AN6007 Team 6

Huang Kun (G2400288F)

Jacqueline Ong Shao Yi (G2400324K)

Limeylia Ng (G2403300B)

Sim Chun Han (G2400312G)

Wu Weiqiang (G2402305F)

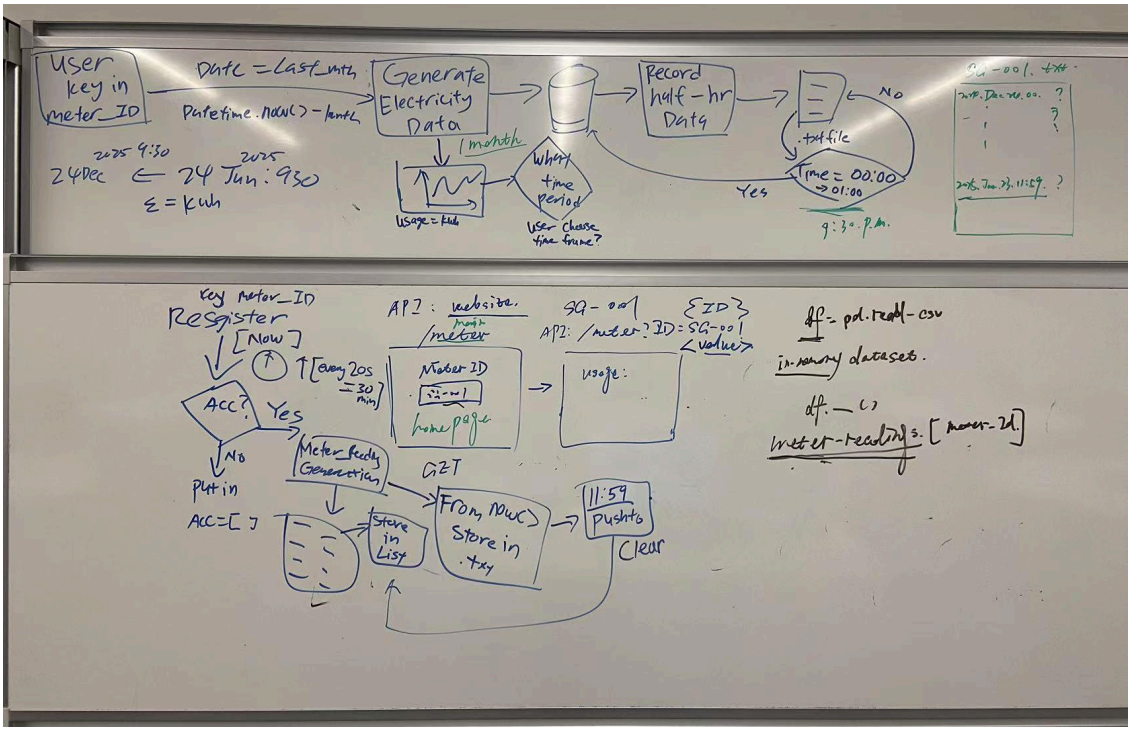
Documentation

Job Distribution and Contribution

Task	Chun Han	Kun	Jacqueline	Limeylia	Weiqiang
Project Planning	x	x	x	x	x
Github Setup		x	x		
Dashboard	x			x	
API Design	x	x	x	x	x
Data Structure Design	x	x	x	x	x
Backend Server	x	x	x	x	x
System Maintenance	x				x
Front-end		x		x	
Testing	x	x	x	x	x

Project Milestones

1. **Week 1 (16-22 January) - COMPLETE**
    a. Requirement Clarification
        i. Review project requirements and clarify with Mr Koh.
    b. Brainstorming → creating process maps, blueprints, and initial ideation for data structures and APIs



2. **Week 2 (23-29 January) - COMPLETE**
    a. Creation of group GitHub. GitHub bootcamp with everyone to understand version control and feature branch usage
    b. Design API endpoints
        i. Designing API for account registration, meter reading posting, electricity consumption query and billing.
        ii. Write documentation to describe the endpoints, including use cases and HTTP method.
3. **Week 3 (30 January - 5 February) - COMPLETE**
    a. API development and testing
        i. Implement API endpoint for account registration
        ii. Implement API endpoint to receive half-hourly meter readings, which is modeled to be done manually by the user.
        iii. Implement API endpoint to query for electricity consumption
        iv. Test for function and error handling
    b. Batch Jobs and Archive
        i. Implement archiving of daily electricity consumption and monthly consumption
        ii. Calculate daily and monthly consumption
        iii. Enable user querying of the electricity consumption
4. **Week 4 (6-12 February) - COMPLETE**
    a. System Maintenance
        i. Create stop server for maintenance period and preparation of readings
    b. Frontend for Flask server
    c. Swagger documentation for APIs (reachable at /apidocs)
5. **Week 5 (13-19 February) - COMPLETE**
    a. Dashboard creation that is the continuation of individual assignment
6. **Week 6** - Not Started

a. Interactive Dashboard with Visualization for Account Owners

API Definition

Route	Description	HTTP Method	Response
/	Main page for users to choose which function they would like to use (Register meter, submit meter reading, Check previous day/month reading)	GET	201:Register successfully 400: Missing or invalid field 409: Meter already registered
/register	For new users to register accounts, including meter ID (in 999-999-999 format), region, area, dwelling_type	POST	201: Meter Registered 400: Fields are invalid or missing 409: Meter already registered.
/meter-readings	Enter meter readings at a specific time	POST	202: Reading saved 400: Invalid Input 403: Meter Does not Exist 500: Internal Server Error
/meters/<meter_id>/daily/latest	Get previous day's electricity reading for a particular meter	GET	200: reading retrieved 404: No reading found or file not found 500: server error

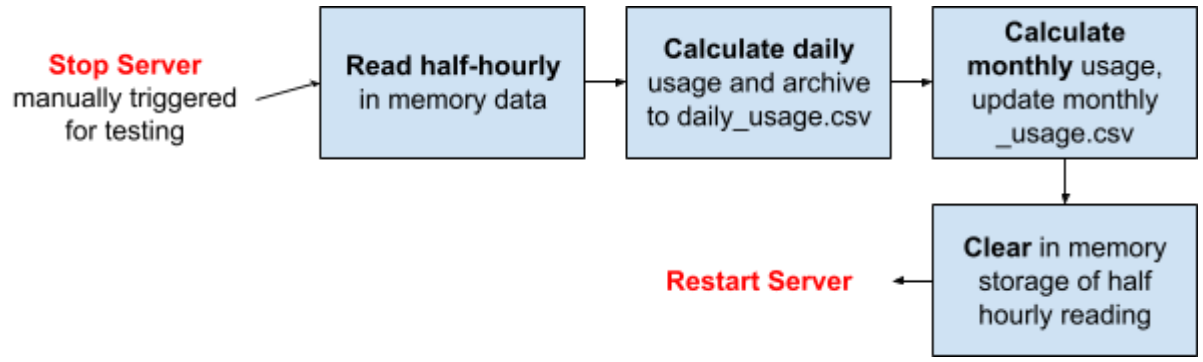
/meters/daily	Get all users daily meter readings. Can be used to analyse electricity trends.	GET	200: readings retrieved 404: No reading found or file not found 500: server error
/meters/<meter_id>/monthly/latest	Get previous months electricity reading for a particular meter	GET	200: readings retrieved 404: No reading found or file not found 500: server error
/meters/monthly	Get all users monthly meter readings. Can be used to analyse electricity trends.	GET	200: readings retrieved 404: No reading found or file not found 500: server error
/stop_server	The server goes on maintenance. Calculate daily usage and monthly usage and archive them on daily_usage.csv and monthly_usage.csv. Clear the in-memory data for the following day operation. Once everything is done, return the server back to live.	POST	200: Server under maintenance. Doing batch jobs

Batch Job

Batch Job Features:

1. Stop Server will be manually triggered for testing. However, in the future, it will automatically run at 12:00 AM daily.
2. Calculate daily usage by subtracting the latest reading with the first reading of the day based on the in-memory data storage.
3. Archive daily usage data to daily_usage.csv
4. Load daily consumption data from daily_usage.csv and sum up the total consumption in the current month.
5. Archive the monthly consumption to monthly_usage.csv.
6. Clear in-memory data for the following day operation.
7. Server is restarted.

Batch Job Illustrations:



Recommendation for Improvement

1. Change server maintenance to be triggered during 0000-0100.
    a. Current assumption is whenever it is manually triggered, it is time for maintenance.
2. Incorporate old server dashboard to the newly developed application.
    a. Region, Area and Dwelling-Type will be a drop down menu in the front end for better user experience.
3. Use more sophisticated libraries like Pydantic and FastAPI to enforce model validation for JSON bodies
    a. This will streamline checking that expected fields for each request are present, for example, that area and region are sent along at /register
4. Maintain the log-in status for users, so users don't need to enter meter_id again for daily usage and monthly billing after logging in.
5. For registration, provide dropdown lists of region, area and dwelling type for users to choose, which is more user-friendly.

ChatGPT Log

Dash DataFrame Filtering

Need:

Confused how to use the selected_region as value input to filter the area dropdown

User:

```
# Layout
layout = html.Div([
    dbc.Container([
        html.H1("Electricity Consumption", className="text-center my-4"),

        dbc.Row([
            dbc.Col([
                html.Label("Select Region"),
                dcc.Dropdown(options=[each for each in df["Region"].unique()], value =
df['Region'].iloc[1], id='region-dropdown')
            ], width=4),
            dbc.Col([
                html.Label("Select Area"),
                dcc.Dropdown(options=[each for each in df["Region" ==
selected_region]["Area"].unique()], value = df['Area'].iloc[1], id='area-dropdown')
            ], width=4),
            dbc.Col([
                html.Label("Select Dwelling Type"),
                dcc.Dropdown(options=[each for each in df["dwelling_type"].unique()], value =
df['dwelling_type'].iloc[1], id='dwelling-dropdown')
            ], width=4)
        ], className="mb-4"),

        dbc.Row([
            dbc.Col([
                dcc.Graph(id="electricity-graph")
            ])
        ])
    ])
])
```

can i filter like this in layout: df["Region" == selected_region]["Area"].

Abacus AI ChatLLM:

No, you cannot filter the DataFrame that way directly in the layout because `selected_region` is not defined at the time the layout is created. The dropdown options need to be available when the page loads, before any user selection is made.

Here's how you can fix this:

1. For initial loading, you should provide all unique areas, then use a callback to update the area dropdown based on the selected region.

2. Here's the corrected code:

‘’’
```
# Layout
layout = html.Div([
    dbc.Container([
        html.H1("Electricity Consumption", className="text-center my-4"),

        dbc.Row([
            dbc.Col([
                html.Label("Select Region"),
                dcc.Dropdown(options=[each for each in df["Region"].unique()],
                             value=df['Region'].iloc[1],
                             id='region-dropdown')
            ], width=4),
            dbc.Col([
                html.Label("Select Area"),
                dcc.Dropdown(id='area-dropdown')  # Options will be set by callback
            ], width=4),
            dbc.Col([
                html.Label("Select Dwelling Type"),
                dcc.Dropdown(options=[each for each in df["dwelling_type"].unique()],
                             value=df['dwelling_type'].iloc[1],
                             id='dwelling-dropdown')
            ], width=4)
        ], className="mb-4"),

        dbc.Row([
            dbc.Col([
                dcc.Graph(id="electricity-graph")
            ])
        ])
    ])
])
```

```

The key changes are:

1. Removed the direct filtering in the area dropdown options
2. Added a new callback to update the area dropdown based on the selected region
3. The area dropdown now starts empty and gets populated when a region is selected

This creates a cascading dropdown where the areas shown are filtered based on the selected region. The callback ensures that only relevant areas are shown for each region.

**Key Learnings**

It is possible to have more than one callback if necessary.
```