ORTA DOĞU TEKNİK ÜNİVERSİTESİ
MIDDLE EAST TECHNICAL UNIVERSITY
KUZEY KIBRIS KAMPUSU ◆ NORTHERN CYPRUS CAMPUS

CNG 495
Fall – 2024
Term Project Progress Report I

Name: Umutcan

Surname: CELIK

SID: 2526200

Project Title: ZeroDay Project

Date of Submission:

Table of Contents

LIST OF FIGURES AND TABLES

Project Description

The main goal of my project is to help new employees start their first day at work smoothly. With this project, companies can show new employees all the important documents on a website before they come to the office. They can also watch safety videos and other important videos online. This way, the first day will be easier, and both the new employee and the company will save time during the hiring process.
The project will be built using Python Flask, Heroku Postgres, Bootstrap, and Blueprint. Python Flask will be used to create the web interface and manage all the logic of the app. Heroku Postgres will store and manage the data in the cloud, making it easier to handle backups and scale up if more users join.
Bootstrap will be used to create a user-friendly interface that works well on all devices, like phones, tablets, and computers. Blueprint will help organize the code into different sections, making it easier to manage and reuse. Together, these tools will make the project more efficient, flexible, and easy to use for everyone.

Cloud Delivery Models

SaaS (Software as a Service):

I will utilize various SaaS solutions to enhance my project functionalities, providing user-friendly services for specific needs without going into specific providers.

PaaS (Platform as a Service):

I will use Heroku to deploy my application because it is a Platform as a Service (PaaS) that makes it easy to develop, run, and manage my Python Flask application. Heroku has many benefits, like automatic scaling, which adds more resources when my application gets more visitors, ensuring it runs smoothly. It also supports continuous integration and deployment (CI/CD), which helps me update my app quickly and efficiently.

IaaS (Infrastructure as a Service):

I will use Heroku Postgres as my database solution. Heroku Postgres provides scalable database services that can easily integrate with my application, allowing me to manage and store data efficiently in the cloud.

Diagrams

Use Case Diagram

You can see the use case diagram at Figure 1.

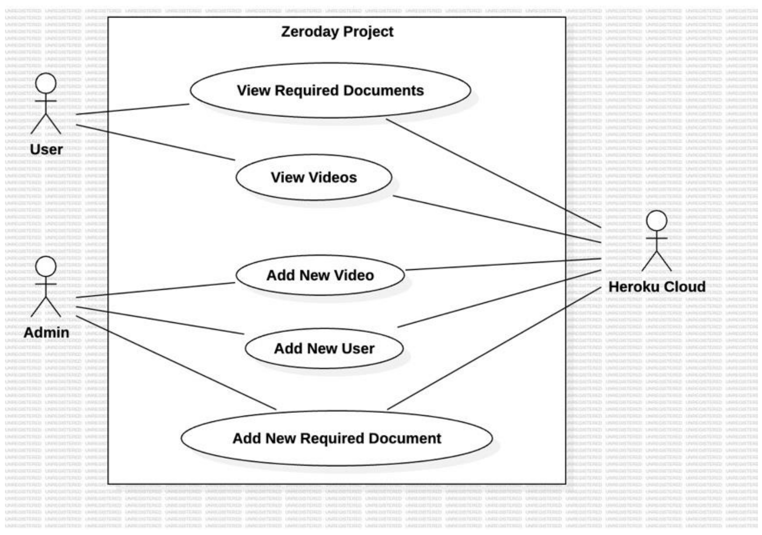


Figure 1: use case diagram of project

Data Flow Diagram

You can see the data flow diagram at Figure 2 and Figure 3.

Context Level:

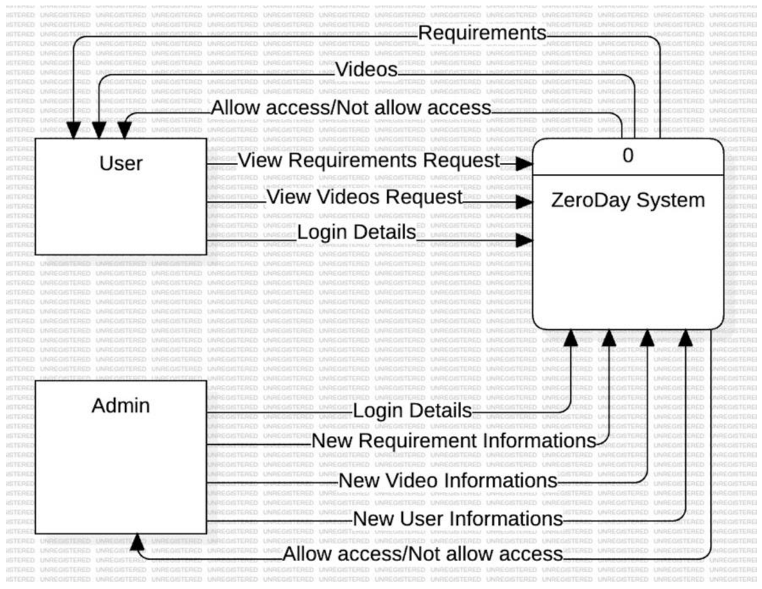


Figure 2: context level of project

Level 0:

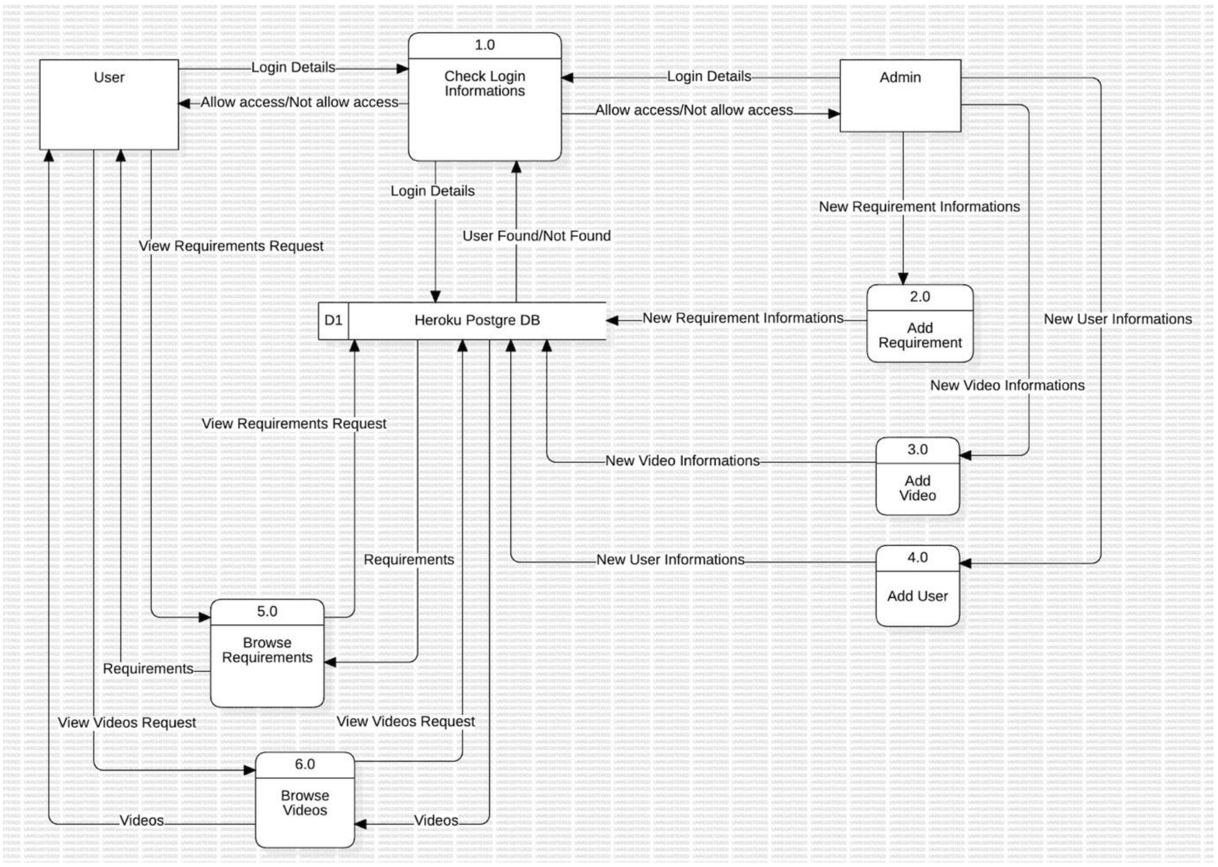


Figure 3: Level 0 of project

Data Types

In my project, the types of data I use are:

Text Data:

- Video Names and Links: In the "videos" section, I show the names and links of the videos that new employees can watch before their first day at work.
- User Login Information: Users log in to the system using a username and a password to access the platform.
- Data Stored in the Database: The database in my project stores important information such as usernames, passwords, video names, links, like/dislike counts, and required documents in text format.

Computation

In my project, computation involves processing data and performing calculations. For example:

- Checking user login information when users access the platform.
- Updating the like/dislike counts for each video when users interact with them.
- Adding new user (create username, password and decide he/she is admin or not).

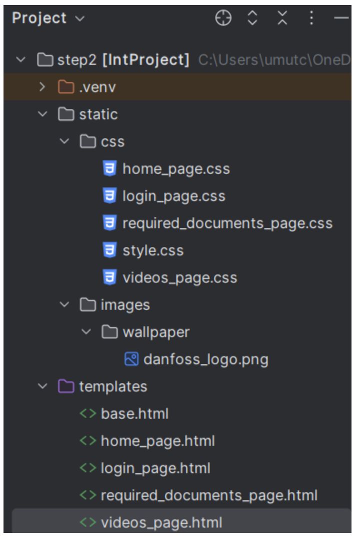
Expected Contribution

The backend, frontend, database and cloud parts of the project will be done by Umutcan Celik.

Milestones Achieved

<u>Week 4 (Oct 21 - 27) -- Week 5 (Oct 28 - Nov 3)</u>

I completed my HTML and CSS codes. In this section, I added comments to keep the codes organized and readable and separated my CSS and HTML codes under static/css and templates files. You can see from Figure 4.



```
Project
step2 [IntProject] C:\Users\umutc\OneD
  .venv
  static
    css
      home_page.css
      login_page.css
      required_documents_page.css
      style.css
      videos_page.css
    images
      wallpaper
        danfoss_logo.png
  templates
    base.html
    home_page.html
    login_page.html
    required_documents_page.html
    videos_page.html
```

Figure 4: HTML and CSS code layout

You can see the interfaces created by the HTML and CSS codes I wrote from figure 5,6,7,8.

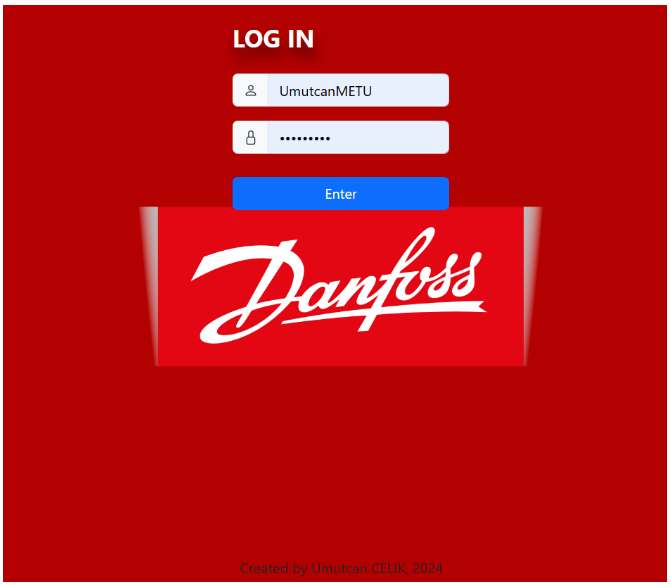
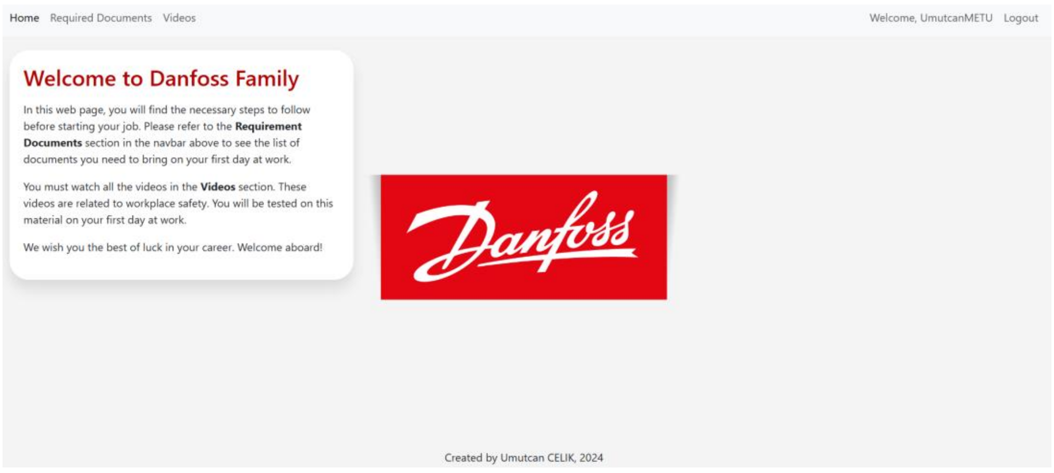


Figure 5: login screen


Home Required Documents Videos
Welcome, UmutcanMETU Logout
Danfoss
Created by Umutcan CELIK, 2024

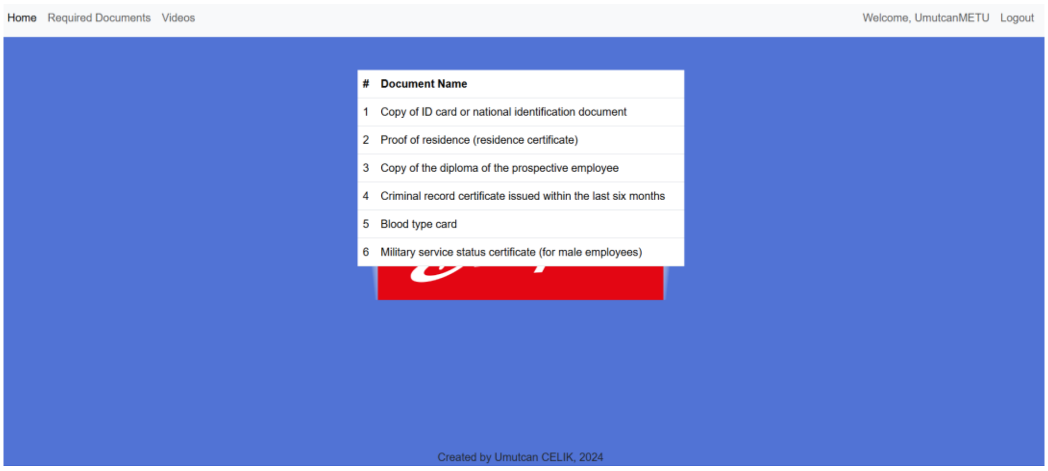
Welcome to Danfoss Family

In this web page, you will find the necessary steps to follow before starting your job. Please refer to the **Requirement Documents** section in the navbar above to see the list of documents you need to bring on your first day at work.

You must watch all the videos in the **Videos** section. These videos are related to workplace safety. You will be tested on this material on your first day at work.

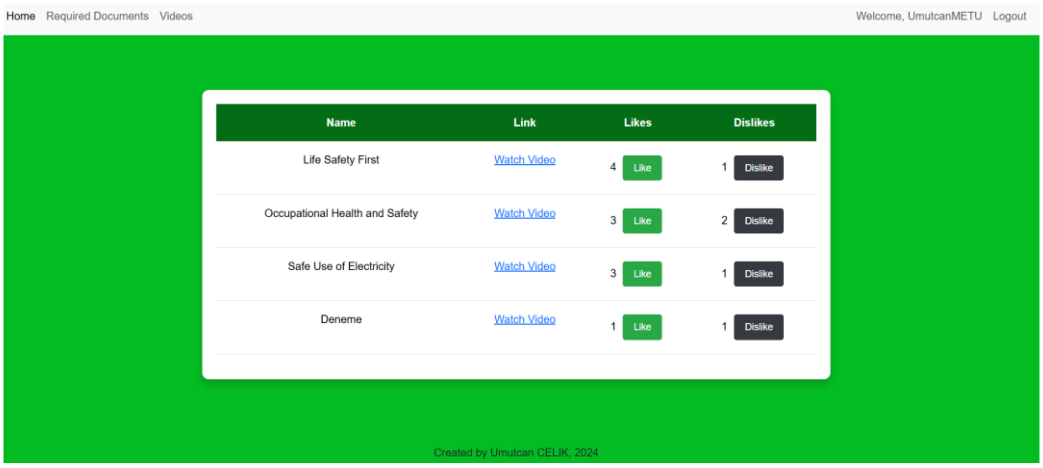
We wish you the best of luck in your career. Welcome aboard!

Figure 6: home page screen


Home Required Documents Videos
Welcome, UmutcanMETU Logout
Created by Umutcan CELIK, 2024

#	Document Name
1	Copy of ID card or national identification document
2	Proof of residence (residence certificate)
3	Copy of the diploma of the prospective employee
4	Criminal record certificate issued within the last six months
5	Blood type card
6	Military service status certificate (for male employees)

Figure 7: documents screen



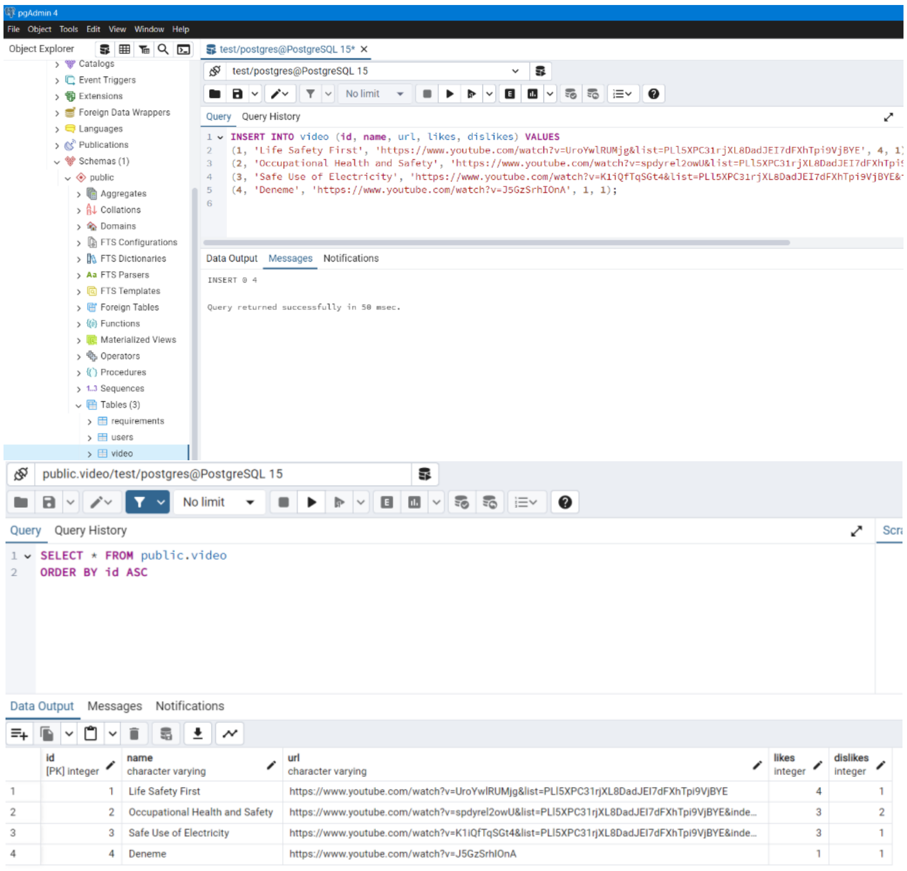
Name	Link	Likes		Dislikes	
Life Safety First	Watch Video	4	Like	1	Dislike
Occupational Health and Safety	Watch Video	3	Like	2	Dislike
Safe Use of Electricity	Watch Video	3	Like	1	Dislike
Deneme	Watch Video	1	Like	1	Dislike

Created by Umutcan CELIK, 2024

Figure 8: videos screen

Week 6 (Nov 4 - 10)

In these weeks, I designed the DB that I will use in PostgreSQL and added a small number of data in them. You can check my work on this from figure 9,10,11.



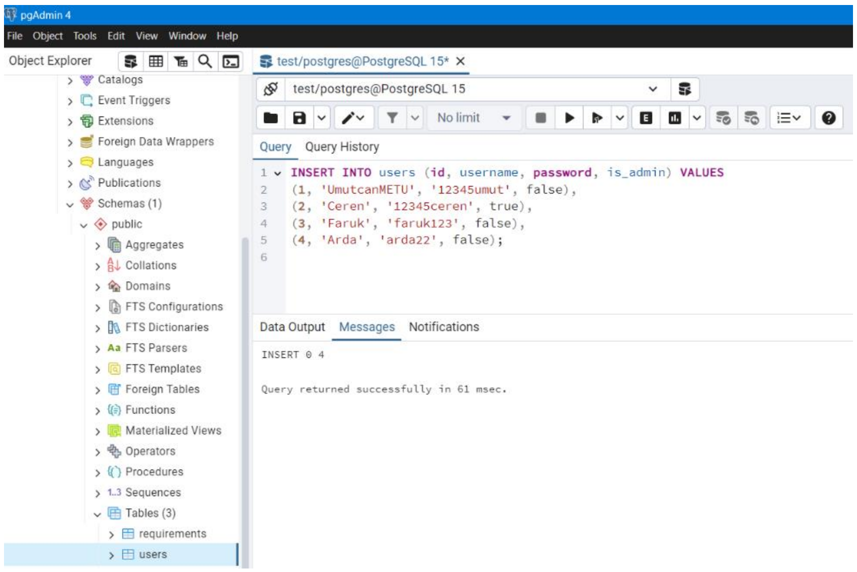
```
INSERT INTO video (id, name, url, likes, dislikes) VALUES
(1, 'Life Safety First', 'https://www.youtube.com/watch?v=UroYwlRUMjg&list=PLl5XPC31rjXL8DadJEI7dFXhTpi9VjBYE', 4, 1)
(2, 'Occupational Health and Safety', 'https://www.youtube.com/watch?v=spdyrel2owU&list=PLl5XPC31rjXL8DadJEI7dFXhTpi
(3, 'Safe Use of Electricity', 'https://www.youtube.com/watch?v=K1iQfTqSGt4&list=PLl5XPC31rjXL8DadJEI7dFXhTpi9VjBYE&
(4, 'Deneme', 'https://www.youtube.com/watch?v=J5GzSrhIOnA', 1, 1);
```

```
INSERT 0 4

Query returned successfully in 58 msec.
```

```
SELECT * FROM public.video
ORDER BY id ASC
```

	id [PK] integer	name character varying	url character varying	likes integer	dislikes integer
1	1	Life Safety First	https://www.youtube.com/watch?v=UroYwlRUMjg&list=PLl5XPC31rjXL8DadJEI7dFXhTpi9VjBYE	4	1
2	2	Occupational Health and Safety	https://www.youtube.com/watch?v=spdyrel2owU&list=PLl5XPC31rjXL8DadJEI7dFXhTpi9VjBYE&inde...	3	2
3	3	Safe Use of Electricity	https://www.youtube.com/watch?v=K1iQfTqSGt4&list=PLl5XPC31rjXL8DadJEI7dFXhTpi9VjBYE&inde...	3	1
4	4	Deneme	https://www.youtube.com/watch?v=J5GzSrhIOnA	1	1

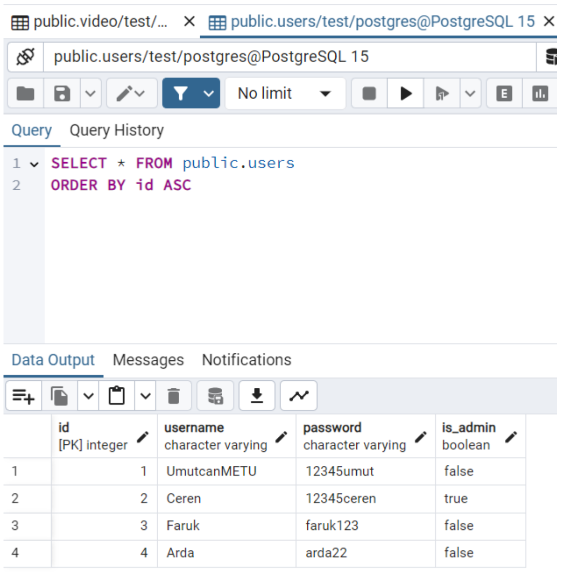
Figure 9: create DB-1



```sql
INSERT INTO users (id, username, password, is_admin) VALUES
(1, 'UmutcanMETU', '12345umut', false),
(2, 'Ceren', '12345ceren', true),
(3, 'Faruk', 'faruk123', false),
(4, 'Arda', 'arda22', false);
```

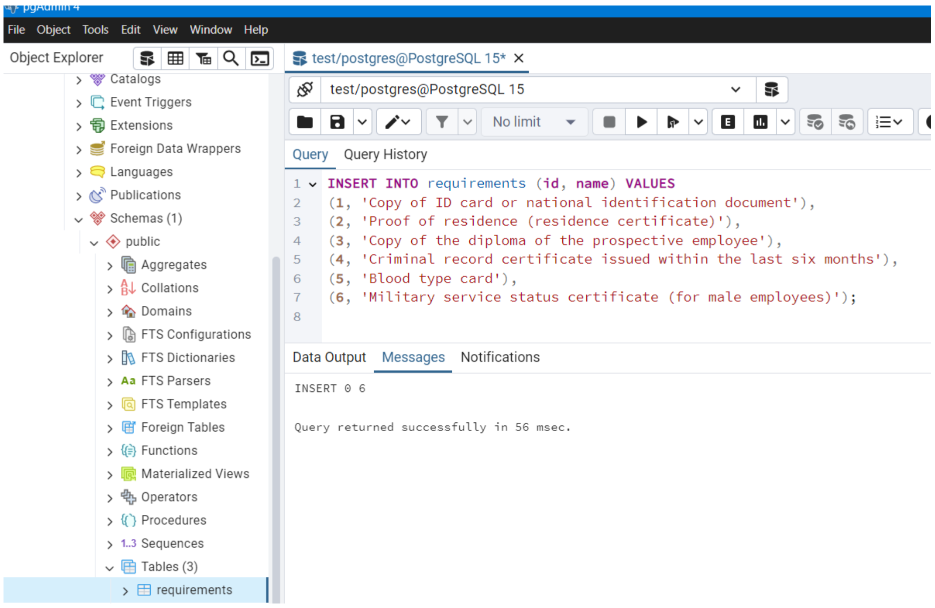
```
INSERT 0 4

Query returned successfully in 61 msec.
```

Figure 10: create DB-2


public.video/test/...
public.users/test/postgres@PostgreSQL 15
public.users/test/postgres@PostgreSQL 15
No limit
Query
Query History
Data Output
Messages
Notifications

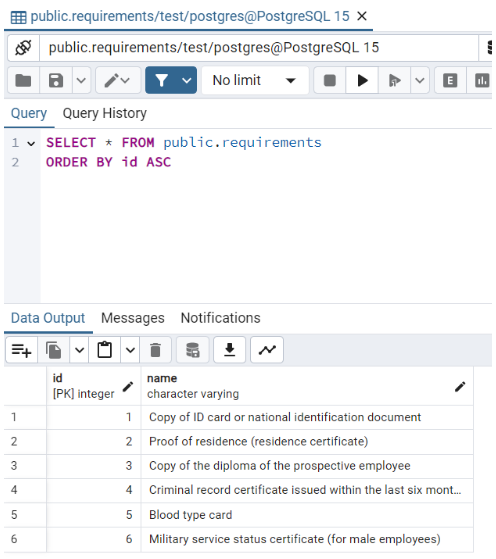
```sql
SELECT * FROM public.users
ORDER BY id ASC
```

	id [PK] integer	username character varying	password character varying	is_admin boolean
1	1	UmutcanMETU	12345umut	false
2	2	Ceren	12345ceren	true
3	3	Faruk	faruk123	false
4	4	Arda	arda22	false



```sql
INSERT INTO requirements (id, name) VALUES
(1, 'Copy of ID card or national identification document'),
(2, 'Proof of residence (residence certificate)'),
(3, 'Copy of the diploma of the prospective employee'),
(4, 'Criminal record certificate issued within the last six months'),
(5, 'Blood type card'),
(6, 'Military service status certificate (for male employees)');
```

```
INSERT 0 6

Query returned successfully in 56 msec.
```


public.requirements/test/postgres@PostgreSQL 15

```
SELECT * FROM public.requirements
ORDER BY id ASC
```

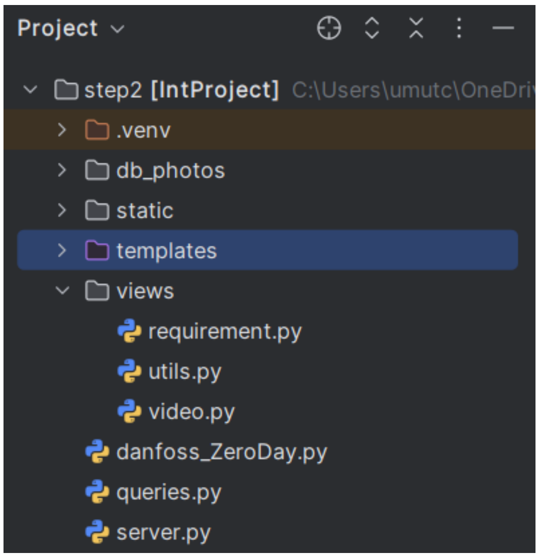
Data Output   Messages   Notifications

	id [PK] integer	name character varying
1	1	Copy of ID card or national identification document
2	2	Proof of residence (residence certificate)
3	3	Copy of the diploma of the prospective employee
4	4	Criminal record certificate issued within the last six mont...
5	5	Blood type card
6	6	Military service status certificate (for male employees)

Figure 11: create DB-3

<u>Week 7 (Nov 11 - 17) -- Week 8 (Nov 18 - 24)</u>

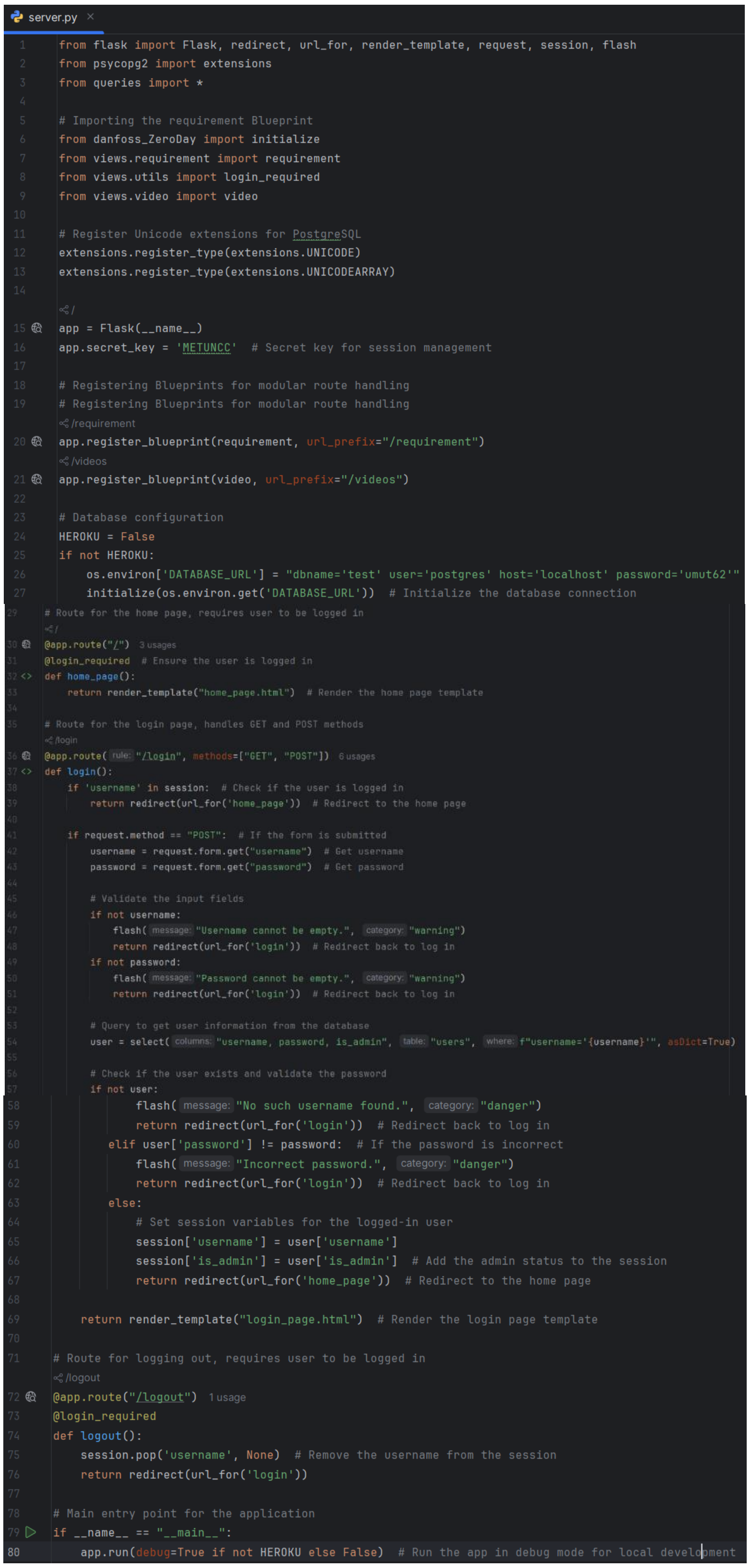
Between these weeks I completed the main part of my code. You can see the file structure of my main code in figure 12. Since I am using Blueprint, I have added these codes by opening an extra file named views.



```
Project
step2 [IntProject] C:\Users\umutc\OneDri
  .venv
  db_photos
  static
  templates
  views
    requirement.py
    utils.py
    video.py
  danfoss_ZeroDay.py
  queries.py
  server.py
```

Figure 12: main codes structure

I want to talk a little bit about the important parts of my code. You can see my server.py code in Figure 13.

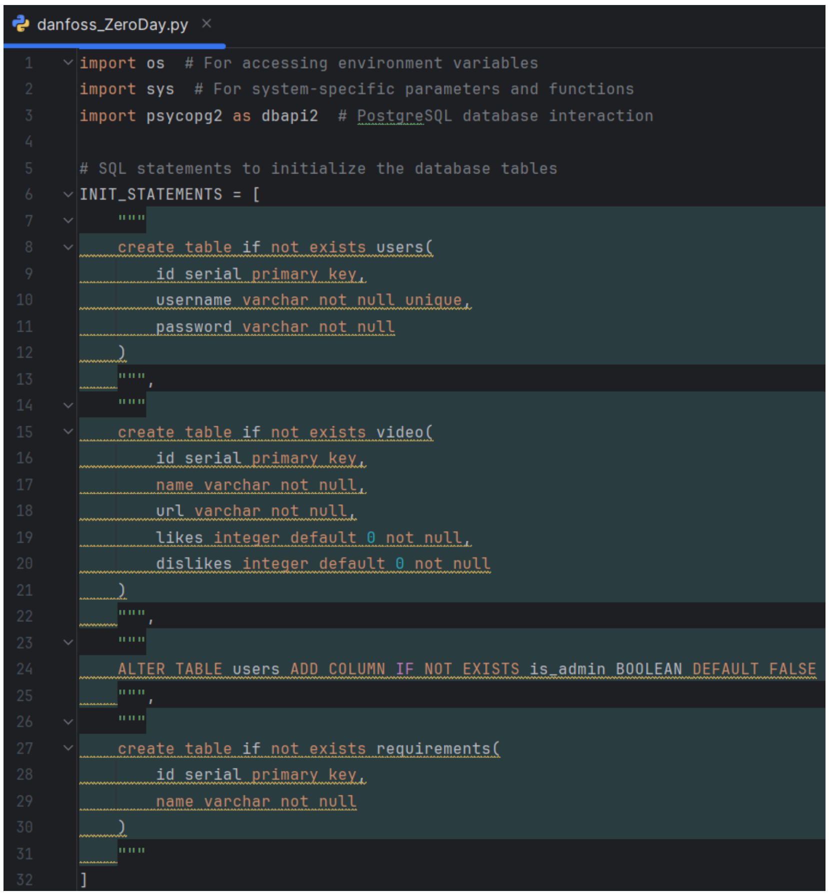


```python
from flask import Flask, redirect, url_for, render_template, request, session, flash
from psycopg2 import extensions
from queries import *

# Importing the requirement Blueprint
from danfoss_ZeroDay import initialize
from views.requirement import requirement
from views.utils import login_required
from views.video import video

# Register Unicode extensions for PostgreSQL
extensions.register_type(extensions.UNICODE)
extensions.register_type(extensions.UNICODEARRAY)

app = Flask(__name__)
app.secret_key = 'METUNCC'  # Secret key for session management

# Registering Blueprints for modular route handling
# Registering Blueprints for modular route handling
app.register_blueprint(requirement, url_prefix="/requirement")
app.register_blueprint(video, url_prefix="/videos")

# Database configuration
HEROKU = False
if not HEROKU:
    os.environ['DATABASE_URL'] = "dbname='test' user='postgres' host='localhost' password='umut62'"
    initialize(os.environ.get('DATABASE_URL'))  # Initialize the database connection
# Route for the home page, requires user to be logged in
@app.route("/")
@login_required  # Ensure the user is logged in
def home_page():
    return render_template("home_page.html")  # Render the home page template

# Route for the login page, handles GET and POST methods
@app.route("/login", methods=["GET", "POST"])
def login():
    if 'username' in session:  # Check if the user is logged in
        return redirect(url_for('home_page'))  # Redirect to the home page

    if request.method == "POST":  # If the form is submitted
        username = request.form.get("username")  # Get username
        password = request.form.get("password")  # Get password

        # Validate the input fields
        if not username:
            flash("Username cannot be empty.", "warning")
            return redirect(url_for('login'))  # Redirect back to log in
        if not password:
            flash("Password cannot be empty.", "warning")
            return redirect(url_for('login'))  # Redirect back to log in

        # Query to get user information from the database
        user = select("username, password, is_admin", "users", f"username='{username}'", asDict=True)

        # Check if the user exists and validate the password
        if not user:
            flash("No such username found.", "danger")
            return redirect(url_for('login'))  # Redirect back to log in
        elif user['password'] != password:  # If the password is incorrect
            flash("Incorrect password.", "danger")
            return redirect(url_for('login'))  # Redirect back to log in
        else:
            # Set session variables for the logged-in user
            session['username'] = user['username']
            session['is_admin'] = user['is_admin']  # Add the admin status to the session
            return redirect(url_for('home_page'))  # Redirect to the home page

    return render_template("login_page.html")  # Render the login page template

# Route for logging out, requires user to be logged in
@app.route("/logout")
@login_required
def logout():
    session.pop('username', None)  # Remove the username from the session
    return redirect(url_for('login'))

# Main entry point for the application
if __name__ == "__main__":
    app.run(debug=True if not HEROKU else False)  # Run the app in debug mode for local development
```

Figure 13: server.py

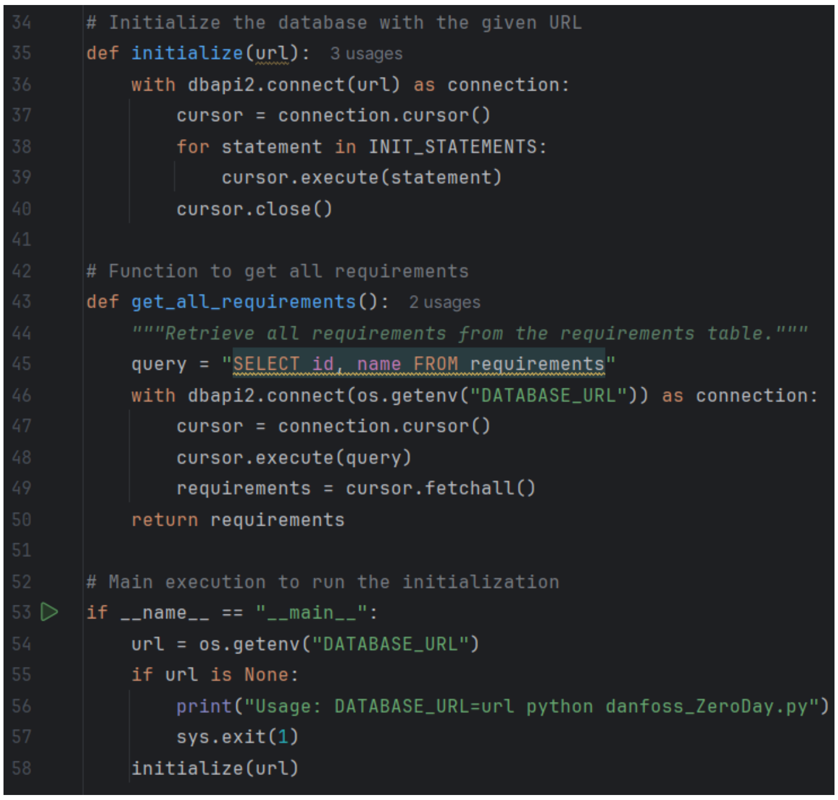
In my code, I created a basic web app structure using Flask and PostgreSQL. First, I imported necessary libraries for routing, templates, and database functions. Then, I set up **Blueprints** to keep things organized, like `/requirement` and `/videos` sections, which helps to manage routes separately. I also configured the PostgreSQL database connection for local development and prepared it for deployment on Heroku. The main parts include a **login system** where users enter a username and password, and I check if the info matches the database. If correct, the user can access the home page; if not, they get a warning. I also added a **logout** function to clear the session. Finally, the app runs in debug mode locally but will be turned off on Heroku. Next steps could involve adding admin features for more control over content.

Now I will talk about my danfoss_ZeroDay.py code where I manage my DB. In my code, I set up the database structure for my app using **PostgreSQL**. First, I imported libraries to work with environment variables and database connections. Then, I created some **SQL statements** that make important tables if they don't already exist. These include a user's table for storing usernames and passwords, a video table for videos with likes and dislikes, and a requirements table for important items. I also added an is_admin column to the users table to identify admin users. The initialize function runs these SQL statements when I connect to the database, setting up the tables automatically. I included a get_all_requirements function to get all entries from the requirements table. Finally, I added a part at the end to make sure the database URL is set; if not, it gives a message and stops. Next steps might include adding more functions to manage data inside these tables more easily. You can examine my code from Figure 14 and 15.


danfoss_ZeroDay.py

```python
import os  # For accessing environment variables
import sys  # For system-specific parameters and functions
import psycopg2 as dbapi2  # PostgreSQL database interaction

# SQL statements to initialize the database tables
INIT_STATEMENTS = [
    """
    create table if not exists users(
        id serial primary key,
        username varchar not null unique,
        password varchar not null
    )
    """,
    """
    create table if not exists video(
        id serial primary key,
        name varchar not null,
        url varchar not null,
        likes integer default 0 not null,
        dislikes integer default 0 not null
    )
    """,
    """
    ALTER TABLE users ADD COLUMN IF NOT EXISTS is_admin BOOLEAN DEFAULT FALSE
    """,
    """
    create table if not exists requirements(
        id serial primary key,
        name varchar not null
    )
    """
]
```

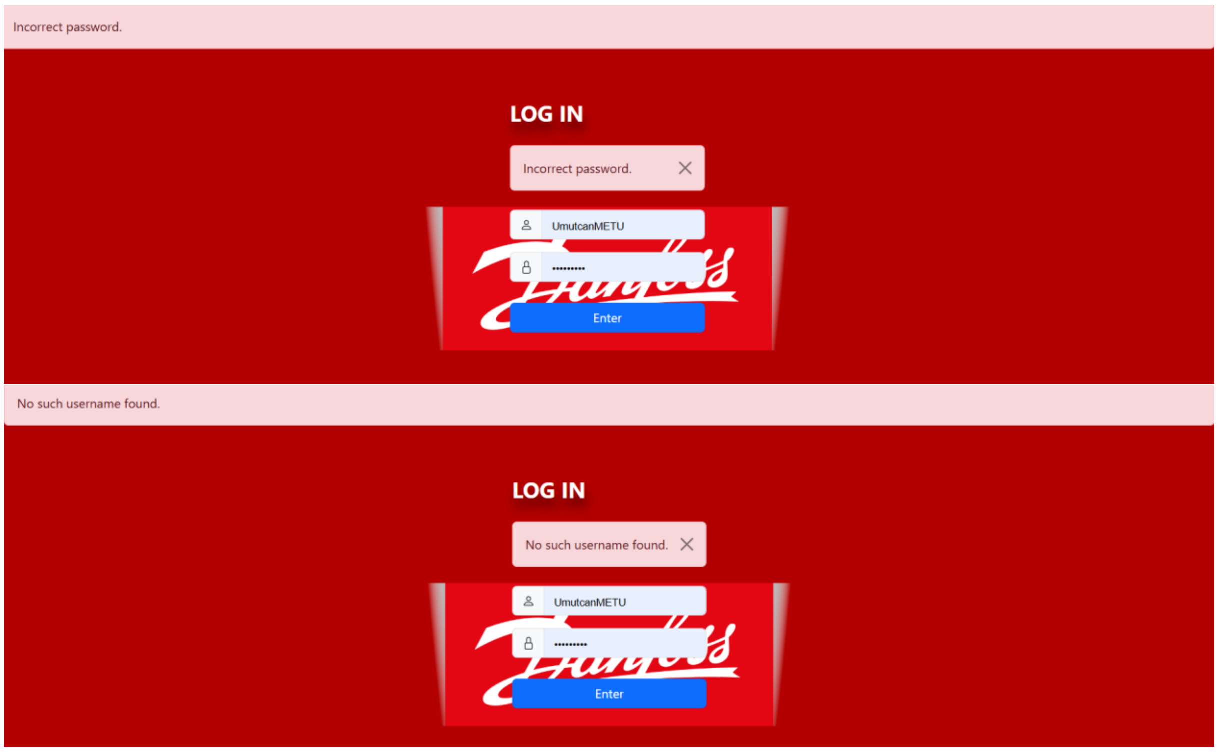
Figure 14: danfoss_ZeroDay.py1



```python
# Initialize the database with the given URL
def initialize(url):  3 usages
    with dbapi2.connect(url) as connection:
        cursor = connection.cursor()
        for statement in INIT_STATEMENTS:
            cursor.execute(statement)
        cursor.close()

# Function to get all requirements
def get_all_requirements():  2 usages
    """Retrieve all requirements from the requirements table."""
    query = "SELECT id, name FROM requirements"
    with dbapi2.connect(os.getenv("DATABASE_URL")) as connection:
        cursor = connection.cursor()
        cursor.execute(query)
        requirements = cursor.fetchall()
    return requirements

# Main execution to run the initialization
if __name__ == "__main__":
    url = os.getenv("DATABASE_URL")
    if url is None:
        print("Usage: DATABASE_URL=url python danfoss_ZeroDay.py")
        sys.exit(1)
    initialize(url)
```

Figure 15: danfoss_ZeroDay.py2

While writing these codes, of course I also did tests, you can find these tests in figure 16.



Incorrect password.

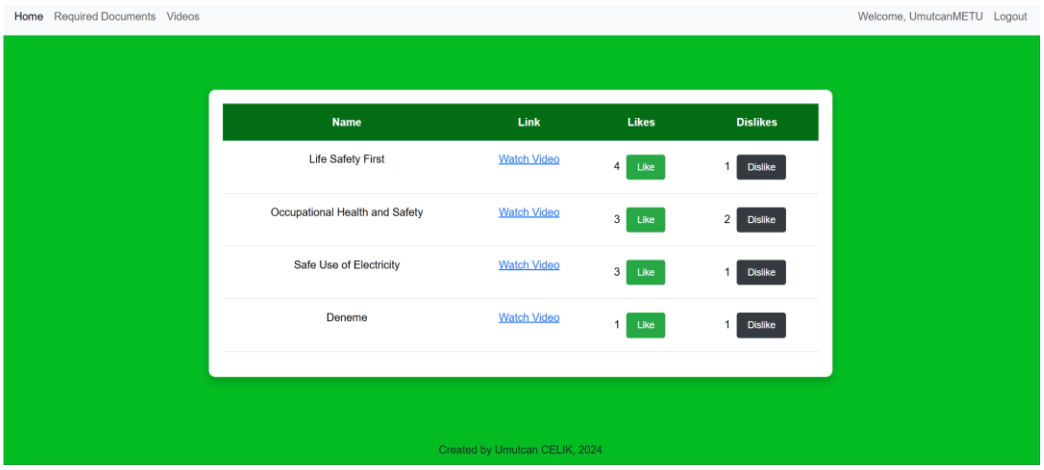
Incorrect password.

No such username found.

No such username found.

Figure 16: test

The test I did in Figure 17 was to check if the DB was updated for the videos.



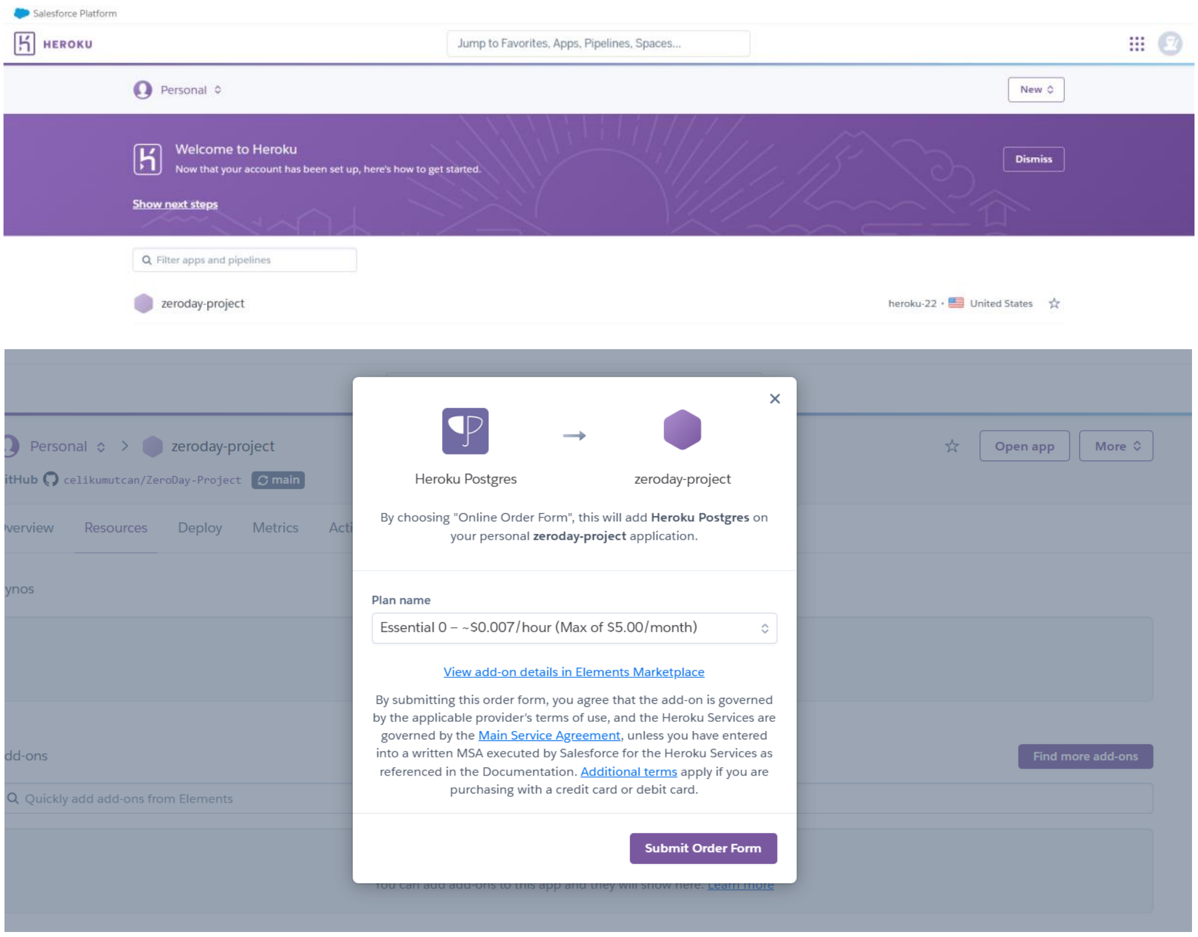
Home Required Documents Videos Welcome, UmutcanMETU Logout

Name	Link	Likes	Dislikes
Life Safety First	Watch Video	4 Like	1 Dislike
Occupational Health and Safety	Watch Video	3 Like	2 Dislike
Safe Use of Electricity	Watch Video	3 Like	1 Dislike
Deneme	Watch Video	1 Like	1 Dislike

Created by Umutcan CELIK, 2024

Figure 17: test DB

After creating a Heroku account, I uploaded a test code and learned how to integrate it with postgresql. You can see this process in figure 18.


Salesforce Platform
HEROKU
Jump to Favorites, Apps, Pipelines, Spaces...
Personal
New
Dismiss
Filter apps and pipelines
Personal > zeroday-project
Open app
More
Overview Resources Deploy Metrics
Find more add-ons
Quickly add add-ons from Elements

Welcome to Heroku

Now that your account has been set up, here's how to get started.

Show next steps

zeroday-project heroku-22 · United States

GitHub celikumutcan/ZeroDay-Project main

Heroku Postgres → zeroday-project

By choosing "Online Order Form", this will add **Heroku Postgres** on your personal **zeroday-project** application.

Plan name

Essential 0 – ~$0.007/hour (Max of $5.00/month)

View add-on details in Elements Marketplace

By submitting this order form, you agree that the add-on is governed by the applicable provider's terms of use, and the Heroku Services are governed by the Main Service Agreement, unless you have entered into a written MSA executed by Salesforce for the Heroku Services as referenced in the Documentation. Additional terms apply if you are purchasing with a credit card or debit card.

Submit Order Form

Figure 18: create Heroku account

The difficulties I had in my codes that I have completed so far were especially difficult to organize and connect the code accordingly, especially since I learned to use Blueprint from the beginning. Even though I didn't take a course on Pyton-Flask, creating this code took me time to research and learn. I made mistakes, but in the end I got a good result. For Heroku, I recommend using the recommended "Salesforce Authenticator" to avoid the "Authenticator" problem. I used "Microsoft Authenticator" and after a while I couldn't access the account. I contacted Help Support and after 2 days I was able to reactivate the account.

Milestones Remained

Week 9 (Nov 25 - Dec 1)

I will add Admin face to my project. For example, Admin will be able to add and remove videos, users and requirements easily on the page. I will also update the interfaces, for example, I will add a nice design to the requirements page.

Week 10 (Dec 2 - 8)

I will add 100-200 users to the DB and complete the necessary tests.

Week 11 (Dec 9 – Dec 15)

I will move my project completely to the Cloud (Heroku) and complete the corrections requested by Hamzeh hocam.

Kodlarımın şimdilik tamamlanmış haline github hesabımdan ulaşabilirsiniz:

https://github.com/celikumutcan/Zeroday_ProjectStep2

Github username: celikumutcan

References

1. *Platform as a service | Heroku*. (n.d.). https://www.heroku.com/platform

2. *Fully Managed database as a service - PostgreSQL | Heroku*. (n.d.). https://www.heroku.com/postgres

3. *Data security in cloud computing using AES under HEROKU cloud*. (2018, April 1). IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/abstract/document/8372705

4. Andersson, N., & Chernov, A. (2016). *Increasing the throughput of a Node.js application : running on the Heroku Cloud App platform*. DIVA. https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A954978&dswid=5210