Here is the folder structure in my root folder
32project
+-- main.py
+-- templates/
    +-- dashboard_admin.html
    +-- dashboard_viewer.html
    +-- index.html

Here is files code:

main.py

```python
from flask import Flask, render_template, redirect, url_for, flash, request, Response, jso
nify
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager, UserMixin, login_user, login_required, logout_user,
current_user
from werkzeug.security import generate_password_hash, check_password_hash
import re
import cv2
import platform
import os
import serial
import time
from dotenv import load_dotenv

# ---------------------- Load Environment ----------------------
load_dotenv()

# ---------------------- GPIO + Arduino Setup ----------------------
IS_PI = platform.system() == "Linux"

if IS_PI:
    import RPi.GPIO as GPIO
    GPIO.setmode(GPIO.BCM)
    print("Running on Raspberry Pi: GPIO enabled")

    # ----- Arduino Serial Setup (for motor control via Arduino) -----
    SERIAL_PORT = os.getenv("ARDUINO_PORT", "/dev/ttyACM0")
    BAUD_RATE = 9600
    try:
        ser = serial.Serial(SERIAL_PORT, BAUD_RATE, timeout=1)
        time.sleep(2)  # allow Arduino to reset
        print(f"Connected to Arduino on {SERIAL_PORT} at {BAUD_RATE} baud")
    except Exception as e:
        ser = None
        print(f"Failed to open Arduino serial port: {e}")
else:
    print("Not running on Raspberry Pi: GPIO simulated")
```

```
    from gpiozero import LED, Device
    from gpiozero.pins.mock import MockFactory
    Device.pin_factory = MockFactory()
    ser = None

# Robot pins
PIN_FORWARD = 17
PIN_BACKWARD = 27
PIN_LEFT = 22
PIN_RIGHT = 23

if IS_PI:
    GPIO.setup(PIN_FORWARD, GPIO.OUT)
    GPIO.setup(PIN_BACKWARD, GPIO.OUT)
    GPIO.setup(PIN_LEFT, GPIO.OUT)
    GPIO.setup(PIN_RIGHT, GPIO.OUT)

# ---------------------- App Setup ----------------------
app = Flask(__name__)
app.config['SECRET_KEY'] = os.getenv('MY_SECRET_KEY') or os.urandom(32)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///site.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)
login_manager = LoginManager(app)
login_manager.login_view = 'login'

# ---------------------- Models ----------------------
class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(100), unique=True, nullable=False)
    email = db.Column(db.String(150), unique=True, nullable=False)
    password = db.Column(db.String(200), nullable=False)
    role = db.Column(db.String(50), default='viewer')  # viewer, operator, admin

@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))

# ---------------------- Camera ----------------------
camera = cv2.VideoCapture(0)

def generate_frames():
    while True:
        success, frame = camera.read()
        if not success:
            continue
        _, buffer = cv2.imencode('.jpg', frame)
```

```
        frame = buffer.tobytes()
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')

@app.route('/video_feed')
@login_required
def video_feed():
    return Response(generate_frames(), mimetype='multipart/x-mixed-replace; boundary=frame
')

# ---------------------- Robot Movement ----------------------
def move_robot(direction):
    # If running on Pi and Arduino serial is available, send the same commands
    # your friend's test Flask app used: F, B, L, R, S.
    if IS_PI and ser is not None:
        try:
            if direction == "forward":
                ser.write(b"F")
            elif direction == "backward":
                ser.write(b"B")
            elif direction == "left":
                ser.write(b"L")

            elif direction == "right":
                ser.write(b"R")
            elif direction == "stop":
                ser.write(b"S")
        except Exception as e:
            print(f"Serial error while sending '{direction}': {e}")
    elif IS_PI:
        # Fallback to direct GPIO control if serial is not available
        GPIO.output(PIN_FORWARD, GPIO.LOW)
        GPIO.output(PIN_BACKWARD, GPIO.LOW)
        GPIO.output(PIN_LEFT, GPIO.LOW)
        GPIO.output(PIN_RIGHT, GPIO.LOW)
        if direction == "forward":
            GPIO.output(PIN_FORWARD, GPIO.HIGH)
        elif direction == "backward":
            GPIO.output(PIN_BACKWARD, GPIO.HIGH)
        elif direction == "left":
            GPIO.output(PIN_LEFT, GPIO.HIGH)
        elif direction == "right":
            GPIO.output(PIN_RIGHT, GPIO.HIGH)
    else:
        print(f"Simulated move: {direction}")

@app.route('/move/<direction>', methods=['POST'])
@login_required
```

```
def move(direction):
    if current_user.role not in ['operator', 'admin']:
        return jsonify({"error": "Access denied"}), 403
    move_robot(direction)
    return jsonify({"status": f"Moved {direction}"}), 200

# ---------------------- Routes ----------------------
@app.route('/')
def home():
    # If user already logged in, send them to their dashboard directly
    if current_user.is_authenticated:
        if current_user.role == 'admin':
            return redirect(url_for('dashboard_admin'))
        elif current_user.role == 'operator':
            return redirect(url_for('dashboard_operator'))
        else:
            return redirect(url_for('dashboard_viewer'))

    # Otherwise show the landing page with base video
    return render_template('index.html')


# ---------------------- Register ----------------------
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        email = request.form['email']
        password = request.form['password']

        # Validation
        if User.query.filter_by(username=username).first():
            flash("Username already exists.", "danger")
            return redirect(url_for('register'))

        if User.query.filter_by(email=email).first():
            flash("Email already registered.", "danger")
            return redirect(url_for('register'))

        if not re.match(r'^[\w\.-]+@[\w\.-]+\.\w+$', email):
            flash("Invalid email format.", "danger")
            return redirect(url_for('register'))

        if not re.match(r'^(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&]).{8,}$', password):
            flash("Password must be at least 8 chars, include 1 uppercase, 1 number, and 1
 special character.", "danger")
            return redirect(url_for('register'))
```

```
        hashed_password = generate_password_hash(password)
        new_user = User(username=username, email=email, password=hashed_password, role='vi
ewer')
        db.session.add(new_user)
        db.session.commit()

        login_user(new_user)
        flash("Registration successful! Welcome to Viewer Dashboard.", "success")
        return redirect(url_for('dashboard_viewer'))

    return render_template('register.html')

# ---------------------- Login ----------------------
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        user = User.query.filter_by(username=username).first()
        if user and check_password_hash(user.password, password):
            login_user(user)
            if user.role == 'admin':
                return redirect(url_for('dashboard_admin'))
            elif user.role == 'operator':
                flash(f"Welcome {user.username}!", "success")
                return redirect(url_for('dashboard_operator'))
            else:
                flash(f"Welcome {user.username}!", "success")
                return redirect(url_for('dashboard_viewer'))
        else:
            flash("Invalid username or password.", "danger")

    return render_template('login.html')

# ---------------------- Logout ----------------------
@app.route('/logout')
@login_required
def logout():
    logout_user()
    flash("Logged out successfully.", "success")
    return redirect(url_for('login'))

# ---------------------- Dashboards ----------------------
@app.route('/dashboard_viewer')
@login_required
def dashboard_viewer():
    return render_template('dashboard_viewer.html')
```

```
@app.route('/dashboard_operator')
@login_required
def dashboard_operator():
    if current_user.role not in ['operator', 'admin']:
        flash("Access denied!", "danger")
        return redirect(url_for('dashboard_viewer'))
    return render_template('dashboard_operator.html')

@app.route('/dashboard_admin')
@login_required
def dashboard_admin():
    if current_user.role != 'admin':
        flash("Access denied!", "danger")
        return redirect(url_for('dashboard_viewer'))
    users = User.query.all()
    existing_operator = User.query.filter_by(role='operator').first()
    return render_template('dashboard_admin.html', users=users, existing_operator=existing
_operator)

# ---------------------- Admin Actions ----------------------
@app.route('/approve_operator/<int:user_id>')
@login_required
def approve_operator(user_id):
    if current_user.role != 'admin':
        flash("Access denied!", "danger")
        return redirect(url_for('dashboard_viewer'))

    existing_operator = User.query.filter_by(role='operator').first()
    if existing_operator:
        flash(f"Cannot promote. Operator '{existing_operator.username}' already exists.",
"warning")
    else:
        user = User.query.get_or_404(user_id)
        if user.role == 'viewer':
            user.role = 'operator'
            db.session.commit()
            flash(f"{user.username} promoted to operator.", "success")
        else:
            flash(f"{user.username} cannot be promoted.", "warning")
    return redirect(url_for('dashboard_admin'))

@app.route('/demote_operator/<int:user_id>')
@login_required
def demote_operator(user_id):
    if current_user.role != 'admin':
        flash("Access denied!", "danger")
        return redirect(url_for('dashboard_viewer'))
```

```
    user = User.query.get_or_404(user_id)
    if user.role == 'operator':
        user.role = 'viewer'
        db.session.commit()
        flash(f"{user.username} demoted to viewer.", "success")
    else:
        flash(f"{user.username} cannot be demoted.", "warning")
    return redirect(url_for('dashboard_admin'))

@app.route('/remove_user/<int:user_id>')
@login_required
def remove_user(user_id):
    if current_user.role != 'admin':
        flash("Access denied!", "danger")
        return redirect(url_for('dashboard_viewer'))

    user = User.query.get_or_404(user_id)
    if user.role != 'admin':
        db.session.delete(user)
        db.session.commit()
        flash(f"{user.username} has been removed.", "success")
    else:
        flash("Cannot remove admin!", "danger")
    return redirect(url_for('dashboard_admin'))

# ---------------------- Auto-create Admin ----------------------
def create_default_admin():
    # Check for existing admin by email or username
    admin = User.query.filter(
        (User.username == 'Admin') |
        (User.email == 'admin@gmail.com')
    ).first()

    if not admin:
        hashed_password = generate_password_hash("Thisisadmin01!")
        admin = User(
            username='Admin',
            email='admin@gmail.com',
            password=hashed_password,
            role='admin'
        )
        db.session.add(admin)
        db.session.commit()
        print("Default admin created!")
    else:
        print("Admin already exists. Skipping creation.")
```

```
# ---------------------- Run App ----------------------
if __name__ == "__main__":
    with app.app_context():
        db.create_all()
        create_default_admin()
    app.run(host="0.0.0.0", port=5000, debug=True)
```

‘‘‘

templates\dashboard_admin.html

‘‘‘html

```
{% extends "base.html" %}

{% block content %}
<div class="dashboard-container">
    <!-- Header -->
    <div class="dashboard-header">
        <h1 class="dashboard-title">Admin Dashboard</h1>
        <a href="{{ url_for('logout') }}" class="btn-logout">Logout</a>
    </div>

    <!-- Flash messages -->
    {% with messages = get_flashed_messages(with_categories=true) %}
      {% if messages %}
        {% set category, message = messages[0] %}
        <div class="flash {{ category }}">{{ message }}</div>
      {% endif %}
    {% endwith %}

    <!-- User Management Table -->
    <div class="user-management">
        <h2>User Management</h2>
        <table>
            <tr>
                <th>Username</th>
                <th>Email</th>
                <th>Role</th>
                <th>Actions</th>
            </tr>
            {% for user in users %}
            <tr>
                <td>{{ user.username }}</td>
                <td>{{ user.email }}</td>
                <td>{{ user.role }}</td>
                <td>
                    {% if user.role == 'viewer' %}
                        {% if not existing_operator %}
```

```
                        <a href="{{ url_for('approve_operator', user_id=user.id) }}" c
lass="btn-primary">Promote to Operator</a>
                    {% else %}
                        <span>Operator exists</span>
                    {% endif %}
                {% elif user.role == 'operator' %}
                    <a href="{{ url_for('demote_operator', user_id=user.id) }}" class=
"btn-primary">Demote to Viewer</a>
                {% endif %}
                {% if user.role != 'admin' %}
                    <a href="{{ url_for('remove_user', user_id=user.id) }}" class="btn
-primary">Remove</a>
                {% endif %}
            </td>
        </tr>
        {% endfor %}
    </table>
</div>

<!-- Camera Section -->
<div class="camera-section">
    <div class="video-wrapper">
        <video id="video" width="640" height="480" autoplay playsinline></video>
        <div id="noCameraMessage">No Camera Detected</div>
    </div>
    <canvas id="snapshot" width="640" height="480" style="display:none;"></canvas>

    <div class="camera-buttons">
        <button id="snapshotBtn" class="btn-primary" disabled>Take Snapshot</button>
        <button id="startBtn" class="btn-primary" disabled>Start Recording</button>
        <button id="stopBtn" class="btn-primary" disabled>Stop Recording</button>
    </div>
    <div id="timer" class="recording-timer">00:00</div>
</div>
</div>

<script>
let mediaStream = null;
let mediaRecorder;
let recordedChunks = [];
let timerInterval;
let seconds = 0;

// Elements
const video = document.getElementById('video');
const noCam = document.getElementById('noCameraMessage');
const startBtn = document.getElementById('startBtn');
const stopBtn = document.getElementById('stopBtn');
```

```
const snapshotBtn = document.getElementById('snapshotBtn');
const timerDisplay = document.getElementById('timer');

// Try to access webcam
navigator.mediaDevices.getUserMedia({ video: true, audio: true })
.then(stream => {
    mediaStream = stream;
    video.srcObject = stream;
    noCam.style.display = "none";
    snapshotBtn.disabled = false;
    startBtn.disabled = false;
})
.catch(() => {
    noCam.style.display = "flex";
    video.style.display = "none";
});

// ---------- SNAPSHOT FUNCTION ----------
snapshotBtn.addEventListener('click', () => {
    if (!mediaStream) return alert("No camera detected.");
    const canvas = document.getElementById('snapshot');
    const context = canvas.getContext('2d');
    context.drawImage(video, 0, 0, canvas.width, canvas.height);
    const imageURL = canvas.toDataURL("image/png");
    const a = document.createElement('a');
    a.href = imageURL;
    a.download = 'snapshot.png';
    a.click();
    alert("Snapshot saved!");
});

// ---------- RECORDING FUNCTIONS ----------
startBtn.addEventListener('click', () => {
    if (!mediaStream) return alert("No camera detected.");

    recordedChunks = [];
    mediaRecorder = new MediaRecorder(mediaStream);
    mediaRecorder.ondataavailable = e => {
        if (e.data.size > 0) recordedChunks.push(e.data);
    };
    mediaRecorder.onstop = saveRecording;

    mediaRecorder.start();
    startBtn.disabled = true;
    stopBtn.disabled = false;
    seconds = 0;
    updateTimer();
    timerInterval = setInterval(updateTimer, 1000);
```

```
});

stopBtn.addEventListener('click', () => {
    if (mediaRecorder && mediaRecorder.state !== "inactive") {
        mediaRecorder.stop();
        clearInterval(timerInterval);
        startBtn.disabled = false;
        stopBtn.disabled = true;
        timerDisplay.textContent = "00:00";
    }
});

function saveRecording() {
    const blob = new Blob(recordedChunks, { type: 'video/webm' });
    const url = URL.createObjectURL(blob);
    const a = document.createElement('a');
    a.href = url;
    a.download = 'recording.webm';
    a.click();
    alert("Recording saved!");
}

function updateTimer() {
    seconds++;
    const mins = String(Math.floor(seconds / 60)).padStart(2, '0');
    const secs = String(seconds % 60).padStart(2, '0');
    timerDisplay.textContent = `${mins}:${secs}`;
}
</script>

<style>
.dashboard-header {
    display: flex;
    align-items: center;
    justify-content: space-between;
    padding: 10px 20px;
}

.dashboard-title {
    margin: 0;
    font-size: 28px;
    font-weight: bold;
}

.btn-logout {
    background: #e74c3c;
    color: white;
    text-decoration: none;
```

```
    padding: 8px 14px;
    border-radius: 6px;
    font-weight: 500;
    transition: 0.3s;
}
.btn-logout:hover {
    background: #c0392b;
}

.video-wrapper {
    position: relative;
    width: 640px;
    height: 480px;
    margin: 20px auto;
    background: #000;
    border-radius: 10px;
    overflow: hidden;
}
#noCameraMessage {
    display: none;
    position: absolute;
    inset: 0;
    background: rgba(20, 20, 20, 0.9);
    color: white;
    font-size: 28px;
    display: flex;
    align-items: center;
    justify-content: center;
}
.camera-buttons {
    text-align: center;
    margin-top: 10px;
}
.recording-timer {
    text-align: center;
    font-weight: bold;
    font-size: 20px;
    margin-top: 10px;
}
</style>
{% endblock %}

‘‘‘
```

templates\dashboard_viewer.html

```
‘‘‘html
{% extends "base.html" %}
{% block content %}
```

```
<div class="dashboard-container">
  <div class="dashboard-header">
    <h1 class="dashboard-title">Viewer Dashboard</h1>
    <div class="dashboard-actions">
      <a href="{{ url_for('logout') }}" class="btn-logout">Logout</a>
    </div>
  </div>

  <p class="dashboard-subtitle">View the camera feed and take snapshots or recordings</p
>

  <div class="camera-section">
    <video id="video" width="640" height="480" autoplay></video>
    <canvas id="snapshot" width="640" height="480" style="display:none;"></canvas>
    <div class="camera-buttons">
      <button onclick="takeSnapshot()" class="btn-primary">Take Snapshot</button>
      <button onclick="startRecording()" class="btn-primary">Start Recording</button
>
      <button onclick="stopRecording()" class="btn-primary">Stop Recording</button>
    </div>
  </div>
</div>

<script>
const video = document.getElementById('video');
navigator.mediaDevices.getUserMedia({ video: true, audio: true })
.then(stream => video.srcObject = stream)
.catch(err => console.error("Camera error:", err));

function takeSnapshot() {
  const canvas = document.getElementById('snapshot');
  const context = canvas.getContext('2d');
  context.drawImage(video, 0, 0, canvas.width, canvas.height);
  alert("Snapshot taken!");
}

let mediaRecorder;
let recordedChunks = [];

function startRecording() {
  recordedChunks = [];
  mediaRecorder = new MediaRecorder(video.srcObject);
  mediaRecorder.ondataavailable = e => { if (e.data.size > 0) recordedChunks.push(e.data
); };
  mediaRecorder.start();
  alert("Recording started");
}
```

```
function stopRecording() {
    mediaRecorder.stop();
    mediaRecorder.onstop = () => {
        const blob = new Blob(recordedChunks, { type: 'video/webm' });
        const url = URL.createObjectURL(blob);
        const a = document.createElement('a');
        a.href = url;
        a.download = 'recording.webm';
        a.click();
        alert("Recording saved");
    };
}
</script>

<style>
.dashboard-container {
    max-width: 900px;
    margin: 50px auto;
    text-align: center;
}

.dashboard-header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin-bottom: 15px;
}

.dashboard-title {
    font-size: 32px;
    color: #0a8f0a;
    font-weight: 700;
    margin: 0;
}

.dashboard-subtitle {
    font-size: 16px;
    color: #666;
    margin-bottom: 30px;
}

.btn-primary {
    padding: 8px 15px;
    background-color: #0a8f0a;
    color: white;
    border: none;
    border-radius: 6px;
    cursor: pointer;
```

```
}
.btn-primary:hover {
    background-color: #087607;
}

.btn-logout {
    background-color: #d9534f;
    color: white;
    padding: 8px 15px;
    border-radius: 6px;
    text-decoration: none;
    font-weight: 600;
    transition: background 0.3s;
}
.btn-logout:hover {
    background-color: #c9302c;
}

.camera-section {
    background: #fff;
    padding: 30px;
    border-radius: 12px;
    box-shadow: 0 10px 25px rgba(0,0,0,0.1);
}

.camera-buttons button {
    margin: 10px;
}
</style>
{% endblock %}
```

‘‘‘

templates\index.html

‘‘‘html

```
{% extends "base.html" %}

{% block title %}WebRover Controller{% endblock %}

{% block background %}
<video class="bg-video" autoplay muted loop playsinline>
    <source src="{{ url_for('static', filename='stock/base.mp4') }}" type="video/mp4">
</video>
{% endblock %}

{% block extra_head %}
<style>
.hero-wrap {
```

```
    min-height: calc(100vh - 120px);
    display: flex;
    flex-direction: column;
    justify-content: center;
}

/* Title */
.hero-title {
    font-size: 54px;
    font-weight: 700;
    color: #ffffff;
    margin-bottom: 16px;
    text-shadow: 0 6px 24px rgba(0,0,0,0.55);
}

/* Description ? crisp, not blurry */
.hero-desc {
    font-size: 20px;
    color: #ffffff;
    max-width: 720px;
    line-height: 1.6;
    margin-bottom: 32px;
    text-shadow: 0 2px 6px rgba(0,0,0,0.4);
}

/* Buttons row */
.hero-buttons {
    display: flex;
    flex-wrap: wrap;
    gap: 18px;
}

/* Both Login & Register: same green style */
.hero-buttons .btn-main {
    display: inline-block;
    padding: 14px 40px;
    border-radius: 18px;
    border: none;
    cursor: pointer;
    font-weight: 700;
    font-size: 22px;
    background-color: #0a8f0a;
    color: #ffffff;
    box-shadow: 0 10px 28px rgba(0,0,0,0.35);
    transition: all 0.2s ease;
}

.hero-buttons .btn-main:hover {
```

```
    background-color: #0a7c0a;
    transform: translateY(-2px);
    box-shadow: 0 14px 34px rgba(0,0,0,0.45);
}
</style>
{% endblock %}

{% block content %}
<div class="hero-wrap">
    <h1 class="hero-title">WebRover Controller</h1>
    <p class="hero-desc">
        Securely control your Raspberry&amp;nbsp;Pi rover from any browser. Assign roles for a
dmins,
        operators, and viewers, stream live video, and send movement commands in real time
.
    </p>

    <div class="hero-buttons">
        <a href="{{ url_for('login') }}" class="btn-main">Login</a>
        <a href="{{ url_for('register') }}" class="btn-main">Register</a>
    </div>
</div>
{% endblock %}


‘‘‘
```