✅ **Backend:** Flask (Python)  
✅ **Database:** PostgreSQL via Render (gratis)  
✅ **Bestandshantering:** Cloudinary (gratis plan)  
✅ **Frontend:** HTML, CSS, en een beetje JavaScript (of later React)  
✅ **Hosting:** Render (voor de backend) en GitHub Pages/Vercel (voor de frontend)

1. CMD (Command Prompt) is de standaard opdrachtprompt in Windows.
   1. Druk op Win + R, typ cmd, en druk op Enter.
2. Navigeer naar de map waar je je project wilt maken met:
   1. C:\Users\diann>cd "C:\Users\Diann\OneDrive\Judo\Judoconnect"
3. Maak de virtuele omgeving
   1. python -m venv venv
4. Check of het er is. Als er een map met venv is dan is het gelukt.
   1. dir
5. Als venv er is, activeer het dan. Je weet dat de activatie is gelukt als je (venv) vóór je prompt ziet.
   1. venv\Scripts\activate
6. Installeer Flask en andere pakketten
   1. pip install flask flask\_sqlalchemy flask\_bcrypt cloudinary
7. Wil je testen of Flask goed is geïnstalleerd? Voer dan dit uit:
   1. python -m flask –version
8. Maak een nieuw bestand app.py = leeg python bestand in cmd
   1. type nul > app.py

Open het bestand in Kladblok (a) of in VS code (b) (beter):

* 1. notepad app.py
  2. code .

1. Zet de Database op: Start een Python-shell en voer het volgende uit om de database aan te maken. Nu ga je in je terminal (CMD of PowerShell) de Python-shell openen:
   1. python
2. Database maken: zorg dat je een Flask-app hebt waarin db is gedefinieerd. Voer de database-commando’s uit. Hiermee wordt het bestand database.db aangemaakt in je projectmap. In de Python-shell typ je regel voor regel:
   1. from app import app, db

with app.app\_context():

db.create\_all()

Let op de spatie vóór db.create\_all() in de tweede regel! Puntjes nog zichtbaar? Druk nogmaals op enter. Flask maakt standaard de database aan in de instance/-map, tenzij je een volledig pad opgeeft in SQLALCHEMY\_DATABASE\_URI. Checken of het is gelukt door in file explorer de instance map te zoeken en te kijken of er een database in staat.

1. Sluit de Python-shell af. In de Python-shell (waar >>> staat), typ:
   1. exit()

Zorg ervoor dat je in de juiste map zit en in de virtuele omgeving (met venv ervoor).

1. Start de Flask Server en Draai de applicatie lokaal:
   1. python app.py
2. Openen bestanden
   1. CMD --> cd "C:\Users\Diann\OneDrive\Judo\Judoconnect" (juiste path) --> code templates/index.html
3. Installeer flask dance voor inloggen gmail in virtuele omgeving
   1. pip install Flask-Dance Flask-OAuthlib

<http://127.0.0.1:5000/admin> = beheerderspagina

App.py:

import os

from flask import request, Flask, render\_template, redirect, url\_for, session, flash

from flask\_sqlalchemy import SQLAlchemy

from flask\_migrate import Migrate

from flask\_dance.consumer.storage.sqla import SQLAlchemyStorage

from flask\_dance.contrib.google import make\_google\_blueprint, google

from flask\_login import LoginManager, login\_user, logout\_user, login\_required, UserMixin

from sqlalchemy import Integer, String, LargeBinary, DateTime

from sqlalchemy.ext.mutable import MutableDict

from sqlalchemy.dialects.postgresql import JSON

from flask\_login import current\_user

from flask\_dance.consumer import oauth\_authorized

os.environ['OAUTHLIB\_INSECURE\_TRANSPORT'] = '1' # ✅ Allow HTTP for local testing

# Maak de Flask-app aan

app = Flask(\_\_name\_\_)

from flask import session

from datetime import timedelta

app.config['PERMANENT\_SESSION\_LIFETIME'] = timedelta(days=1) # Houd sessies een dag geldig

@app.before\_request

def make\_session\_permanent():

session.permanent = True

# Configuratie voor de database

app.config["SQLALCHEMY\_DATABASE\_URI"] = "sqlite:///videos.db"

app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False

app.secret\_key = 'je-geheime-sleutel' # Verander dit naar iets unieks en geheim

# Initialiseer SQLAlchemy en Migrate

db = SQLAlchemy(app)

migrate = Migrate(app, db)

class Video(db.Model):

id = db.Column(db.Integer, primary\_key=True)

youtube\_url = db.Column(db.String(255), nullable=False)

embedded\_url = db.Column(db.String(255), nullable=False)

approved = db.Column(db.Boolean, default=False) # Goedkeuringsstatus

# Definieer de 'Technique' model voor de database

class Technique(db.Model):

id = db.Column(db.Integer, primary\_key=True)

name = db.Column(db.String(100), nullable=False)

description = db.Column(db.String(500), nullable=True)

category = db.Column(db.String(100), nullable=False) # bv. 'staande technieken'

approved = db.Column(db.Boolean, default=False) # Of de techniek goedgekeurd is

# Model voor OAuth-tokens

class OAuth(db.Model):

id = db.Column(db.Integer, primary\_key=True)

provider = db.Column(db.String(50), nullable=False)

token = db.Column(MutableDict.as\_mutable(JSON), nullable=False) # Gebruik JSON i.p.v. Binary

user\_id = db.Column(db.Integer, db.ForeignKey('user.id'))

email = db.Column(db.String(150), unique=True, nullable=False)

name = db.Column(db.String(150), nullable=False)

# Configureer de opslag in Flask-Dance

oauth\_storage = SQLAlchemyStorage(OAuth, db.session)

# Model voor gebruikers

class User(db.Model, UserMixin):

id = db.Column(db.Integer, primary\_key=True)

email = db.Column(db.String(150), unique=True, nullable=False)

name = db.Column(db.String(150), nullable=False)

# Login manager instellen

login\_manager = LoginManager()

login\_manager.init\_app(app)

login\_manager.login\_view = 'google\_login'

@login\_manager.user\_loader

def load\_user(user\_id):

# Zorg ervoor dat je 'user\_id' omgezet wordt naar een integer

return db.session.get(User, int(user\_id))

# Configuraties voor Google OAuth2 via Flask-Dance

google\_bp = make\_google\_blueprint(client\_id='1070657144197-jdv61agehj4bk817tsenk15pa0qkhlf6.apps.googleusercontent.com',

client\_secret='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',

redirect\_to='google.authorized', # Zorg ervoor dat dit overeenkomt met de redirect URI

scope=["openid",

"<https://www.googleapis.com/auth/userinfo.email>",

"<https://www.googleapis.com/auth/userinfo.profile>"],

storage=SQLAlchemyStorage(OAuth, db.session, user=lambda: current\_user)

)

app.register\_blueprint(google\_bp, url\_prefix='/google\_login')

import json

# Define the oauth\_authorized signal to manually associate the token with the user

@oauth\_authorized.connect\_via(google\_bp)

def google\_oauth\_authorized(sender, token, \*\*extra):

#print("🔄 OAuth authorized callback gestart...") # ✅ Dit wordt aangeroepen

if not google.authorized:

print("⚠️ Google is niet geautoriseerd, doorsturen naar login...")

return redirect(url\_for('google.login'))

#print(f"✅ OAuth-token ontvangen: {token}") # ✅ Token ophalen gelukt?

# Haal de user info op van Google

resp = google.get('/oauth2/v1/userinfo')

if not resp.ok:

print("❌ Kon geen gebruikersinformatie ophalen!")

return 'Failed to fetch user info', 400

user\_info = resp.json()

#print(f"✅ Gebruikersinfo ontvangen: {user\_info}") # ✅ Check wat binnenkomt

email = user\_info.get('email')

if not email:

print("❌ Geen e-mailadres gevonden!")

return 'No email found', 400

# Zoek of de gebruiker al bestaat

user = User.query.filter\_by(email=email).first()

if user is None:

print(f"🆕 Nieuwe gebruiker aanmaken: {email}")

user = User(email=email, name=user\_info.get('name', 'Onbekend'))

db.session.add(user)

db.session.commit()

# Zoek of er al een OAuth-entry is

oauth\_entry = OAuth.query.filter\_by(email=email, provider='google').first()

if oauth\_entry:

#print("🔄 OAuth-entry gevonden, token bijwerken...")

oauth\_entry.token = token # \*\*Token bijwerken in plaats van een nieuwe rij\*\*

else:

print("🆕 Nieuwe OAuth-entry aanmaken")

oauth\_entry = OAuth(provider='google', token=token, user\_id=user.id, email=email, name=user\_info.get('name', 'Onbekend'))

db.session.add(oauth\_entry)

db.session.commit()

# Log de gebruiker in

print(f"✅ {email} is nu ingelogd!")

login\_user(user)

if email == "[judoconnect@gmail.com](mailto:judoconnect@gmail.com)":

return redirect(url\_for('admin')) # Stuur alleen admin naar de adminpagina

else:

return redirect(url\_for('home')) # Stuur andere gebruikers naar de homepagina

@app.route("/upload\_video", methods=["GET", "POST"])

def upload\_video():

if request.method == "POST":

youtube\_url = request.form["youtube\_url"]

if "youtube.com/watch?v=" in youtube\_url or "youtu.be/" in youtube\_url:

video\_id = youtube\_url.split("v=")[-1].split("&")[0] if "v=" in youtube\_url else youtube\_url.split("/")[-1]

embedded\_url = f"https://www.youtube.com/embed/{video\_id}"

# Video opslaan in de database als niet-goedgekeurd

new\_video = Video(youtube\_url=youtube\_url, embedded\_url=embedded\_url, approved=False)

db.session.add(new\_video)

db.session.commit()

return redirect(url\_for("upload\_video"))

return render\_template("upload\_video.html")

@app.route("/", methods=["GET", "POST"])

def home():

#print(f"📌 Huidige sessie: {session}") # Log de volledige sessie

if not current\_user.is\_authenticated:

return redirect(url\_for("google.login"))

#print("✅ Gebruiker is ingelogd, doorsturen naar uploadpagina...")

return render\_template("home.html") # Hier wordt home.html geladen

# return redirect(url\_for('upload\_video'))

@app.route("/admin")

@login\_required

def admin():

if current\_user.email != "[judoconnect@gmail.com](mailto:judoconnect@gmail.com)":

return redirect(url\_for("home"))

# Haal technieken op die nog niet goedgekeurd zijn

unapproved\_techniques = Technique.query.filter\_by(approved=False).all()

return render\_template('admin.html', techniques=unapproved\_techniques)

# Route om een techniek goed te keuren

@app.route('/approve\_technique/<int:id>', methods=['POST'])

@login\_required

def approve\_technique(id):

if current\_user.email != "[judoconnect@gmail.com](mailto:judoconnect@gmail.com)":

return redirect(url\_for("home")) # Alleen admin mag goedkeuren

technique = Technique.query.get(id)

if technique:

technique.approved = True

db.session.commit()

return redirect(url\_for('admin'))

# Route om een techniek af te wijzen

@app.route('/reject\_technique/<int:id>', methods=['POST'])

@login\_required

def reject\_technique(id):

if current\_user.email != "[judoconnect@gmail.com](mailto:judoconnect@gmail.com)":

return redirect(url\_for("home")) # Alleen admin mag verwijderen

technique = Technique.query.get(id)

if technique:

db.session.delete(technique)

db.session.commit()

return redirect(url\_for('admin'))

@app.route("/videos")

def videos():

videos = Video.query.filter\_by(approved=True).all()

return render\_template("videos.html", videos=videos)

@app.route('/logout')

def logout():

session.clear() # Wis de hele sessie

return redirect(url\_for('home')) # Stuur terug naar de homepagina

# Voer de app uit

if \_\_name\_\_ == "\_\_main\_\_":

with app.app\_context():

db.create\_all() # Zorgt ervoor dat de tabellen worden aangemaakt in de database

app.run(debug=True)

@app.route("/check\_login")

def check\_login():

if current\_user.is\_authenticated:

return f"Ingelogd als {current\_user.email}"

return "Niet ingelogd"

@app.route('/tachiwaza')

def tachiwaza():

return render\_template('tachiwaza.html')

@app.route('/newaza')

def newaza():

return render\_template('newaza.html')

@app.route('/exercises')

def exercises():

return render\_template('exercises.html')

videos = []

@app.route('/O-soto-gari', methods=['GET', 'POST'])

def osotogari():

global videos

if request.method == 'POST':

youtube\_url = request.form['youtube\_url']

# Controleer of de URL een geldige YouTube-URL is

if "youtube.com" in youtube\_url:

videos.append({'url': youtube\_url, 'votes': 0})

else:

return "Ongeldige YouTube URL", 400

return render\_template('tachiwaza/osotogari.html', videos=videos)

@app.route('/vote/<int:video\_id>')

def vote(video\_id):

global videos

if 0 <= video\_id < len(videos):

videos[video\_id]['votes'] += 1

return redirect(url\_for('osotogari'))

@app.route('/Tachi-waza', methods=['GET', 'POST'])

def tachi\_waza():

global videos

if request.method == 'POST':

youtube\_url = request.form['youtube\_url']

if "youtube.com" in youtube\_url:

videos.append({'url': youtube\_url, 'votes': 0})

else:

return "Ongeldige YouTube URL", 400

return render\_template('tachi\_waza.html', videos=videos)

from routes.tachiwaza import tachiwaza\_bp

# Registreer Blueprints

app.register\_blueprint(tachiwaza\_bp)

Tachiwaza.py

from flask import Blueprint, render\_template, request, jsonify

tachiwaza\_bp = Blueprint('tachiwaza', \_\_name\_\_)

# Voorbeeldlijst van technieken (eventueel later uit een database halen)

techniques = [

"O-soto-gari", "De-ashi-barai", "Okuri-ashi-barai", "Hiza-guruma", "Ko-soto-gake",

"O-uchi-gari", "Ko-uchi-gari", "O-soto-guruma", "O-soto-otoshi", "Ko-soto-gari",

"Sasae-tsuri-komi-ashi", "Harai-tsuri-komi-ashi", "O-soto-gake", "Ko-uchi-maki-komi",

"Ashi-guruma", "Uchi-mata", "Uki-goshi", "Kubi-nage", "Tsuri-goshi", "Koshi-guruma",

"Harai-goshi", "Hane-goshi", "Ushiro-goshi", "Tsuri-komi-goshi", "Utsuri-goshi",

"O-goshi", "Ko-tsuri-goshi", "O-guruma", "Yama-arashi", "Obi-goshi", "Kata-seoi",

"Ippon-seoi-nage", "Kata-guruma", "Seoi-otoshi", "Hidari-kata-seoi", "Morote-seoi-nage",

"Eri-seoi-nage", "Tai-otoshi", "Uki-otoshi", "Sokui-nage", "Kata-ashi-dori", "Ryo-ashi-dori",

"Tomoe-nage", "Yoko-tomoe-nage", "Soto-maki-komi", "Yoko-gake", "Tani-otoshi", "Sumi-gaeshi",

"Kani-basami", "Ura-nage", "Yoko-guruma", "Ude-hishigi-juji-katame", "Ude-garami", "Ude-hishigi",

"Yoko-hiza-katame", "Kami-ude-hishigi-juji-katame", "Yoko-ude-hishigi", "Kami-hiza-katame",

"Ude-hishigi-henka-waza", "Gyaku-juji", "Shime-garami", "Hiza-katame", "Hara-katame",

"Ashi-katame", "Ude-garami-henka-waza", "Kesa-garami", "Kuzure-kami-shiho-garami",

"Gyaku-kesa-garami", "Gyaku-te-kubi", "Hiji-maki-komi", "Kanuki-katame", "Ude-hishigi-hiza-katame",

"Kata-juji-jime", "Name-juji-jime", "Gyaku-juji-jime", "Yoko-juji-jime", "Ushiro-jime",

"Okuri-eri-jime", "Kata-ha-jime", "Hadaka-jime", "Ebi-garami", "Tomoe-jime", "Narabi-juji-jime",

"Katate-jime", "Sode-guruma", "Hidari-ashi-jime", "Kagato-jime", "Kami-shiho-jime",

"Kami-shiho-ashi-jime", "Kami-shiho-basami", "Gyaku-okuri-eri", "Gaeshi-jime",

"Gyaku-gaeshi-jime", "Kesa-katame", "Kata-katame", "Kami-shiho-katame",

"Kuzure-kami-shiho-katame", "Ushiro-kesa-gatame", "Gyaku-kesa-katame", "Yoko-shiho-katame",

"Makura-kesa-katame", "Mune-katame", "Tate-shiho-katame", "Kuzure-kesa-katame",

"Kata-osae-katame", "Ura-katame", "Kashira-katame", "Ura-shiho-katame", "Kami-sankaku-katame",

"Kuzure-yoko-shiho", "Tate-sankaku-katame", "Uki-katame"

]

techniques\_list = {technique: [] for technique in techniques}

# Dictionary voor video's per techniek

videos\_data = {technique: [] for technique in techniques}

import json

# Laad de video's van een bestand

def load\_videos():

try:

with open('videos.json', 'r') as file:

return json.load(file)

except FileNotFoundError:

return {}

# Sla de video's op in een bestand

def save\_videos(videos):

with open('videos.json', 'w') as file:

json.dump(videos, file)

# Laad de video's in je route

saved\_videos = load\_videos()

@tachiwaza\_bp.route('/tachiwaza/<technique>', methods=['GET', 'POST'])

def technique\_page(technique):

global videos\_data

if technique not in techniques\_list:

return "Deze techniek bestaat niet.", 404 # Geef een 404 als de techniek niet bestaat

# Laad video's van de techniek uit de opgeslagen gegevens

videos = saved\_videos.get(technique, [])

if request.method == 'POST':

youtube\_url = request.form['youtube\_url']

if "youtube.com" in youtube\_url:

videos.append({'url': youtube\_url, 'votes': 0})

saved\_videos[technique] = videos # Update de video's voor deze techniek

save\_videos(saved\_videos) # Sla de video's op na het toevoegen

else:

return "Ongeldige YouTube URL", 400

return render\_template('tachiwaza/technique.html', technique=technique, videos=videos)

# Het opslaan van stemmen

@tachiwaza\_bp.route('/tachiwaza/<technique>/vote', methods=['POST'])

def vote\_video(technique):

try:

user\_id = request.remote\_addr # Gebruik IP als tijdelijke identificatie

data = request.get\_json() # Zorg ervoor dat we JSON krijgen

video\_index = int(data['video\_index'])

rating = int(data['rating'])

# Controleer of techniek bestaat in saved\_videos

if technique not in saved\_videos:

return jsonify({'error': f'Technique {technique} niet gevonden'}), 400

# Controleer of de video-index geldig is

if video\_index >= len(saved\_videos[technique]):

return jsonify({'error': f'Video index {video\_index} is buiten bereik'}), 400

video = saved\_videos[technique][video\_index]

# Voeg of update de stem van de gebruiker

if not isinstance(video['votes'], dict): # Controleer of votes een dictionary is

video['votes'] = {}

video['votes'][user\_id] = rating # Nu is votes een dictionary en werkt het opslaan correct

# Bereken het gemiddelde

video['average\_rating'] = sum(video['votes'].values()) / len(video['votes'])

# Update het aantal stemmen

total\_votes = len(video['votes'])

save\_videos(saved\_videos) # Opslaan naar JSON

return jsonify({

'average\_rating': round(video['average\_rating'], 1),

'total\_votes': total\_votes

})

except Exception as e:

return jsonify({'error': str(e)}), 500