



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: 11.12.2024

Table of Contents

Project Description.....	4
Cloud Delivery Models	4
<i>SaaS (Software as a Service):.....</i>	<i>4</i>
<i>PaaS (Platform as a Service):</i>	<i>4</i>
<i>IaaS (Infrastructure as a Service):.....</i>	<i>4</i>
Diagrams	5
<i>Use Case Diagram</i>	<i>5</i>
<i>Data Flow Diagram.....</i>	<i>5</i>
Data Types	6
Computation	7
Expected Contribution	7
Milestones Achieved.....	7
<i>Week 4 (Oct 21 - 27) -- Week 5 (Oct 28 - Nov 3)</i>	<i>7</i>
<i>Week 6 (Nov 4 - 10)</i>	<i>9</i>
<i>Week 7 (Nov 11 - 17) -- Week 8 (Nov 18 - 24)</i>	<i>11</i>
Milestones Remained	16
<i>Week 9 (Nov 25 - Dec 1).....</i>	<i>16</i>
<i>Week 10 (Dec 2 - 8)</i>	<i>16</i>
<i>Week 11 (Dec 9 – Dec 15)</i>	<i>16</i>
References.....	17

LIST OF FIGURES AND TABLES

Figure 1: use case diagram of project.....	5
Figure 2: context level of project	5
Figure 3: Level 0 of project.....	6
Figure 4: HTML and CSS code layout.....	7
Figure 5: login screen.....	8
Figure 6: home page screen.....	8
Figure 7: documents screen.....	8
Figure 8: videos screen.....	9
Figure 9: create DB-1.....	9
Figure 10: create DB-2.....	10
Figure 11: create DB-3.....	11
Figure 12: main codes structure	11
Figure 13: server.py.....	12
Figure 14: danfoss_ZeroDay.py1	13
Figure 15: danfoss_ZeroDay.py2	14
Figure 16: test.....	14
Figure 17: test DB.....	15
Figure 18: create Heroku account	15

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.

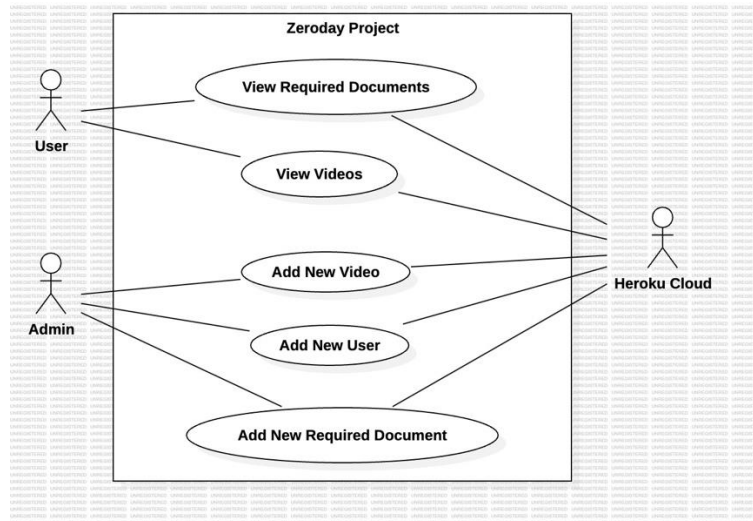


Figure 1: use case diagram of project

Data Flow Diagram

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

Context Level:

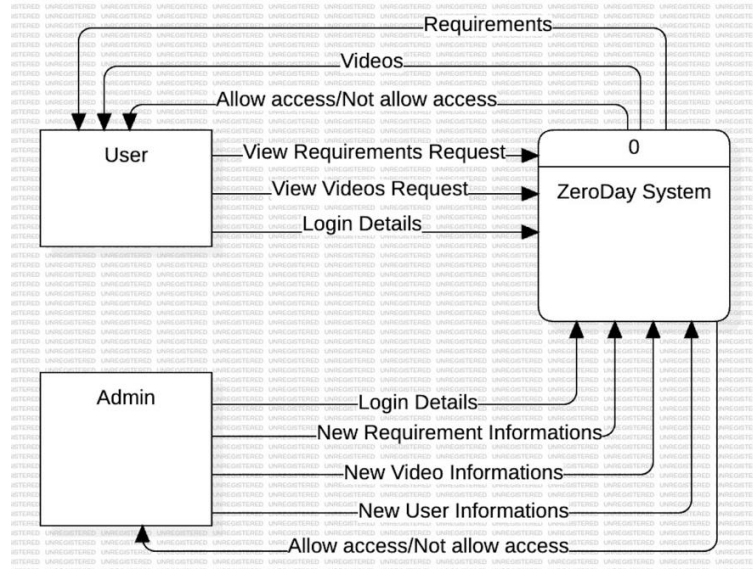


Figure 2: context level of project

Level 0:

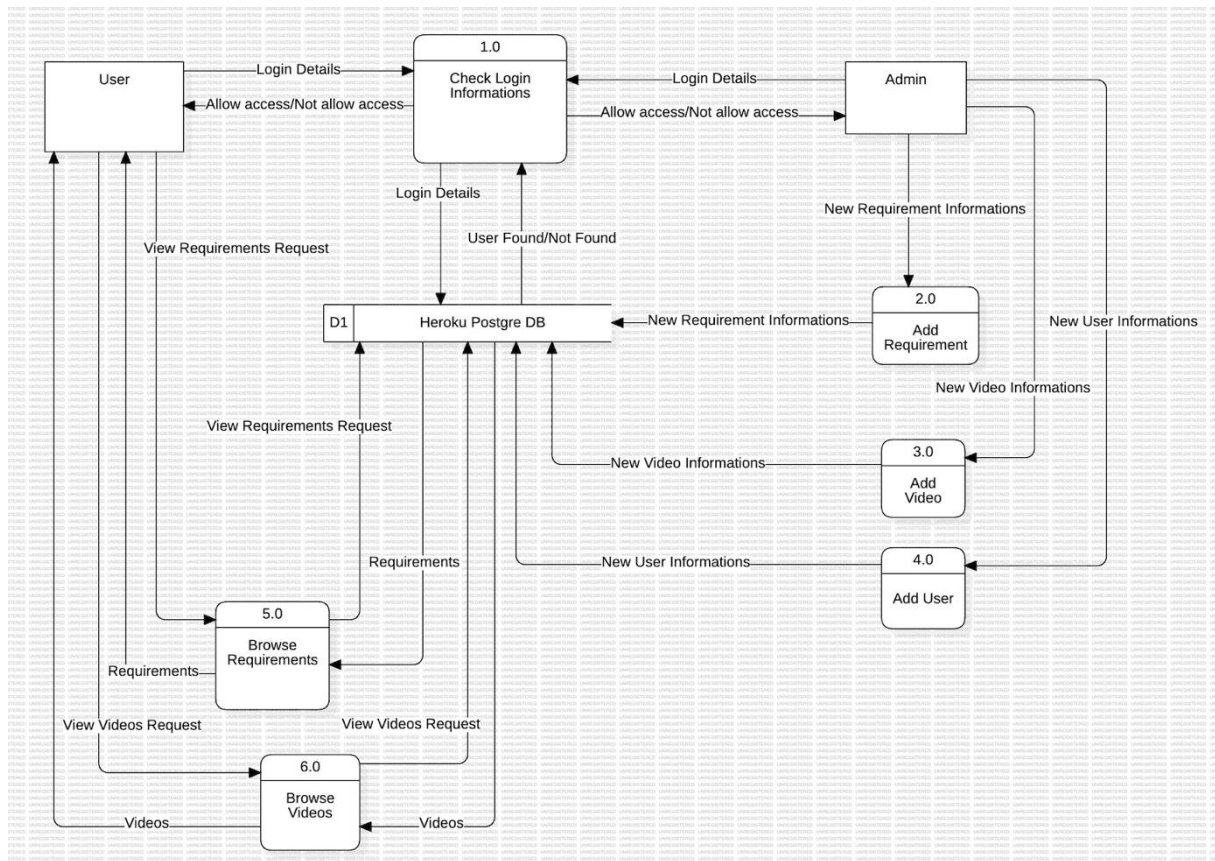


Figure 3: Level 0 of project

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

Week 4 (Oct 21 - 27) -- Week 5 (Oct 28 - Nov 3)

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.

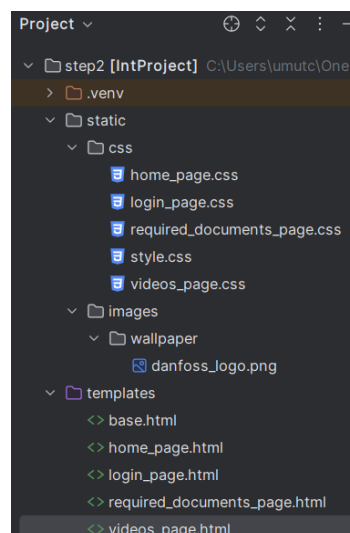


Figure 4: HTML and CSS code layout

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

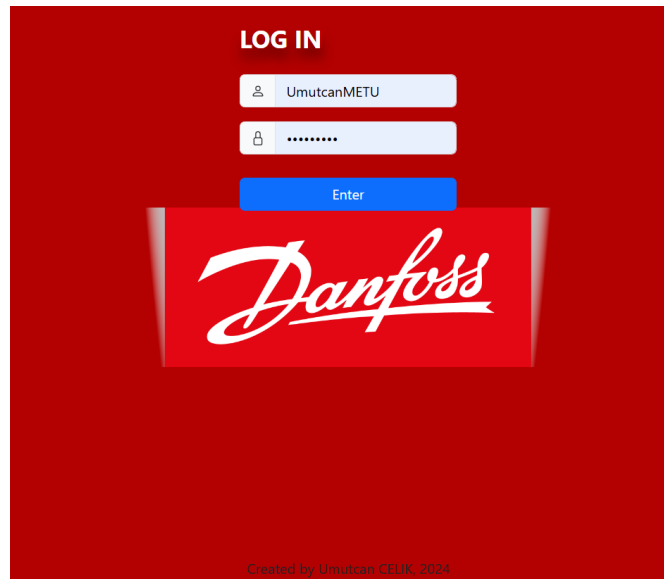


Figure 5: login screen

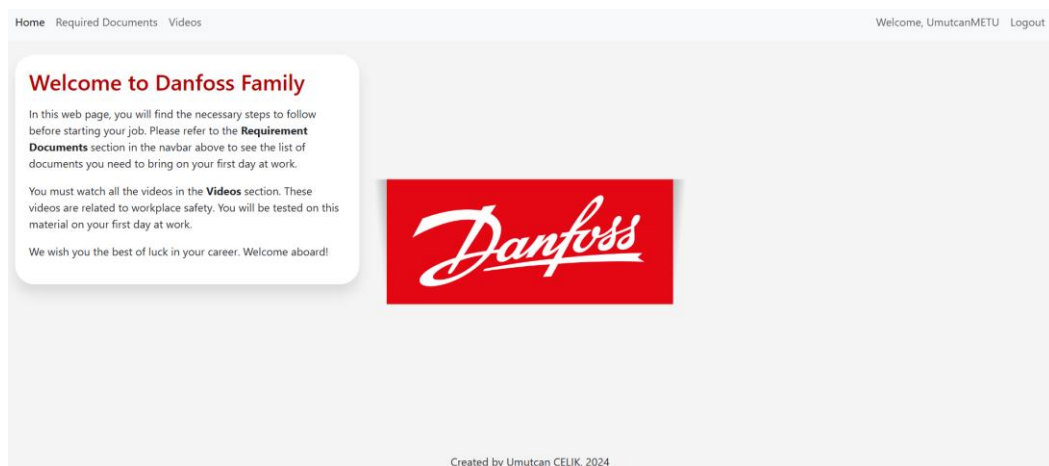


Figure 6: home page screen

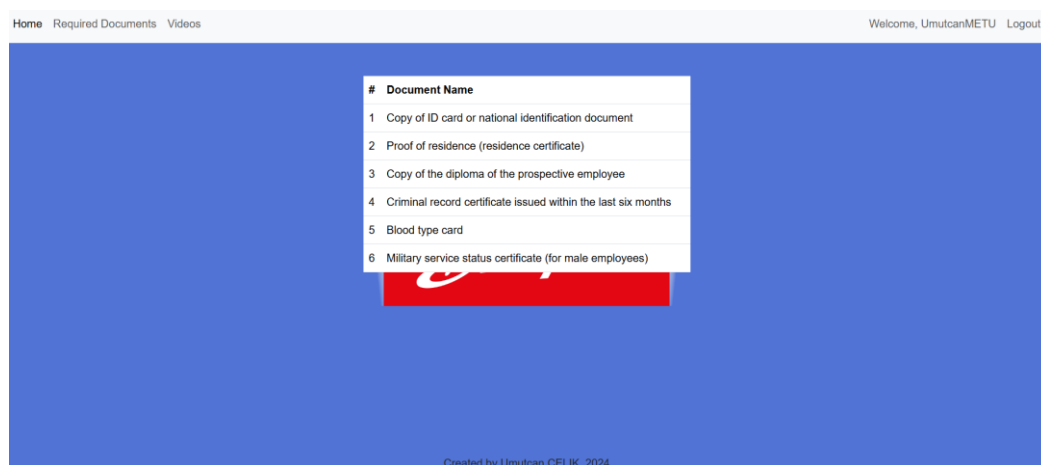


Figure 7: documents screen

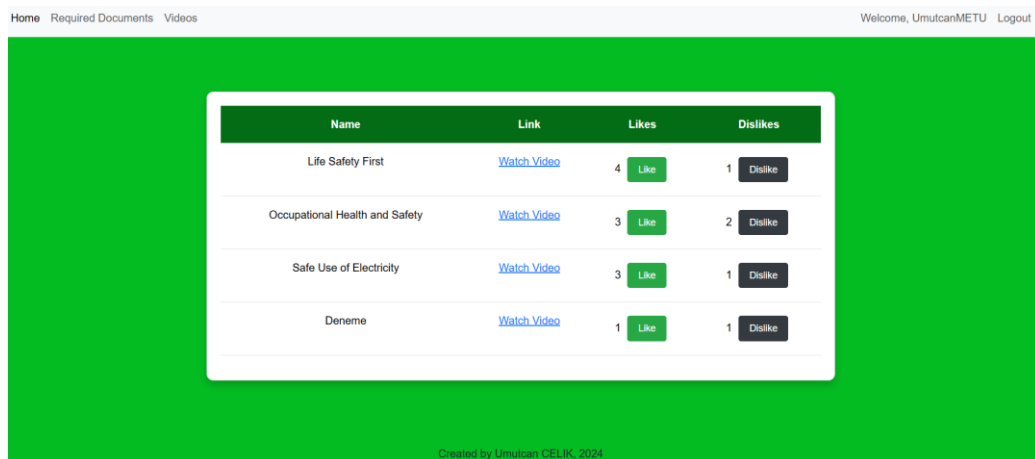


Figure 8: videos screen

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.

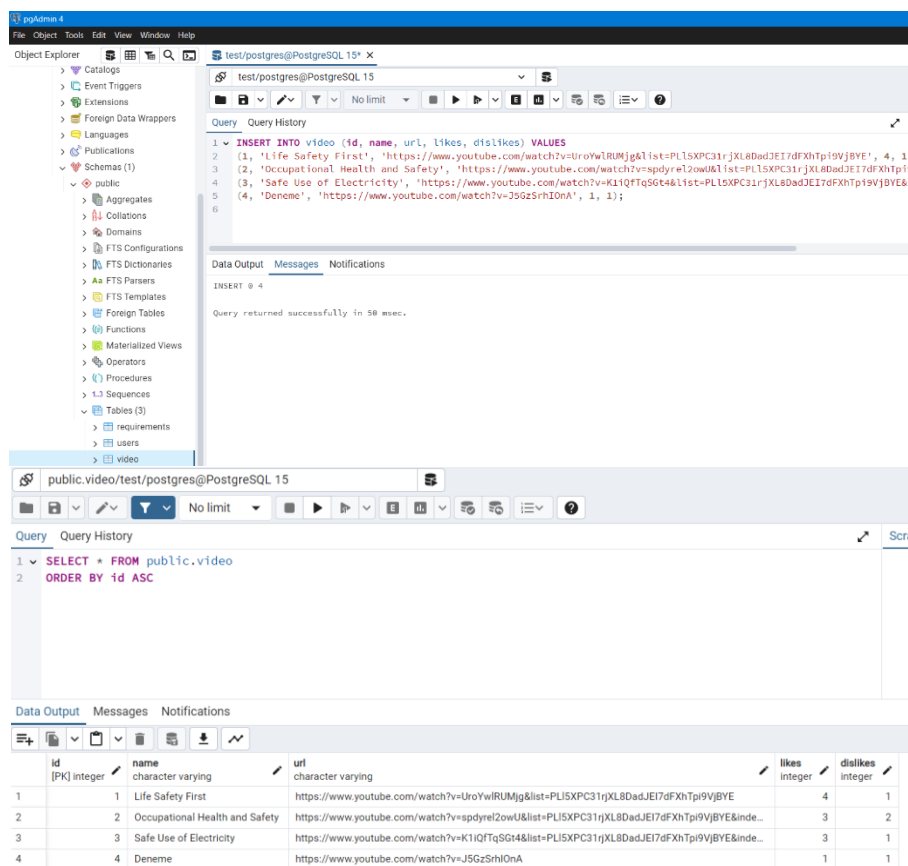


Figure 9: create DB-1

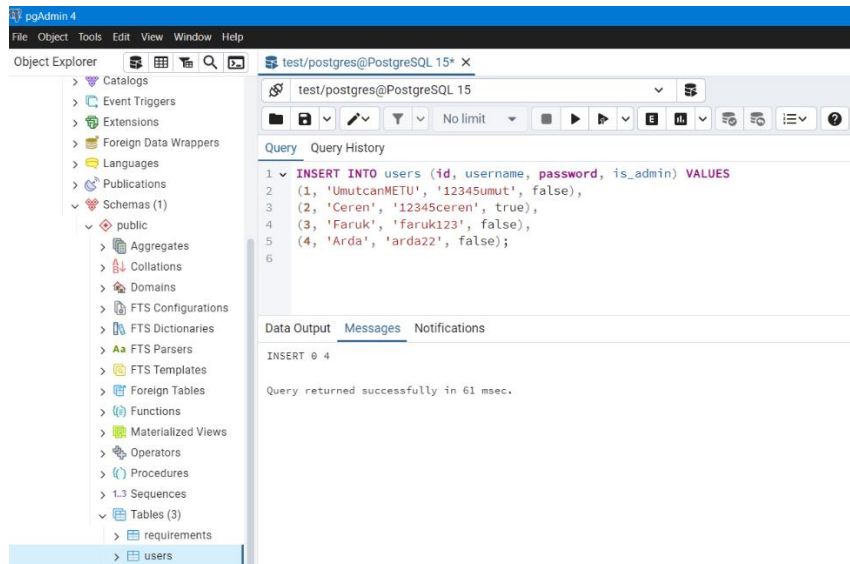


Figure 10: create DB-2

public.video/test/... X public.users/test/postgres@PostgreSQL 15 X

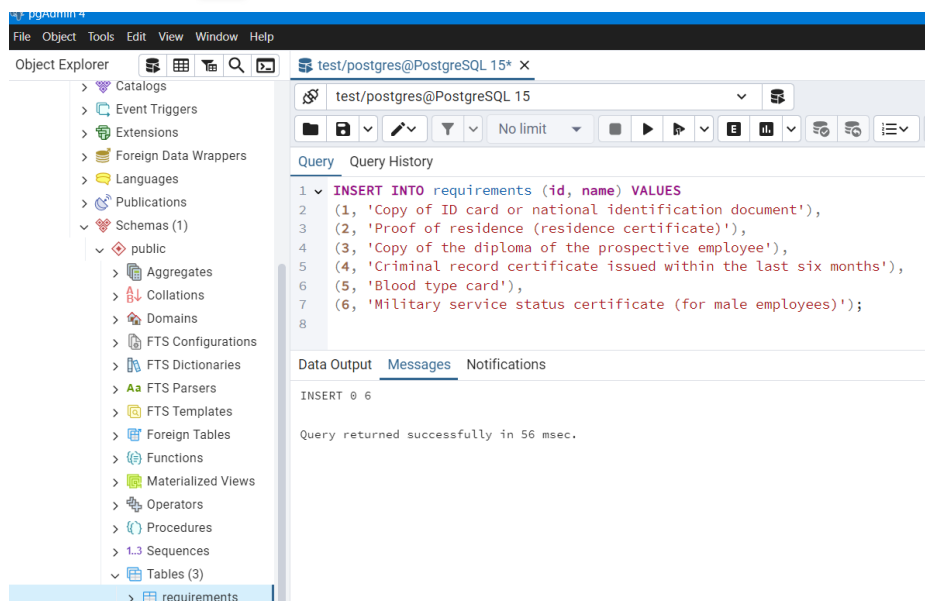
public.users/test/postgres@PostgreSQL 15

Query Query History

```
1 SELECT * FROM public.users
2 ORDER BY id ASC
```

Data Output Messages Notifications

	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



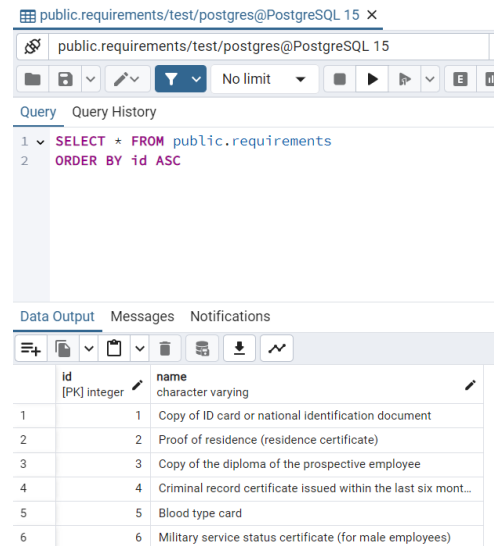


Figure 11: create DB-3

Week 7 (Nov 11 - 17) -- Week 8 (Nov 18 - 24)

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.

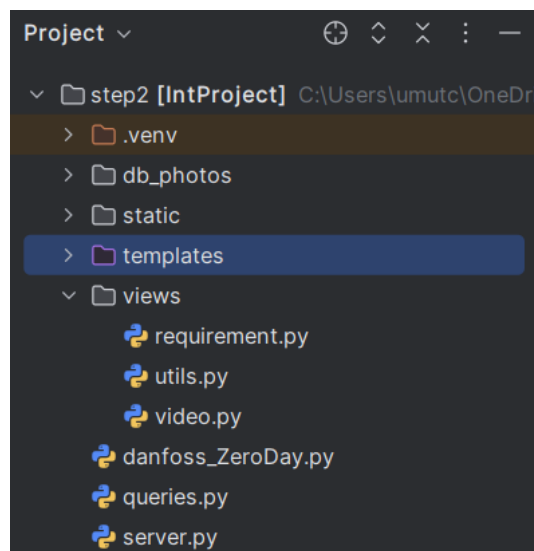


Figure 12: main codes structure

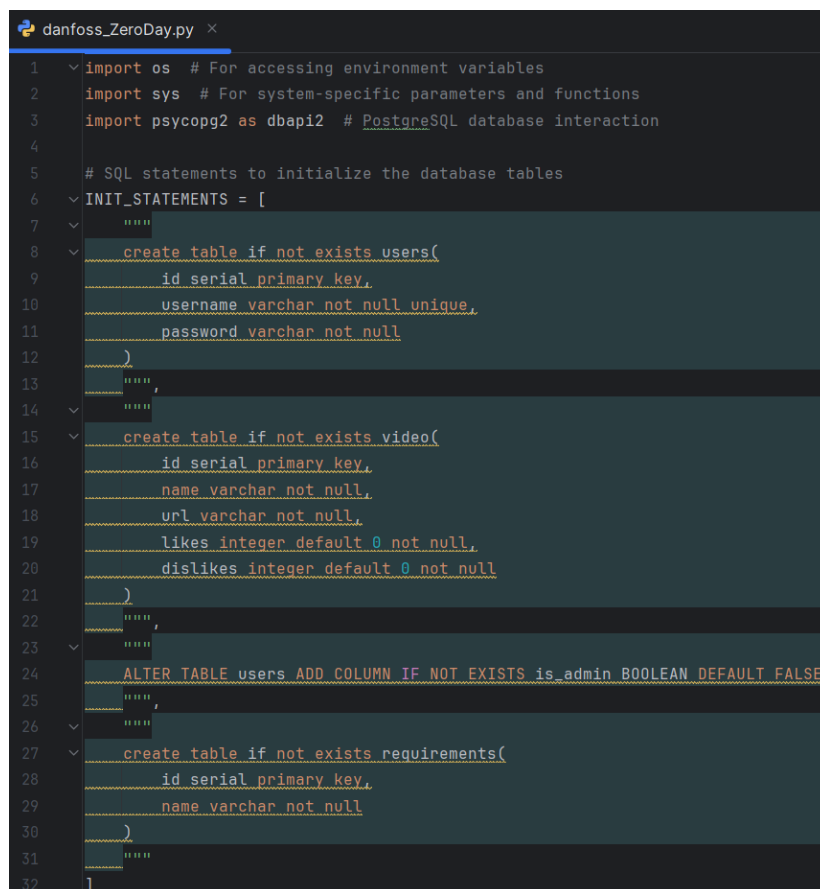
I want to talk a little bit about the important parts of my code. You can see my server.py code in Figure 13.

```
server.py x
1 from flask import Flask, redirect, url_for, render_template, request, session, flash
2 from psychopg2 import extensions
3 from queries import *
4
5 # Importing the requirement Blueprint
6 from danfoss_ZeroDay import initialize
7 from views.requirement import requirement
8 from views.utils import login_required
9 from views.video import video
10
11 # Register Unicode extensions for PostgreSQL
12 extensions.register_type(extensions.UNICODE)
13 extensions.register_type(extensions.UNICODEARRAY)
14
15 <!--
16 app = Flask(__name__)
17 app.secret_key = 'METUNCC' # Secret key for session management
18
19 # Registering Blueprints for modular route handling
20 # Registering Blueprints for modular route handling
21 </requirement
22 app.register_blueprint(requirement, url_prefix="/requirement")
23 </videos
24 app.register_blueprint(video, url_prefix="/videos")
25
26 # Database configuration
27 HEROKU = False
28 if not HEROKU:
29     os.environ['DATABASE_URL'] = "dbname='test' user='postgres' host='localhost' password='umut62'"
30     initialize(os.environ.get('DATABASE_URL')) # Initialize the database connection
31
32 # Route for the home page, requires user to be logged in
33 <!--
34 @app.route("/") 3 usages
35 @login_required # Ensure the user is logged in
36 <--> def home_page():
37     return render_template("home_page.html") # Render the home page template
38
39 # Route for the login page, handles GET and POST methods
40 <!-- Login
41 @app.route("/login", methods=["GET", "POST"]) 6 usages
42 <--> def login():
43     if 'username' in session: # Check if the user is logged in
44         return redirect(url_for('home_page')) # Redirect to the home page
45
46     if request.method == "POST": # If the form is submitted
47         username = request.form.get("username") # Get username
48         password = request.form.get("password") # Get password
49
50         # Validate the input fields
51         if not username:
52             flash(message="Username cannot be empty.", category="warning")
53             return redirect(url_for('login')) # Redirect back to log in
54         if not password:
55             flash(message="Password cannot be empty.", category="warning")
56             return redirect(url_for('login')) # Redirect back to log in
57
58         # Query to get user information from the database
59         user = select(columns="username, password, is_admin", table="users", where=f"username='{username}'", asDict=True)
60
61         # Check if the user exists and validate the password
62         if not user:
63             flash(message="No such username found.", category="danger")
64             return redirect(url_for('login')) # Redirect back to log in
65         elif user['password'] != password: # If the password is incorrect
66             flash(message="Incorrect password.", category="danger")
67             return redirect(url_for('login')) # Redirect back to log in
68         else:
69             # Set session variables for the logged-in user
70             session['username'] = user['username']
71             session['is_admin'] = user['is_admin'] # Add the admin status to the session
72             return redirect(url_for('home_page')) # Redirect to the home page
73
74     return render_template("login_page.html") # Render the login page template
75
76 # Route for logging out, requires user to be logged in
77 <!-- Logout
78 @app.route("/logout") 1 usage
79 @login_required
80 def logout():
81     session.pop('username', None) # Remove the username from the session
82     return redirect(url_for('login'))
83
84 # Main entry point for the application
85 if __name__ == "__main__":
86     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 x
1  import os # For accessing environment variables
2  import sys # For system-specific parameters and functions
3  import psycopg2 as dbapi2 # PostgreSQL database interaction
4
5  # SQL statements to initialize the database tables
6  INIT_STATEMENTS = [
7      """
8      create table if not exists users(
9          id serial primary key,
10         username varchar not null unique,
11         password varchar not null
12     )
13     """,
14     """
15     create table if not exists video(
16         id serial primary key,
17         name varchar not null,
18         url varchar not null,
19         likes integer default 0 not null,
20         dislikes integer default 0 not null
21     )
22     """,
23     """
24     ALTER TABLE users ADD COLUMN IF NOT EXISTS is_admin BOOLEAN DEFAULT FALSE
25     """,
26     """
27     create table if not exists requirements(
28         id serial primary key,
29         name varchar not null
30     )
31     """,
32 ]
```

Figure 14: `danfoss_ZeroDay.py`

```

34 # Initialize the database with the given URL
35 def initialize(url): 3 usages
36     with dbapi2.connect(url) as connection:
37         cursor = connection.cursor()
38         for statement in INIT_STATEMENTS:
39             cursor.execute(statement)
40         cursor.close()
41
42 # Function to get all requirements
43 def get_all_requirements(): 2 usages
44     """Retrieve all requirements from the requirements table."""
45     query = "SELECT id, name FROM requirements"
46     with dbapi2.connect(os.getenv("DATABASE_URL")) as connection:
47         cursor = connection.cursor()
48         cursor.execute(query)
49         requirements = cursor.fetchall()
50     return requirements
51
52 # Main execution to run the initialization
53 > if __name__ == "__main__":
54     url = os.getenv("DATABASE_URL")
55     if url is None:
56         print("Usage: DATABASE_URL=url python danfoss_ZeroDay.py")
57         sys.exit(1)
58     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.

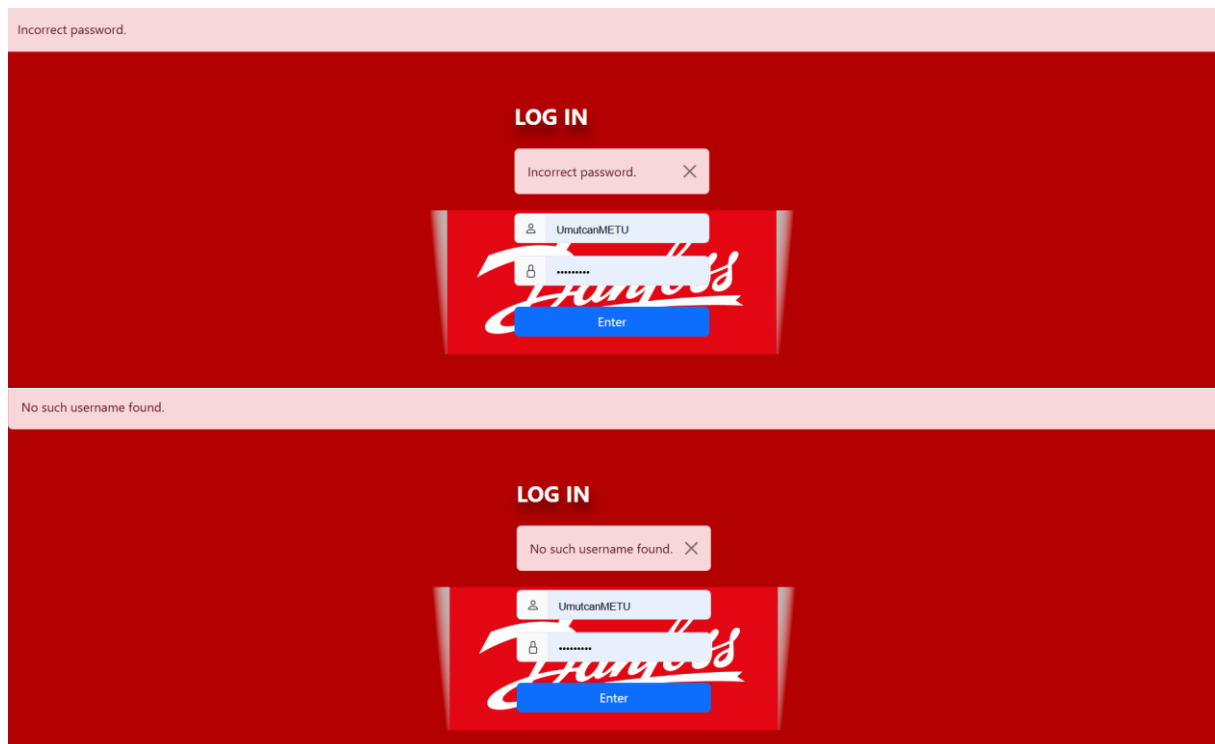
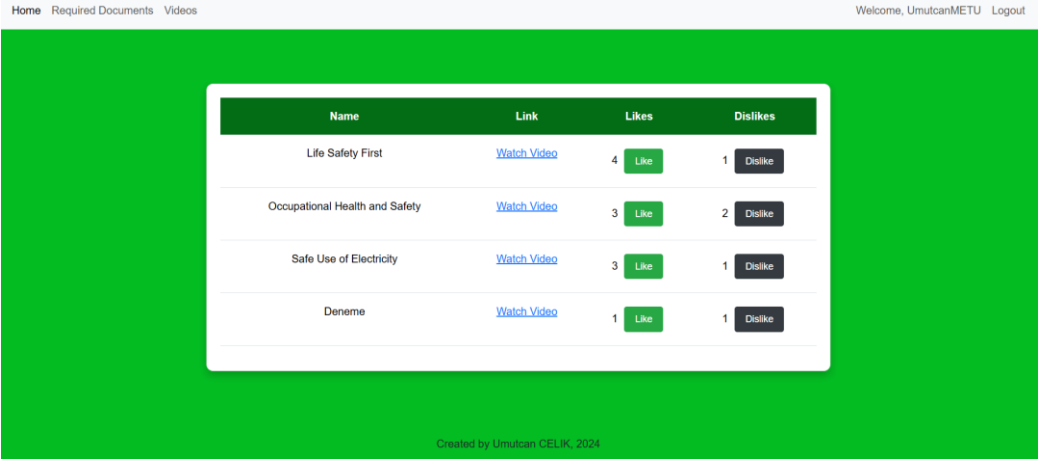


Figure 16: test

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



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

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.

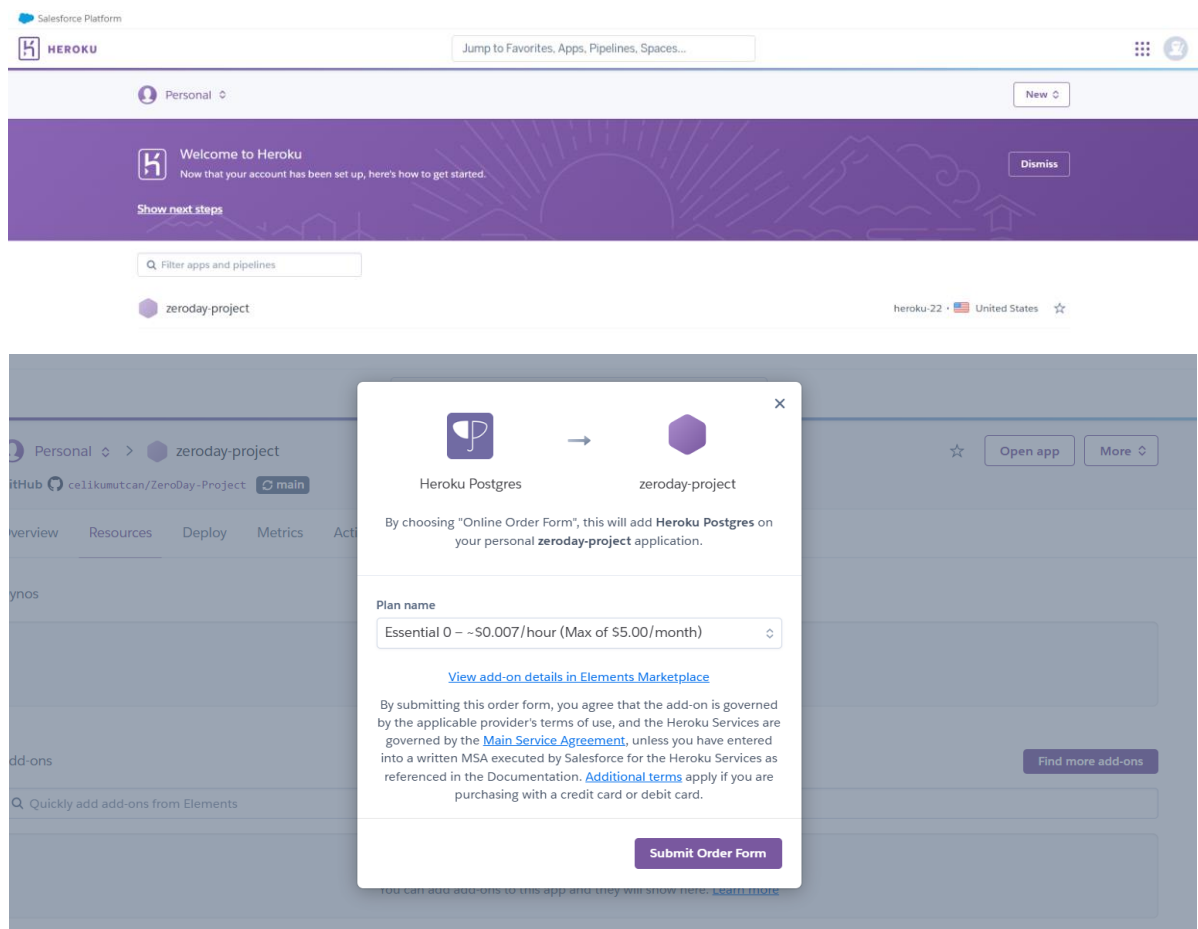


Figure 18: create Heroku account

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 Python-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.

You can find the completed code for now on my github account:

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>