# Icon Description automatically generated

FIBnessApp

Spring 2023

Projecte de Tecnologies de la Informació

Isaac Bassas

Mikhail Kovalev

Guillem Peralta

Marc Tacons

Universitat Politècnica de Catalunya

Department of Information Technology

# Introducció

La idea inicial del projecte era fer un projecte que ens permetès integrar ML per resoldre un problema real, com pot ser la necessitat de fer exercici. En un principi, l'aplicació seria capaç no només de comptar les repeticions, sinó també de detectar les postures dels diferents exercicis i determinar si eren correctes o no. Finalement, degut a que aquesta ultima part hauria requerit de tornar a entrenar el model, un ámbit en el que no teniem experiencia, vam acabar per descartar la idea, i centrarnos en obtenir uns bons resultats en la medicio de repeticions. A mes a mes, tot i que inicialment no teniem clar si ho fariem, finalment hem integrat l'aplicació amb Kubernetes i CockroachDB, lo que ens permet tenir un sistema molt flexible i escalable.

# 

# Motivació/Descripció del projecte

Les diferents possibilitats inicials de projecte incloïen capturar dades sobre tot tipus d'objectes, com cotxes, fauna i flora, o contar persones. Tot i així, finalment vam decidir centrarnos en un objectiu molt mes directe i pràctic per al usuari mig, i és així com va sorgir la idea de una app de fitness. Les activitats esportives formen part del dia a dia de moltes persones, i és per això que vam decidir que la millor manera de procedir seria fent algun projecte que pogués tenir una utilitat real de cara a un usuari final, i no tant per empreses de un sector específic. Per últim, part de la motivació també era poder penjar l'aplicació a la PlayStore / AppStore i poder veure com els usuaris interactuaven amb el nostre treball.

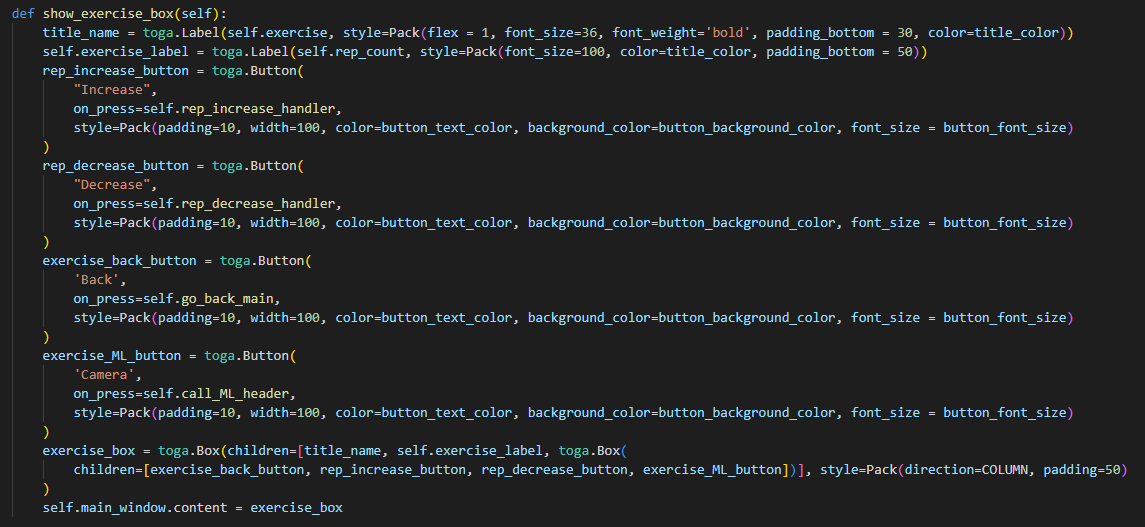
# 

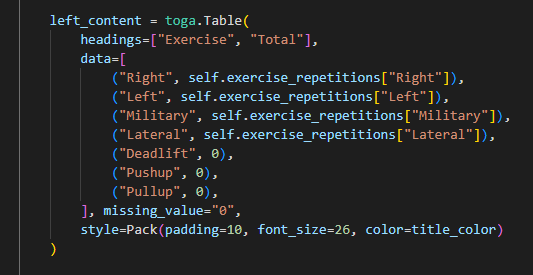
# Front End

Aplicació mòbil principalment pensada per Android i IOS, tot i que es podria utilitzar a Windows i Mac també. Per fer-la s’ha utilitzat un framework de python anomenat Beeware, que permet programar una sola vegada y exportar a qualsevol dispositiu amb una sola comanda “briefcase build [Android/IOS]”. El framework funciona molt similar a la majoria de eines per desarrollar aplicacions mòbils: Crees un canvas, on hi pots ficar diferents caixes en posicions i sentits. Dins de les caixes podem ficar qualsevol tipus de widgets, com per exemple botons, caixes de text, imatges, inputs d'usuari, taules de contingut o menus.

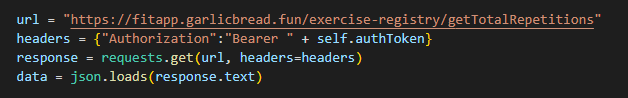
Les diferents pantalles (Windows) que es mostren a l'usuari son les següents:

* Starting, que mostrar el menú inicial per poder registrarse (Sign Up), entrar amb el teu compte (Log In) y un botó per tancar l'aplicació (Exit), així com el nom de l'aplicació en gran y el logo de la app i de la UPC.
* Login, window per fer el login, amb els text y password inputs i els botons per entrar y sortir de la finestra
* Signup, window per fer el signup, amb els text inputs per introduir la informació de l'usuari i els botons per entrar y sortir de la finestra.
* Main, la pàgina principal del usuari, on es mostra tota la seva informació dels exercicis, i es permet fer noves repeticions.
* Exercise, finestra generica utilitzada per a tots els exercicis que mostra per pantalla el número de repeticions realitzat en aquella sessió ademés dels botons per incrementar i decrementar manualment les repeticions, i el botó per iniciar el ML.
* Profile, una finestra on es mostra la informació de l'usuari que ha iniciat sessió, actualment només el nom i el email.

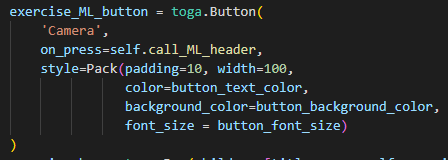




El frontend es comunica amb el backend mitjançant la llibreria requests de python, i codifica la informació a ser enviada amb la llibreria json. Les dades d'usuari son empaquetades en un json, per enviarles a la BD, i la BD retorna un altre fitxer json, amb la informació dels exercicis de l'usuari.



Donat que l'aplicació està feta amb Python3, per poder utilitzar la part de Machine Learning només ha calgut crear una funció amb tot el codi, i cridar quan es prem el botó per començar a fer l'exercici. També s’han inclòs les llibreries de tensorflow, numpy, matplotlib i openCV per a que tot funcioni adequadament.



# 

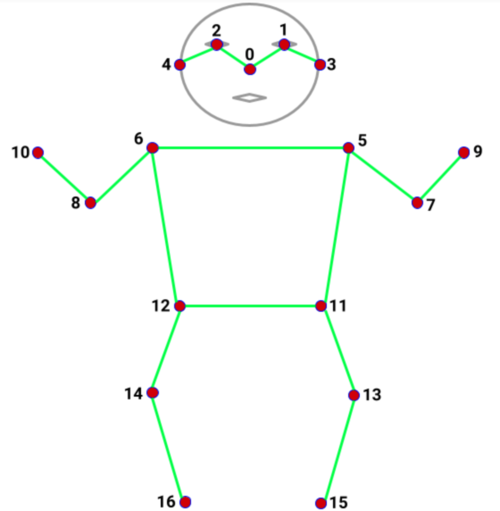
# Machine Learning

Com ja s’ha mencionat anteriorment, volem que la nostre aplicació don-hi feedback en temps reals dels exercicis que està realitzan un usuari que utilitza la nostre aplicació. Això va suposar haver d’investigar una mica ja que el nostre grup de *Machine Learning* no teniem molta idea ja que és un tema que hem tocat molt poc. Així doncs, després d’estar buscant informació i seguint els consells del professorat vam trobar l’algoritme que voliem utilitzar. Aquest es coneix amb el nom de *Movenet Lightning,* un model d’estimació de postura que es desenvolupat a partir de *TensorFlow* que és capaç de detectar la postura corporal d’una persona. Aquest model es tracta d’una xarxa convolucional que ha sigut entrenada amb una gran varietat de datasets i trainingsets. És per això que aquest model supera a l’anterior versió *PoseNet*(també de *TensorFlow)*. El que fa *Movenet Lightning* és detectar la postura corporal d’una sola persona a partir d’una imatge o frame. En el cas de que hi hagi més d’una persona el model intentarà detecar la postura d’aquella que estigui més próxima al centre de la imatge. No obstant, és recomana que pel bon funcionament de l’algoritme és millor que només surti una persona. A més a més, també cal destacar que *Movenet* *Lightning* és una versió que té menys capacitat que la seva versió germana *Movenet Thunder*. Tot i que aquesta darrera és més precisa en quant a detecció de postura vam decidir utilitzar la versió “light” perquè, tot i no ser tan precisa, és un model més lleuger que proporciona un bon rendiment en la majoria de dispositius de la nostre actualitat. Aleshores, amb la finalitat de que qualsevol persona amb un smartphone pogués fer ús de la nostre aplicació vam decantar-nos per elegir aquesta opció.

## Com funciona?

Com ja s’ha explicat anteriorment *Movenet Lightning* és una xarxa convolucional que es capaç de predir l’ubicació/posició de les articulacions humanes d’una sola persona. Aquest model pot correr en temps real a més de 50 fps en els navegadors de qualsevol ordinador actual a través de *Tensorflow,js* o en qualsevol dispositiu mòbil mitjançant *TensorFlow Lite* (aquest és el nostre cas).

No entrarem en detalls tècnics per explicar exactament com s’ha creat i com funciona el model sino que en aquesta secció s’explicarà una mica com realitza la detecció la postura l’algoritme de *Movenet* Lightning. El que fa aquest model és detectar fins a 17 articulacions o més ben dit 17 *keypoints* del cos d’una persona. Aquest 17 *keypoints* són: nas, ull esquerre, ull dret, orella esquerre, orella dreta, espatlla esquerre, espatlla dreta, colze esquerre, colze dret, canell dret, canell esquerre, maluc esquerre, maluc dret, genoll esquerre, genoll dret, turmell esquerre i turmell dret.

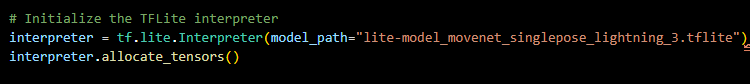


Perquè el model funcioni correctament i pugui fer correctament les prediccions és necessari que el frame del video o la imatge que se li passin com a input segueixi un fromat int32 amb una relació 192x192x3. Pel que fa al output, l’algoritme de *Movenet Lighnting* retorna un vector de float32 que té el següent format: [1, 1, 17, 3]. Aquest es tracta d’un vector multidimensional i així a ull és una mica difícil de visualitzar però el que fa el model és retornar en els 2 primers elements les coordenades xy de cada un dels *keypoints* que hem mencionat anteriorment seguint el mateix ordre. Finalment l’últim element ens indica el que es coneix com a *confidence threshold* que ens permet saber amb quina confiança a predit el model cada un dels *keypoints*. Un exemple de coordenades i *confidence threshold* d’un *keypoint* sería [0.496572, 0.52227, 0.68976]. Un cop explicat com s’ha de treballar amb l’algoritme podem passar a veure com hem integrat l’algoritme de *Machine Learning* amb la nostra aplicació.

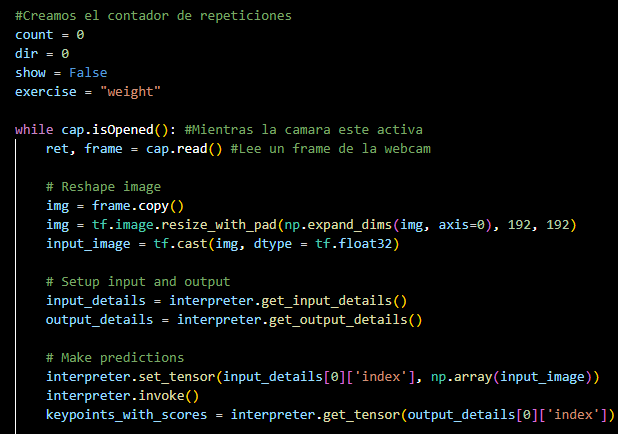
## Integració amb la nostra aplicació

Per treballar amb el model de *Movenet Lightning* hem utilitzat python com a llenguatge de programació ja que és el llenguatge que més s’utilitza per aplicacions que fan servir Intel·ligència Artificial o *Machine Learning* i, per tant, no tindrem problemes a l’hora de programar. D’altra banda, les tres llibreries principals que hem utilitzat per treballar amb *Movenet Lightning* han sigut tres:

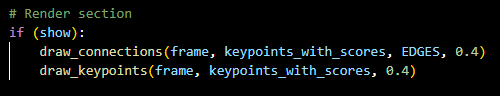
* **TensorFlow:** llibreria opensource desenvolupada per Google que s’utilitza en aplicacions d’intel·ligència artificial i machine learning que es necessaria per executar i interactuar amb el nostre model.
* **NumPy:** llibreria de python que dona suport per crear matrius i vectors multidimensionals. Necessitem aquesta llibreria ja que l’output que ens retorna *Movenet Lighting* és un vector de varies dimensions.
* **Cv2:** llibreria de visió artificial desenvolupada per Intel que s’utilitza en aplicacions de reconeixement d’objectes, detecció de moviments, de postura, etc. Aquesta llibreria serà necessària.

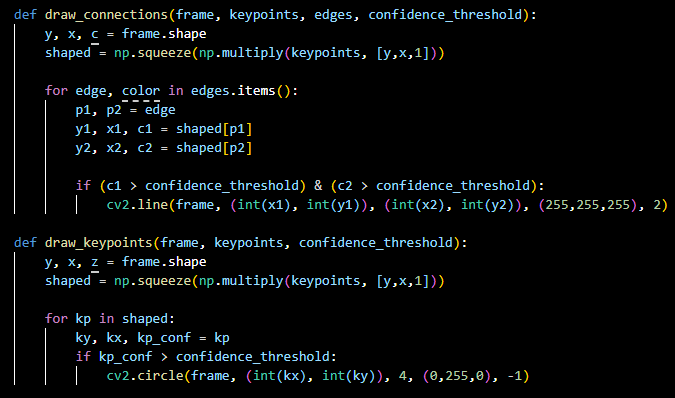
Dit això passem a l’explicació del codi que hem realitzat amb l’algorisme de *Machine Learning*. Primer de tot, carreguem el model *Movenet* *Lightning* i cridem al interpreter de *TensorFlow* perquè sàpiga quin és le model que ha d’utilitzar.

Després d’això, a través de la llibreria cv2, detectem la càmara del dispositiu i obtenim els frames un per un mitjançant cap.read( ). Com ja s’ha explicat anteriorment, els frames necessiten tenir un format int32 de 192x192x3 píxels i, per tant, hem de fer un resize dels frames que obtenim de la càmara. Seguidament, invoquem l’*interpreter* de *TensorFlow* i obtenim la predicció de cada un dels *keypoints* a través del output del model.



Amb aquests punts ja podem dibuixar l’esquelet virtual perquè sigui més visuals. Per dibuixar tant els punts com les arestes utilitzem les següents funcions les quals se’ls passa cada un dels punts que es volen dibuixar. Cal destacar que perquè l’esquelet virtual sigui el més precís possible hem decidit que tots els punts i les arestes es dibuixaran sempre i quan el *confidence threshold* sigui més gran que 0.4 (que és un valor força assequible).

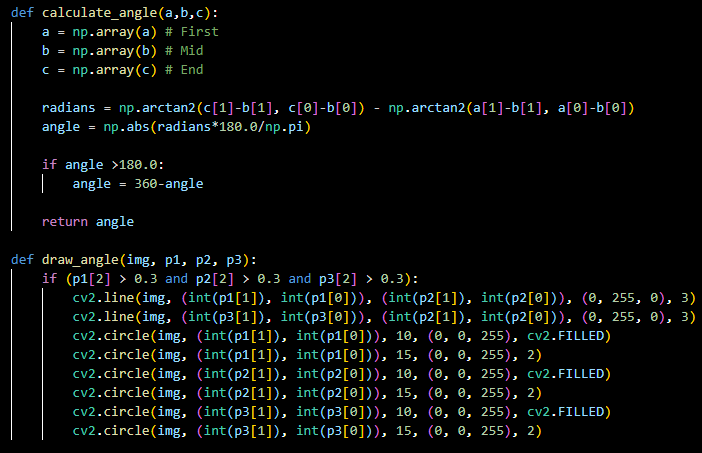




Ja per últim només quedaria parla de l’última part del codi que es el contador de repeticions. Per qüestions de temps i per altres dificultats només hem pogut crear el codi per comptar les repeticions de 4 exercicis:

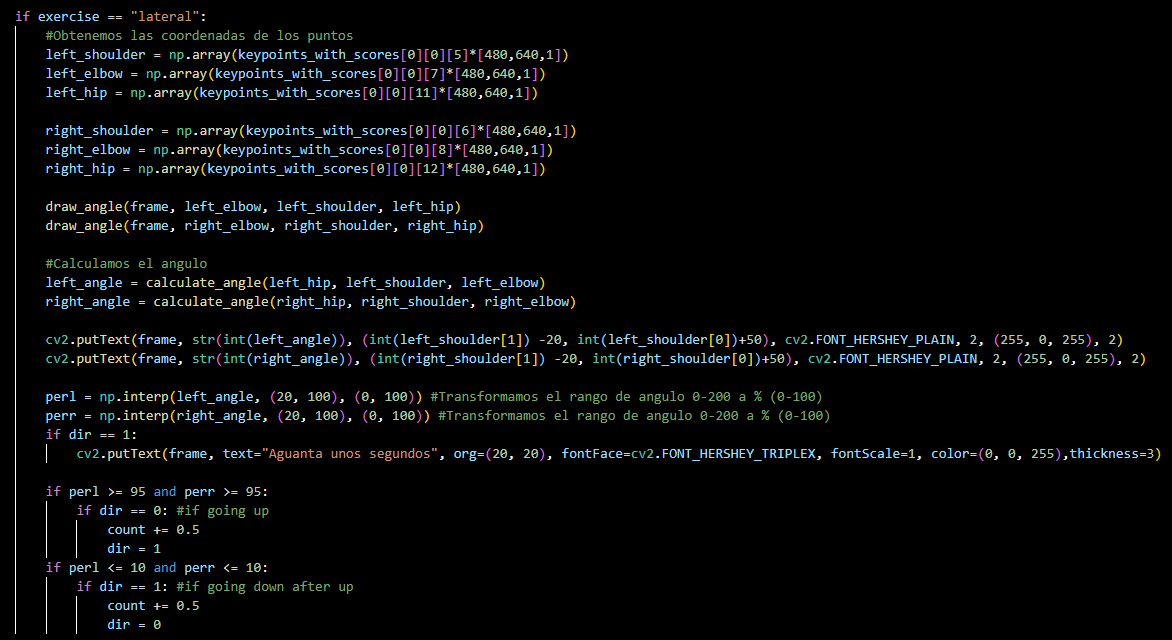
* **Weight left i Weight right:** Aquest exercici esta pensat per quan l’usuari realitza algun tipus d’exercici físic amb peses, concretement quan aquest entrena els bíceps fent varies series. Per poder comptar el número de repeticions el que fem és obtenir les coordenades de l’espatlla, el colze i el canell per calcular l’angle amb el qual incrementerem el contador. Finalment, transformem l’angle en un interval de percentatges i fem que si aquest és més gran de 75 sumem mitja repetició al contador i quan és més petit de 25 sumem la mitja repetició que li falta. El codi es veu a continuació.





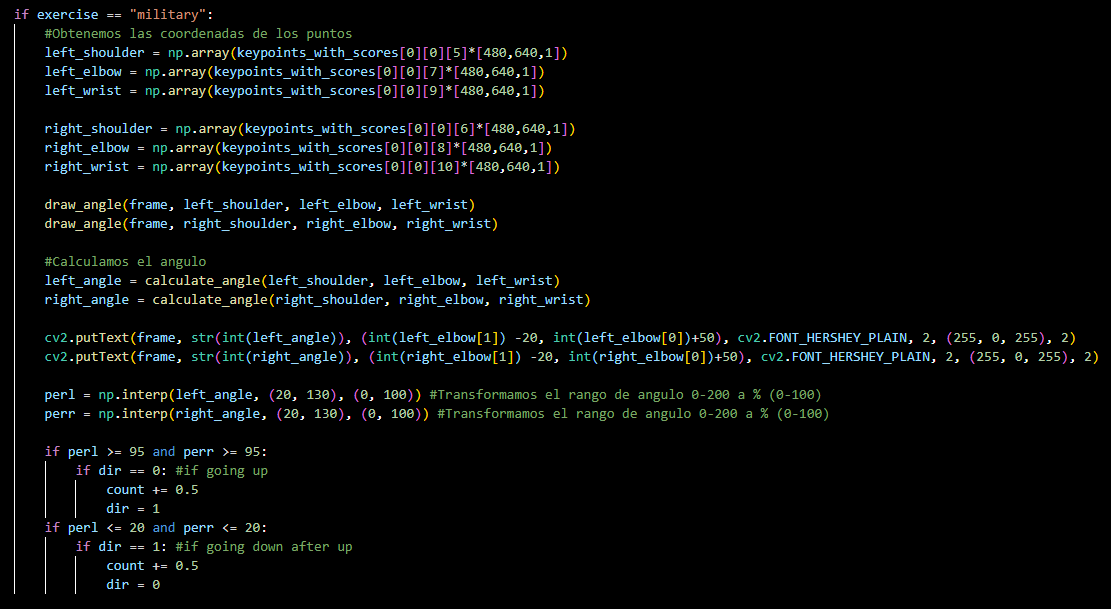
* **Lateral:** Aquest exercici esta pensat per quan l’usuari realitza alguna mena d’entrenament lateral amb els braços. En aquest cas agafem les coordenades de l’espatlla, el colze i la cintura per calcular l’angle entre aquests i poder comptar el número de repeticions. El mecanisme és el mateix que a l’anterior exercici.





* **Military:** Aquest exercici esta pensar per quan l’usuari realitza peses però aquest cop amb les dues mans. Per comptar les repeticions es segueix el mateix mecanisme que el de Weight left pero en aquest cas es té en compte l’angle d’ambdues parts del cos (esquerra i dreta).





# 

# Arquitectura - Backend-Node js

Per a programar els endpoints que comunicarà el frontend amb la base de dades hem decidit utilitzar node js, utilitzant el framework express degut a la seva simplesa i flexibilitat.

Per a l’arquitectura hem decidit dividir el backend en dos, un on tindrem tot el relacionat amb els usuaris, i l’altre amb els exercicis. En ambdós casos tenim el codi replicat, només canviem les funcions (el fitxer src/controllers/endpoint\_controllers.js). Ho hem decidit fer així en cas que haguem d’afegir noves funcionalitats en només un dels dos casos tenir ja l’estructura preparada per a fer-ho sense problemes.

Després tenim 2 fitxers, db.js i index.js i 5 carpetes diferents, les quals són:

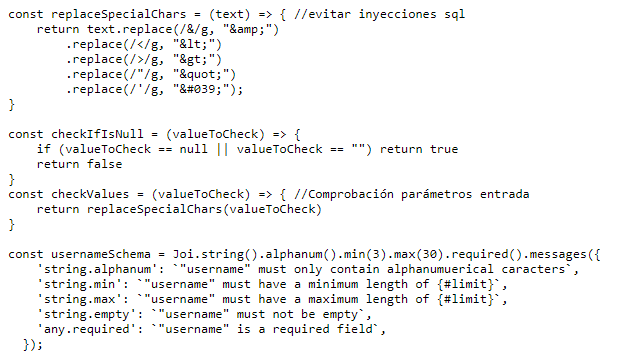
* **Controllers:** Hi tenim el fitxer endpoint\_controllers.js, on tenim el codi per a les funcions que necessitem per tractar amb els usuaris, és a dir, el login, el register i agafar la informació de l’usuari. Tant en el register com en el login el funcionament és l’esperat, és a dir, en el register es comprova que l’usuari no existeix, i si no existeix llavors es guarden les seves dades a la base de dades. En el cas del login es comprova que existeix, i si es el cas, que les contrasenyes coincideixin. En el cas de retornar la informació de l’usuari simplement és fer una consulta a la base de dades de l’usuari en qüestió i retornar tot el que es té sobre aquell usuari.

Ara bé, per a assegurar la seguretat i fiabilitat de les dades hem afegit dos fases per a la validació dels paràmetres de l’usuari.

La primera fase de validació és per a llançar un missatge d’error en cas que no es compleixin certs requisits que hem establert amb anterioritat, com pot ser, que els paràmetres d’entrada d’usuari i password tinguin entre 3 i 30 paraules alfanumèriques, que es un paràmetre obligatori..etc. Això s’aconsegueix mitjançant el mòdul de Joi, i l'apliquem a cada paràmetre d’entrada que ha hagut d’escriure l’usuari.

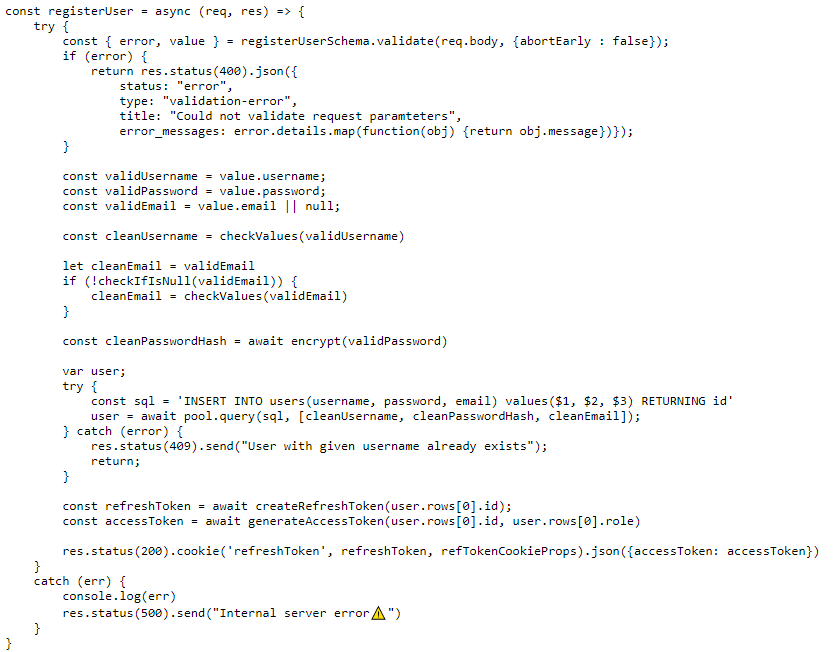
La segona fase consisteix en que si l’usuari ens ha passat caràcters que no volem (“<”, “>”...), i se'ns ha escapat de la primera validació, ara es substitueixen aquests paràmetres per d’altres que no afecten en absolut, per exemple, per espais en blanc o caracters que acceptem com vàlids.

Exemples de les validacions:



També hem fet un format d’error on es donem cert tipus d’informació: el que ha ocorregut, quin és el problema, com es pot solucionar..etc. I ho enviem al frontend en format JSON. Allà es separa els diferents tipus d’informació sobre l’error i es serveixen a l’usuari, sempre tenint cura de no donar informació que un usuari no hauria de saber i que pot posar en perill la seguretat de l’aplicació.

El register i login serien:

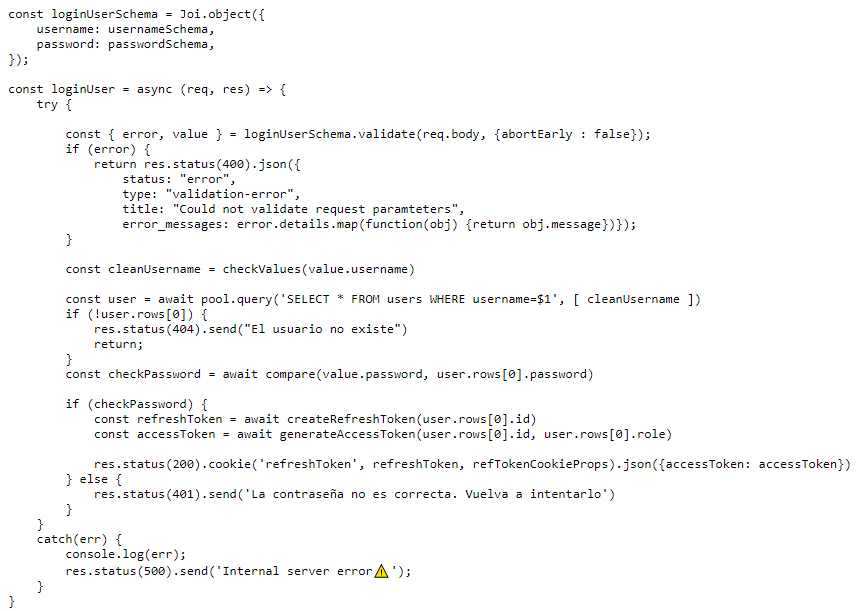


També cal destacar que a las consultas sql, en comptes de passar directament els paràmetre de l’usuari per dur a terme aquestes consultes, per exemple, el username, password i email, utilitzem consultes parametritzades o preparades, i simplement substituïm els paràmetres genèrics per als valors que es vol . D’aquesta manera ens protegim d’atacs d’injecció sql (una de les mesures).

Exemple:



En el cas dels exercicis el funcionament és el mateix. Tenim 2 fases de validació dels paràmetres d’entrada, i en cas que tot estigui correcte, s’envien les sentències sql corresponents a la base de dades i es retorna el feedback o els resultats que volia l’usuari.



* **Database:** Conté el fitxer db.js amb les comandes per a la creació de la base de dades.
* **Helpers:** Conté dos fitxers, els quals són generateToken.js i handlerBcrypt.js. El primer conté el codi per a gestionar els tokens (validar, signar, refrescar), i en el segon el codi per a encriptar la contraseña dels usuaris. Es fa el hash de la contrasenya i se li aplica un salt mitjançant el mòdul *bcryptjs.*
* **Middleware:** Conté el fitxer auth.js amb el codi per a l’autentificació dels tokens generats.
* **Routes:** Conté les rutes que es criden al frontend i s’associen amb les funciones situades a *Controllers.*

En el fitxer **db.js**, tenim la connexió amb la base de dades, i a **index.js** tenim definits certs paràmetres de configuració, com es el port en que rebem les peticions, que serà el 3000, protecció contra atacs DoS mitjançant el mòdul express-rate-limit, el qual et permet definir un màxim de peticions en un cert temps que pots rebre. En **index.js** també tenim protecció contra atacs xss mitjançant el mòdul *helmet,* el qual ens posa a la capçalera certs paràmetres de protecció per evitar aquests atacs, per exemple.

Index.js:



Tota la gestió de la sessió de l’usuari és tracta a través de Json Web Tokens i de Refresh Tokens.

L’objectiu de tenir estructurat així el backend és per a assegurar la bona lectura l’escalabilitat del backend. Les JWT ens donin l'escalabilitat basant-se en el fet que aquells es guarden dins d’emmagatzematge de l’usuari i fan que els servidors no hagin de mantenir estat de les sessions de l’autenticació. Els refresh tokens facin que el servei sigui segur i còmode per l’usuari, però requereixen mecanismes de revocació i manteniment de sessions.

# Base de dades

A l'hora de dissenyar l’aplicació, hem vist que les dades que volíem guardar s’ajustaven bé al model relacional SQL i, per tant, hem decidit usar una base de dades que segueix aquest model. Inicialment, hem usat la BD PostgreSQL, ja que ens donava tota la funcionalitat que volíem, prestava rendiment bon, tenia molt material disponible en internet i finalment era una aplicació de què teníem coneixement bon perquè l'hem estudiat en una assignatura de l'especialitat.

Quan hem començat a desplegar el backend en els servidors, hem vist que volíem tenir una arquitectura distribuïda i hem donat compte que la BD que hem seleccionat inicialment no s'ajustava com volíem a aquest model i que hi havia alternatives més interessants que milloraven aquest aspecte. Al final, hem decidit usar la base de dades CockroachDB, que també és compatible amb els controladors de PostgresDB, que ha sigut molt còmode per nosaltres donat que he pogut utilitzar-lo sense fer molts canvis al programa de backend i ens ha deixat la possibilitat de l’ús de PostgreSQL per desenvolupar el backend localment.

A mesura que estàvem desplegant l'aplicació, hem hagut de fer alguns canvis a l’arquitectura i hem vist alguns aspectes molt interessant de la base de dades. Un d’ells ha sigut que a diferència de PostgreSQL, la BD que hem utilitzat preferia fer ús de UUID en lloc d'ID seqüencials. Un altre aspecte que hem vist és que CockroachDB dóna més garanties per les transaccions ACID que Postgres.

Després d’aplicar i integrar tot, hem obtingut un lloc d’emmagatzematge de dades segur i distribuït que, a més, ens permetia usar l’arquitectura de microserveis escalable, ja que hem pogut fer que totes les dades es guardaven en una BD que podia ser accedida en paral·lel. Hem guardat les dades en 3 taules relacionades entre elles amb l’ús d'intrínseques de SQL. Amb l’ús de Kubernetes, hem usat 3 instàncies paral·leles del servidor de BD que faries tal que fallada d’un servidor no ha de crear cap interrupció del servei per l'usuari.

# Kubernetes

Un dels punts interessants que volíem estudiar amb el nostre projecte ha sigut l'arquitectura serverless amb Kubernetes, ja que ens dona molts avantatges comparat amb l’arquitectura tradicional. Les millores de la nova arquitectura inclouen la reducció dels costos associats amb el manteniment, l’escalabilitat i la possibilitat del desplegament ràpid de les aplicacions de backend. Aquestes característiques han fet que el nou paradigma ha buscat molta popularitat i suport pels molts proveïdors dels serveis i han motivat creació de noves aplicacions que depenen de la nova arquitectura.

Com s’ha discutit abans, hem anat desenvolupant el nostre sistema amb aquesta mentalitat i com el resultat hem obtingut una arquitectura distribuïda que faria ús dels microserveis independents. En el nostre model, hem fet dues aplicacions que donaven servei per l'usuari. La primera aplicació s’encarregava d’autenticació i creació dels tokens. La segona ens donava el servei d'emmagatzematge de les dades generades per l’usuari. A més d’això, hem programat les aplicacions tal que es puguin ser escalades horitzontalment, segons la demanda potencial del servei. Aplicant aquesta arquitectura, hem aconseguit que els serveis es puguin funcionar de manera independent, donant-nos la resiliència, seguretat i escalabilitat. Per exemple, si es produeix disrupció d’una de les aplicacions usades, l’altra pot seguir funcionant donant un servei parcial a l’usuari. Similarment, si cau un servidor, l’arquitectura de Kubernetes ho he de detectar i restaurar el servei sense que l’usuari no noti el problema.

Per fer que els serveis es facin disponible a l’usuari, hem usat un equilibrador de càrrega basat en Nginx. Similarment, als altres serveis, hem situat el servei dins del clúster de Kubernetes per obtenir l’escalabilitat ràpida integrar-ho millor amb les altres aplicacions.

Finalment, en referència del desplegament físic, hem utilitzat 5 màquines virtuals per crear un clúster local. Aquest clúster consistia en un node Master i 4 nodes Worker. Hem utilitzat l’aplicació Flannel per proveir la xarxa dins del clúster.

Combinant-ho tot, hem creat una arquitectura segura, distribuïda i tolerant a fallades, amb totes les aplicacions corrent dins del nostre clúster petit, que pot ser extrapolat a un proveïdor del servei comercial. Cada component té diverses còpies dins del clúster i en conjunt, hem obtingut un disseny que permet que es perdin els servidors sense que l’usuari nota cap diferència del servei. A més, l’arquitectura pot escalar-se segons la demanda del servei. Tot i que no hem aplicat cap política d'escalat dinàmic donat que no teníem la demanda dels usuaris reals, creiem que hi puguin ser aplicades per ajustar a la demanda.

# Resultats obtinguts

En general els resultats que hem obtingut han sigut els que esperàvem. Hem pogut comprovar de primera mà que el model de *machine learning Movenet Lightning* funciona correctament i hem pogut observar el seu potencial. Tot i que a vegades no detecta molt bé els *keypoints* del cos (això és normal ja que es tracta d’un model de menys capacitat que és més ràpid però menys precís) hem comprovat que en la majoria d’exercicis que hem desenvolupat es compten bé les repeticions. No obstant, com a punt final voldríem destacar que en un principi teniem pensat reentrenar el model perquè aquest fos capaç no només de comptar el número de repeticions sino que també fos capaç d’indicar al usuari si estava o no realitzant correctament l’exercici. Però per falta de temps i també per problemes que vam tenir a l’hora d’intentar entrenar al model (a part de que necessitàvem un training set bastant gran) finalment no vam poder implementar-ho al projecte final. Tanmateix, creiem que els resultats obtinguts han sigut molt bons.

# 

# Bibliografia

<https://www.linkedin.com/advice/0/how-do-you-prevent-cross-site-scripting-xss-sql-injection>

<https://planetscale.com/blog/how-to-prevent-sql-injection-attacks-in-node-js>

<https://www.cloudflare.com/learning/serverless/why-use-serverless/>

<https://developers.google.com/ml-kit/vision/pose-detection/android?hl=es-419#kotlin>

<https://github.com/googlesamples/mlkit/tree/master/android/vision-quickstart>

<https://github.com/tensorflow/tfjs-models/tree/master/pose-detection#example-code-and-demos>

<https://github.com/tensorflow/tfjs-models/tree/master/pose-detection/src/movenet>

<https://blog.tensorflow.org/2021/05/next-generation-pose-detection-with-movenet-and-tensorflowjs.html>

<https://storage.googleapis.com/movenet/MoveNet.SinglePose%20Model%20Card.pdf>

<https://tfhub.dev/google/lite-model/movenet/singlepose/lightning/3>

<https://tfhub.dev/google/lite-model/movenet/singlepose/thunder/3>

<https://www.youtube.com/watch?v=06TE_U21FK4>

<https://www.youtube.com/watch?v=fiDaAc7z_kQ>

<https://www.tensorflow.org/hub/tutorials/movenet?hl=es-419>

<https://blog.tensorflow.org/2021/08/pose-estimation-and-classification-on-edge-devices-with-MoveNet-and-TensorFlow-Lite.html>

<https://www.tensorflow.org/lite/tutorials/pose_classification?hl=es-419>

<https://www.youtube.com/watch?v=SSW9LzOJSus>

<https://www.youtube.com/watch?v=1hpc70_OoAg>

<https://www.youtube.com/watch?v=i3OdKwuBjeM>

<https://beeware.org/>