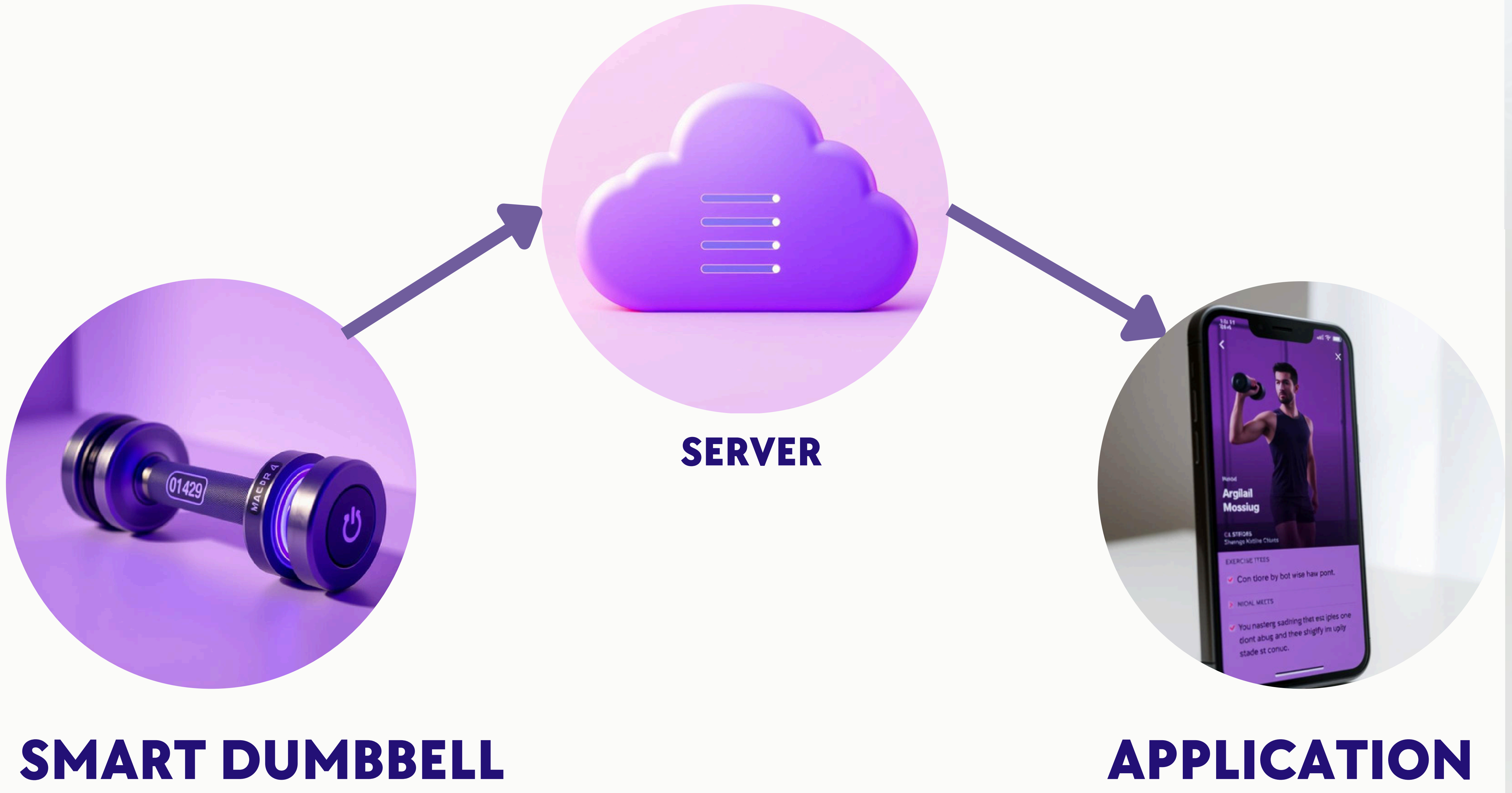
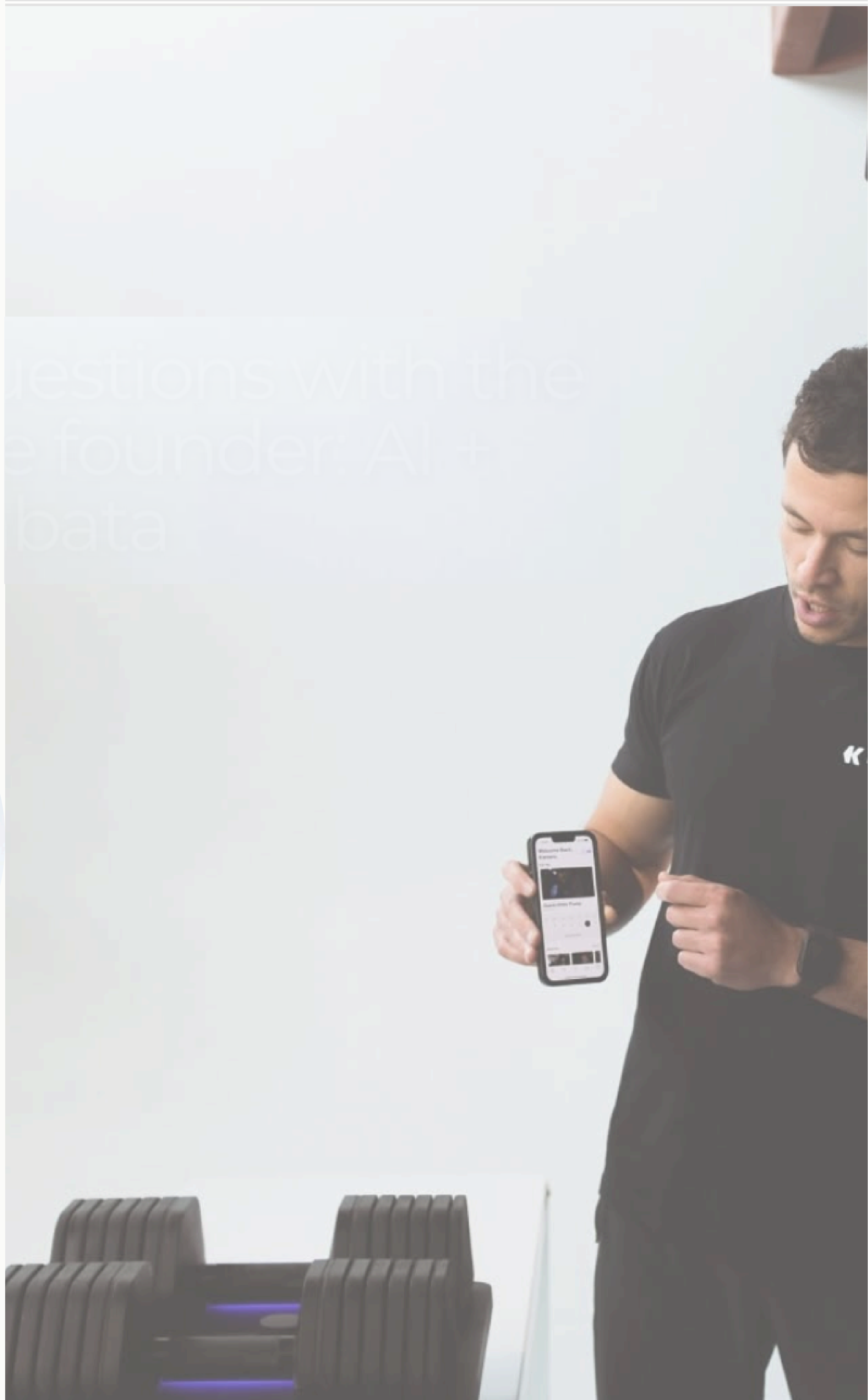
IOT PROJECT

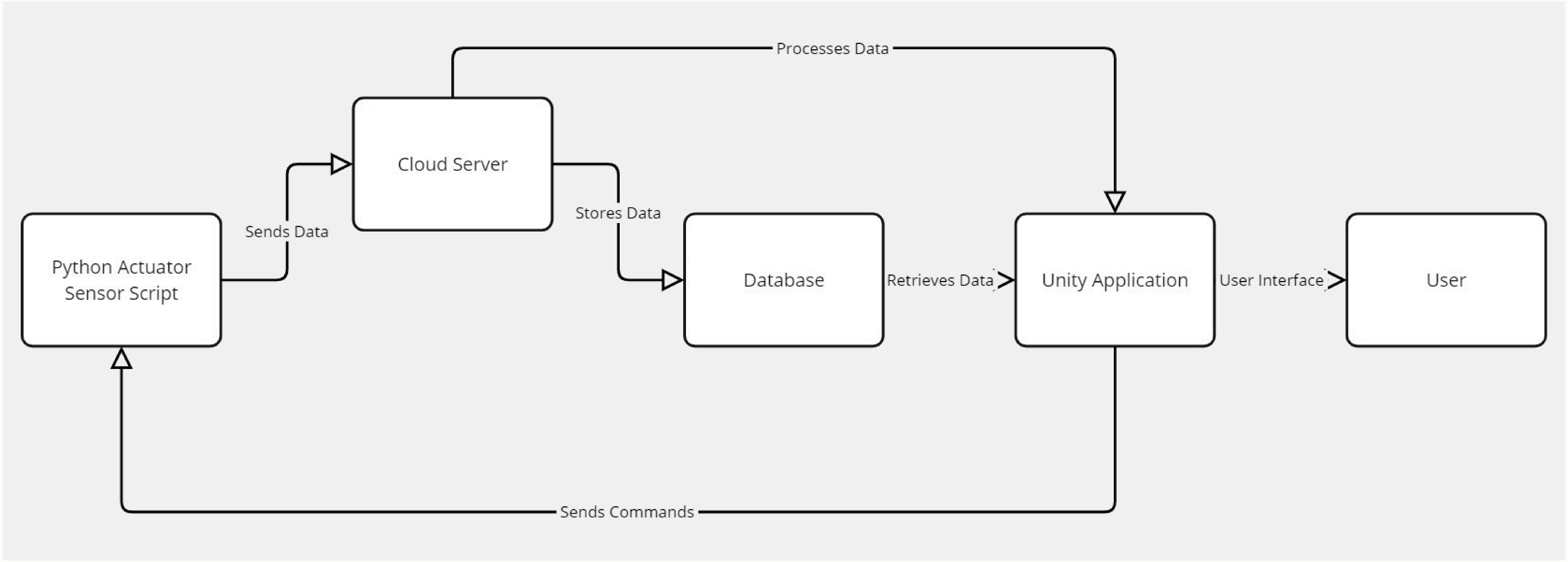
SMART DUMBBELLS

INTRODUCTION



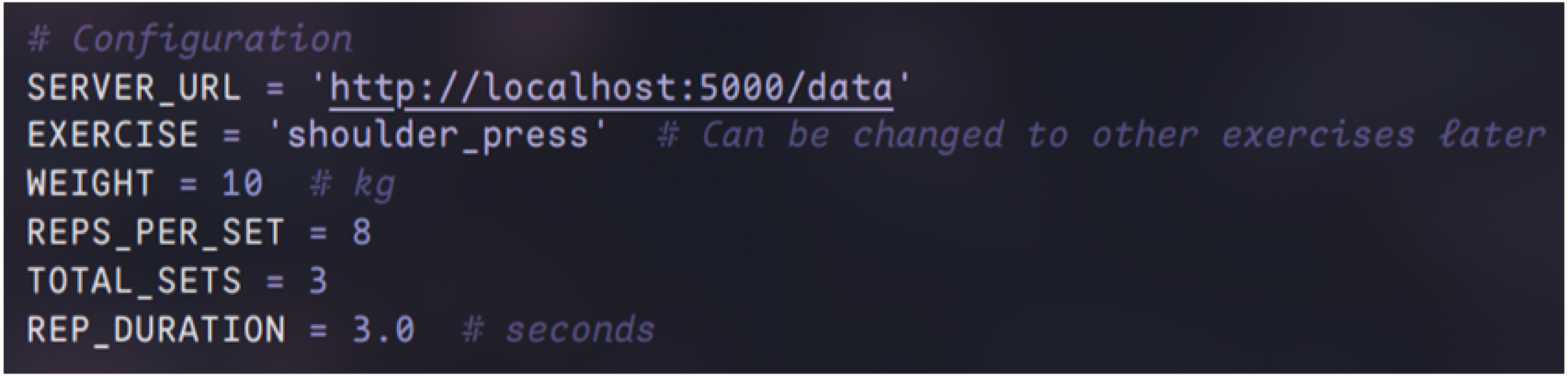


DESIGN DATA FLOW DIAGRAM



Python script : sensor/actuator

The main configuration parameters



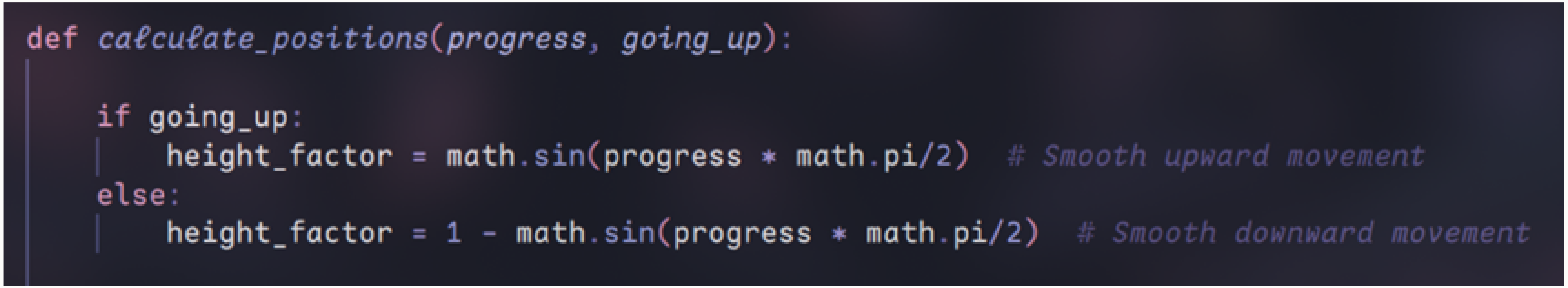
```python
# Configuration
SERVER_URL = 'http://localhost:5000/data'
EXERCISE = 'shoulder_press'  # Can be changed to other exercises later
WEIGHT = 10  # kg
REPS_PER_SET = 8
TOTAL_SETS = 3
REP_DURATION = 3.0  # seconds
```

Python script : sensor/actuator

Calculate positions for both dumbbells and elbows based on exercise progress

progress: 0 to 1 representing movement progress

going_up: True if movement is upward, False if downward

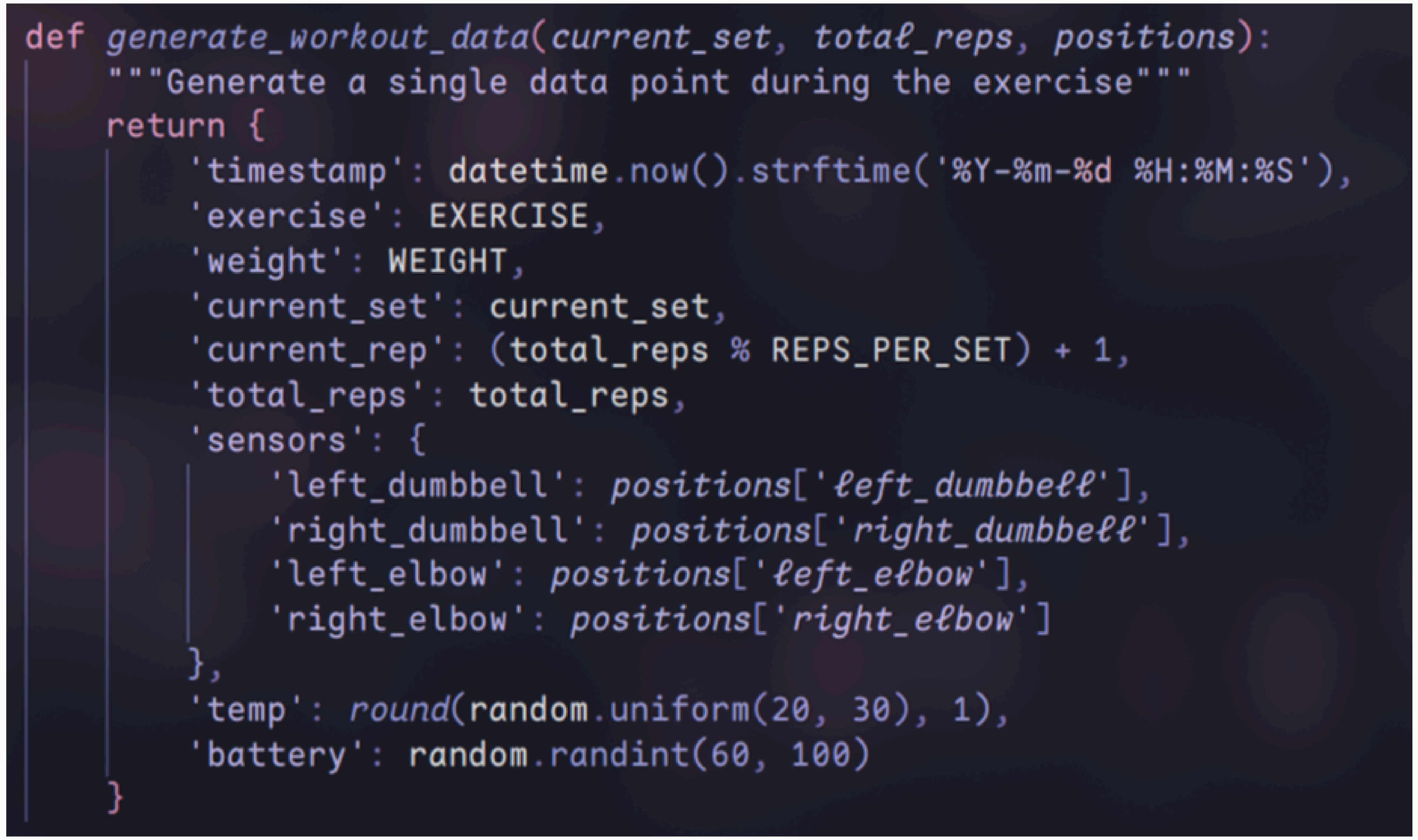


```python
def calculate_positions(progress, going_up):

    if going_up:
        height_factor = math.sin(progress * math.pi/2)  # Smooth upward movement
    else:
        height_factor = 1 - math.sin(progress * math.pi/2)  # Smooth downward movement
```

Python script : sensor/actuator

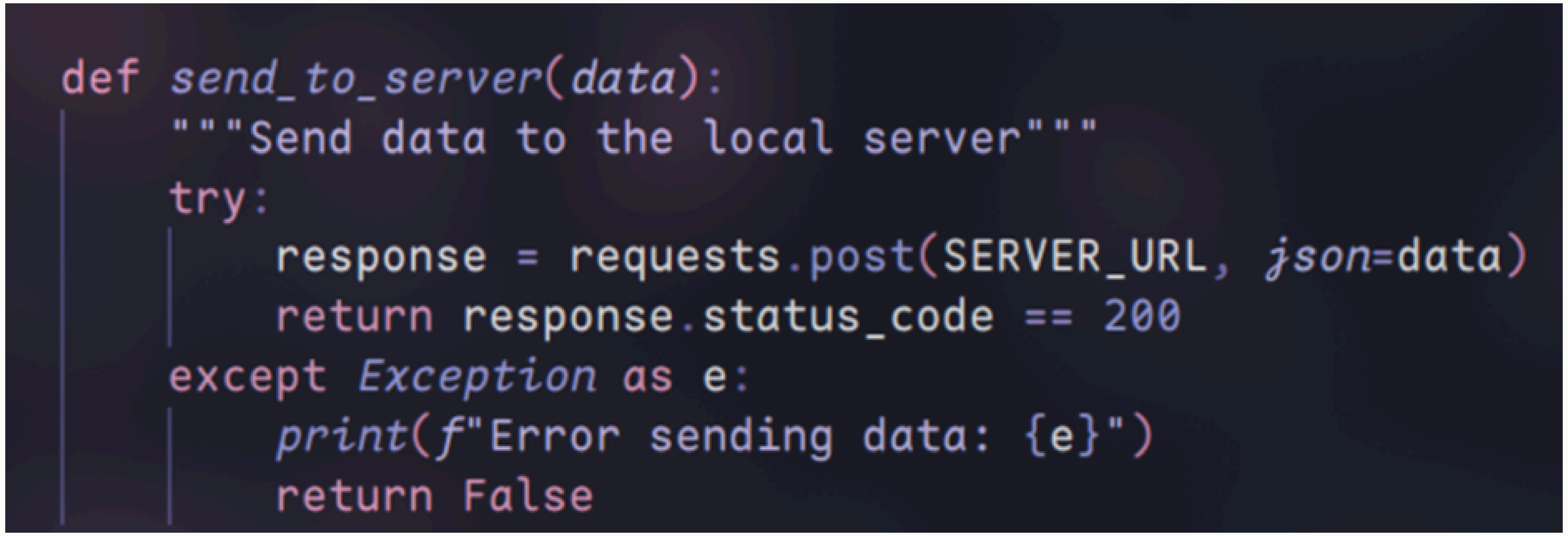
This function gathers all the important information about a single movement during the exercise. This includes things like the time, the weight used, which set and rep you're on, and where your arms and dumbbells are. It then packages all this information neatly into a single message that can be sent to the server.



```python
def generate_workout_data(current_set, total_reps, positions):
    """Generate a single data point during the exercise"""
    return {
        'timestamp': datetime.now().strftime('%Y-%m-%d %H:%M:%S'),
        'exercise': EXERCISE,
        'weight': WEIGHT,
        'current_set': current_set,
        'current_rep': (total_reps % REPS_PER_SET) + 1,
        'total_reps': total_reps,
        'sensors': {
            'left_dumbbell': positions['left_dumbbell'],
            'right_dumbbell': positions['right_dumbbell'],
            'left_elbow': positions['left_elbow'],
            'right_elbow': positions['right_elbow']
        },
        'temp': round(random.uniform(20, 30), 1),
        'battery': random.randint(60, 100)
    }
```

Python script : sensor/actuator

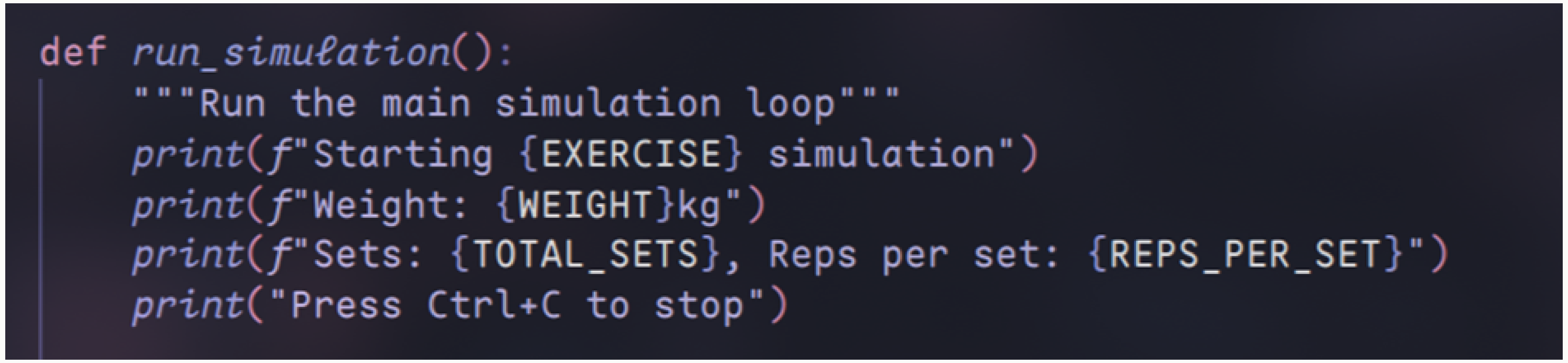
This function is responsible for establishing a connection with the local server and sending the generated workout data to it.



```python
def send_to_server(data):
    """Send data to the local server"""
    try:
        response = requests.post(SERVER_URL, json=data)
        return response.status_code == 200
    except Exception as e:
        print(f"Error sending data: {e}")
        return False
```

Python script : sensor/actuator

This is the main function that controls the entire workout simulation. It starts by showing you important details about the workout, like what exercise you'll be doing, how much weight you'll be using, and how many sets and reps are planned."



```python
def run_simulation():
    """Run the main simulation loop"""
    print(f"Starting {EXERCISE} simulation")
    print(f"Weight: {WEIGHT}kg")
    print(f"Sets: {TOTAL_SETS}, Reps per set: {REPS_PER_SET}")
    print("Press Ctrl+C to stop")
```

THE CLOUD SIMULATION OVERVIEW

We have created two scripts to handle the cloud simulation and data flow, which are:

–server.py

–database.py

presentation of server.py script

server.py :the core of the cloud simulation, managing communication between IoT devices, the database, and the Unity app.

Cloud Server Features

1-Device Registration:

Device ➡ Server ➡ new-objects.json ➡ Validation ➡ connected-object.json

2-Real-time data streaming

T Device ➡ Cloud Server ➡SSE➡ Unity App

Cloud Server Features

3-Database storage and integration.

we have used SQLite database to store user, device, and relationship data persistently

4-User Authentication

User enters credentials ➡ Server validates them ➡ Session is created

Device Communication and Commands

User ➡ Server ➡ Socket ➡ Device ➡ Response ➡ Server ➡ UI.

Data base management

1. User Registration

User ➡ Server ➡ Users Table (user details stored)

2. Device Registration

Device ➡ Server ➡ Objects Table (device details stored)

Objects Table ➡ Users_Objects Table (link device to owner)

3. Data Queries

Users Table ➡ Users_Objects Table ➡ Objects Table ➡ Server (retrieve user-specific device details)

Integration with Unity App

1. IoT Device (Smart Dumbbell)

Collects workout data (e.g., repetitions, duration, weight).

Device ➡ Sends Data ➡ Cloud Server

2. Cloud Server

Cloud Server ➡ Streams Data ➡ Unity App

3. Unity App

Unity App ➡ Updates UI ➡ Display Live Stats to User

Strengths of Our Code

-Robustness: Handles errors gracefully (device disconnections, invalid inputs...)

-Scalability: Supports adding more devices and users without significant changes .Designed for easy transition to larger frameworks and databases (e.g., PostgreSQL, Django).

-Real-World Simulation : Simulates a cloud backend for IoT devices with real-time data handling .Provides a seamless connection between devices, the server, and the Unity app.

-Security + Efficiency:

presentation of database.py script

database.py : Provides structured and persistent storage for the cloud simulation system.

Database Initialization:

database.py ➡ Creates Tables ➡ Users, Devices, User-Device Relationships

Data Integrity:

Foreign Keys ➡ Enforce Valid Relationships ➡ User ↔ Device

Persistent Data Storage:

Server Actions ➡ Store User/Device Details ➡ SQLite Database

Cascading Deletes:

Delete User/Device ➡ Automatically Remove Links ➡ Clean Database

IOT PROJECT

THE UNITY APP

MADE WITH Unity®