Phizi

Application Design Document

Members:

Batya Bialik, Itay Mor, Aviehu Yudilevitch, Hila Klein

Contents

Chapter 1

Use Cases

1.1)

Use-case name		**Pose matching request**
Actors		Server, Client
Preconditions		The client connected to the server and the user logged in to the system
Normal flow	Description	The server sends to the client a position the user should perform.
	Post conditions	The position is shown on the user's screen.
Alternative flows		• Communication error between the client and the server.

1.2)

Use-case name		**Pose matching performance**
Actors		Server, Client, User, Database
Preconditions		Pose matching request completed successfully. The customer has a camera installed on his device.
Normal flow	Description	The user performs the requested pose. The client examines the pose and send the result to the server and show feedback to the user.
	Post conditions	The database is updated accordingly to the user progress.
Alternative flows		• Communication error between the client and the server. • Database failed to update.

2.1)

Use-case name		**Register a new user**
Actors		User, Server, Client
Preconditions		The Server is running, and a valid data base exist. The user is logged-in as a therapist.
Normal flow	Description	• The therapist enters all the needed information (email, password, medical information etc.) • The client sends the server a request containing the information. • The server adds the user's information to the DB and responds with status 200
	Post conditions	A new patient is registered.
Alternative flows		• Communication error between the client and the server and an error is displayed. • Email already taken or Password isn't secured enough and an error is displayed.

2.2)

Use-case name		**login**
Actors		User, Client, Server
Preconditions		The user is registered to the system
Normal flow	Description	The user clicks on the "login" button. A "login" request is sent from the client to the server. The server sends an indication about a successfully login to the client.
	Post conditions	The user is successfully logged-in and now can use the system's interface.
Alternative flows		• Communication error between the client and the server. • Failed to log-in. A message will be shown on the user's screen.

2.3)

Use-case name		**Logout user**
Actors		User, Server, Client
Preconditions		The Server is running, valid data base exists and user is logged in.
Normal flow	Description	• The user presses the logout button. • The client sends a request to the server to log out the user. • The server logs out the user
	Post conditions	User is logged out of the system and can't preform any action until he logs back in
Alternative flows		• Communication error between the client and the server and an error is displayed.

3.1)

Use-case name		**Select session**
Actors		User, Server, Client
Preconditions		The Server is running, user is logged in and the client is running.
Normal flow	Description	• The client sends a request to the server to receive a list of sessions available to the user. • The server responds with the user's possible sessions. • The client selects the wanted session to play.
	Post conditions	An image displaying the first pose that the user needs to achieve.
Alternative flows		• Communication error between the client and the server and an error is displayed. • Session started before the earlier session ended and an error is displayed and the earlier session ends.

3.2)

Use-case name		**Play session**
Actors		User, Client
Preconditions		The Server is running, user is logged in and the client is running. The user has selected a session to play.
Normal flow	Description	• The user presses the start session button. • The client starts the session for the user to play.
	Post conditions	• A 2D skeleton is attached to the user's video. • The user skeleton will be matched against the target pose on each frame.
Alternative flows		• Communication error between the client and the server and an error is displayed. • Session started before the earlier session ended and an error is displayed and the earlier session ends.

3.3)

Use-case name		**End session**
Actors		User, Server, Client
Preconditions		The Server is running, user is logged in and the client is running
Normal flow	Description	• The last target-pose of the session is matched successfully. • An animation will be displayed to the user. • The client sends a request to the server with the user's score. • The server stops the user's session and saves the relevant data to the DB. • The client ends the session for the user.
	Post conditions	The client is running but the session has ended. The DB contains a new record for that session

Alternative flows		• Communication error between the client and the server and an error is displayed. • No current running session for user and an error is displayed.

4)

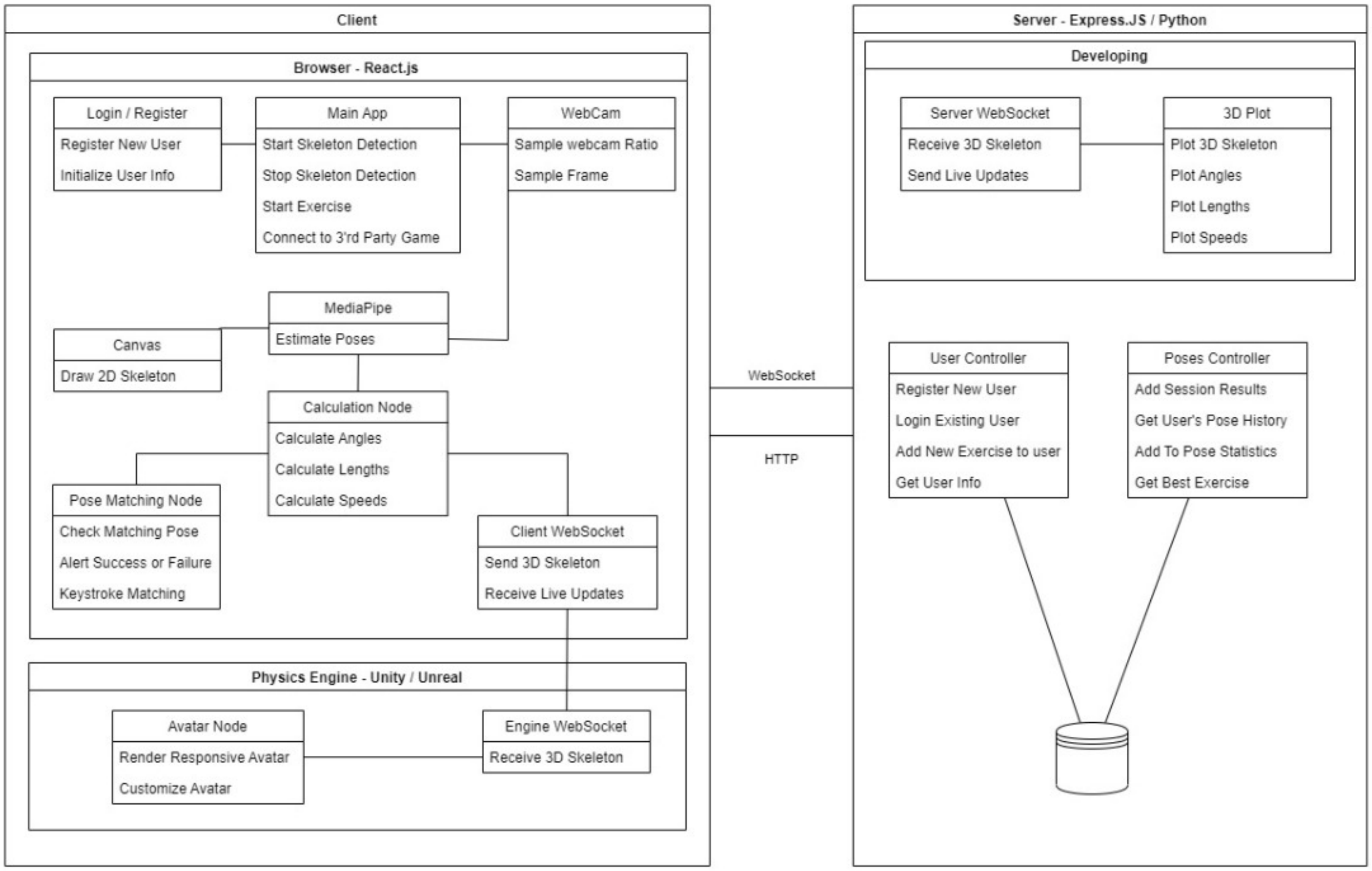
Use-case name		**Add pose**
Actors		User, Server, Client
Preconditions		The Server is running, user is logged in as a therapist and the client is running
Normal flow	Description	• The therapist navigates to add pose page. • The therapist selects to upload an existing image or take a picture himself. • The selected image is passed through the pose-estimating model. • The Client sends the pose to the server. • The Server stores the new pose in the DB.
	Post conditions	A new pose is added to the DB and a success message is displayed to the user.
Alternative flows		• Communication error between the client and the server and an error is displayed.

5)

Use-case name		**Add session**
Actors		User, Server, Client
Preconditions		The Server is running, user is logged in as a therapist and the client is running
Normal flow	Description	• The therapist navigates to add session page. • The therapist fills up the requested data to a session (poses, difficulty, goals, etc.) • The Client sends the session to the server. • The Server stores the new session in the DB.
	Post conditions	A new session is added to the DB and a success message is displayed to the user.
Alternative flows		• Communication error between the client and the server and an error is displayed.

Chapter 2

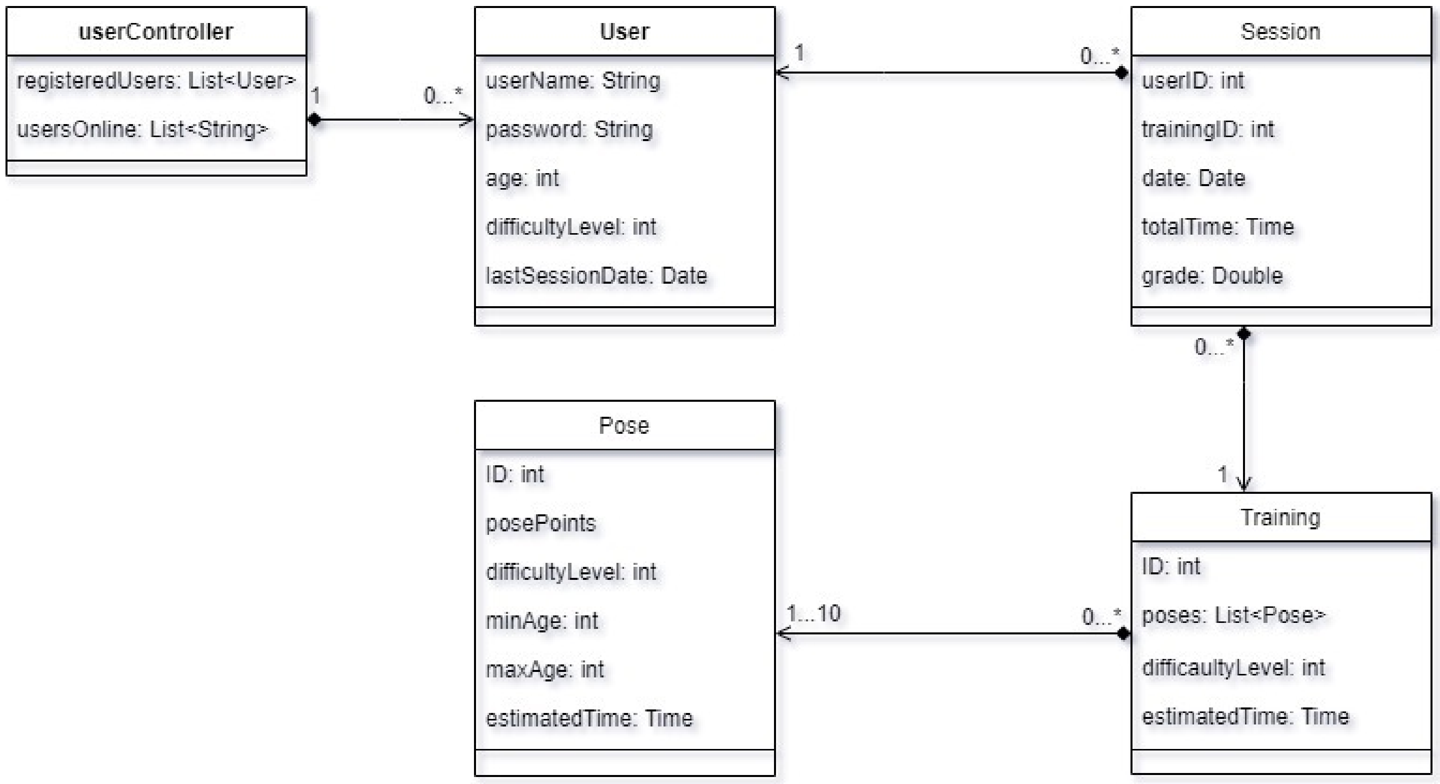
System Architecture



Chapter 3

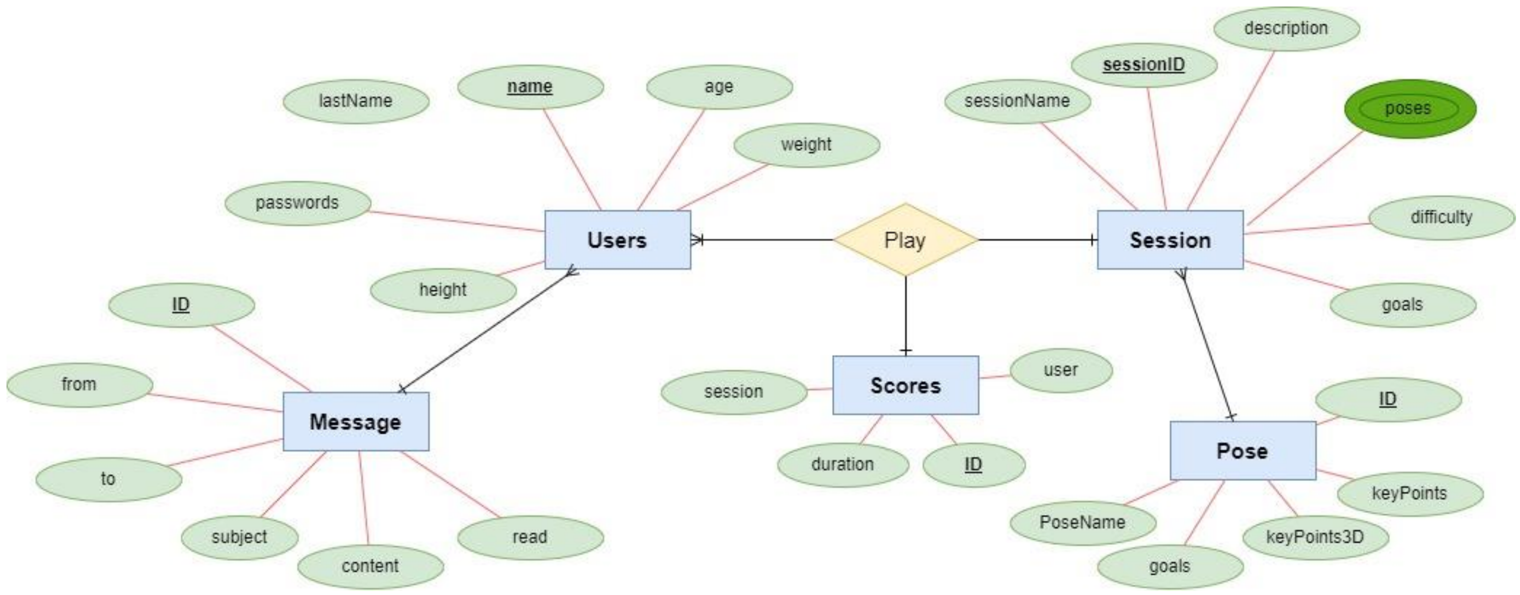
Data Model

3.1) Description of Data Objects and their relationships (3.2)



3.2) Databases

ERD



Tables:

Users:

name	password	email	age	weight	height	BMI	goals

Poses:

name	goals	Keypoints	Keypoints3D

Scores:

ID	user	session	duration

Sessions:

name	description	difficulty	poses	goals

Messages:

ID	from	to	subject	content	read

Main transactions:

Register- the user table modifies by adding a new record of the new user who registered.

Add Pose – The pose table is modified by adding a new record.

Add Session – The sessions table is modified by adding a new record.

Session end – A new score record is added to the scores' table.

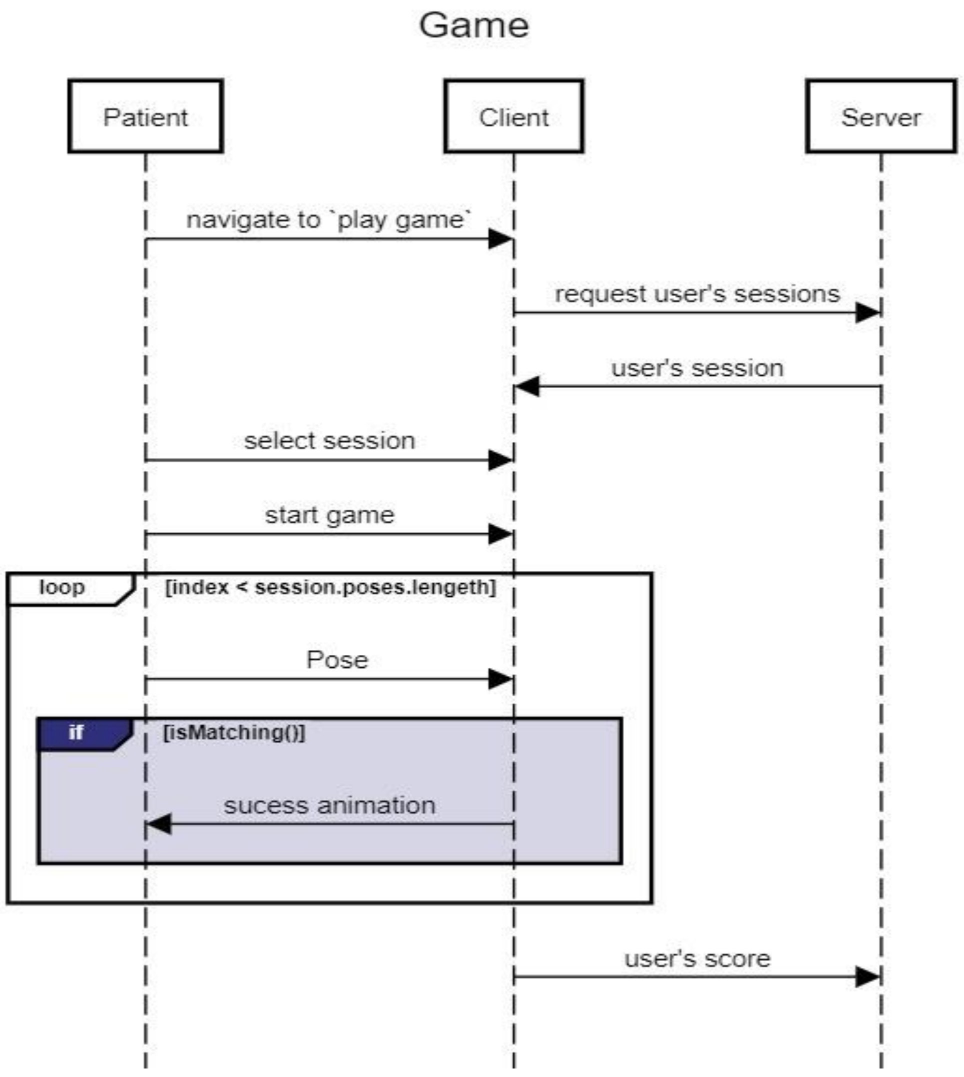
Login – A record is pulled from the users' table in the database based on the email adress and password of the user.

Chapter 4

Behavioral Analysis

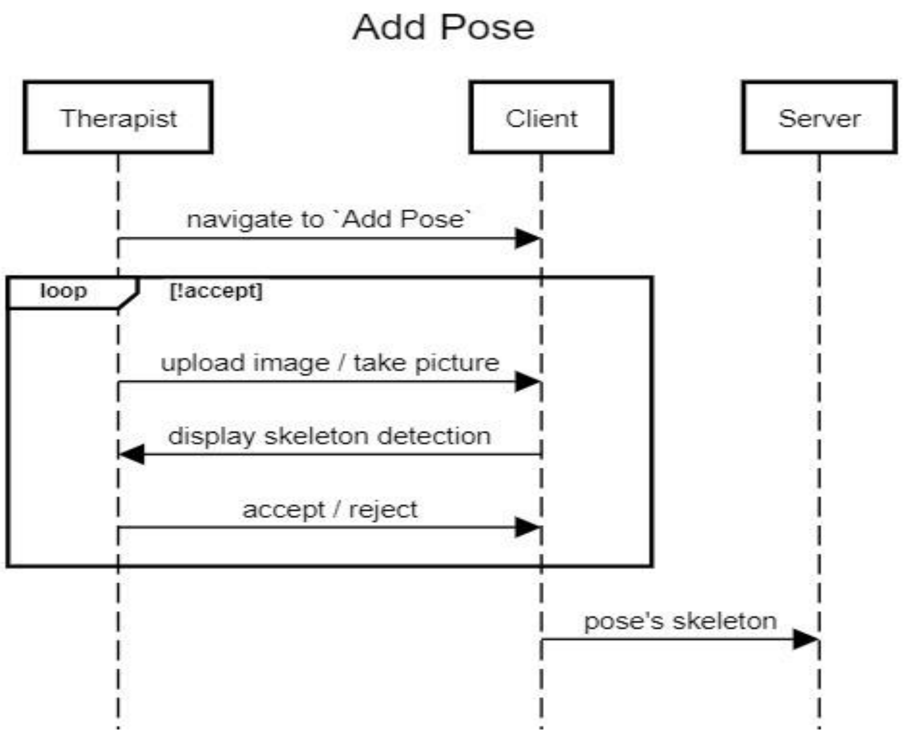
4.1) Sequence Diagrams

1. Game



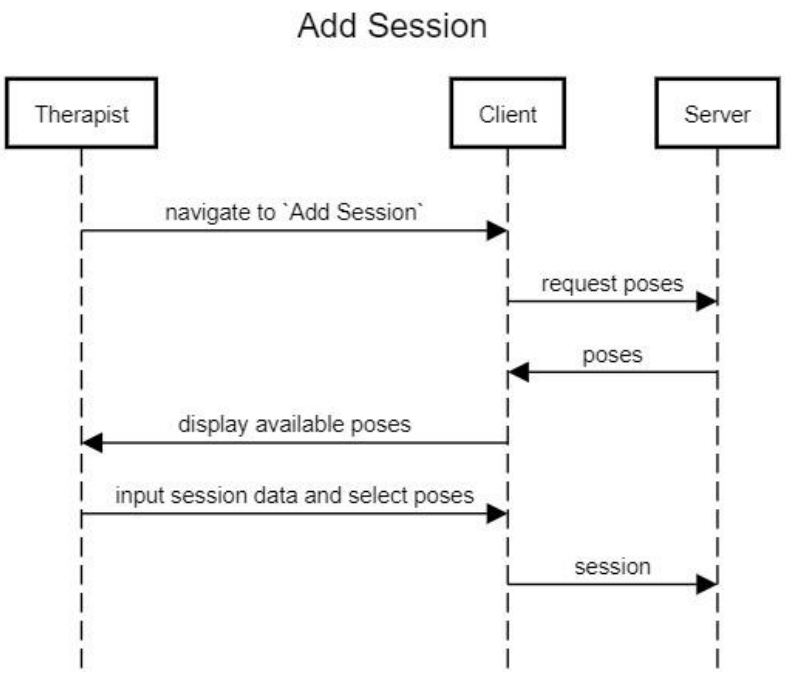
4.1
4.2
4.3

2. Add pose

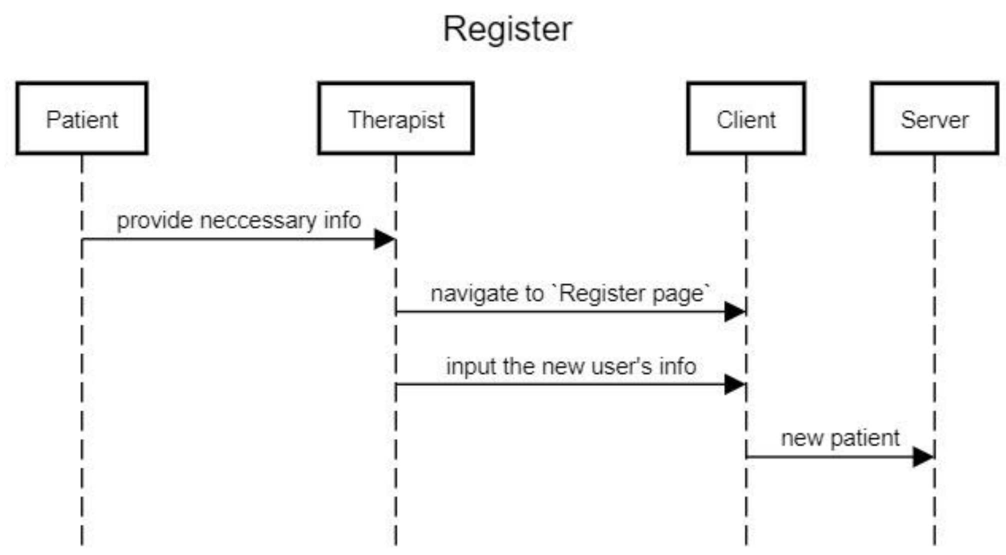


4.2)
4.3)
4.4)
4.5)
4.6)

3. Add session



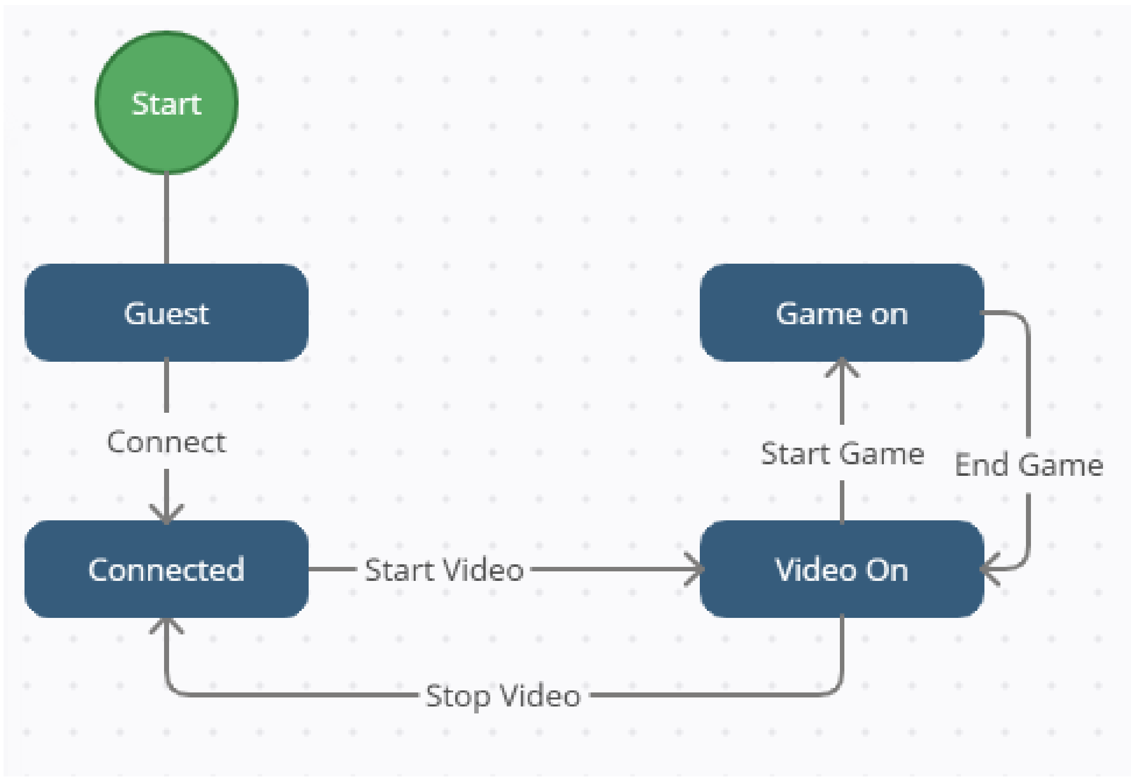
4. Register



4.2) Events

- Upon user registration, the system will save the user information in the database in a secured way.
- Upon starting a game, a pose will be displayed on the user screen.
- Upon the user successfully mitigate the displayed pose, the next pose is presenting or the game ends.

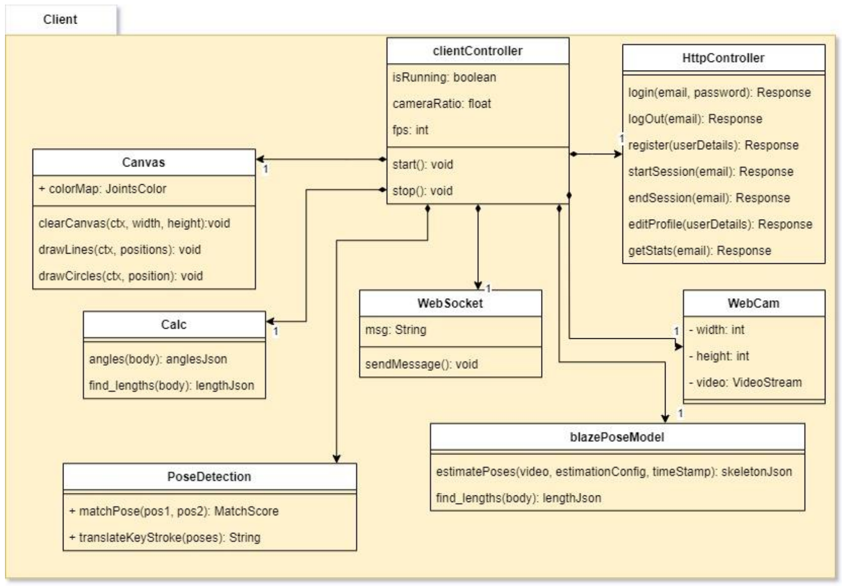
4.3) States

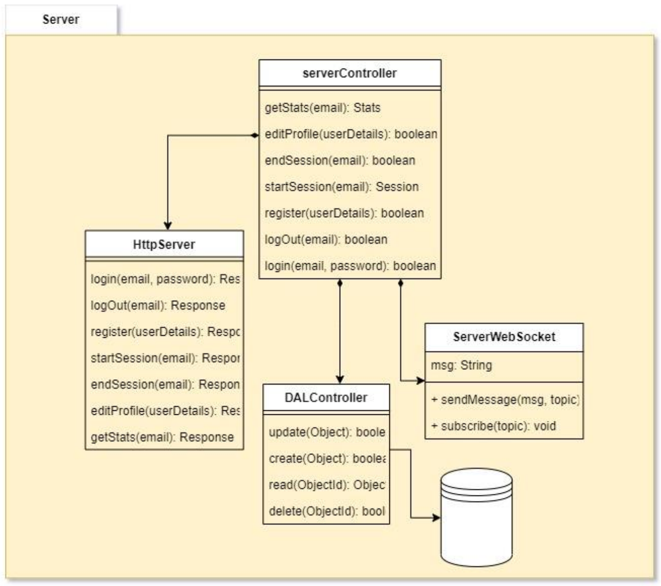

Start
Guest
Connect
Connected
Start Video
Video On
Start Game
Game on
End Game
Stop Video

Chapter 5

Object-Oriented Analysis

5.1) Class Diagrams


Client
clientController
isRunning: boolean
cameraRatio: float
fps: int
start(): void
stop(): void
HttpController
login(email, password): Response
logOut(email): Response
register(userDetails): Response
startSession(email): Response
endSession(email): Response
editProfile(userDetails): Response
getStats(email): Response
Canvas
+ colorMap: JointsColor
clearCanvas(ctx, width, height):void
drawLines(ctx, positions): void
drawCircles(ctx, position): void
Calc
angles(body): anglesJson
find_lengths(body): lengthJson
WebSocket
msg: String
sendMessage(): void
WebCam
- width: int
- height: int
- video: VideoStream
blazePoseModel
estimatePoses(video, estimationConfig, timeStamp): skeletonJson
find_lengths(body): lengthJson
PoseDetection
+ matchPose(pos1, pos2): MatchScore
+ translateKeyStroke(poses): String
1


Server
serverController
getStats(email): Stats
editProfile(userDetails): boolean
endSession(email): boolean
startSession(email): Session
register(userDetails): boolean
logOut(email): boolean
login(email, password): boolean
HttpServer
login(email, password): Res
logOut(email): Response
register(userDetails): Respo
startSession(email): Respor
endSession(email): Respon
editProfile(userDetails): Res
getStats(email): Response
ServerWebSocket
msg: String
+ sendMessage(msg, topic)
+ subscribe(topic): void
DALController
update(Object): boole
create(Object): boolea
read(ObjectId): Objec
delete(ObjectId): bool

5.2) Class Description

<u>Client Package</u>

BlazePoseModel – an API for pose detection developed by Google that can compute (x, y, z) coordinates of 33 skeleton key points.

<u>Main methods:</u>
- estimatePoses: (async)
    - invariants – the user's webcam is on during the whole session.
    - pre-condition - the user is logged-in.
    - post-conditions – a set of 33 3D key points is exported.

Calc - Analyzes the 33 skeleton key points (x, y, z) blazepose output and provides additional data about the skeleton.

Main methods:

- calcAngles:
    - pre-conditions – the blazepose model exported a set of 3D key points.
    - post-conditions – a dictionary of joints and their corresponding angles has been created.

- calcLength:
    - pre-conditions – the blazepose model exported a set of 3D key points.
    - post-conditions – a dictionary of body parts and their corresponding lengths has been created.

Canvas – Draws a 2D skeleton on the client's video stream.

Main methods:

- drawLines:
    - invariants – the user's webcam is on during the whole session.
    - pre-conditions - the blazepose model exported a set of 3D key points and a list of pose pairs was declared.
    - Post-conditions – lines representing the body parts (thigh, arm, etc.) as a 2D skeleton is presented on the client screen.

- drawCircles:
    - invariants – the user's webcam is on during the whole session.
    - pre-conditions - the blazepose model exported a set of 3D key points.

    - Post-conditions – points representing the body key points (nose, shoulder, elbow, etc.) as a 2D skeleton is presented on the client screen.

PoseDetection – Analyzes data about the user's movement (including the 3D key points, angles, etc.)

<u>Main methods:</u>

- matchPose:
    - invariants – the user's webcam is on during the whole session.
    - pre-conditions - the blazepose model exported a set of 3D key points and calc class had finished its analyzes.
    - Post-conditions – A score of the accuracy of the user's pose compared to the pose he was asked to preform.

ClientController – Controls the client-side operations.

<u>Main methods:</u>

- Start game:
    - pre-conditions – the client is logged-in, his webcam is on, web socket is successfully connected.
    - post-conditions – 2D and 3D skeletons are presented based on the client's video. In addition, a json file of the user's data (joints, angles, 3D keypoints, timestamp, etc.)

- stop game:
    - pre-conditions – the user is in "start" state.
    - post-condition – the session has been stopped and the canvas is cleared.

HttpController – Responsible for all the client-server communication related to user's opeations.

<u>Main methods:</u> (all the functions are async)

- register:
    - pre-conditions – the user has not been registered to the system.
    - post-conditions – the user has been successfully registered and the user's data (mail, hashed password, medical-condition, etc.) has been sent to the server.

- login:
    - pre-conditions – the user has registered to the system.
    - post-conditions – the user is logged in and now can start using the app.

- startSession:
    - invariants - the user is logged in and his webcam is on.
    - pre-conditions - the user is logged in and his webcam is on.
    - post-conditions - a session corresponding the user's data has been created and presented on the user's screen.

- endSession:
    - pre-conditions - a user is in the startSession state.
    - post-conditions - details about the user's current session (score, total time, etc.) have been sent to the server.

- getStats:
    - pre-conditions – the user has registered to the system.
    - post-conditions – statistics representing the user's progress is presented on the user's screen.

Server Package

ServerController – Controls the server-side operations.

Main methods: (All the methods are async)

- register:
  - pre-conditions – the user's data is not in the database.
  - post-conditions – the user has been successfully registered and the user's data (mail, hashed password, medical-condition, etc.) is stored in the database.

- login:
  - pre-conditions – the user's data is in the database.
  - post-conditions – an indication about the login is sent to the client.

- startSession:
  - pre-conditions - the user's data is in the database.
  - post-conditions - a session corresponding the user's data has been sent to the client.

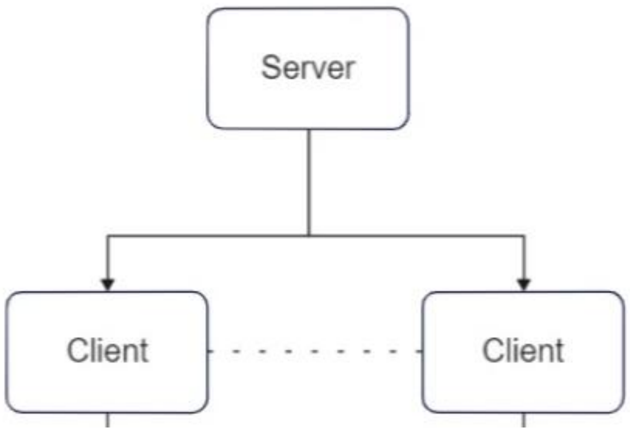
- endSession:
  - pre-conditions - the user's data is in the database.
  - post-conditions - details about the user's current session (score, total time, etc.) are updated and saved in the server.

- getStats:
  - pre-conditions - the user's data is in the database. post-conditions – statistics representing the user's progress has been sent to the client.

HttpServer – The purpose of this class is to wrap all the serverContoller's messages in order to send them properly to the client-side.

DALController – The purpose of this class is to manage all communication with the database.

5.3) Packages



Client – Web application that captures the user's webcam and provides a 3D skeleton with additional calculations in a json format via websocket.
Runs on the user's device.
Server – Deployed on a cloud platform.

- Http – handles the user's information and determines the session per user.
- WebSocket – a development communication with the client.

5.4) Unit Testing

Calc -

- calcAngles:
  - Test the method with assert of error if given a wrong 3D key points.
  - Test the method with assert of correct angle between 3 3D points.
  - Test the method with assert of wrong angle between 3 3D points.

- calcLength:
  - Test the method with assert of error if given a wrong 3D key points.
  - Test the method with assert of correct length between 2 3D points.
  - Test the method with assert of wrong length between 2 3D points.

PoseDetection –

- matchPose:
  - Test the method with disconnected camera and expect an exception.
  - Test the method with 2 resemble set of points and assert correction.
  - Test the method with 2 different set of points and assert failure.

ClientController –

- Start game:
  - Test the method while user webcam if off and expect an exception.
  - Test the method while user is logged-out and expect an exception.
  - Test the method while web socket is not connected

    - and expect an exception.
    - Test the method when meet all the pre-condition and check the exported json file.

- Stop game:
    - Test the method while user has not started and expect an exception.
    - Test the method while user is in "start" state and assert canvas is clean.

HttpController –

- register:
    - Test the method with registered email and expect an exception.
    - Test the method with weak password and expect an exception.
    - Test the method with correct data and assert user exists.

- login:
    - Test the method with unregistered username and expect an exception.
    - Test the method with registered user and correct password and assert user logged in.
    - Test the method with registered user and uncorrect password and expect an exception.

- startSession:
    - Test the method while user is logged-out and expect an exception.
    - Test the method while user webcam if off and expect an exception.
    - Test the method with valid user and assert session presented

- endSession:
    - Test the method while user has not started session and expect an exception.
    - Test the method after user started session and assert the information sent.
- getStats:
    - Test the method while user is logged-out and expect an exception.
    - Test the method with valid user and assert stats presented

ServerController –

- register:
    - Test the method while user's data is not in the database and expect an exception.
    - Test the method with weak password and expect an exception.
    - Test the method with correct data and assert user created.

- login:
    - Test the method with unregistered username and expect an exception.
    - Test the method with registered user and correct password and assert user logged in.
    - Test the method with registered user and uncorrect password and expect an exception.
- startSession:
    - Test the method while user is logged-out and expect an exception.
    - Test the method while user webcam if off and expect an exception.
    - Test the method with valid user and assert session sent

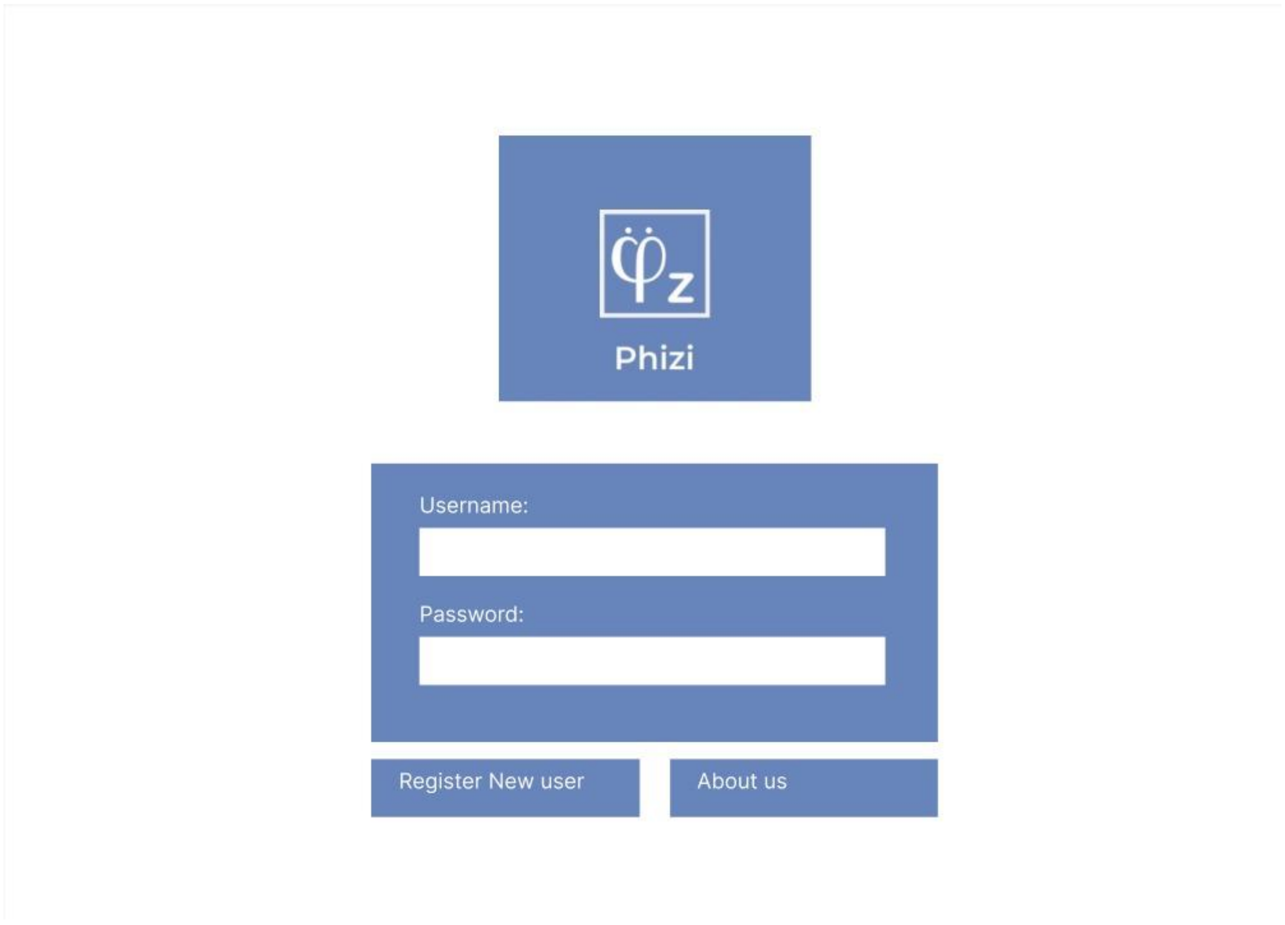
- endSession:
    - Test the method while user has not started session and expect an exception.
    - Test the method after user started session and assert the information received.

- getStats:
    - Test the method while user is logged-out and expect an exception.
    - Test the method with valid user and assert stats sent

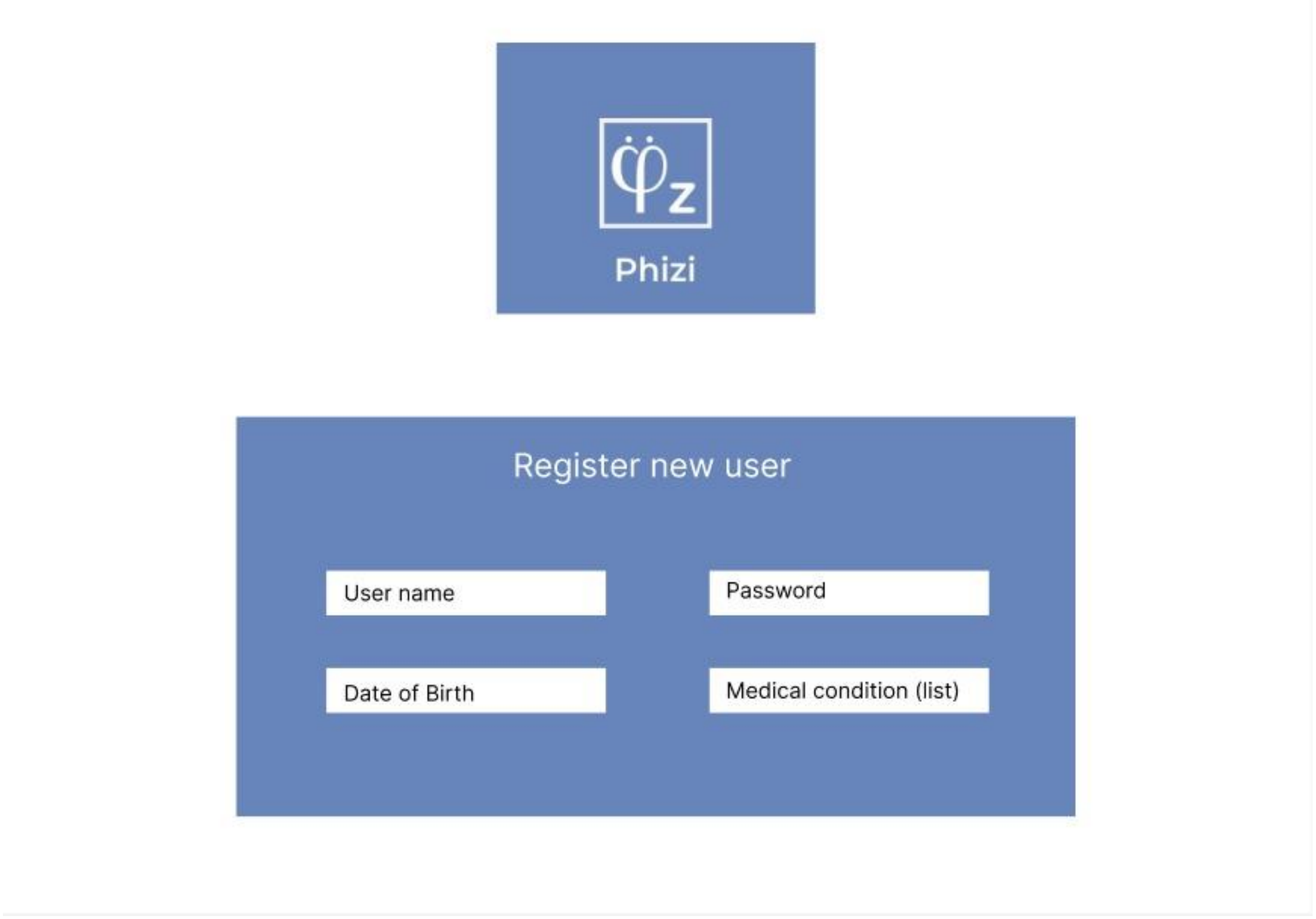
Chapter 6

User Interface Draft

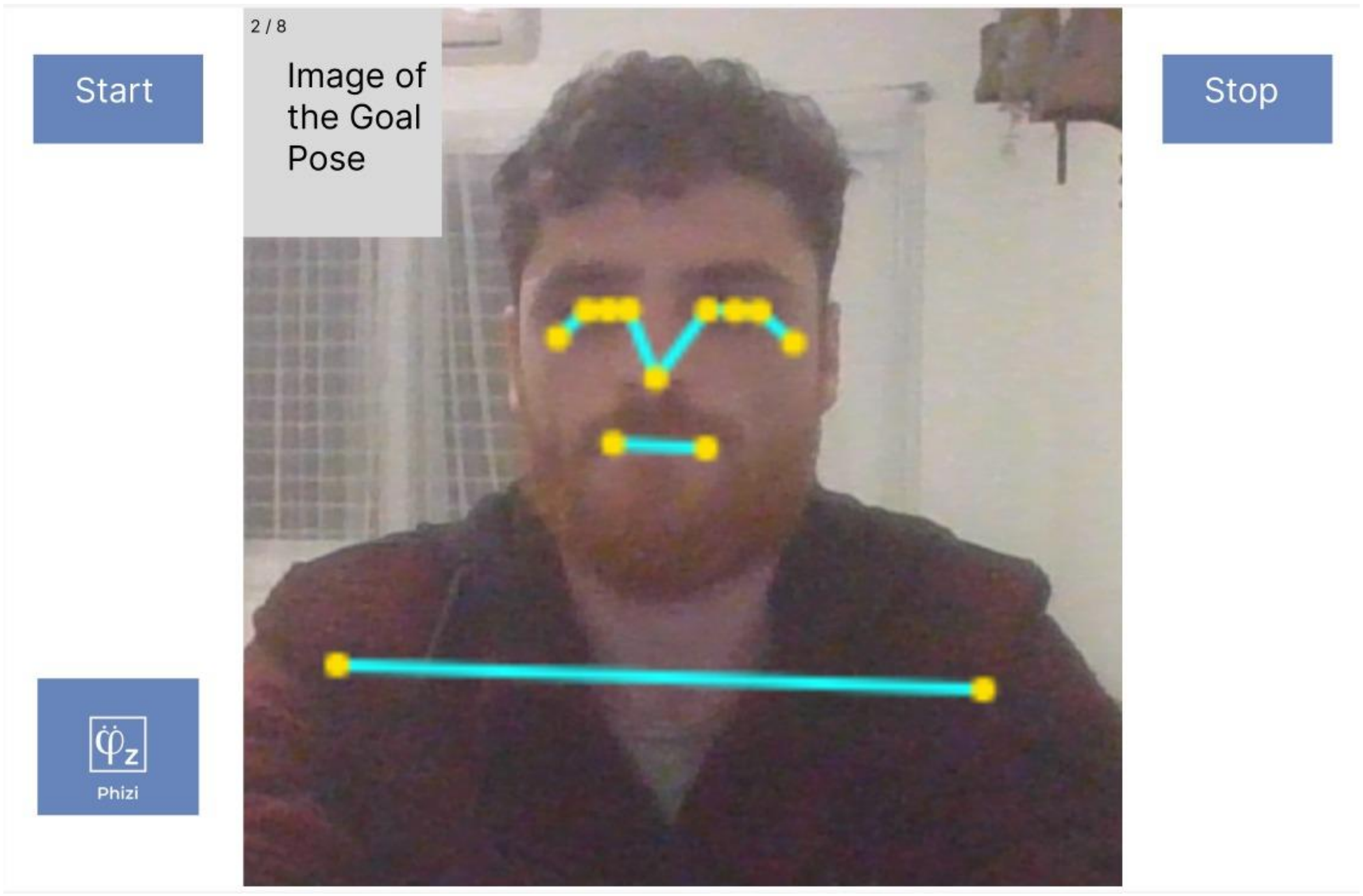
<u>Login Page:</u>



Register Page:



Main Page:



- המציג הינו דוגמן שלדים

Chapter 7

Testing

To test the entire system, we plan on creating a test plan that includes:

- Server API tests that will test all the available routes.
- Unit testing for complicated calculations or functions.
- tests for each of the non-functional constraints listed in the ARD. For example:

Performance: We will conduct tests to measure the frame rate of the system and ensure that it meets the minimum requirement of 30FPS. This will be done by running the system under various conditions and measuring the frame rate.

Reliability: To test the reliability of the pose-matching feature, we will conduct tests with users of different physical attributes to ensure that the system can accurately match poses regardless of the user’s physical attributes.

Usability: To test the usability of the system, we will conduct user testing with children to ensure that the system is easy to use and understandable for them.

Format: To test the format of the points output, we will verify that the output file is in JSON format and that it contains the expected data.

For each subsystem, we will create specific tests to verify its functionality. These tests will include steps to conduct each test, expected results, and actual results obtained.

Server Test Suite

tests/controllers/poseController.test.js

Pose controller

POST /api/poses/addPose

- should create a new pose
- should avoid create a new pose with the same name

POST /api/poses/updatePose

- should update a pose by name

GET /api/poses/getPose

- should return a pose by name
- should return an error if pose is not found

GET /api/poses/deletePose

- should delete a pose by name
- should return an error if pose is not found

GET /api/poses/getAllPoses

- should return all poses

tests/controllers/scoreController.test.js

Score controller

POST /api/scores/addScore

- should create a new score
- should return an error if user is missing

GET /api/scores/getSessionScores

- should return scores by session
- should return an empty array if could not found scores

GET /api/scores/getUserScores

- should return scores by user email
- should return an empty array if could not found scores

tests/controllers/sessionController.test.js

Session controller

POST /api/sessions/addSession

- should create a new session
- should avoid create a new session with the same name

POST /api/sessions/updateSession

- should update a session by name

GET /api/sessions/getSession

- should return a session by name
- should return a 404 error if session is not found

GET /api/sessions/deleteSession

- should delete a session by name
- should return an error if session is not found

GET /api/sessions/getAllSessions

- should return all sessions

tests/controllers/userController.test.js

User controller

POST /api/users/register

- should create a new user
- should return an error if email is missing

POST /api/users/updateUser

- should update a user by email

GET /api/users/getUser

- should return a user by email
- should return a 404 error if user is not found

GET /api/users/getAllUsers

- should return all users

<u>Client Test Suite</u>

getPoints.test.js

get2DPositions

get2DPositions_success

- should return expected result when given valid input

get2DPositions_withPointsBelowThreshold

- should return expected result when given input with points below threshold

get2DPositions_allPointsBelowThreshold

- should return expected result when all points are below threshold

get3DPositions

get3DPositions_success

- should return expected result when given valid input

get3DPositions_withPointsBelowThreshold

- should return expected result when given input with points below threshold

get3DPositions_allPointsBelowThreshold

- should return expected result when all points are below threshold

testCalc.test.js

dist

dist_validPoints1

- should return the distance between two valid points

dist_validPotins2

- should return the distance between two valid points

dist_nullPoint

- should return -1 if one of the points is null

dist_nullPoints

- should return -1 if both points are null

findCoord

findcoord_withMatchingPart

- should return the coordinates of a matching part

findcoord_withoutMatchingPart

- should return null if there is no matching part

findcoord_withEmptyBody

- should return null if the body is empty

findAngle

findAngle_validPoints1

- should return the angle between three valid points

findAngle_validPoints2

- should return the angle between three valid points findAngle_nullPoints

- should return NaN if one of the points is null

calculateAngle

calculateAngle_validPoints1

- should return an object with name and angle properties for three valid points

calculateAngle_validPoints2

- should return an object with name and angle properties for three valid points

calculateAngle_nullPoints

- should return -1 if one of the points is null

calculateAngle_withOnlyTwoPoints

- should return -1 if there are only two points

calcAngles

calcAngles_validPositions

- should return an array of angles for valid positions

calcAngles_missingPositions

- should return an array of -1 values if positions are missing

calcAngles_emptyPositions

- should return an array of -1 values if positions are empty

poseMatching.test.js

isMatching

isMatchingWithinThreshold_true

- should return true if the pose is matching within the threshold

isMatchingFalse

- should return false if the pose is not matching

isMatchingFalse_userWithoutAllPoints

- should return false if the user pose does not have all points