National College of Ireland

Configuration Manual

MSc Academic Internship
MSCCYB1_JAN23B_0

Abdul Basit Dalvi
Student ID: 22134697

School of Computing
National College of Ireland

Supervisor: Mr. Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Abdul Basit Dalvi
Student ID:	22134697
Programme:	MSCCYB1_JAN23B_O **Year:** 22-2023
Module:	MSc Academic Internship
Lecturer:	Mr. Vikas Sahni
Submission Due Date:	14 Dec, 2023
Project Title:	Next-Generation Compliance Support Tool: Leveraging Machine Learning to Optimize Implementation and Audit Preparedness
Word Count:	2043 (excluding References) **Page Count:** 11

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Abdul Basit Dalvi

Date: 12/12/2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Abdul Basit Dalvi
Student ID: 22134697

1 Introduction

The compliance support tool, a product of extensive academic research outlined in the main thesis, has been comprehensively documented in this manual. It includes technical specifications, installation guidelines, and operational instructions for the application. Insights into the development environment, encompassing hardware and software features, details on software components and their versions, initial installation steps, and guidelines for correct operational procedures have been provided in this manual. While the main thesis delved into the architecture and design phase, this manual has served as a practical guide for users, ensuring a seamless understanding of the compliance support tool's functionality and implementation

2 System Configuration

In the subsequent section, a concise overview is provided of the hardware specifications utilized during the development and testing phases of the compliance support tool. Additionally, a detailed inventory is presented, cataloging the diverse array of software tools, frameworks, and solutions strategically employed throughout the application's development life cycle.

2.1 Hardware and OS Configuration

Below is the hardware and OS configuration of the system used during development of the tool:

Component	Details
Operating System	Windows 11 Home Single Language (64-bit)
Processor	AMD Ryzen 5 5600H with Radeon Graphics, 6 cores, 12 logical processors
Memory(RAM)	16.0 GB installed RAM
System type	x64-based PC
Virtual Memory	Total: 29.8 GB, Available: 15.2 GB
Development Environment	Python virtual environment, Visual Studio Code

2.2 Prerequisites

The tool requirement is very basic and has following prerequisites:

1. Windows OS / Linux OS / Mac OS
2. Internet Connection
3. Python (Python, n.d.)
4. Visual Studio Code (Visual Studio Code, n.d.)

2.3 Software Version

- Python
  - Version: 3. 11
  - Python is a dynamic and versatile high-level programming language which played a crucial role in the functionality of our application.
  - To confirm the Python version installed, users can utilize the terminal or command prompt. By entering the command:

    python --version

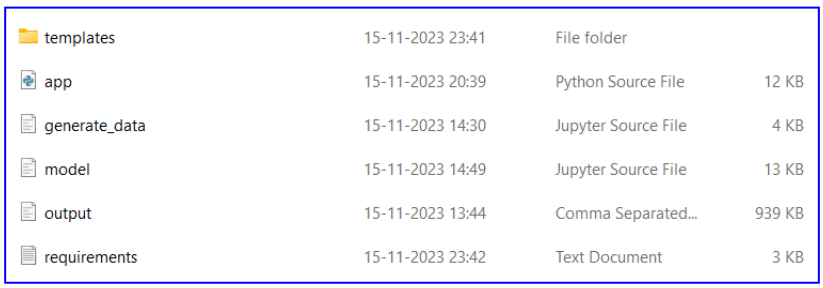
    it can be verified that Python 3. 11 is successfully installed on the system.
- Visual Studio Code
  - Version 1.85
  - Provided a robust and user-friendly integrated development environment.

2.4 Libraries used

Library	Details
CSV Module	The CSV module efficiently handled tabular data through functionalities to parse CSV data. It facilitated the reading and writing of CSV files.
Random Module	The standard Python random module generated crucial pseudo-random numbers for creating random data or making selections.
Faker Library	Employed Faker library offered a wide range of functionalities to create realistic and randomized data. This enhanced the testing and development capabilities to handle various scenarios effectively.
Pandas (Python Data Analysis Library)	A powerful data manipulation tool used to efficiently organize and process tabular data. The DataFrame structure eased integration with machine learning workflows.
Scikit-Learn	Scikit-Learn was used for machine learning which incorporated the DecisionTreeClassifier for building and training decision tree models. The library's tools contributed to a robust machine learning pipeline i.e. from development to evaluation and deployment.

3 Project Structure

The project has been divided into two parts. Where one part is related to synthetic data generation and model building. Whereas the other part is related to the integration of the build model into the web app using the flask framework of python.



The details for each are as follows:

1. generate_data.ipynb: This is the jupyter notebook which includes the code related to the synthetic data generation as there is no availability of real data and also done the preprocessing of the random data to make the clear and processed csv file as output.

2. model.ipynb: This is the jupyter notebook which includes the data for making the model for the prediction.
3. output.csv: This is the csv file which has been generated from the generate_data.ipynb.
4. app.py: This is the entry point of the flask app.
5. requirements.txt: This is the text file which consists of all the packages with their version to be installed using pip in python to run the web app
6. templates: This is the folder being used by flask app. It contains the views i.e., the html files which can be seen as views to user.

4 Steps to Configure and Run the Flask Application

- Download and Install Python 3. 11 : Visit the official Python website (https://www. python. org/downloads/) to download and install Python 3. 11. Follow the installation instructions for the operating system.
- On Windows, use the Command Prompt or PowerShell. On macOS and Linux, use the terminal.
- Create a Virtual Environment: Run the following command to create a virtual environment named 'venv':

  python -m venv

  This step isolates your project's dependencies from the global Python environment.

- Activate the virtual environment
  - On Windows, activate the virtual environment using:

    venv\Scripts\Activate
  - On macOS/Linux, use:

    source venv/bin/activate

  The command prompt or terminal prompt should change to indicate the active virtual environment.
- Install Required Packages: Run the following command to install the dependencies listed in the 'requirements. txt' file:
  pip install -r requirements. txt
- Set the FLASK_APP Environment Variable
  - On Windows, use the following command:

    set FLASK_APP=app. Py
  - On macOS/Linux, use:

    export FLASK_APP=app. Py

  This step tells Flask which Python file represents your application.
- Run the Flask Application: Start the Flask development server with the following command:

  flask run

  The application should now be running locally. You'll see output indicating the server address (usually http://127. 0. 0. 1:5000/).
- Access the Application: Open a web browser and navigate to the provided server address (e. g. , http://127. 0. 0. 1:5000/). You should see your Flask application.

5 Source Code Walkthrough

5.1 Data Generation Script (generate_data. py)

This script generates synthetic data for the tool using the Faker library.

5.1.1 Libraries Used:

import csv: This is being used for creating csv of generated data.
import random: This is being used to create random generated data from given set of values.
from faker import Faker: This is being used to generate fake data as there is no real data available.

5.1.2 Data Generation Logic:

It creates a synthetic dataset with details such as company name, employee range, number of branches, etc. The script utilizes Faker for realistic data generation and random module for variability.

5.1.3 Output:

The generated data is saved in a CSV file named 'output. csv'.

5.1.4 Coding Steps

- Import the required libraries i.e. csv, random and faker.
- Set the seed for reproducibility : Sets the seed for the random number generator to 42, ensuring that subsequent random processes will produce the same sequence of numbers, facilitating reproducibility in the code.
- Define the ranges: Defines ranges and lists for various parameters related to a company's information security, such as company name, employee range, number of branches, number of network devices, number of workstations, types of information, processes and frequency of data transfers, cryptographic controls, event logging mechanisms, and information security events.
- Generate CSV data: Generates CSV data for 10,000 rows, where each row represents information related to a company's security. The data includes company name, employee count, number of branches, number of network devices, total workstations, count of workstations with Windows OS, count of workstations with Linux OS, type of information, frequency of data transfers, cryptographic controls, event logging mechanisms, and information security events, with values randomly chosen based on the defined ranges and lists.
- Write to CSV file: Writes the generated CSV data (stored in the 'rows' list) to a file named 'output.csv'. The header, containing the column names, is first written to the CSV file, followed by the data rows. The file is opened in write mode ('w') with the 'newline' parameter set to an empty string to ensure proper line endings in the CSV file. The 'csv.writer' object is used to write the header and rows to the file.

```python
import csv
import random
from faker import Faker

fake = Faker()

# Set the seed for reproducibility
random.seed(42)

# Define the ranges
company = ['A', 'B']
employee_range = (100, 5000)
number_of_branches = (1, 10)
number_of_network_devices = (1, 50)
number_of_workstations = (1, 50)
types_of_information = [4, 3, 2, 0]
processes_and_frequency_of_data_transfers = ['daily', 'weekly', 'monthly', 'annually', 'no transfer']
cryptographic_controls = ['Strong Cryptographic controls', 'Good Cryptographic control',
                          'Medium Cryptographic control', 'low Cryptographic control', 'No sensitive financial transactions ']
event_logging_mechanisms = ['in-house logging', 'handled by third-party', 'not required']
information_security_events = ['in-house logging', 'handled by third-party', 'not required']

# Generate CSV data
rows = []
for _ in range(10000):
    total_workstations = random.randint(*number_of_workstations)
    windows_os = random.randint(0, total_workstations)
    linux_os = total_workstations - windows_os
```

```python
    row = [
        random.choice(company),
        random.randint(*employee_range),
        random.randint(*number_of_branches),
        random.randint(*number_of_network_devices),
        total_workstations,
        windows_os,
        linux_os,
        random.choice(types_of_information),
        random.choice(processes_and_frequency_of_data_transfers),
        random.choice(cryptographic_controls),
        random.choice(event_logging_mechanisms),
        random.choice(information_security_events)
    ]
    rows.append(row)

# Write to CSV file
header = ['company', 'employee_range', 'number_of_branches', 'number_of_network_devices',
          'number_of_workstations', 'windows_os', 'linux_os', 'types_of_information',
          'processes_and_frequency_of_data_transfers', 'cryptographic_controls',
          'event_logging_mechanisms', 'information_security_events']
with open('output.csv', 'w', newline='') as csvfile:
    csv_writer = csv.writer(csvfile)
    csv_writer.writerow(header)
    csv_writer.writerows(rows)

print("CSV file generated successfully.")
```

```
CSV file generated successfully.
```

5.2 Machine Learning Model Script (model.py):

This script implements a Decision Tree Classifier using Scikit-Learn to predict the company based on input parameters.

5.2.1 Libraries Used:

import pandas as pd: This library being used to handle data and also to do manipulation on the generated data.

from sklearn. model_selection import train_test_split: This is beign used to split the data into two sets of dataset such as train set and test set.

from sklearn. tree import DecisionTreeClassifier: This is beign used to use decision tree classifier for the generation of the model using Decision Tree

from sklearn. preprocessing import LabelEncoder: This is being used to convert the object data type(string data type) into numerical representation to feed the data into model.

5.2.2 Data Loading and Preprocessing:

The synthetic data generated is loaded into a Pandas DataFrame. Categorical columns are encoded using LabelEncoder so that it should be feed as numerical data in model.

5.2.3 Model Training:

A Decision Tree model is created and trained on the data.

5.2.4 Prediction Example:

Example input values are provided, and the trained model is used to predict the company.

5.2.5 Coding Steps

- Import the required libraries.
- Load the CSV data into a DataFrame: Uses the pandas library to load the data from the 'output.csv' file into a DataFrame named 'df', allowing for easy manipulation and analysis of the tabular data in a structured format.
- Convert categorical columns to numerical using Label Encoding: Initializes four LabelEncoder objects named label_encoder_process, label_encoder_cryptographic, label_encoder_event_logging, and label_encoder_information_security. These label encoders can be used to convert categorical columns in a DataFrame to numerical values using label encoding,
- Separate features (X) and target variable (y): Separates the DataFrame df into features (X) and the target variable (y). The features include columns such as 'employee_range', 'number_of_branches', 'number_of_network_devices', 'number_of_workstations', 'windows_os', 'linux_os', 'types_of_information', 'processes_and_frequency_of_data_transfers', 'cryptographic_controls', 'event_logging_mechanisms', and 'information_security_events'. The target variable (y) is the 'company' column, indicating the company name.
- Split the data into training and testing sets: Uses the train_test_split function from scikit-learn to split the data into training and testing sets. The features (X) and target variable (y) are split into training sets (X_train and y_train) and testing sets (X_test and y_test). The parameter test_size=0.2 specifies that 20% of the data will be used for testing, and random_state=42 sets the seed for reproducibility.
- Create a Decision Tree model: Creates a Decision Tree model using the DecisionTreeClassifier from scikit-learn. The model is then trained on the training data (X_train and y_train) using the fit method, with random_state=42 setting the seed for reproducibility.
- Make predictions using the trained model: Make predictions using the trained Decision Tree model (model) on new data.

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, classification_report
```

Python

```python
# Load the CSV data into a DataFrame
df = pd.read_csv('output.csv')
```

Python

```python
# Convert categorical columns to numerical using Label Encoding
label_encoder_process = LabelEncoder()
label_encoder_cryptographic = LabelEncoder()
label_encoder_event_logging = LabelEncoder()
label_encoder_information_security = LabelEncoder()
```

Python

```python
df['processes_and_frequency_of_data_transfers'] = label_encoder_process.fit_transform(
    df['processes_and_frequency_of_data_transfers'])
df['cryptographic_controls'] = label_encoder_cryptographic.fit_transform(df['cryptographic_controls'])
df['event_logging_mechanisms'] = label_encoder_event_logging.fit_transform(df['event_logging_mechanisms'])
df['information_security_events'] = label_encoder_information_security.fit_transform(
    df['information_security_events'])
```

```python
# Separate features (X) and target variable (y)
X = df[['employee_range', 'number_of_branches', 'number_of_network_devices', 'number_of_workstations',
        'windows_os', 'linux_os', 'types_of_information', 'processes_and_frequency_of_data_transfers',
        'cryptographic_controls', 'event_logging_mechanisms', 'information_security_events']]
y = df['company']
```

Python

```python
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Python

```python
# Create a Decision Tree model
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)
```

Python

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

```python
# Example input values for prediction
new_data = pd.DataFrame({
    'employee_range': [3088],
    'number_of_branches': [3],
    'number_of_network_devices': [24],
    'number_of_workstations': [25],
    'windows_os': [8],
    'linux_os': [17],
    'types_of_information': [2],
    'processes_and_frequency_of_data_transfers': ['weekly'],
    'cryptographic_controls': ['Medium Cryptographic control'],
    'event_logging_mechanisms': ['not required'],
    'information_security_events': ['not required']
})
```

Python

```python
# Convert categorical columns to numerical using Label Encoding
new_data['processes_and_frequency_of_data_transfers'] = label_encoder_process.transform(
    new_data['processes_and_frequency_of_data_transfers'])
new_data['cryptographic_controls'] = label_encoder_cryptographic.transform(
    new_data['cryptographic_controls'])
new_data['event_logging_mechanisms'] = label_encoder_event_logging.transform(
    new_data['event_logging_mechanisms'])
new_data['information_security_events'] = label_encoder_information_security.transform(
    new_data['information_security_events'])
```

Python

```python
# Make predictions using the trained model
prediction = model.predict(new_data)
```

Python

```python
# Display the predicted company
print("Predicted Company:", prediction[0])
```

Python

```
Predicted Company: B
```

```python
# Example input values for prediction
new_data = pd.DataFrame({
    'employee_range': [2353],
    'number_of_branches': [4],
    'number_of_network_devices': [15],
    'number_of_workstations': [41],
    'windows_os': [7],
    'linux_os': [34],
    'types_of_information': [3],
    'processes_and_frequency_of_data_transfers': ['daily'],
    'cryptographic_controls': ['No sensitive financial transactions '],
    'event_logging_mechanisms': ['in-house logging'],
    'information_security_events': ['not required']
})
```

Python

```python
# Convert categorical columns to numerical using Label Encoding
new_data['processes_and_frequency_of_data_transfers'] = label_encoder_process.transform(
    new_data['processes_and_frequency_of_data_transfers'])
new_data['cryptographic_controls'] = label_encoder_cryptographic.transform(
    new_data['cryptographic_controls'])
new_data['event_logging_mechanisms'] = label_encoder_event_logging.transform(
    new_data['event_logging_mechanisms'])
new_data['information_security_events'] = label_encoder_information_security.transform(
    new_data['information_security_events'])
```

Python

```python
# Make predictions using the trained model
prediction = model.predict(new_data)
```

Python

```python
# Display the predicted company
print("Predicted Company:", prediction[0])
```

Python

```
Predicted Company: B
```

5.3 Flask Web Application (app.py)

This script creates a web application using Flask, allowing users to input parameters and receive control suggestions.

5.3.1 Libraries used

from flask import Flask, render_template, request, redirect, url_for, session: Here the Flask has been imported to run the app in web, render_template is being used to display the required view to the use, request is beign used to get and receive the request, redirect is beign used to redirect the user to certain view after certain completion of task, url_for is being used to call the route for any particular function of view.

import os: Here os module has been imported to create the unique secret key of the flask web app.

5.3.2 Flask Routes

The application has routes for index, controls, and result pages.

5.3.3 Model Integration

The model script is integrated to predict the company based on user inputs.

5.3.4 User Input Handling

User inputs are received through web forms, processed, and used for model prediction.

5.3.5 Session Management

Flask session is utilized to store and pass data between routes.

5.3.6 Coding Steps

- The code is a Python script that utilizes the Flask web framework to create a web application for predicting and analyzing security controls for companies based on user input. The application consists of three routes: '/' or '/index', '/controls', and '/result'.
- In the 'index' route, user inputs such as company information and security-related parameters are collected via an HTML form. The input data is then processed using a pre-trained decision tree classifier. The prediction is stored in the Flask session, and the user is redirected to the 'controls' route.
- In the 'controls' route, users are prompted to provide additional security controls based on the predicted company type ('A' or 'B'). The entered controls are then redirected to the 'result' route, where the controls and their values are displayed to the user.
- The code uses the Flask framework to handle web requests, Pandas for data manipulation, and scikit-learn for machine learning tasks such as label encoding and decision tree classification.

```
from flask import Flask, render_template, request, redirect, url_for, session
import os
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder

app = Flask(__name__)

app.secret_key = os.urandom(24).hex()
app.jinja_env.auto_reload = True
app.config['TEMPLATES_AUTO_RELOAD'] = True

@app.route('/')
@app.route('/index', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        company_name = request.form['compnay_name']
        employees = request.form['employees']
        branches = request.form['branches']
        network = request.form['network']
        workstations = request.form['workstations']
        windows = request.form['windows']
        linux = request.form['linux']
        type_i = request.form['type_i']
        frequency = request.form['frequency']
        cryptographic = request.form['cryptographic']
        logging = request.form['logging']
        assessing = request.form['assessing']
```

```
    df = pd.read_csv('output.csv')
    label_encoder_process = LabelEncoder()
    label_encoder_cryptographic = LabelEncoder()
    label_encoder_event_logging = LabelEncoder()
    label_encoder_information_security = LabelEncoder()
    df['processes_and_frequency_of_data_transfers'] = label_encoder_process.fit_transform(
    df['processes_and_frequency_of_data_transfers'])
    df['cryptographic_controls'] = label_encoder_cryptographic.fit_transform(df['cryptographic_controls'])
    df['event_logging_mechanisms'] = label_encoder_event_logging.fit_transform(df['event_logging_mechanisms'])
    df['information_security_events'] = label_encoder_information_security.fit_transform(
        df['information_security_events'])
    X = df[['employee_range', 'number_of_branches', 'number_of_network_devices', 'number_of_workstations',
    'windows_os', 'linux_os', 'types_of_information', 'processes_and_frequency_of_data_transfers',
    'cryptographic_controls', 'event_logging_mechanisms', 'information_security_events']]
    y = df['company']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    model = DecisionTreeClassifier(random_state=42)
    model.fit(X_train, y_train)
    new_data = pd.DataFrame({
        'employee_range': [employees],
        'number_of_branches': [branches],
        'number_of_network_devices': [network],
        'number_of_workstations': [workstations],
        'windows_os': [windows],
        'linux_os': [linux],
        'types_of_information': [type_i],
        'processes_and_frequency_of_data_transfers': [frequency],
        'cryptographic_controls': [cryptographic],
        'event_logging_mechanisms': [logging],
        'information_security_events': [assessing]
    })
```

```
    new_data['processes_and_frequency_of_data_transfers'] = label_encoder_process.transform(
    new_data['processes_and_frequency_of_data_transfers'])
    new_data['cryptographic_controls'] = label_encoder_cryptographic.transform(
        new_data['cryptographic_controls'])
    new_data['event_logging_mechanisms'] = label_encoder_event_logging.transform(
        new_data['event_logging_mechanisms'])
    new_data['information_security_events'] = label_encoder_information_security.transform(
        new_data['information_security_events'])
    prediction = model.predict(new_data)
    print(prediction[0])
    session['company_name'] = company_name
    session['company_prediction'] = prediction[0]
    return redirect(url_for('controls'))
return render_template('index.html')
```

```
@app.route('/controls', methods=['GET', 'POST'])
def controls():
    if request.method == 'POST':
        control1 = request.form['control1']
        control2 = request.form['control2']
        control3 = request.form['control3']
        control4 = request.form['control4']
        control5 = request.form['control5']
        control6 = request.form['control6']
        control7 = request.form['control7']
        control8 = request.form['control8']
        control9 = request.form['control9']
        control10 = request.form['control10']
        control11 = request.form['control11']
        control12 = request.form['control12']
        control13 = request.form['control13']
        control14 = request.form['control14']
        control15 = request.form['control15']
        control16 = request.form['control16']
        if session['company_prediction'] == 'A':
            control17 = request.form['control17']
            control18 = request.form['control18']
            control19 = request.form['control19']
        if session['company_prediction'] == 'B':
            control20 = request.form['control20']
            control21 = request.form['control21']
        if session['company_prediction'] == 'A':
            return redirect(url_for('result',
                                    control1=control1,
                                    control2=control2,
                                    control3=control3,
```

```
        if session['company_prediction'] == 'B':
            return redirect(url_for('result',
                                    control1=control1,
                                    control2=control2,
                                    control3=control3,
                                    control4=control4,
                                    control5=control5,
                                    control6=control6,
                                    control7=control7,
                                    control8=control8,
                                    control9=control9,
                                    control10=control10,
                                    control11=control11,
                                    control12=control12,
                                    control13=control13,
                                    control14=control14,
                                    control15=control15,
                                    control16=control16,
                                    control20=control20,
                                    control21=control21))

        return render_template('controls.html')

    return render_template('controls.html')

@app.route('/result', methods=['GET','POST'])
def result():
    if session['company_prediction'] == 'A':
        control1 = request.args.get('control1')
        control2 = request.args.get('control2')
        control3 = request.args.get('control3')
```

```
    if session['company_prediction'] == 'B':
        control1 = request.args.get('control1')
        control2 = request.args.get('control2')
        control3 = request.args.get('control3')
        control4 = request.args.get('control4')
        control5 = request.args.get('control5')
        control6 = request.args.get('control6')
        control7 = request.args.get('control7')
        control8 = request.args.get('control8')
        control9 = request.args.get('control9')
        control10 = request.args.get('control10')
        control11 = request.args.get('control11')
        control12 = request.args.get('control12')
        control13 = request.args.get('control13')
        control14 = request.args.get('control14')
        control15 = request.args.get('control15')
        control16 = request.args.get('control16')
        control20 = request.args.get('control20')
        control21 = request.args.get('control21')
    return render_template('results.html',
                           control1=int(control1),
                           control2=int(control2),
                           control3=int(control3),
                           control4=int(control4),
                           control5=int(control5),
                           control6=int(control6),
                           control7=int(control7),
                           control8=int(control8),
                           control9=int(control9),
                           control10=int(control10),
```

References

Python. (n.d.). Retrieved from https://www.python.org/downloads/release/python-3110/
Visual Studio Code. (n.d.). Retrieved from https://code.visualstudio.com/download