SINGAPORE UNIVERSITY OF SOCIAL SCIENCES

ICT239

End-of-Course Assessment – January Semester 2021

Web Application Development

INSTRUCTIONS TO STUDENTS:

1. This End-of-Course Assessment paper contains **TWO (2)** questions and comprises **SIXTEEN (16)** pages (including the cover page and appendix).

2. You are to include the following particulars in your submission: Course Code, Title of the ECA, SUSS PI No., Your Name, and Submission Date.

3. Late submission will be subjected to the marks deduction scheme. Please refer to the Student Handbook for details.

IMPORTANT NOTE

ECA Submission Deadline: Sunday, 23 May 2021, 12 noon

Please Read This Information before You Start Working on your ECA

This ECA carries 70% of the course marks and is a compulsory component. It is to be done individually and not collaboratively with other students.

Submission

You are to submit the ECA assignment in exactly the same manner as your tutor-marked assignments (TMA), i.e. using Canvas. Submission in any other manner like hardcopy or any other means will not be accepted.

Electronic transmission is not immediate. It is possible that the network traffic may be particularly heavy on the cut-off date and connections to the system cannot be guaranteed. Hence, you are advised to submit your assignment the day before the cut-off date in order to make sure that the submission is accepted and in good time.

Once you have submitted your ECA assignment, the status is displayed on the computer screen. You will only receive a successful assignment submission message if you had applied for the e-mail notification option.

ECA Marks Deduction Scheme

Please note the following:
a) Submission Cut-off Time – Unless otherwise advised, the cut-off time for ECA submission will be at ***12:00 noon*** *on the day of the deadline. All submission timings will be based on the time recorded by Canvas.*
b) Start Time for Deduction – Students are given a grace period of 12hours. Hence calculation of late submissions of ECAs will begin at ***00:00 hrs*** *the following day (this applies even if it is a holiday or weekend) after the deadline.*
c) How the Scheme Works – From 00:00 hrs the following day after the deadline, ***10 marks*** *will be deducted for each* ***24-hour block****. Submissions that are subject to more than 50 marks deduction will be assigned* ***zero mark****. For examples on how the scheme works, please refer to Section 5.2 Para 1.7.3 of the Student Handbook.*

Any extra files, missing appendices or corrections received after the cut-off date will also not be considered in the grading of your ECA assignment.

Plagiarism and Collusion

Plagiarism and collusion are forms of cheating and are not acceptable in any form of a student's work, including this ECA assignment. You can avoid plagiarism by giving appropriate references when you use some other people's ideas, words or pictures (including diagrams). Refer to the American Psychological Association (APA) Manual if you need reminding about quoting and referencing. You can avoid collusion by ensuring that your submission is based on your own individual effort.

The electronic submission of your ECA assignment will be screened through a plagiarism detecting software. For more information about plagiarism and cheating, you should refer to the Student Handbook. SUSS takes a tough stance against plagiarism and collusion. Serious cases will normally result in the student being referred to SUSS's Student Disciplinary Group. For other cases, significant marking penalties or expulsion from the course will be imposed.

Answer all questions. (Total 100 marks)

This Resit ECA is based on the TMA for July 2020 semester, which you have taken. You may reuse the source code of your TMA as a base to revise and extend further if it is applicable. Similarly, source code from the review materials posted on the Resit group on Canvas is made available for your use for this ECA. More specifically, the zipped source code file can be retrieved from the following URL under the Modules subfolder.

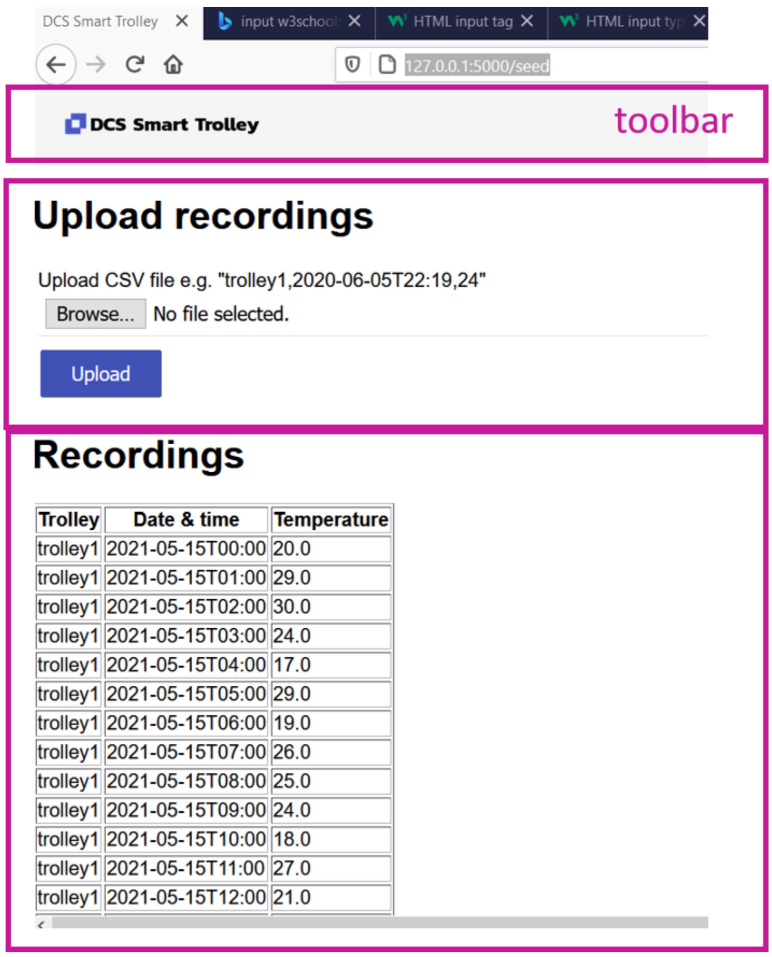
https://canvas.suss.edu.sg/courses/34268/files/5916730?module_item_id=429050

Question 1

In TMA Q3, you are asked to develop a function to upload data in the CSV format. One input data file that has 3 day worth of recordings starting from 21 to 23 May 20201. It consists of 3 fields, Trolley, Data & Time, and Temperature, can be retrieved from the following link:

https://github.com/paulhjwu/ICT239-Resit/blob/master/seed_demo_ECA.csv

Through a URL http://127.0.0.1:5000/seed, the data can be uploaded, via the section "Upload recordings", and stored in the backend database that can be displayed in the section "Recordings" as shown in the following Diagram Q1(a).



Upload recordings

Upload CSV file e.g. "trolley1,2020-06-05T22:19,24"

Recordings

Trolley	Date & time	Temperature
trolley1	2021-05-15T00:00	20.0
trolley1	2021-05-15T01:00	29.0
trolley1	2021-05-15T02:00	30.0
trolley1	2021-05-15T03:00	24.0
trolley1	2021-05-15T04:00	17.0
trolley1	2021-05-15T05:00	29.0
trolley1	2021-05-15T06:00	19.0
trolley1	2021-05-15T07:00	26.0
trolley1	2021-05-15T08:00	25.0
trolley1	2021-05-15T09:00	24.0
trolley1	2021-05-15T10:00	18.0
trolley1	2021-05-15T11:00	27.0
trolley1	2021-05-15T12:00	21.0

Diagram Q1(a): "Upload recordings" and "Recordings" Sections of the view of http://127.0.0.1/seed

This question concerns the implementation of Diagram Q1(a), which is to be done according to the MVC design principles.

(a) Regarding the “Upload recordings” section, read the following flask source code and partial HTML code that are used to process the request for http://127.0.0.1:5000/seed, and explain what this code means by providing the answers after reading the codes.

```
import csv

# To be completed in question Q1(b)(ii) below
from datetime import …
from flask import …

app = Flask(__name__)

import repo

# This route is for adding test data and is not part of the migration
@app.route("/seed", methods=['GET','POST'])
def seed():
    if request.method == 'GET':
        recordings = repo.get_recordings()
        return render_template("seed.html", recordings=recordings)

    elif request.method == 'POST':
        print(request.form.get('type'))
        file = request.files.get('file')
        repo.upload_recording(file)
        return redirect(url_for('seed'))

if __name__ == "__main__":
    app.run(debug=True, host='0.0.0.0')
```

Backend Flask code app.py including Controller Route “/seed”

```
{% extends "base.html" %}
{% block body %}
<h1>Upload recordings</h1>
<form action="/seed" method="post" enctype="multipart/form-data">
    <div>
        <label for='upload'>Upload CSV file e.g. "trolley1,2020-06-05T22:19,24"</label>
        <input id='upload' name='file' type='file' accept='.csv' required>
    </div>
    <div>
        <input type="submit" name="type" value="Upload"/>
    </div>
</form>
{% endblock %}
```

Frotend HTML Code in seed.html, Section “Upload recordings” - base.html HTML can be found below

Read and comprehend the above two source codes and fill in the blanks to describe the functioning of the source codes.

1. When______ is accessed via the http_____ request a html webpage _____ will be displayed
2. Before that, a list of the current recordings will be retrieved by a method _____________ and stored in the variable ________.
3. The recordings are passed into the template system through ___________________ method to be displayed together with the webpage in the Recordings section.
4. When the ______ button is pressed in Diagram Q1(a), users can select a file of the ___ type to be uploaded.
5. When the ______ button is pressed, a http ____ request, the variable file is assigned to the request parameter named ____.
6. The method ___________________ is used to upload the content of the file and update the backend database.
7. After the upload action is completed, the view will display _________ webpage again due to the _______________________ statement.
8. However, this time, the content displayed in the __________ section will include the recordings that were just uploaded.

(15 marks)

(b) Regarding the "Recordings" section, a part of the code has been completed, please fill in the remaining codes that will allow the display of the content of the recordings. Bear in mind, the render template method is called at the first place with the content of the recordings variable passed in to the template system.

```
    <h1>Recordings</h1>
    <table border=1>
        <tr>
            <th>Trolley</th>
            <th>Date & time</th>
            <th>Temperature</th>
        </tr>

        // Fill in the remaining codes here

    </table>
```

<u>Frotend HTML Code in seed.html – Section "Recordings"</u>

(5 marks)

(c) Given the base.html and the directory structure of the source codes as shown below:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>DCS Smart Trolley</title>
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <link rel="stylesheet" type="text/css" media="screen" href="{{ url_for('static', filename='index.css') }}">
</head>
<body>
    <div class="toolbar">
        <span class="logo"><img id="logo-img" src="{{ url_for('static', filename='logo.png') }}"></span>
        <span class="logout"><a href="#">Logout</a></span>
    </div>
    {% block body %}{% endblock %}
</body>
</html>
```

base.html HTML Code

```
/home/user/Projects/ECA
└─ Q1/
    ├── repo.py
    ├── app.py
    ├── seed_demo_ECA.csv
    ├── templates/
    │   ├── base.html
    │   └── seed.html
    ├── static/
    │   ├── logo.png
    │   ├── user.png
    │   └── index.css
    ├── requirements.txt
    └── venv
```

Dictory structure of source codes for Q1

(i) Complete the development **repo.py** by completing the **following TWO (2)** methods which was imported in app.py

- upload_recording

(7 marks)

- get_recordings

(4 marks)

and inserting additional codes into the following code skeleton of repo.py

```
import csv
from datetime import datetime, timedelta
import io
import pymongo

connection = pymongo.MongoClient('mongodb://localhost:27017')
db = connection['cluster']

def create_recording(trolley, _datetime, temperature):

     // Insert codes here to complete the method that populates the db.readings collection

def upload_recording(file):

   //  Insert codes here, before calling the create_recording method defined earlier, to complete the method

       create_recording(trolley, _datetime, temperature)

def get_recordings(_date=None):
   filter = {}
   recordings = []

    // Insert codes here to complete the method

   return recordings
```

Skeleton Codes for repo.py

Incorporate the **Backend Flask Controller Code** into **app.py**, and the two sections of **Frotend HTML Code** with base.html into **seed.html**, and

(ii) Complete the app.py program by completing the two "from … import ..." import statements.

(3 marks)

(iii) Complete seed.html by incorporating base.html into seed.html

(3 marks)

(iv) Specify the content of requirements.txt and steps to create the virtual environment venv.

(2 marks)

(v) Complete index.css development to display the view as shown in Diagram Q1(a)

(6 marks)

(vi) Do a recording to show the application can be run successfully

(5 marks)

Question 2

In TMA Q1, a Webpage is required to be developed similar to that as shown in Diagram Q2(a) below.

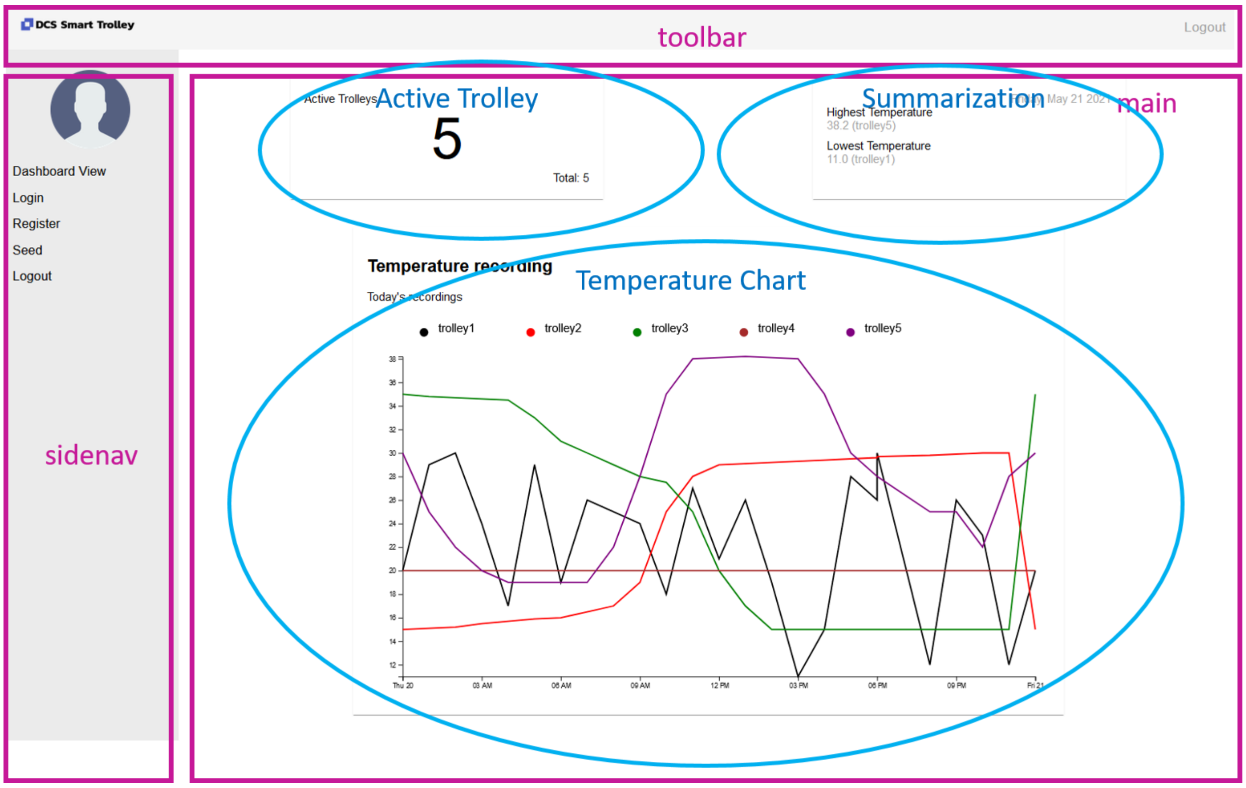


Diagram Q2(a)

The webpage is constituted with 3 main sections:

1. The toolbar section that consists of the company logo and the Logout link
2. The side navigation, or sidenav, section that consists of icon of users, user id, and links to the Dashboard, Account, Configure Email, Download views, and Logout interface and functions
3. The main section that contains the following three views:
   a. Active Trolleys view: show the number of the current active trolleys and total trolleys for a specified date;
   b. Summarization view: show the trolley with the highest and lowest temperatures for a specified date;
   c. Temperature Chart view: show a chart of temperature recording for the current date (say, 'Today'). The x-axis refers to the hours along a day; y-axis refers to the temperature value; and legend refers to trolley id.

(a) In order for the 3 views in the main section to be displayed readily, it is recommended that two additional collections, besides **db.recordings**, in MongoDB to be formulated to store (1) the trolley collection, **db.trolleys**, where the trolleys and their latest readings of temperature (up to today) is stored, and (2) the stats collection, **db.stats**, where the trollies with the highest and lowest readings of temperature (up to today).

```
@app.route("/", methods=['GET'])
def index():
    active_trolleys = repo.get_trolleys_count(datetime.now())
    total_trolleys = repo.get_trolleys_count()
    stat = repo.get_latest_stat()
    return render_template("dashboard.html",
           active_trolleys=active_trolleys, total_trolleys=total_trolleys, stat=stat)

@app.route("/api/recordings", methods=['GET'])
def recordings():
    _date = request.args.get('date')
    if _date is not None:
        try:
            _date = date.fromisoformat(_date)
        except ValueError:
            return {'error': 'date format should be YYYY-MM-DD'}, 400
    recordings = repo.get_recordings(_date)
  return jsonify({'recordings': recordings})
```

<u>Backend Flask Controller Code for Routes "/" and "/api/recordings"</u>

Given that additional controller codes for routes "/" and "/api/recordings" and the JavaScript to render the graphics of the Temperature Chart view (see Appendix 1)

(i) Revise your solution to Q1(c).v in **repo.py** to include the creation and update of the documents in **db.trolleys** and **db.stats** collections. So that when http://127.0.0.1/ is visited, a view as shown in Diagram 2(a) will be displayed. (20 marks)

(ii) Do a recording to show the application can be run successfully (5 marks)

(b) User login management is foundational component of Website development. In Seminar 5 of the review materials you are introduced to a module called loginAuth.

The recording of Seminar 5 can be found below, the explanation of loginAuth is from 15:00 onwards:

https://suss.zoom.us/rec/play/7MgF6XmtioS4MZrPW-xEYxIq2jMqcpsuqf9mDdxPNy5LFLmn2dqf1iJaAWN4W-S_7BZWW_KbNAHVu0T3.4Nr-JPL9PeZ5Of8r

The source codes covered in Seminar 5 can be found under the subfolder loginAuth of the zipped source code file.

Review the loginAuth module of the review materials and incorporate the flask-login module into the development of this solution.

(i) Develop the following webpage accessed by “127.0.0.1:5000/login” and “127.0.0.1:5000/register” with the associated CSS codes.

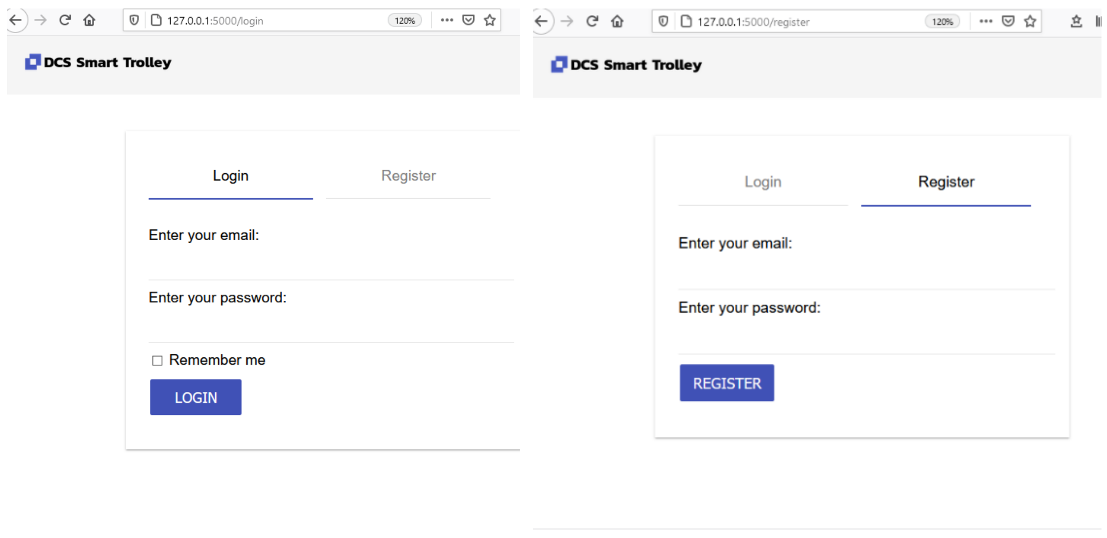
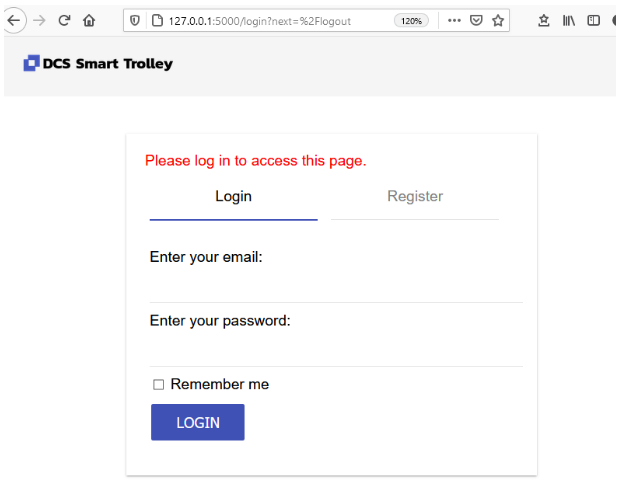


Figure Q2(a): The display of login.html and register.html

(10 marks)

(ii) Do a recording of the run of the above solution to demonstrate the following features:

- When 127.0.0.1:5000/ is visited, it will ask the visitors to login first.
- Without logging in, a visitor trying to access protected links such as 127.0.0.1:5000/dashboard it would be routed to the login link, and an error message "Please log in to access this page." will be displayed as follows:


127.0.0.1:5000/login?next=%2Flogout

120%

DCS Smart Trolley

Please log in to access this page.

Login | Register

Enter your email:

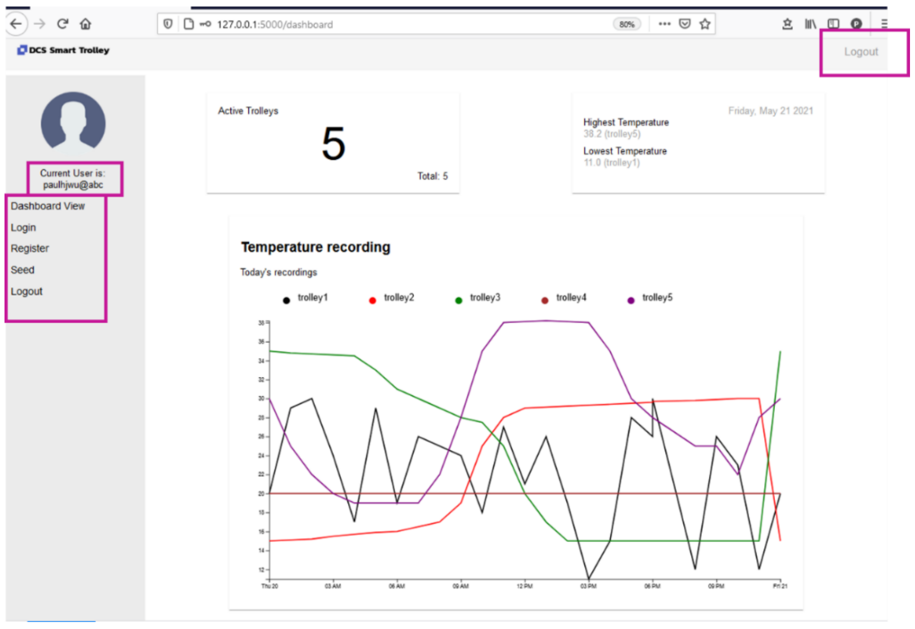
Enter your password:

☐ Remember me

LOGIN

Figure Q2(c): The display of register.html

- Also as shown in Figure Q2(c), there is no Logout link when a visitor is yet to register and login.
- Register a user with email id "test@abc" and password of your choice.
- Login as the test@abc user, the website will display the same view as Diagram Q2(a) above. In addition, the sidebar view will now display all the links including the email id of the current user such as shown below.
- When a link in the sidebar is clicked after logging in, it will work accordingly.


127.0.0.1:5000/dashboard

80%

DCS Smart Trolley

Logout

Current User is: paulhjwu@abc

Dashboard View

Login

Register

Seed

Logout

Active Trolleys

5

Total: 5

Friday, May 21 2021

Highest Temperature
38.2 (trolley5)

Lowest Temperature
11.0 (trolley1)

Temperature recording

Today's recordings

(15 marks)

Suggest you organize the above codes in the following directory structure following the loginAuth example whereby most of **models.py** codes are incorporated into **auth.py**.

```
/home/user/Projects/ECA
├── Q1/
│   ...
└── Q2/
    ├── init.py
    ├── repo.py
    ├── auth.py
    ├── app.py
    ├── seed_demo_ECA.csv
    ├── templates/
    │   ├── base.html
    │   ├── dashboard.html
    │   ├── login.html
    │   ├── register.html
    │   └── seed.html
    ├── static/
    │   ├── logo.png
    │   ├── user.png
    │   ├── index.css
    │   └── dashboard.js
    ├── requirements.txt
    └── venv
```

Appendix 1: dashboard.js source code

```
const STROKES = ["black", "red", "green", "brown", "purple"];

// Get the today's date as "YYYY-MM-DD" string

let chart_date = new Date();
const offset = chart_date.getTimezoneOffset();
chart_date = new Date(chart_date.getTime() - (offset*60*1000));
chart_date = chart_date.toISOString().split('T')[0];

// Fetch the data and plot the chart for today's readings only

fetch(`/api/recordings?date=${chart_date}`).then(response => {
    if (response.status === 200) {
        return response.json();
    } else {
        return Promise.reject();
    }
}).then(data => {

    let recordings = data.recordings; // input format: [{trolley: x, datetime: y,
temperature: z}, ...]

    // Reformat the data for plotting chart

    let chart_data = {}; // output format: {x: [{datetime: y, temperature: z}, ...], ...}
    let temperatures = []; // format: [z, ...]

    // Convert data from input format to output format in chart_data

    for (let recording of recordings) {

        if (!(recording.trolley in chart_data)) {
            chart_data[recording.trolley] = [];
        }

        // Only include those recordings recorded later than today and before tomorrow
        chart_data[recording.trolley].push({
            datetime: new Date(recording.datetime),
            temperature: parseFloat(recording.temperature),
        });

        temperatures.push(recording.temperature);
    }

    // Sort the chart data by datetime

    for (const trolley in chart_data) {
        chart_data[trolley].sort((a, b) => a.datetime - b.datetime);
    }

    //
    // Render and plot the chart
    //

    // Prepare the Canvas Area
    const margin = { top: 20, right: 20, bottom: 30, left: 50 };
    const width = 960 - margin.left - margin.right;
    const height = 500 - margin.top - margin.bottom;

    const today_date = new Date().setHours(0, 0, 0, 0);
    const x = d3.scaleTime().range([0, width])
        .domain([today_date, today_date + 24 * 60 * 60 * 1000])
        .nice()

    const y = d3.scaleLinear().range([height, 0])
        .domain([d3.min(temperatures), d3.max(temperatures)])

    // Canvas the Legend Area
    const svg_legend = d3.select("#chart-legend").append("svg")
    .attr("width", width + margin.left + margin.right)
    .attr("height", 50)
    .append("g")
    .attr("transform",
        "translate(" + margin.left + "," + margin.top + ")");

    // Canvas the Chart Area
    const svg = d3.select("#chart").append("svg")
        .attr("width", width + margin.left + margin.right)
        .attr("height", height + margin.top + margin.bottom)
        .append("g")
        .attr("transform",
            "translate(" + margin.left + "," + margin.top + ")");

    // Add the X Axis
    svg.append("g")
        .attr("transform", "translate(0," + height + ")")
        .call(d3.axisBottom(x));

    // Add the Y Axis
    svg.append("g")
        .call(d3.axisLeft(y));

    // Plot the data on the chart
    let count = 0;
    for (let trolley in chart_data) {
        const valueline = d3.line()
            .x(d => x(d.datetime))
            .y(d => y(d.temperature));

        aData=chart_data[trolley]
        svg.append("path")
            .datum(aData)
            .attr("class", "line")
            .style("stroke", STROKES[count % STROKES.length])
            .attr("d", valueline);

        svg_legend.append("circle")
            .attr("cx", 30 + count * 150)
            .attr("cy", 20)
            .attr("r", 6)
            .style("fill", STROKES[count % STROKES.length]);

        svg_legend.append("text")
            .attr("x", 50 + count * 150)
            .attr("y", 20)
            .text(trolley)
            .attr("alignment-baseline", "middle");
        count += 1;
    }

}).catch(reason => {
    console.log(reason);
})
```

----- END OF ECA PAPER -----