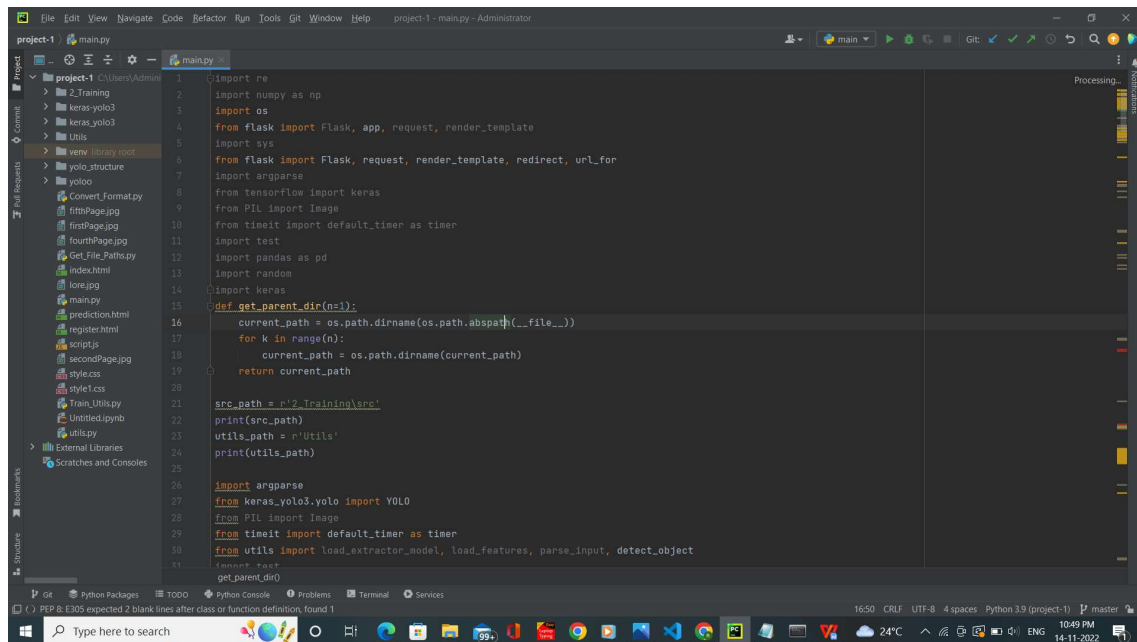


## SPRINT-4

Team ID	PNT2022TMID07004
Project Name	Project -AI-Based Localization and Classification of Skin Disease with Erythema



The screenshot displays the PyCharm IDE interface. The left sidebar shows a project structure with folders like 'project-1', '2.Training', 'keras\_yolo3', and 'Utils'. The main editor window shows a Python script named 'main.py' with the following code:

```
1 import re
2 import numpy as np
3 import os
4 from flask import Flask, app, request, render_template
5 import sys
6 from flask import Flask, request, render_template, redirect, url_for
7 import argparse
8 from tensorflow import keras
9 from PIL import Image
10 from timeit import default_timer as timer
11 import test
12 import pandas as pd
13 import random
14 import keras
15 def get_parent_dir(n=1):
16     current_path = os.path.dirname(os.path.abspath(__file__))
17     for k in range(n):
18         current_path = os.path.dirname(current_path)
19     return current_path
20
21 src_path = r'2.Training\src'
22 print(src_path)
23 utils_path = r'Utils'
24 print(utils_path)
25
26 import argparse
27 from keras_yolo3.yolo import YOLO
28 from PIL import Image
29 from timeit import default_timer as timer
30 from utils import load_extractor_model, load_features, parse_input, detect_object
31 import test
32 get_parent_dir()
```

The bottom status bar indicates the file encoding is UTF-8, the editor has 4 spaces, and the Python version is 3.9 (project-1). The system tray at the bottom shows the date and time as 14-11-2022, 10:49 PM.

The screenshot shows an IDE window titled "project-1 - main.py - Administrator". The left sidebar displays a project tree with folders like "2\_Training", "keras\_yolo3", "yolo\_structure", and "yolo". The main editor shows the code for "main.py", which is a Flask application. The code includes imports for "os", "path", and "Flask". It sets the "TF\_CPP\_MIN\_LOG\_LEVEL" to 3 and defines several paths for data, images, and model weights. The application is initialized with "app = Flask(\_\_name\_\_)" and "from cloudant.client import Cloudant". The status bar at the bottom indicates the file encoding is UTF-8, 4 spaces, Python 3.9, and the current branch is master.

```
17 os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"
18
19 data_folder = os.path.join(get_parent_dir(1), "yolo_structure", "Data")
20
21 image_folder = os.path.join(data_folder, "Source_Images")
22
23 os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"
24
25 data_folder = os.path.join(get_parent_dir(1), "yolo_structure", "Data")
26
27 image_folder = os.path.join(data_folder, "Source_Images")
28
29 image_test_folder = os.path.join(image_folder, "Training_Images")
30
31 detection_results_folder = os.path.join(image_folder, "Test_Image_Detection_Results")
32
33 detection_results_file = os.path.join(detection_results_folder, "Detection_Results.csv")
34
35 model_folder = os.path.join(data_folder, "Model_Weights")
36
37 model_weights = os.path.join(model_folder, "trained_weights_final.h5")
38 model_classes = os.path.join(model_folder, "data_classes.txt")
39
40 anchor_path = os.path.join(src_path, "keras_yolo3", "model_data", "yolo_anchors.txt")
41
42 FLAGS = None
43
44 app = Flask(__name__)
45
46 from cloudant.client import Cloudant
47 get_parent_dir()
```

The screenshot shows the same IDE window, but the code in "main.py" is more complete. It includes the "app" initialization, the "from cloudant.client import Cloudant" import, and the creation of a database client. The code defines routes for "index()", "home()", and "afterreg()", which uses the Cloudant client to create a database and store data. The status bar at the bottom indicates the file encoding is UTF-8, 4 spaces, Python 3.9, and the current branch is master.

```
64 app = Flask(__name__)
65
66 from cloudant.client import Cloudant
67 client = Cloudant.iam("8a78d8cb-ff5b-4e07-a4e1-d7b1d7a774f6-blumix", "zk059Fa0P9aIh3u0ifrnY4uL0LhE4Vnhy8KSLVFe08", connect=True)
68
69 my_database = client.create_database("my_database")
70
71 @app.route("/")
72 def index():
73     return render_template("index.html")
74
75 @app.route("/index.html")
76 def home():
77     return render_template("index.html")
78
79 @app.route("/afterreg", methods=['POST'])
80 def afterreg():
81     x = [x for x in request.form.values()]
82     print(x)
83     data = {
84         'id': x[1],
85         'name': x[0],
86         'psw': x[2]
87     }
88     print(data)
89
90 query = {'_id': {'$eq': data['_id']}}
91
92 data = my_database.get_document(my_id)
93
94 get_parent_dir()
```

The screenshot shows the VS Code editor with the file `main.py` open. The code implements a registration and login system. It starts by querying a database for documents. If the list is empty, it registers a new user by creating a document in the database and rendering the `register.html` template with a success message. If the list is not empty, it renders `register.html` with a message indicating the user is already a member. A `login` endpoint is defined, which renders `login.html`. An `afterlogin` endpoint is also defined, which takes user ID and password from the form, queries the database, and renders `login.html` with a message if the username is not found. The code also includes a `get_parent_dir` function.

```
docs = my_database.get_query_result(query)
print(docs)

print(len(docs.all()))

if (len(docs.all()) == 0):
    url = my_database.create_document(data)
    return render_template("register.html", pred="Registration Successful, please login using your details")
else:
    return render_template("register.html", pred="You are already a member, please login using your details")

@app.route("/login")
def login():
    return render_template("login.html")

@app.route("/afterlogin", methods=['POST'])
def afterlogin():
    user = request.form['_id']
    passw = request.form['psw']
    print(user, passw)

    query = {'_id': {'$eq': user}}

    docs = my_database.get_query_result(query)
    print(docs)

    print(len(docs.all()))

    if (len(docs.all()) == 0):
        return render_template("login.html", pred="The username is not found")

    get_parent_dir()
```

The screenshot shows the VS Code editor with the file `main.py` open. The code implements a logout and prediction system. It starts by querying a database for documents. If the list is empty, it renders `login.html` with a message indicating the username is not found. If the list is not empty, it checks if the user ID and password match the first document in the list. If they match, it redirects to the `prediction` endpoint. If they don't match, it prints "Invalid User". A `logout` endpoint is defined, which renders `logout.html`. A `prediction` endpoint is also defined, which uses `argparse` to parse command-line arguments for `input_path` and `output`. The code also includes a `get_parent_dir` function.

```
print(len(docs.all()))

if (len(docs.all()) == 0):
    return render_template("login.html", pred="The username is not found")
else:
    if ((user == docs[0][0]['_id'] and passw == docs[0][0]['psw'])):
        return redirect(url_for("prediction"))
    else:
        print("Invalid User")

@app.route("/logout")
def logout():
    return render_template("logout.html")

@app.route("/prediction", methods=['GET', 'POST'])
def res():
    parser = argparse.ArgumentParser(argument_default=argparse.SUPPRESS)

    parser.add_argument(
        "--input_path",
        type=str,
        default=image_test_folder,
        help="Path to image. default is"
        + image_test_folder,
    )

    parser.add_argument(
        "--output",
        type=str,
        default=detection_results_folder,
        help="Output path for detection results. Default is"
    )

    get_parent_dir()
```

