CEA SIMULATOR

Technologies Used:

Front-end: HTML, CSS, JavaScript, AJAX
Back-end: Python (Django)
Databases: InfluxDB, MySQL

Work Flow:

1. User creates an account or logs in to the existing account.
2. Parameters such as temperature, CO2, date, and location are selected by the user.
3. These parameters are passed to the Crop Growth Simulator and three CSV files (username_distri.csv, username_fruitdev.csv, username_growth.csv) are generated.
4. The CSV files are stored in influxDB
5. The data is being passed as an input to the Google Visualization API.
6. The output from the Google Visualization API is the graph that is being displayed.
7. Responsive table is also being generated using the data

Folder & Files Description:

1. **influxdb/**
    a. **dbinflux.py** - This is used to pull the data from NREL website by using the API
    b. **final1.csv** - Latitude and Longitude information for all locations in USA

2. **task1/**
    a. **maps_app/**
        i. **models.py** - contains the MySQL db table information
        ii. **sim.py** & **simulator.p**y - it executes the simulation
        iii. **urls.py** - contains the path to the views
        iv. **views.py** - contains all the backend code
    b. **static/** - contains the webpage designs in css/ , fonts/, js/ and the images in images/
    c. **task1/**
        i. **settings.py** - OS Path, static file and database configurations
    d. **templates/maps_app** - it contains the webpages
        i. **chart.html** - chart is being displayed by Google's Visualization API
        ii. **index.html** - first page of the website
        iii. **mainpage.html** - home page of the website (after user logged in)
        iv. **mappage.html** - page where user is being asked for input
        v. **table1.html** & **table2.html** - table view of the graph that is being displayed

Detailed Description:

1. **index.html:**

   Front-end:

   Sign-in Module:

   - In the sign-in tab username and password are requested from the user and passed to the server.

   Sign-up Module:

   - In the sign-up tab Firstname, Lastname, username, mail-id, and password is requested from the user and passed to the server.

   Back-end:

   Sign-in:

   - The username and password are received at the views.py file using the POST method.
   - The credentials are received by the function named “home”.
   - The password is encrypted using SHA and verified with the MySQL database and access is permitted.
   - The imports required are user, auth and pymysql

   Sign-up:

   - All the information is stored in the MySQL database. The password will be stored only in an encrypted format.
   - The table used is “auth_user”. It can be changed if required.

2. **mainpage.html:**

   Front-end:

   - The webpage consists of three cards. One card for each model.
   - The first card when clicked directs to mappage.html
   - Second card when clicked directs to energymodel.html and the third directs to energysupply.html

3. **mainpage.html:**

   ❖ The API-key for the google maps plugin should be changed.
   ❖ The variable “geojson_url” is used to load the latitude and the longitude data.
   ❖ The function “initMap()” inside the inbuilt script is responsible to load the google maps.
   ❖ The function “loadMarker()” places markers on the map based on the given latitude and longitude information.

Front-end:

- The webpage comprises 5 input fields. The temperature, co2, location, and date range is obtained as input from the user.
- When the user hits the 'run simulation’ button the data is passed through GET method to the server.

Back-end:

views.py:

- The values are obtained by the function called mappage.
- The data is obtained using the .get() method.
- The dates are converted to julian dates in order to specify the range in the query to pull the data from the influxDB database.
- The year is appended to the location_id. That ‘locationid_year’ will be the table name.
- The db query is applied and the data is fetched from the database and stored in the variable ‘result’.
- Also, the username of the current session is obtained using ‘request.user’ method.
- The username, resultset, temperature, co2, and the julian dates are passed to the sim.py file.
- After the simulator’s execution, two functions are called. pytojsfruit() and pytojsgrowth(). The fruitdev and growth files are processed and the data is extracted.
- The data and the result is passed to the chart.html file.

sim.py:

- The data from the function mappage is obtained and passed to the simulator.py file.
- The simulator is started.

simulator.py:

- The simulator generated three files username_distri, username_FruitDev, username_grwoth.
- The ‘adddatatodb()’ function adds the data to the influxdb database.

4. chart.html:

Front-end:

- The webpage consists of two charts.
- The table icon in the first chart directs to table1.html and in the second chart it directs to table2.html.

Back-end:

- The Google Visualisation API is used to produce the charts.
- The first and second charts have the id’s graph1 and graph2 respectively.
- There are two script tags. The first script tag produces the graph for the FruitDev results. It plots the harvest day vs the number of Trusses
- The second script tag produces the graph for the growth results. It plots the harvest day and Average radiation.
- Before mapping, the data should be converted to an array and then passed to the API.

5. **table1.html & table2.html:**

- ❖ The tables are generated dynamically using ajax.
- ❖ The variable “radiation_data” is used for rows and “cell_data” is used for columns.
- ❖ The logic varies based on the number of rows and columns required.

Storing Data to influxDB:

1. The dbinflux.py inside the influxdb folder consists of the code to pull data from the NREL website.
2. The python code uses the API to pull data for the required year.
3. The attributes that are required must be changed with caution. When more than 6 attributes are requested the API does not work.
4. The data is pulled using the ‘pandas dataframe’ data structure.
5. The data is converted to json format before pushing it into the database.
6. JSON format:-

```
json_body = [
{
    "measurement": dbname,
    "tags": {
     },
    "fields": {
    },
    "time": row[0]
}
]
```

7. The json format order is important. The time variable should be in the end.
8. The max-series limit should be changed in the configuration file as per the user’s requirement.