar stayed roughly over the mid foot (within a threshold) throughout the lift. This provides better leverages to move the weight, and helps to decrease risk of injury.

Knees and Hips extending at the same rate - In order for the athletes centre of gravity to remain in the right place when holding a bar on their back, it's important that the knees don't extend quicker than the hips, or vice versa, this could cause the athlete to fall forward or backward, which can be very dangerous when using heavier weight.

The Athlete's back is straight - When deadlifting, the most important thing is to keep your back straight, many people have injured themselves when lifting with a back that is too rounded or too arched. This is one of the most common gym injuries.

Implementation of the above criteria:

Bar Positioned over Midfoot

To ensure the bar was over the mid foot, we used the foot size from the ankle and made sure that the bar didn't deviate from the x value being a certain threshold above or below the x value of the middle of the foot.

Knees and Hips extending at the same rate

To make sure the knees and hips extended at the same rate we needed to keep track of all of the previous angles at the knees and hips. We then got the absolute value of the difference between the i^{th} and $i+1^{th}$ elements and compared them to make sure the difference was similar.

The Athlete performs the squat with correct depth

To ensure the athlete hits depth. We just needed to check that the y value of the hips was less that the y value of the knees at at least one point in time during the lift.

The Athlete's hips stay on the bench

To ensure the Athlete's hips stayed on the bench, we took the first hips point as the bench point. We then needed to make sure that the hips stayed within the y values threshold to make sure they didn't raise off the bench.

The Athlete's feet don't leave the floor

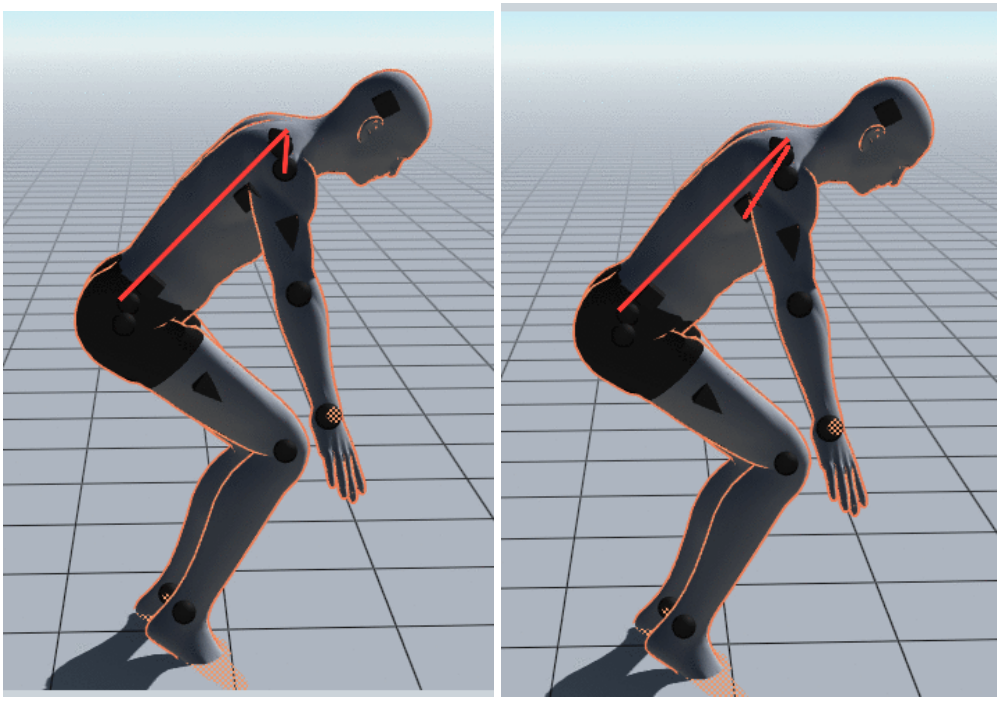
To implement making sure the athlete's feet don't leave the floor we did something very similar to the hips leaving the bench. We needed to get the feet point, that was the first instance of the feet, and ensure they didn't rise by cross referencing each of the y values.

The bar doesn't go past the chest

We already have a chest point. So we just needed to make sure the x value of the bar didn't exceed the x value of the chest

The Athlete's back is straight

To make sure the athlete's back stayed straight we had to get multiple angles. We drew the line between the mid point of the hips and the neck. This would act as our spine. We then needed to get the angle at the shoulder, and the angle at the chest, and get the maximum of these angles as the back angle for this frame. The angles we are getting are shown in the figures below.



We added these angles to a list, and got the average back angle over the entire lift to see if the athlete's back was arched or rounded too far.

4.5. Data

When the user is looking at the body tracking page they can see lots of information as to what has gone wrong with their lift. This was implemented by having nested dictionaries for each score with the tips for each one inside. You can see the example in sample code. We were passed a dictionary from the body tracking function that tells us which parts the athlete earned points in, and which points they didn't, allowing us to dynamically change the content of the page depending on the scores they received. This was stored in the database along with an infoid. This was unique to each set of information and we used this when looking at the stash to determine what information we needed to show when they looked back on past videos. The name of the video stored in the S3 bucket was "username/body/Squat-infoid.mp4" This allowed us to parse the link for the video to get the infoid and use this to get what elements were scored on from the database for the user. All data needed was also stored in the database using similar models to the user.py file in section 5.1.

4.6. RPE Customisation

We receive the RPE values and speed after the user records their video. In order to customise the rpe we take into consideration the 2 previous lines of best fit (which are set as default from the beginning) and use regression to find relationship and line of best fit between the three sets containing speeds and use this as the newest RPE scale. Having 2 previous sets of data helps us to eliminate an outlier, e.g. if a set is perceived an RPE 9 by us but an RPE 6 by the user, there won't be a huge jump in data unless the user consistently indicated that their RPE 6 speed is the same as our default for RPE 9.

5. Sample Code

5.1. User.py - showing the class implemented to store the user details in the database.

```python
from ..wsgi import db
from flask_login import UserMixin
from werkzeug.security import generate_password_hash, check_password_hash


You, a month ago | 2 authors (You and others)
class User(UserMixin, db.Model):

    __tablename__ = "account"

    uid = db.Column(db.Integer, primary_key=True)
    f_name = db.Column(db.String(255))
    l_name = db.Column(db.String(255))
    email = db.Column(db.String(255))
    password = db.Column(db.String(255))
    username = db.Column(db.String(255))
    # created_on = db.Column(db.String(255))

    def __init__(self, f_name, l_name, email, password, username):
        self.f_name = f_name
        self.l_name = l_name
        self.email = email
        self.password = generate_password_hash(password)
        self.username = username
        # self.created_on = created_on

    def set_password(self, password):
        self.password = generate_password_hash(
            password,
            method="sha256"
        )

    def check_password(self, password):
        return check_password_hash(self.password, password)

    def get_id(self):       You, 4 months ago • finished login and added logout
        return self.uid

    def is_authenticated(self):
        return self.authenticated

    def __repr__(self):
        return f"<User {self.username}>"
```

5.2. user_authenticate.py - to allow the user to login and register

```python
@auth_bp.route("/register", methods=["GET", "POST"])
def register():
    form = RegisterForm()
    if form.validate_on_submit():
        try:
            new_user = User(f_name=form.f_name.data, l_name=form.l_name.data, email=form.email.data, password=form.password.data, username=form.username.data)
            db.session.add(new_user)
            db.session.commit()
            id = new_user.get_id()
            default = "2.5, 2.25, 2, 1.75, 1.5, 1.3, 1, 0.75, 0.5, 0.3"
            squat = Squat(userid=id, version0=default, version1=default, version2=default)
            bench = Bench(userid=id, version0=default, version1=default, version2=default)
            deadlift = DeadLift(userid=id, version0=default, version1=default, version2=default)
            db.session.add(squat)
            db.session.add(bench)
            db.session.add(deadlift)
            db.session.commit()
            flash("Created User {} {} with username {}".format(form.f_name.data, form.l_name.data, form.username.data))
            return redirect("/login")
        except exc.IntegrityError:
            flash("User {} already exists. Please Log In".format(form.username.data))
            return redirect("/login")
    return render_template("register.html", form=form)


@auth_bp.route("/login", methods=["GET", "POST"])
def login():
    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(username=form.username.data).first()
        flash("Login requested for user {}, remember me= {}".format(form.username.data, form.remember.data))
        if user:
            if check_password_hash(user.password, form.password.data):
                logout_user()
                login_user(user, remember=form.remember.data)
                flash("User {} Logged in successfully !".format(form.username.data))
                return redirect("/")
            flash("Invalid Username or Password")      You, a month ago • merge develop
        return redirect("/")
    return render_template("login.html", form=form)
```

5.3. Video_process.py - the process video function for tracking the

```python
def process_video(videourl, username, colour, lift, infoid):
    colours = {
               "redGradient": [(138, 0, 0), (255, 0, 0)],
               "blueGradient": [(11, 3, 216), (0, 212, 255)],
               "greenGradient": [(29, 86, 6), (64, 255, 255)],
               "yellowGradient": [(148, 116, 2), (255, 254, 0)]
               }
    print(colour)
    bar_speed = 0
    rest_time = 0
    # is_moving = False
    # rep_checked = False
    concentric = False
    last_direction = concentric
    # x-move = 0
    processed = False
    # current_app.logger.info("Processing Video")
    fps = 20
    fin = False
    upper_colour_range = colours[colour][0]
    lower_colour_range = colours[colour][1]
    print(upper_colour_range, lower_colour_range)
    past_points = deque(maxlen=32)
    # displacement = 0
    # velocity_ms = 0
    y_velocity_ms = 0      You, 3 months ago • get rep counting and bar speed in m/s
    avg_velocity = 0
    count = 0
    rep_count = 0
    lastx = None
    lasty = None
    radius = None
    peak_velocity = 0
    bb_radius = 25
    direction = ""
    avg_velocities = []
    # velocities = []
    history = []
    peak_vel = []
    rep = False
    # fourcc = cv2.VideoWriter_fourcc(*"AVC1")
    cap = cv2.VideoCapture(videourl)
    frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    # cv2.VideoWriter_fourcc('M', 'J', 'P', 'G')
    fps = int(cap.get(cv2.CAP_PROP_FPS))
    fourcc = cv2.VideoWriter_fourcc(*'H264')
    out = cv2.VideoWriter("static/video/bar.mp4", fourcc, fps, (frame_width, frame_height))
    print("Starting Bar Tracking")
    if cap.isOpened() is False:
        print('No Video File Found')
        # current_app.logger.error("Error opening video stream or file")
    print("File Found")
    while cap.isOpened():
        ret, frame = cap.read()
        if ret is True:
            # set the frame size to we can determine the pixels per metre
            # frame = imutils.resize(frame, width=800, height=600)
            # remove noise and convert to HSV color space
            noise_remove = cv2.GaussianBlur(frame, (11, 11), 0)
            hsv = cv2.cvtColor(noise_remove, cv2.COLOR_BGR2HSV)
            # create a mask for the selected colour to track it
            mask = cv2.inRange(hsv, upper_colour_range, lower_colour_range)
            mask = cv2.erode(mask, None, iterations=2)
            mask = cv2.dilate(mask, None, iterations=2)
            # find the outline of the circular end of the bar and a variable for the centre
            edges = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
            edges = imutils.grab_contours(edges)
            centre = None
            # if an outline is found
            if len(edges) > 0:
                # find the largest outline from the masked hsv image (which will be the end of the bar) and the mi
                outline = max(edges, key=cv2.contourArea)
                ((x, y), r) = cv2.minEnclosingCircle(outline)
                if radius is None:
                    radius = r
                    mmppx = bb_radius / radius
                    print(f"first radius {radius}... mm per pixel {mmppx}")
                    print(colour)
                if lastx is None:
                    lastx = x
                    lasty = y
                # get coordinates of centre of circle
                m = cv2.moments(outline)
                # centre = (x, y)
                centre = (int(m["m10"] / m["m00"]), int(m["m01"] / m["m00"]))
                # proceed if radius meets size requirements
                # pixel_dist_x = lastx - x
                pixel_dist_y = lasty - y
                lastx = x
                lasty = y
                # mm_dist_x = pixel_dist_x * mmppx
                mm_dist_y = pixel_dist_y * mmppx
                y_velocity_ms = mm_dist_y * fps / 1000
```

```
            if r > 10:
                # sys.stdout.flush
                # draw the circle and centre
                cv2.circle(frame, (int(x), int(y)), int(radius), (0, 255, 255), 2)
                cv2.circle(frame, centre, 5, (0, 0, 255), -1)
                # cv2.rectangle(frame, centre, 5, (0, 128, 255), -1)
                past_points.appendleft(centre)
                if mm_dist_y > radius / 4:
                    # is_moving = True
                    # rep_checked = False
                    # distance_moved = math.sqrt((pixel_dist_x ** 2) + (pixel_dist_y ** 2)) * mmppx
                    # velocity_ms = distance_moved * fps / 1000
                    rest_time = 0
                    if y_velocity_ms < 0.01 and y_velocity_ms > -0.01:
                        # is_moving = False
                        y_velocity_ms = 0
                        rest_time += 1

                history.append((centre[0], centre[1], y_velocity_ms))

                rep, info = check_for_rep(history, concentric, last_direction)
                fin = more_x_movement(history)

                if rep is True:
                    rep_count += 1
                    history = []
                    avg_velocity = info[0]
                    avg_velocities.append(info[1])
                    peak_vel.append(info[0])
                    if rep_count == 1:
                        avg_velocity = info[0]
                        peak_velocity = info[1]
                    else:
                        avg_velocity = avg_velocities[-1]
                        peak_velocity = peak_vel[-1]
                        v_loss = (avg_velocities[0] - avg_velocities[-1]) / avg_velocities[0] * 1000
                        if v_loss > 20:
                            fin = True
            for i in np.arange(1, len(past_points)):
                if past_points[i - 1] is None or past_points[i] is None:
                    continue
                # check if enough frames have been captured to see a direction change
                if count >= 10 and i == 1 and past_points[-10] is not None:
                    # reinitialise the direction variable for when the bar is stationary
                    direction = ""
                    last_direction = concentric
                    # check for significant movement up or down
                    if past_points[-10][1] > past_points[i][1]:
                        direction = "Concentric"
                        concentric = True
                    else:
                        direction = "Eccentric"
                        concentric = False
                    # use more points when checking for a rep as a small movement could qualify as a rep
                    if len(past_points) > 20:
                        if past_points[-20][1] > past_points[i][1]:
                            concentric = True
                        else:
                            concentric = False
                    cv2.line(frame, past_points[i - 1], past_points[i], (0, 0, 255), 5)
            cv2.putText(frame, f"Peak Velocity {peak_velocity}m/s", (10, frame.shape[0] - 30), cv2.FONT_HERSHEY_SIM
            cv2.putText(frame, f"Rep: {rep_count}", (10, 60), cv2.FONT_HERSHEY_SIMPLEX, 0.65, (0, 0, 255), 3)
            cv2.putText(frame, direction, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.65, (0, 0, 255), 3)
            cv2.putText(frame, f"Average Velocity {avg_velocity}m/s", (10, frame.shape[0] - 10), cv2.FONT_HERSHEY_S
            cv2.putText(frame, f"EndSet {fin}", (10, frame.shape[0] - 50), cv2.FONT_HERSHEY_SIMPLEX, 0.65, (0, 0, 2
            out.write(frame)
            print("Processing...")
            # last_direction = concentric
        else:
            out.write(frame)
            current_app.logger.info("Finished Processing Video")
            break

    count += 1
print("Finished Processing")
out.write(frame)
print(avg_velocities)
if len(avg_velocities) > 1:
    bar_speed = avg_velocities[-1]
    if bar_speed == 0:
        bar_speed = avg_velocities[-2]
print(bar_speed)
cap.release()
processed = True
print(processed)
vid = username + f"/bar/{lift}-{infoid}.mp4"
aws.S3_upload("bbtrack-bucket", username, "../static/video/video2.avi", vid)
return bar_speed
```

5.4. Check_points.py - where we get the points we don't have from the probability map

```python
def get_point_estimations(pts, POSE_NAMES, head_length):
    shin_length = int(head_length * 2)
    # i = 0
    # while i <= 1:
    if None not in pts:
        return pts
    try:
        # check ankle points
        if pts[POSE_NAMES["RANKLE"]] is None and pts[POSE_NAMES["LANKLE"]] is not None:
            pts[POSE_NAMES["RANKLE"]] = pts[POSE_NAMES["LANKLE"]][0], pts[POSE_NAMES["LANKLE"]][1] + 1
        elif pts[POSE_NAMES["LANKLE"]] is None and pts[POSE_NAMES["RANKLE"]] is not None:
            pts[POSE_NAMES["LANKLE"]] = pts[POSE_NAMES["RANKLE"]][0], pts[POSE_NAMES["RANKLE"]][1] + 1

        # check knee points
        if pts[POSE_NAMES["RKNEE"]] is None and pts[POSE_NAMES["LKNEE"]] is not None:
            pts[POSE_NAMES["RKNEE"]] = pts[POSE_NAMES["LKNEE"]][0], pts[POSE_NAMES["LKNEE"]][1] + 1
        elif pts[POSE_NAMES["LKNEE"]] is None and pts[POSE_NAMES["RKNEE"]] is not None:
            pts[POSE_NAMES["LKNEE"]] = pts[POSE_NAMES["RKNEE"]][0], pts[POSE_NAMES["RKNEE"]][1] + 1
        # if we have both knees and no ankles
        if pts[POSE_NAMES["LANKLE"]] is None and pts[POSE_NAMES["RANKLE"]] is None and pts[POSE_NAMES["LKNEE"]] is
            pts[POSE_NAMES["LANKLE"]] = (pts[POSE_NAMES["LKNEE"]][0], pts[POSE_NAMES["LKNEE"]][1] - shin_length)
            pts[POSE_NAMES["RANKLE"]] = (pts[POSE_NAMES["RKNEE"]][0], pts[POSE_NAMES["RKNEE"]][1] - shin_length)
        # and vice versa
        elif pts[POSE_NAMES["LANKLE"]] is not None and pts[POSE_NAMES["RANKLE"]] is not None and pts[POSE_NAMES["LK
            pts[POSE_NAMES["LKNEE"]] = (pts[POSE_NAMES["LANKLE"]][0], pts[POSE_NAMES["LANKLE"]][1] + shin_length)
            pts[POSE_NAMES["RKNEE"]] = (pts[POSE_NAMES["RANKLE"]][0], pts[POSE_NAMES["RANKLE"]][1] + shin_length)
        # check hand points
        if pts[POSE_NAMES["RHAND"]] is None and pts[POSE_NAMES["LHAND"]] is not None:
            pts[POSE_NAMES["RHAND"]] = pts[POSE_NAMES["LHAND"]][0], pts[POSE_NAMES["LHAND"]][1] + 1
        elif pts[POSE_NAMES["LHAND"]] is None and pts[POSE_NAMES["RHAND"]] is not None:
            pts[POSE_NAMES["LHAND"]] = pts[POSE_NAMES["RHAND"]][0], pts[POSE_NAMES["RHAND"]][1] + 1

        # check elbow points
        print("HERE")      You, a month ago • try more ways to get video to write
        if pts[POSE_NAMES["RELBOW"]] is None and pts[POSE_NAMES["LELBOW"]] is not None:
            pts[POSE_NAMES["RELBOW"]] = pts[POSE_NAMES["LELBOW"]][0], pts[POSE_NAMES["LELBOW"]][1] + 1
        elif pts[POSE_NAMES["LELBOW"]] is None and pts[POSE_NAMES["RELBOW"]] is not None:
            pts[POSE_NAMES["LELBOW"]] = pts[POSE_NAMES["RELBOW"]][0], pts[POSE_NAMES["RELBOW"]][1] + 1
        # if we have both elbows and no hands
        if pts[POSE_NAMES["LHAND"]] is None and pts[POSE_NAMES["RHAND"]] is None and pts[POSE_NAMES["LELBOW"]] is n
            pts[POSE_NAMES["LHAND"]] = (pts[POSE_NAMES["LELBOW"]][0], pts[POSE_NAMES["LELBOW"]][1] -  (variable) sh
            pts[POSE_NAMES["RHAND"]] = (pts[POSE_NAMES["RELBOW"]][0], pts[POSE_NAMES["RELBOW"]][1] - shin_length)
        # and vice versa
        elif pts[POSE_NAMES["LHAND"]] is not None and pts[POSE_NAMES["RHAND"]] is not None and pts[POSE_NAMES["LELB
            pts[POSE_NAMES["LELBOW"]] = (pts[POSE_NAMES["LHAND"]][0], pts[POSE_NAMES["LHAND"]][1] + shin_length)
            pts[POSE_NAMES["RELBOW"]] = (pts[POSE_NAMES["RHAND"]][0], pts[POSE_NAMES["RHAND"]][1] + shin_length)
        # Do the same for hips
        if pts[POSE_NAMES["RHIP"]] is None and pts[POSE_NAMES["LHIP"]] is not None:
            pts[POSE_NAMES["RHIP"]] = pts[POSE_NAMES["LHIP"]][0], pts[POSE_NAMES["LHIP"]][1] + 1
        elif pts[POSE_NAMES["LHIP"]] is None and pts[POSE_NAMES["RHIP"]] is not None:
            pts[POSE_NAMES["LHIP"]] = pts[POSE_NAMES["RHIP"]][0], pts[POSE_NAMES["RHIP"]][1] + 1
    except Exception:
        return "NOT ENOUGH POINTS"
        # i += 1
    return pts
```

5.5. Deadlift.py - where we analyse the body for the deadlift

```
if head_length is None:
    try:
        head_length = getDistance(pts[POSE_NAMES["HEAD"]], pts[POSE_NAMES["NECK"]])
    except Exception:
        head_length = None
    if head_length:
        foot_size = head_length * .85
    else:
        try:
            forearm_right = getDistance(pts[POSE_NAMES["RHAND"]], pts[POSE_NAMES["RELBOW"]])
        except Exception:
            print("couldnt get one or both forearm lengths.")
        try:
            forearm_left = getDistance(pts[POSE_NAMES["LHAND"]], pts[POSE_NAMES["LELBOW"]])
        except Exception:
            print("couldnt get one or both forearm lengths.")
        if forearm_left and forearm_right:
            foot_size = (forearm_right + forearm_left) / 2
        elif forearm_left and not forearm_right:
            foot_size = forearm_left
        elif forearm_right and not forearm_left:
            foot_size = forearm_right
        head_length = (foot_size / 85) * 100
# head_length = getDistance(pts[POSE_NAMES["HEAD"]], pts[POSE_NAMES["NECK"]])
# plotPoint(frame, pts[POSE_NAMES["HEAD"]], 17)
# plotPoint(frame, pts[POSE_NAMES["NECK"]], 17)
# Get the Bar Distance from the ankle
pts = get_point_estimations(pts, POSE_NAMES, head_length)
print_pose_elements(Pn)

if None not in (pts[POSE_NAMES["RANKLE"]], pts[POSE_NAMES["RHIP"]], pts[POSE_NAMES["RHAND"]], pts[POSE_NAMES["RE
    # ================================================================================
    # SCORE CHECK 1
    # Check that the par is positioned over mid foot
    # ================================================================================
    # Get average forearm length to estimate foot size
    forearm_right = getDistance(pts[POSE_NAMES["RHAND"]], pts[POSE_NAMES["RELBOW"]])
    forearm_left = getDistance(pts[POSE_NAMES["LHAND"]], pts[POSE_NAMES["LELBOW"]])
    foot_size = (forearm_right + forearm_left) / 2
    fs2 = head_length * .85
    print('FOOTSIZE1:', foot_size, "FOOTSIZE2:", fs2)
    # Get the midpoint of the hands incase the video is at a slight angle and add it to pts
    hand_point = getMidPoint(pts, POSE_NAMES, "RHAND", "LHAND")
    pts.append(hand_point)
    # plotPoint(frame, hand_point, 17)
    # plotPoint(frameWhite, hand_point, 17)

    ankle_point = getMidPoint(pts, POSE_NAMES, "LANKLE", "RANKLE")
    pts.append(ankle_point)
    # plotPoint(frame, ankle_point, 18)
    # plotPoint(frameWhite, ankle_point, 18)

    bar_x_distance_ankles = abs(hand_point[0] - ankle_point[0])
    print("BAR X DISTANCE FROM ANKLES: ", bar_x_distance_ankles)

    if (fs2 / 2) + bar_distance_threshold >= bar_x_distance_ankles >= (fs2 / 2) - bar_distance_threshold:
        print("Bar in correct Position", bar_x_distance_ankles)
    else:
        bar_not_over_mid_foot_count += 1
        print("Bar not in correct Position", bar_x_distance_ankles)
    # print(bar_x_distance_ankles, score, ankle_point[0], foot_size / 2)

    # ================================================================================
    # SCORE CHECK 2
    # Check that the athlete's back is straight
    # ================================================================================
    hips_point = getMidPoint(pts, POSE_NAMES, "LHIP", "RHIP")
    pts.append(hips_point)
    # plotPoint(frame, hips_point, 19)
    # plotPoint(frameWhite, hips_point, 19)

    shoulders_point = getMidPoint(pts, POSE_NAMES, "LSHOULDER", "RSHOULDER")
    pts.append(shoulders_point)

    back_angle_chest = getAngleC(hips_point, pts[POSE_NAMES["CHEST"]], pts[POSE_NAMES["NECK"]])
    if shoulders_point == pts[POSE_NAMES["NECK"]]:
        shoulders_point = shoulders_point[0] + 1, shoulders_point[1] + 1
    back_angle_shoulder = getAngleC(hips_point, shoulders_point, pts[POSE_NAMES["NECK"]])
    back_angles.extend([back_angle_chest, back_angle_shoulder])
    cv2.putText(frame, f"Score:  {score}", (10, 90), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0, 0, 255), 2, lineType=cv.
    cv2.putText(frameWhite, f"Chest Angle:  {back_angle_chest}", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0, 0
    cv2.putText(frameWhite, f"Shoulder Angle:  {back_angle_shoulder}", (10, 60), cv2.FONT_HERSHEY_SIMPLEX, 0.75,
    cv2.putText(frameWhite, f"Score:  {score}", (10, 90), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0, 0, 255), 2, lineTy
    # ================================================================================
    # SCORE CHECK 3
    # Check that the athlete's knees and hips extend at the same time
    # ================================================================================
    knees_point = getMidPoint(pts, POSE_NAMES, "LKNEE", "RKNEE")
    print(hips_point, ankle_point, knees_point, pts[POSE_NAMES["NECK"]])
    if ankle_point == knees_point:
        knees_point = knees_point[0] + 1, knees_point[1] + 1
    knee_angle = getAngleC(hips_point, ankle_point, knees_point)
    hip_angle = getAngleC(knees_point, pts[POSE_NAMES["NECK"]], hips_point)
    knee_angles.append(knee_angle)
    hip_angles.append(hip_angle)

    print("KNEE ANGLE: ", knee_angle, "HIP ANGLE: ", hip_angle)
else:
    print("Not Enough points")
```

```
            for pair in POSE_PAIRS:
                partA = pair[0]
                partB = pair[1]

                if pts[partA] and pts[partB]:
                    cv2.line(frame, pts[partA], pts[partB], (0, 255, 255), 2)
                    cv2.line(frameWhite, pts[partA], pts[partB], (0, 255, 255), 2)

            out.write(frame)
            out2.write(frameWhite)
        else:
            break
        out.write(frame)
        out2.write(frameWhite)
    out.release()
    out2.release()
    cap.release()
    cv2.destroyAllWindows()
    k = 1
    while k < len(knee_angles):
        diff_knees = abs(knee_angles[k] - knee_angles[k - 1])
        diff_hips = abs(hip_angles[k] - hip_angles[k - 1])
        # print(knee_angles[k], hip_angles[k])
        if knee_hips_angle_threshold >= abs(diff_knees - diff_hips):
            print('yes', k)
        else:
            print("NO", k)
            knee_hips_rate_is_not_equal = True
            print(diff_knees, diff_hips)
        k += 1
    if bar_not_over_mid_foot_count <= 7:
        score += 1
        scores["Bar Position"] = True
    average_back_angle = sum(back_angles) / len(back_angles)
    if average_back_angle <= back_angle_threshold:
        score += 1
        scores["Back Angle"] = True
    if not knee_hips_rate_is_not_equal:
        score += 1
        scores["Knee and Hip Extension"] = True
    # print("SCORE: ", score)
    # print("AVG BACK ANGLE: ", average_back_angle)
    # print("BAR POSITION: ", bar_not_over_mid_foot_count)
    # print(knee_angles, hip_angles)
    print(scores)
    return scores
```

5.6. body_functions/functions.py - some useful functions we needed to analyse the points

```python

def getMidPoint(pts, POSE_NAMES, partA, partB):
    x_coord = (pts[POSE_NAMES[partA]][0] + pts[POSE_NAMES[partB]][0])/2
    y_coord = (pts[POSE_NAMES[partA]][1] + pts[POSE_NAMES[partB]][1])/2
    return (int(x_coord), int(y_coord))


def getDistance(pointA, pointB):
    # get the distance between 2 points
    return math.sqrt(((pointA[0] - pointB[0]) ** 2) + ((pointA[1] - pointB[1]) ** 2))


def angleC(a, b, c):
    if a == 0:
        a = 0.001
    if b == 0:
        b = 0.001
    if c == 0:
        c = 0.001

    return math.degrees(math.acos((c**2 - b**2 - a**2)/(-2.0 * a * b)))


def getAngleC(pointA, pointB, pointC):
    AB_dist = getDistance(pointA, pointB)
    BC_dist = getDistance(pointB, pointC)
    AC_dist = getDistance(pointA, pointC)
    return(max(angleC(BC_dist, AC_dist, AB_dist), angleC(AC_dist, AB_dist, BC_dist)))


def print_pose_elements(Pn):
    for k, v in Pn.items():
        print(k, v)


def diff(a, b):
    return abs(a - b)

```

5.7. Body_constants.py - where we choose what messages to show

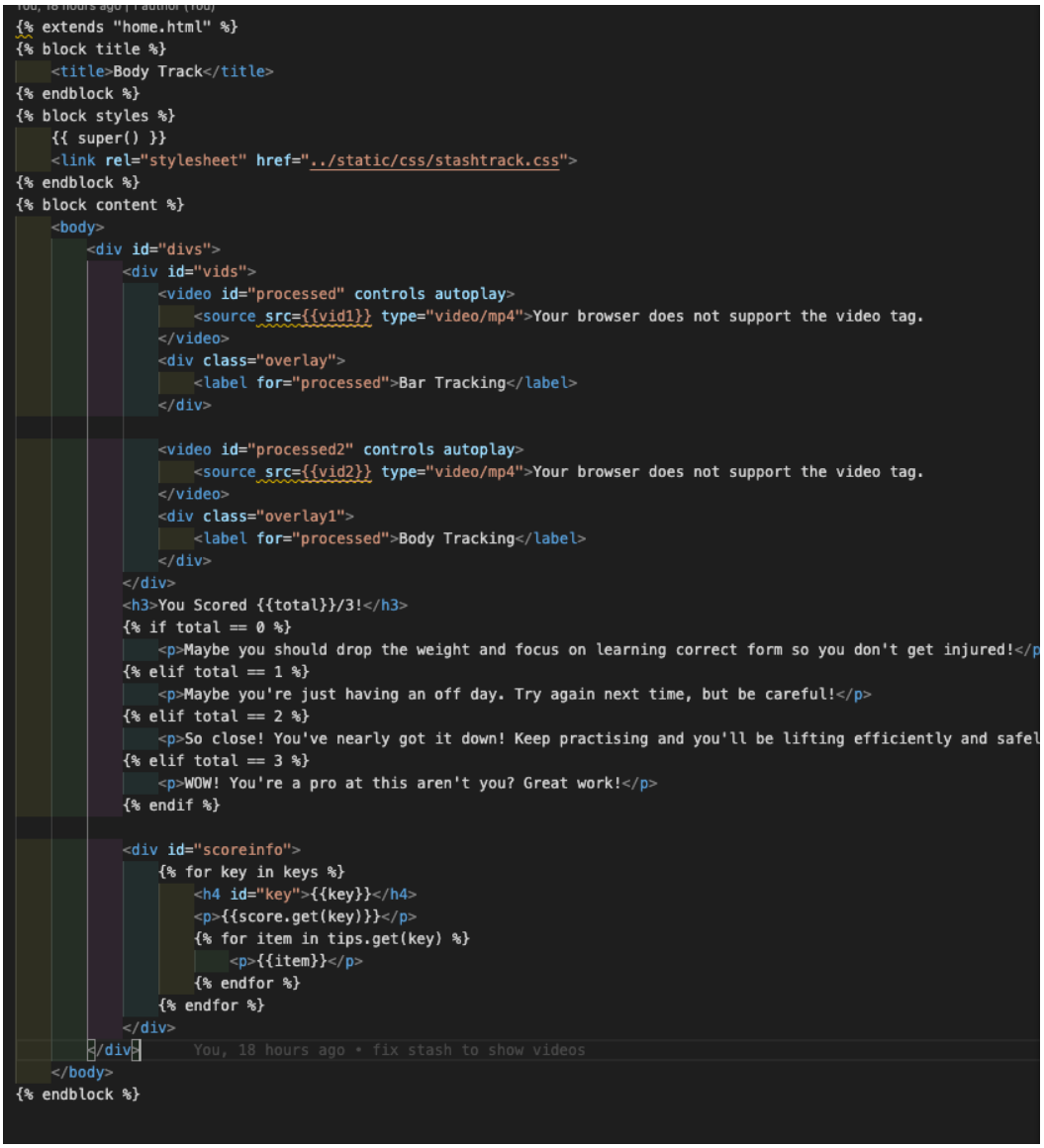
```
You, 15 hours ago | 1 author (You)
dictionary = {'Depth': {"SCORE": "Well Done! You hit the correct depth for a Squat when your hip went below your knees",
                        "NOSCORE": "Opps! You didn't hit depth!",
                        "TIPS": ["In order to hit depth your hips need to drop below the tops of your knees.",
                                 "Try to widen or narrow your stance to a position you can comfortably hit this position.",
                                 "You can also try to widen or narrow the angles of your feet. A good tip is to have them pointin in the same direction as your kn
                        },
              "Bar Position": {"SCORE": "Well done! You kept the bar in a straight line above your mid foot throughout the lift. Keep it up!",
                               "NOSCORE": "Oops! You allowed the bar to go forward too much throughout the lift.",
                               "TIPS": ["Try to sit back more against the weight push your hips backwards",
                                        "Going too far forward can cause your centre of gravity to move and, in turn cause you to lose balance"]
                               },
              "Back Angle": {"SCORE": "Well done! Your back stayed at a consistently straight angle throughout the lift. Keep it up!",
                             "NOSCORE": "Oops! Your back didnt stay consistently striaght throughout the lift.",
                             "TIPS": ["Try creating a stronger brace, and pushing out into your belt to maintain tenstion in your back.",
                                      "You may find that dropping your shoulder blades down and tucking your chin will help you to maintain this position effectivel
                             },
              "Knee and Hip Extension": {"SCORE": "Well done! Your knees and hips extended and contracted simultaneously throughout the lift. Keep it up!",
                                         "NOSCORE": "Oops! Your knees and hips didn't extend and contract simultaneously  throughout the lift.",
                                         "TIPS": ["Try to not let your hips rise before the weight begins to move or at a faster pace than your upper body.",
                                                  "Think about the cue \"Chest Up\" when performing the lift to ensure the weight moves optimally.",
                                                  "Extending your knees and hips at the same time allows for better leverage when moving the weight on the bar."]
                                         },
              "Hips on Bench": {"SCORE": "Well done! You kept your hips on the bench throughout the lift. Keep it up!",
                                "NOSCORE": "Oops! You didn't keep your hips on the bench throughout the lift.",
                                "TIPS": ["Try to keep tension throughout the lift. A sudden burst of strength can cause the hips to shoot up",
                                         "Push up powerfully, but controlled. A lack of control can also cause your hips to leave the bench."]
                                },
              "Feet on Floor": {"SCORE": "Well done! You kept your feet on the floor throughout the lift. Keep it up!",
                                "NOSCORE": "Oops! You didn't keep your feet on the floor throughout the lift.",
                                "TIPS": ["Think about pushing through your heels and trying to push your body up the bench.",
                                         "Try a different stange. Bring your feet closer/further to the bench. You may also benefit from a wider/ narrowe
                                },
              "Bar Past Chest": {"SCORE": "Well done! You kept the bar in a straight line above your mid foot throughout the lift. Keep it up!",
                                 "NOSCORE": "Oops! You allowed the bar to go forward too much throughout the lift.",
                                 "TIPS": ["Try to sit back more against the weight push your hips backwards",
                                          "Going too far forward can cause your centre of gravity to move and, in turn cause you to lose balance"]
                                 }      You, 15 hours ago • flake8
              }

```

5.8. Video.js -to record and send the video

```
let constraintObj = {       You, 4 months ago • get video to record, need to work out how to save…
    audio: false,
    video: true
};


if (navigator.mediaDevices === undefined) {
    navigator.mediaDevices = {};
    navigator.mediaDevices.getUserMedia = function(constraintObj) {
        let getUserMedia = navigator.webkitGetUserMedia || navigator.mozGetUserMedia;
        if (!getUserMedia) {
            return Promise.reject(new Error('getUserMedia is not implemented in this browser'));
        }
        return new Promise(function(resolve, reject) {
            getUserMedia.call(navigator, constraintObj, resolve, reject);
        });
    }
}else{
    navigator.mediaDevices.enumerateDevices()
    .then(devices => {
        devices.forEach(device=>{
            console.log(device.kind.toUpperCase(), device.label);

        })
    })
    .catch(err=>{
        console.log(err.name, err.message);
    })
}

navigator.mediaDevices.getUserMedia(constraintObj)
.then(function(mediaStreamObj) {

    let video = document.querySelector('video');
    if ("srcObject" in video) {
        video.srcObject = mediaStreamObj;
    } else {

        video.src = window.URL.createObjectURL(mediaStreamObj);
    }

    video.onloadedmetadata = function(ev) {

        video.play();
    };

    let start = document.getElementById("startbtn");
    let stop = document.getElementById("stopbtn");
    let vidSave = document.getElementById("vid");
    vidSave.style.display = "none"
    let save = document.getElementById("saveVid")
    let mediaRecorder = new MediaRecorder(mediaStreamObj);
    let chunks = [];

    start.addEventListener('click', (ev) =>{
        mediaRecorder.start();
        var y = document.getElementById("vid1");
        var x = document.getElementById("vid");
        if (y.style.display === "none") {
            y.style.display = "block";
            x.style.display = "block";
        }
        else {
            x.style.display = "block";
            y.style.display = "block";
        }
        console.log(mediaRecorder.state);
    })
    stop.addEventListener('click', (ev) =>{
        mediaRecorder.stop();
        var x = document.getElementById("vid1");
        var y = document.getElementById("vid");
        if (x.style.display === "none") {
            x.style.display = "block";
            y.style.display = "none";
        }
        else {
            x.style.display = "none";
            y.style.display = "block";
        }
        console.log(mediaRecorder.state);
    });
    mediaRecorder.ondataavailable = function(ev) {
        chunks.push(ev.data);
    }
    mediaRecorder.onstop = (ev) =>{
        let blob = new Blob(chunks, { 'type' : 'video/mp4;' });
        chunks = [];
        let videoURL = window.URL.createObjectURL(blob);
        vidSave.src = videoURL;
        let formData = new FormData();
        formData.append('video', blob, "video.mp4")

        save.addEventListener('click', (ev) =>{
            // var file = new File([blob], "video.mp4",{type: "video/mp4"});
            // getSignedRequest(file);
```

```
            $.ajax ({
                url: "/upload",
                type: "POST",
                data: formData,
                cache: false,
                processData: false,
                contentType: false
            }).done(function(response){
                console.log(response);
            })
        })
    })


.catch(function(err) {
    console.log(err.name, err.message);
});
```

5.9. Stashtrack.html - an example of how we use the dictionaries to show the information to the user



```
{% extends "home.html" %}
{% block title %}
    <title>Body Track</title>
{% endblock %}
{% block styles %}
    {{ super() }}
    <link rel="stylesheet" href="../static/css/stashtrack.css">
{% endblock %}
{% block content %}
    <body>
        <div id="divs">
            <div id="vids">
                <video id="processed" controls autoplay>
                    <source src={{vid1}} type="video/mp4">Your browser does not support the video tag.
                </video>
                <div class="overlay">
                    <label for="processed">Bar Tracking</label>
                </div>

                <video id="processed2" controls autoplay>
                    <source src={{vid2}} type="video/mp4">Your browser does not support the video tag.
                </video>
                <div class="overlay1">
                    <label for="processed">Body Tracking</label>
                </div>
            </div>
            <h3>You Scored {{total}}/3!</h3>
            {% if total == 0 %}
                <p>Maybe you should drop the weight and focus on learning correct form so you don't get injured!</p
            {% elif total == 1 %}
                <p>Maybe you're just having an off day. Try again next time, but be careful!</p>
            {% elif total == 2 %}
                <p>So close! You've nearly got it down! Keep practising and you'll be lifting efficiently and safel
            {% elif total == 3 %}
                <p>WOW! You're a pro at this aren't you? Great work!</p>
            {% endif %}

            <div id="scoreinfo">
                {% for key in keys %}
                    <h4 id="key">{{key}}</h4>
                    <p>{{score.get(key)}}</p>
                    {% for item in tips.get(key) %}
                        <p>{{item}}</p>
                    {% endfor %}
                {% endfor %}
            </div>
        </div>      You, 18 hours ago • fix stash to show videos
    </body>
{% endblock %}
```

5.10. Polyreg.py - function that calculates the line of best fit

```python
import numpy as np
import matplotlib
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline
matplotlib.use('TkAgg')


def data_rep(a, b, c):
    x = [1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6, 7, 7, 7, 8, 8, 8, 9, 9, 9, 10, 10, 10]
    p = []

    i = 0
    while i < len(a):
        p.append(a[i])
        p.append(b[i])
        p.append(c[i])
        i += 1

    model = make_pipeline(PolynomialFeatures(2), LinearRegression())
    model.fit(np.array(x).reshape(-1, 1), p)
    x_reg = np.arange(1, 11)
    y_reg = model.predict(x_reg.reshape(-1, 1))

    return x_reg, y_reg
```

6. Problems Solved

We ran into many problems during the process of creating our app. The first of which was the S3 bucket. It took a lot of configuration to allow access to the bucket both from localhost and from Heroku. We managed to get it working no problem with local host after some time, but still had issues with allowing heroku to use it, access keys etc were not working. We think if we had hosted it on AWS and had we known that we would have been using an S3 bucket prior to hosting on heroku we definitely would have hosted there instead.

Another problem we had was when starting the body tracking. There were a lot of options for us, OpenPose, OpenVino and more, a lot of time was spent with the trial and error method to get these working, OpenPose was especially difficult but was far too slow and required a bit more modification to the existing framework. OpenVino, after a long time of trying, was deemed too difficult to get started and we decided to try it with OpenCV as we came across an instance of it being used with object recognition. Thankfully it worked for us, and although it's quite slow to use, we were able to get it running consistently in our app and generating correct information.

Nearer to the end, we also had problems with Heroku. The slug size for heroku is quite small, and with the size of our app we consistently had issues with the app not being able to deploy and needing to take things out that we could deal without for the time being for testing purposes which led to some annoying waiting times as the deploy could take upwards of 15-20 minutes and then we would be let know that the deploy had failed. We managed to solve these eventually, but we would definitely use an alternative hosting platform, or a paid version of heroku if we were to do something like this again in the future.

7. Results

We found after our unit testing, end to end testing, cross browser testing, accuracy testing and user testing, that the project did achieve the goal of capturing the rpe when recording the lift, and tracking the key points and giving information based on this. We found that the accuracy on average, when body tracking was about 74.64% compared to 83.78% when tracking without a bar, which wouldn't get in the way of some points such as hands or ankles. The feedback on our user testing was very impactful on the reality of this app becoming a viable product. We found that overall, the coaches that answered our questionnaire after using the app found that the rpe was a great piece of information for them, but argued that the body tracking could be different for every individual. This means that If we were to succeed with this app becoming a product, we would need to speed up the video processing and also cater towards learning peoples form in order for coaches to be able to make use of the data. The athletes however, loved the body part tracking. Perhaps this is due to having extra information, although not particularly useful to them but can be quite interesting for them to see how their body moves and maybe enlighten them as to more efficient ways of lifting to prevent injury. The people who weren't involved with any coaches and had little experience in lifting weight also found body tracking to be very useful in expanding their knowledge with regards to correct and safe form to practise. They found little to no use from the RPE section, as this is primarily used in strength training and wouldn't be widely used outside of this area i.e. the average gym-goer. More information on the test results is included in the testing manual. Ultimately, our app does provide enough accuracy to be useful to amateur athletes and coaches, but there is without a doubt potential for growth and for a successful application given enough time, resources and knowledge. The scope and potential for this technology is ever growing as is the amount of athletes entering the strength training realm. We have learned a lot over the course of the year developing this app and have certainly made some mistakes along the way, but given the opportunity, the app could become a necessity in any gym or training setup. Some of the potential future work that we conduct is explained below.

8. Future Work

8.1. Hardware

In the future there are quite a lot of things given we had the time, resources, money etc. that we would do differently. First of all we think that the idea of using a coloured piece of paper, no matter how many variants there are is not inclusive to all potential users, i.e. colour blind. We would like to implement some real hardware that perhaps uses magnets to stay on the bar to detect the speed and bar path while connecting via bluetooth or wifi.

8.2. Heart Rate

As RPE is so subjective, we know that there is one thing that can never lie about how difficult an exercise is, the heart. If possible we could include a heart rate monitor, or pair the app with a fitbit or apple watch to get the heart rate, using past data to determine the level of exertion.

We would also begin with the idea that we would create a native app out of this. We are only able to use it as a web app for the time being and ios/android capabilities would be necessary to make this product viable in the industry.

8.3. Exercise Recognition

We could also add in exercise recognition, so the user wouldn't need to go through the steps of selecting a colour or lift, they can simply record or upload a video to be processed. And go from there.

8.4. Faster Body Tracking

The body tracking portion of the app is very slow at the best of times, so we would host this elsewhere on a server with better capabilities than our computers to allow this portion to run smoothly.

8.5. Hosting

Although it would be better to be a native app as opposed to a web app, we would choose another hosting platform to heroku, or use the paid version of heroku to allow the app to run on this, due to the large file sizes and amount of requests made, it's simply not viable to use the free version of heroku for a product like this.

8.6. Adding a Social Aspect

We would love to make this a community hub for some of the powerlifting clubs. Allowing the coaches to give their gym programmes in the app, track all progress including bodyweight, nutrition, sleep and many more. To become an all round fitness app for coaches and their clients where clients can chat amongst each other, direct message their coach, ask questions on a forum and share their progress between each other or on other social media apps.