1. **Tech Description - Team Energy 5 - SolarEase**

2. **Description of all hardware components and software modules/frameworks used**

Programming Languages:

- Backend
  - Python
- Frontend
  - HTML (Hypertext Markup Language)
  - CSS (Cascading Style Sheets)
  - JS (JavaScript)

2.1. **Base HTML**

```
<!doctype html>
<html lang="en">

    <head>
        <meta charset="UTF-8">
        <meta name="viewport"
            content="width=device-width, initial-scale=1, user-scalable=no" />
        <title>{% block title %}Flask App{% endblock %}</title>
        <link rel="stylesheet"
            href="{{ url_for('static', filename='css/main.css') }}">
        <noscript>
            <link rel="stylesheet"
                href="{{ url_for('static', filename='css/noscript.css') }}">
        </noscript>
        {% block head %}{% endblock %}
    </head>

    <body class="is-preload">

        <!-- Page Wrapper -->
        <div id="page-wrapper">

            <!-- Header -->
            <header id="header">
                <h1><a href="{{ url_for('main.index') }}">Solar Ease</a></h1>
                <nav>
                    <a href="#menu">Menu</a>
                </nav>

                <!-- Menu -->
                <nav id="menu">
                    <div class="inner">
                        <h2>Menu</h2>
                        <ul class="links">
                            <li><a
```

```
                                    href="{{ url_for('main.index') }}">Home</a>
</li>
                            <li><a href="{{ url_for('main.power_mix') }}">Power
                                    Mix</a></li>
                            <li><a
                                    href="{{ url_for('main.roof_calculator')
}}">Roof
                                    Calculator</a></li>
                        </ul>
                        <a href="#" class="close">Close</a>
                    </div>
                </nav>
            </header>

            <section id="banner">
                <div class="inner">
                    <div>
                        {% block content_header %}{% endblock %}
                    </div>
                </section>

                <section id="wrapper">
                    <div class="wrapper">
                        {% block content %}{% endblock %}
                    </div>
                </section>

                <section id="footer">
                    <div class="inner">
                        {% block content_footer %}{% endblock %}
                    </div>
                </section>

            </div>

            <!-- Scripts -->
            <script
                src="{{ url_for('static', filename='js/jquery.min.js') }}">
</script>
            <script
                src="{{ url_for('static', filename='js/jquery.scrollex.min.js')
}}"></script>
            <script
                src="{{ url_for('static', filename='js/browser.min.js') }}">
</script>
            <script
                src="{{ url_for('static', filename='js/breakpoints.min.js') }}">
</script>
            <script
                src="{{ url_for('static', filename='js/util.js') }}"></script>
            <script
                src="{{ url_for('static', filename='js/main.js') }}"></script>
            {% block scripts %}{% endblock %}
        </body>
```

```
    </html>
```

Description

This is the base template for the Flask application. It sets up the basic structure of the webpage, including the header, navigation menu, content sections, and footer. It also includes links to external CSS and JavaScript files.

Components

1. **DOCTYPE and HTML Tag**:
    - Defines the document type and root element.
2. **Head Section**:
    - **Meta Tags**:
        - Character set (`UTF-8`).
        - Viewport settings for responsive design.
    - **Title Block**:
        - Placeholder for the page title.
    - **CSS Links**:
        - Main stylesheet.
        - Noscript stylesheet for users with JavaScript disabled.
    - **Head Block**:
        - Placeholder for additional head content.
3. **Body Section**:
    - **Body Class**:
        - Sets the class for preloading styles.
    - **Page Wrapper Div**:
        - **Header Section**:
            - **Header ID**: `header`.
            - **Logo**:
                - Link to the home page with the text "Solar Ease".
            - **Navigation**:
                - Link to the menu.
        - **Menu Section**:
            - **Menu ID**: `menu`.
            - **Inner Div**:
                - **Menu Heading**: "Menu".
                - **Links List**:
                    - Home, Power Mix, and Roof Calculator links.
                - **Close Link**: Link to close the menu.
        - **Banner Section**:
            - **Banner ID**: `banner`.
            - **Inner Div**:
                - Placeholder for content header.
        - **Wrapper Section**:
            - **Wrapper ID**: `wrapper`.

        - **Inner Wrapper Div**:
            - Placeholder for main content.
    - **Footer Section**:
        - **Footer ID**: `footer`.
        - **Inner Div**:
            - Placeholder for footer content.

4. **Scripts**:
    - **JavaScript Includes**:
        - jQuery.
        - jQuery Scrollex.
        - Browser.
        - Breakpoints.
        - Util.
        - Main.
    - **Scripts Block**:
        - Placeholder for additional scripts.

2.2. **Index (Landing Page)**

2.2.1. **Backend**

2.2.1.1. Function: `index`

```
@main.route("/")
def index():
    return flask.render_template("index.html")
```

Route: `/`

Methods: `GET`

Description

The `index` function is a route handler that renders the `index.html` template when the root URL ("/") is accessed via a GET request.

Functionality Breakdown

1. **Route Definition**:
    - The function is mapped to the root URL route ("/"), which triggers the function when accessed.
2. **Rendering Template**:
    - The function uses Flask's `render_template` method to serve the `index.html` template.
    - This allows the web application to present the content defined within the `index.html` template file to the user.

2.2.2. **Frontend**

2.2.2.1. Content Header Block

```
{% block content_header %}
<div class="logo">
    <span class="icon">
        <img
            style="display: inline-block; width: 100%; height: 100%; object-fit:
cover;"
            src="{{ url_for('static', filename='images/logo.png') }}"
            alt>
    </span>
</div>
<h2>Welcome to Solar Ease</h2>
<p>Bringing Neighbors Together for Local Solar Empowerment</p>
{% endblock %}
```

Description

The **content_header** block defines the header section of the web page content, including the logo and introductory text.

Functionality Breakdown

1. **Logo Section**:
    - **Container**: A `<div>` with the class "logo" to encapsulate the logo.
    - **Icon**: A `<span>` element with the class "icon" that contains an `<img>` element.
    - **Image**: The `<img>` element displays the logo image sourced from the static folder and styled to fit the container using inline CSS.
2. **Header Text**:
    - **Title**: An `<h2>` element displaying "Welcome to Solar Ease".
    - **Subtitle**: A `<p>` element providing a brief introduction with the text "Bringing Neighbors Together for Local Solar Empowerment".

2.2.2.2. Section: Identifying a Problem

```
<section id="one" class="wrapper spotlight style1">
    <div class="inner">
        <a href="#" class="image"><img
                src="{{ url_for('static', filename='images/identifying-the-
problem.webp') }}"
                alt></a>
        <div class="content">
            <h2 class="major">Identifying a problem</h2>
            <p class="justifier">Harnessing solar energy is a powerful step
                toward a sustainable future, offering a clean, renewable source
                of power that reduces our carbon footprint, mitigates climate
                change, and supports self-sufficient, resilient energy
                communities. However, the transition is hindered by a lack of
```

```
                real-time data, revenue channels, and financial resources.
                Decisions are often based on rough estimates, and excess PV
                energy is sold to the grid at low prices, making the model
                unsustainable.</p>
        </div>
    </div>
</section>
```

Description

This section describes a problem related to the harnessing of solar energy, detailing the challenges and the current unsustainable model.

Functionality Breakdown

1. **Section Container**:
    - **Element**: `<section>` with id "one", class "wrapper spotlight style1".
    - **Purpose**: Acts as a container for the entire problem-identifying section.
2. **Inner Container**:
    - **Element**: `<div>` with class "inner".
    - **Purpose**: Wraps the content to provide inner styling and alignment.
3. **Image Link**:
    - **Element**: `<a>` with an embedded `<img>`.
    - **Image Source**: The image is sourced from the static folder and displayed using the `<img>` tag.
    - **Purpose**: Provides a visual element related to the problem being described.
4. **Content Container**:
    - **Element**: `<div>` with class "content".
    - **Purpose**: Contains the text content of the section.
5. **Major Heading**:
    - **Element**: `<h2>` with class "major".
    - **Text**: "Identifying a problem".
    - **Purpose**: Serves as the main heading for the section.
6. **Description Paragraph**:
    - **Element**: `<p>` with class "justifier".
    - **Text**: Describes the challenges in transitioning to solar energy, including lack of real-time data, revenue channels, and financial resources.
    - **Purpose**: Provides a detailed explanation of the problem.

2.2.2.3. Section: Our Solution

```
<section id="two" class="wrapper alt spotlight style2">
    <div class="inner">
        <a href="#" class="image"><img
                src="{{ url_for('static', filename='images/our-solution.webp') }}"
                alt></a>
        <div class="content">
            <h2 class="major">Our solution</h2>
```

```
            <p class="justifier">We propose autonomous mini-grids to reduce
                dependence on utility
                providers. These mini-grids provide real-time data on energy
                production and consumption, financial tracking, and monthly
                projections, enabling efficient investment in PVs and strategic
                energy procurement. Equipped with sensors, smart meters, and IoT
                devices, this can connect 6-8 buildings. Trained on four years
                of
                energy bills and online weather data, the system offers data
                visualization and monthly projections.</p>
        </div>
    </div>
</section>
```

Description

This section outlines the proposed solution for the identified problem, focusing on autonomous mini-grids and their benefits.

Functionality Breakdown

1. **Section Container**:
    - **Element**: `<section>` with id "two", class "wrapper alt spotlight style2".
    - **Purpose**: Acts as a container for the entire solution section.
2. **Inner Container**:
    - **Element**: `<div>` with class "inner".
    - **Purpose**: Wraps the content to provide inner styling and alignment.
3. **Image Link**:
    - **Element**: `<a>` with an embedded `<img>`.
    - **Image Source**: The image is sourced from the static folder and displayed using the `<img>` tag.
    - **Purpose**: Provides a visual element related to the proposed solution.
4. **Content Container**:
    - **Element**: `<div>` with class "content".
    - **Purpose**: Contains the text content of the section.
5. **Major Heading**:
    - **Element**: `<h2>` with class "major".
    - **Text**: "Our solution".
    - **Purpose**: Serves as the main heading for the section.
6. **Description Paragraph**:
    - **Element**: `<p>` with class "justifier".
    - **Text**: Describes the proposed autonomous mini-grids, detailing their functionalities, benefits, and the technology used.
    - **Purpose**: Provides a detailed explanation of the solution.

2.2.2.4. Section: How We Are Different

```
<h2 class="major">How We Are Different</h2>
<div class="table-wrapper">
    <table>
        <thead>
            <tr>
                <th style="text-align: center;"></th>
                <th style="text-align: center;">CREM</th>
                <th style="text-align: center;">Solar Aurora</th>
                <th style="text-align: center;">1komma5</th>
                <th style="text-align: center;">SolarEase</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>Energy Management System</td>
                <td style="text-align: center;">&#10003;</td>
                <td style="text-align: center;"></td>
                <td style="text-align: center;">&#10003;</td>
                <td style="text-align: center;">&#10003;</td>
            </tr>
            <tr>
                <td>Data Analytics / Forecasting</td>
                <td style="text-align: center;"></td>
                <td style="text-align: center;">&#10003;</td>
                <td style="text-align: center;">&#10003;</td>
                <td style="text-align: center;">&#10003;</td>
            </tr>
            <tr>
                <td>Consulting</td>
                <td style="text-align: center;">&#10003;</td>
                <td style="text-align: center;">&#10003;</td>
                <td style="text-align: center;"></td>
                <td style="text-align: center;">&#10003;</td>
            </tr>
            <tr>
                <td>Sell/Buy through Mini-Grid</td>
                <td style="text-align: center;"></td>
                <td style="text-align: center;"></td>
                <td style="text-align: center;"></td>
                <td style="text-align: center;">&#10003;</td>
            </tr>
        </tbody>
    </table>
</div>
```

Description

This section displays a comparison table highlighting the unique features of SolarEase compared to other competitors.

Components

1. **Heading**:
    - **Tag**: `<h2>`
    - **Class**: `major`
    - **Content**: "How We Are Different"
2. **Table Wrapper**:
    - **Tag**: `<div>`
    - **Class**: `table-wrapper`
3. **Table**:
    - **Tag**: `<table>`
4. **Table Head**:
    - **Tag**: `<thead>`
    - **Content**: Contains a row with five columns.
        - **Columns**:
            - **First Column**: Empty.
            - **Second to Fifth Columns**: Competitor names ("CREM", "Solar Aurora", "1komma5", "SolarEase").
        - **Text Alignment**: Centered using `style="text-align: center;"`.
5. **Table Body**:
    - **Tag**: `<tbody>`
    - **Content**: Contains four rows comparing different features across competitors.
        - **Rows**:
            - **First Row**:
                - **Feature**: "Energy Management System"
                - **Competitor Checkmarks**: CREM (✔), 1komma5 (✔), SolarEase (✔).
            - **Second Row**:
                - **Feature**: "Data Analytics / Forecasting"
                - **Competitor Checkmarks**: Solar Aurora (✔), 1komma5 (✔), SolarEase (✔).
            - **Third Row**:
                - **Feature**: "Consulting"
                - **Competitor Checkmarks**: CREM (✔), Solar Aurora (✔), SolarEase (✔).
            - **Fourth Row**:
                - **Feature**: "Sell/Buy through Mini-Grid"
                - **Competitor Checkmarks**: SolarEase (✔).
        - **Text Alignment**: Centered using `style="text-align: center;"`.

2.2.2.5. Section: Our Features

```
<h2 class="major">Our Features</h2>
<section class="features">
    <article>
        <a href="{{ url_for('main.power_mix') }}" class="image"><img
                src="{{ url_for('static', filename='images/power-mix.webp') }}"
                alt></a>
        <h3 class="major">Power Mix</h3>
        <p class="justifier">This website can calculate the power mix
            for a home given the
```

```
            start and end date of the period. The provided values are
            the overall consumption, the PV generation and how much
            energy has to be taken from the main grid, and how much can
            be sold to the main grid. This website calculates the
            investment needed and profit made for the given period and
            provides an in-depth cost analysis of the generation and
            main grid cost. </p>
        <a href="{{ url_for('main.power_mix') }}" class="special">Learn
            more</a>
    </article>
    <article>
        <a href="{{ url_for('main.roof_calculator') }}"
            class="image"><img
                src="{{ url_for('static', filename='images/roof-calculator.webp')
}}"
                alt></a>
        <h3 class="major">Roof Calculator</h3>
        <p class="justifier">This website also has a feature called roof
            calculator. The
            user has to just enter his address and select the rooftop
            and he will be provided with information on how big the
            rooftop is, what position the solar panels have, and how
            much energy can be generated from this rooftop. </p>
        <a href="{{ url_for('main.roof_calculator') }}"
            class="special">Learn more</a>
    </article>
</section>
```

Description

This section showcases the key features of the SolarEase website, namely "Power Mix" and "Roof Calculator".

Components

1. **Heading**:
    - **Tag**: `<h2>`
    - **Class**: `major`
    - **Content**: "Our Features"
2. **Features Section**:
    - **Tag**: `<section>`
    - **Class**: `features`
3. **Feature Articles**:
    - **Tag**: `<article>`
4. **Power Mix Article**:
    - **Link**:
        - **Tag**: `<a>`
        - **Attributes**:
            - `href`: Links to the power mix feature (`{{ url_for('main.power_mix') }}`)
            - `class`: `image`
        - **Image**:

        - **Tag**: `<img>`
        - **Attributes**:
            - `src`: Path to the power mix image (`{{ url_for('static', filename='images/power-mix.webp') }}`)
            - `alt`: Alternative text for the image.
    - **Feature Title**:
        - **Tag**: `<h3>`
        - **Class**: `major`
        - **Content**: "Power Mix"
    - **Description**
        - **Tag**: `<p>`
        - **Class**: `justifier`
        - **Content**: A detailed description of the Power Mix feature.
    - **Learn More Link**:
        - **Tag**: `<a>`
        - **Attributes**:
            - `href`: Links to the power mix feature (`{{ url_for('main.power_mix') }}`)
            - `class`: `special`
        - **Content**: "Learn more"
5. **Roof Calculator Article**:
    - **Link**:
        - **Tag**: `<a>`
        - **Attributes**:
            - `href`: Links to the roof calculator feature (`{{ url_for('main.roof_calculator') }}`)
            - `class`: `image`
        - **Image**:
            - **Tag**: `<img>`
            - **Attributes**:
                - `src`: Path to the roof calculator image (`{{ url_for('static', filename='images/roof-calculator.webp') }}`)
                - `alt`: Alternative text for the image.
    - **Feature Title**:
        - **Tag**: `<h3>`
        - **Class**: `major`
        - **Content**: "Roof Calculator"
    - **Description**
        - **Tag**: `<p>`
        - **Class**: `justifier`
        - **Content**: A detailed description of the Roof Calculator feature.
    - **Learn More Link**:
        - **Tag**: `<a>`
        - **Attributes**:
            - `href`: Links to the roof calculator feature (`{{ url_for('main.roof_calculator') }}`)
            - `class`: `special`

- **Content**: "Learn more"

2.2.2.6. Section: Our Team

```
<h2 class="major">Our Team</h2>
<div class="box alt">
    <div class="features2 row gtr-uniform">
        <article class="col-4"><span class="image fit"><img
                    src="{{ url_for('static', filename='images/pic05.jpg') }}"
                    alt></span>
            <h3 class="major">Ivana Peneva</h3>
            <p>M.Sc. Information Systems</p>
            <a href="https://www.linkedin.com/in/ivana-peneva-267927239/"
                class="special">Contact</a>
        </article>
        <article class="col-4"><span class="image fit"><img
                    src="{{ url_for('static', filename='images/pic06.jpg') }}"
                    alt></span>
            <h3 class="major">Asad Khan</h3>
            <p>M.Sc Management and Technology</p>
            <a href="https://www.linkedin.com/in/khanasad94/"
                class="special">Contact</a>
        </article>
        <article class="col-4"><span class="image fit"><img
                    src="{{ url_for('static', filename='images/pic07.jpg') }}"
                    alt></span>
            <h3 class="major">Maqarios Saleh</h3>
            <p>M.Sc. Computer Science</p>
            <a href="https://www.linkedin.com/in/maqarios/"
                class="special">Contact</a>
        </article>
    </div>
</div>
```

Description

This section introduces the team members involved in the project, providing their names, qualifications, and contact links.

Functionality Breakdown

1. **Section Heading**:
    - **Element**: `<h2>` with class "major".
    - **Text**: "Our Team".
    - **Purpose**: Acts as the title for the team section.
2. **Box Container**:
    - **Element**: `<div>` with class "box alt".
    - **Purpose**: Provides an alternate styling for the inner content, typically with a different background or border.

3. **Features Row Container**:
    - **Element**: `<div>` with class "features2 row gtr-uniform".
    - **Purpose**: Contains the individual team member articles, displayed in a row with uniform spacing.
4. **Team Member Article: Ivana Peneva**:
    - **Element**: `<article>` with class "col-4".
    - **Components**
        - **Image**: `<span>` with class "image fit" and embedded `<img>`, displaying Ivana's picture.
        - **Heading**: `<h3>` with class "major", text "Ivana Peneva".
        - **Qualification**: `<p>`, text "M.Sc. Information Systems".
        - **Contact Link**: `<a>` with class "special", text "Contact", linking to Ivana's LinkedIn profile.
    - **Purpose**: Provides information about Ivana Peneva, her qualifications, and a link to contact her.
5. **Team Member Article: Asad Khan**:
    - **Element**: `<article>` with class "col-4".
    - **Components**
        - **Image**: `<span>` with class "image fit" and embedded `<img>`, displaying Asad's picture.
        - **Heading**: `<h3>` with class "major", text "Asad Khan".
        - **Qualification**: `<p>`, text "M.Sc Management and Technology".
        - **Contact Link**: `<a>` with class "special", text "Contact", linking to Asad's LinkedIn profile.
    - **Purpose**: Provides information about Asad Khan, his qualifications, and a link to contact him.
6. **Team Member Article: Maqarios Saleh**:
    - **Element**: `<article>` with class "col-4".
    - **Components**
        - **Image**: `<span>` with class "image fit" and embedded `<img>`, displaying Maqarios's picture.
        - **Heading**: `<h3>` with class "major", text "Maqarios Saleh".
        - **Qualification**: `<p>`, text "M.Sc. Computer Science".
        - **Contact Link**: `<a>` with class "special", text "Contact", linking to Maqarios's LinkedIn profile.
    - **Purpose**: Provides information about Maqarios Saleh, his qualifications, and a link to contact him.

2.3. **Power Mix**

2.3.1. **Backend**

2.3.1.1. Function: `power_mix`

```python
@main.route("/power_mix")
def power_mix():
    return flask.render_template("power_mix.html")
```

Route: `/power_mix`

Methods: `GET`

Description

The `power_mix` function renders the `power_mix.html` template when the `/power_mix` endpoint is accessed via a GET request.

Functionality Breakdown

1. **Route Definition**:
   - The function is mapped to the `/power_mix` URL route, triggering the function when accessed.
2. **Rendering Template**:
   - The function uses Flask's `render_template` method to serve the `power_mix.html` template.
   - This allows the web application to present the power mix data or interface defined within the `power_mix.html` template file to the user.

2.3.1.2. Function: `calculate_power_mix`

```python
@main.route("/calculate_power_mix", methods=["POST"])
def calculate_power_mix():
    data = flask.request.json
    start_date_str = data.get("start_date")
    end_date_str = data.get("end_date")

    if not start_date_str or not end_date_str:
        return flask.jsonify({"error": "Invalid date range"}), 400

    try:
        start_date = datetime.strptime(start_date_str, "%Y-%m-%d")
        end_date = datetime.strptime(end_date_str, "%Y-%m-%d")
    except ValueError:
        return flask.jsonify({"error": "Invalid date format"}), 400

    with open(
        os.path.join(
            os.path.join(os.path.dirname(os.path.abspath(__file__)), "data"),
            "simulation.json",
        ),
        "r",
    ) as f:
        simulation_data = json.load(f)

    total_generation = 0
    total_consumption = 0

    for record in simulation_data["Data"]:
        record_date = datetime.strptime(record["date"], "%Y-%m-%d")
        if start_date <= record_date <= end_date:
            total_generation += record["generation_kWh"]
            total_consumption += record["consumption_kWh"]

    remaining_consumption = total_consumption - total_generation

    # Calculate total cost and profit
```

```
    government_price_per_kWh = 0.4
    generation_cost = total_generation * government_price_per_kWh
    consumption_cost = total_consumption * government_price_per_kWh

    initial_investment = simulation_data["PV System Cost"]["Total Cost"]

    profit_earned = total_generation * government_price_per_kWh

    return flask.jsonify(
        {
            "total_generation": round(total_generation),
            "total_consumption": round(total_consumption),
            "remaining_consumption": round(remaining_consumption),
            "generation_cost": round(generation_cost),
            "consumption_cost": round(consumption_cost),
            "initial_investment": round(initial_investment),
            "profit_earned": round(profit_earned),
        }
    )
```

Route: `/calculate_power_mix`

Methods: `POST`

Description

The `calculate_power_mix` function calculates power generation, consumption, and associated costs for a specified date range based on historical simulation data. It processes JSON data from a POST request, performs necessary calculations, and returns the results as a JSON response.

Functionality Breakdown

1. **Data Extraction and Validation**:
    - Extracts JSON data from the request, specifically the start and end dates.
    - Validates the presence of the start and end dates, returning an error if either is missing.
    - Attempts to parse the date strings into `datetime` objects, returning an error if the format is invalid.
2. **Loading Simulation Data**:
    - Opens and reads the `simulation.json` file, which contains historical power generation and consumption data.
    - Initializes variables to store total power generation and consumption.
3. **Calculating Generation and Consumption**:
    - Iterates over the records in the simulation data.
    - For each record within the specified date range, adds up the `generation_kWh` and `consumption_kWh`.
4. **Calculating Costs and Profit**:
    - Calculates the remaining consumption by subtracting total generation from total consumption.
    - Computes the generation and consumption costs based on a predefined price (`government_price_per_kWh`).
    - Retrieves the initial investment cost from the simulation data.

- Calculates the profit earned from the power generation.

5. **Returning the Results**:
   - Returns a JSON response containing:
     - `total_generation`: Total power generated.
     - `total_consumption`: Total power consumed.
     - `remaining_consumption`: Power consumption minus generation.
     - `generation_cost`: Cost of generated power.
     - `consumption_cost`: Cost of consumed power.
     - `initial_investment`: Initial cost of the PV system.
     - `profit_earned`: Profit from generated power.
   - These values are rounded for readability.

2.3.2. **Frontend**

2.3.2.1. Start and End Date Selectors

```
<div class="row">
    <div class="col-6 col-12">
        <label for="start-date">Start Date</label>
        <ul class="actions" name="start-date" id="start-date">
            <li>
                <div class="col-6 col-12">
                    <label for="start-date-day">Day</label>
                    <select name="start-date-day" id="start-date-day">
                        {% for day in range(1, 32) %}
                        <option value="{{ day }}">{{ day }}</option>
                        {% endfor %}
                    </select>
                </div>
            </li>
            <li>
                <div class="col-6 col-12">
                    <label for="start-date-month">Month</label>
                    <select name="start-date-month" id="start-date-month">
                        <option value="1">January</option>
                        <option value="2">February</option>
                        <option value="3">March</option>
                        <option value="4">April</option>
                        <option value="5">May</option>
                        <option value="6">June</option>
                        <option value="7">July</option>
                        <option value="8">August</option>
                        <option value="9">September</option>
                        <option value="10">October</option>
                        <option value="11">November</option>
                        <option value="12">December</option>
                    </select>
                </div>
            </li>
            <li>
```

```
                <div class="col-6 col-12">
                    <label for="start-date-year">Year</label>
                    <select name="start-date-year" id="start-date-year">
                        {% for year in range(2020, 2025) %}
                        <option value="{{ year }}">{{ year }}</option>
                        {% endfor %}
                    </select>
                </div>
            </li>
        </ul>
        <label for="end-date">End Date</label>
        <ul class="actions" name="end-date" id="end-date">
            <li>
                <div class="col-6 col-12">
                    <label for="end-date-day">Day</label>
                    <select name="end-date-day" id="end-date-day">
                        {% for day in range(1, 32) %}
                        <option value="{{ day }}" {% if day == 31 %}selected{%
                            endif %}>{{ day }}</option>
                        {% endfor %}
                    </select>
                </div>
            </li>
            <li>
                <div class="col-6 col-12">
                    <label for="end-date-month">Month</label>
                    <select name="end-date-month" id="end-date-month">
                        <option value="1">January</option>
                        <option value="2">February</option>
                        <option value="3">March</option>
                        <option value="4">April</option>
                        <option value="5">May</option>
                        <option value="6">June</option>
                        <option value="7">July</option>
                        <option value="8">August</option>
                        <option value="9">September</option>
                        <option value="10">October</option>
                        <option value="11">November</option>
                        <option value="12" selected>December</option>
                    </select>
                </div>
            </li>
            <li>
                <div class="col-6 col-12">
                    <label for="end-date-year">Year</label>
                    <select name="end-date-year" id="end-date-year">
                        {% for year in range(2020, 2025) %}
                        <option value="{{ year }}" {% if year == 2022
                            %}selected{% endif %}>{{ year }}</option>
                        {% endfor %}
                    </select>
                </div>
            </li>
        </ul>
```

```
        <ul class="col-6 actions">
            <li><button id="calculate-button"
                    class="button">Calculate</button></li>
        </ul>
    </div>
</div>
```

Description

This code block provides a user interface for selecting the start and end dates with dropdowns for day, month, and year.

Components

- **Start Date Selection**:
  - **Day Selector**: Dropdown menu for selecting the day (1-31).
  - **Month Selector**: Dropdown menu for selecting the month (January-December).
  - **Year Selector**: Dropdown menu for selecting the year (2020-2025).
- **End Date Selection**:
  - **Day Selector**: Dropdown menu for selecting the day (1-31), with 31 pre-selected.
  - **Month Selector**: Dropdown menu for selecting the month (January-December), with December pre-selected.
  - **Year Selector**: Dropdown menu for selecting the year (2020-2025), with 2022 pre-selected.
- **Calculate Button**: Button to initiate the calculation process.

```
document.getElementById('calculate-button').addEventListener('click', function ()
{
    const startDate = new Date(
        document.getElementById('start-date-year').value,
        document.getElementById('start-date-month').value - 1,
        document.getElementById('start-date-day').value
    );
    const endDate = new Date(
        document.getElementById('end-date-year').value,
        document.getElementById('end-date-month').value - 1,
        document.getElementById('end-date-day').value
    );

    if (endDate >= startDate) {
        fetch('/calculate_power_mix', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({
                start_date: startDate.toISOString().split('T')[0],
                end_date: endDate.toISOString().split('T')[0]
            })
        })
```

```
            .then(response => response.json())
            .then(data => {
                if (data.error) {
                    alert("Error: " + data.error);
                    hideAllSections();
                } else {
                    showAllSections();
                    generationComsumptionChart(data.total_generation,
data.total_consumption);
                    generationComsumptionChart2(data.total_generation,
data.total_consumption);
                    investmentProfitChart(data.initial_investment,
data.profit_earned);
                    drawColumnChart(data.consumption_cost, data.generation_cost);
                }
            });
    } else {
        alert("End date must be greater than or equal to start date.");
        hideAllSections();
    }
});
```

Description

This script adds functionality to the "Calculate" button, enabling it to fetch and process data based on user-selected start and end dates.

Functionality

1. **Event Listener**:
   - **Calculate Button**: Adds a click event listener to the calculate button.
2. **Date Retrieval and Validation**:
   - **Start Date**: Retrieves and constructs the start date from the selected values.
   - **End Date**: Retrieves and constructs the end date from the selected values.
   - **Validation**: Ensures that the end date is not earlier than the start date. If invalid, an alert is shown.
3. **API Request**:
   - **Request**: Sends a POST request to the `/calculate_power_mix` endpoint with the start and end dates in ISO format.
   - **Response Handling**:
     - **Error Handling**: Displays an alert and hides sections if there's an error.
     - **Success Handling**: Displays the relevant charts and sections based on the response data.
4. **Chart Drawing Functions**:
   - **generationComsumptionChart**: Draws the generation vs. consumption pie chart.
   - **generationComsumptionChart2**: Draws the consumption breakdown pie chart.
   - **investmentProfitChart**: Draws the investment vs. profit pie chart.
   - **drawColumnChart**: Draws the cost analysis column chart.

2.3.2.2. Google Charts Initialization and Section Visibility Management

```
google.charts.load('current', { 'packages': ['corechart', 'bar'] });
google.charts.setOnLoadCallback(hideAllSections);

function hideAllSections() {
    document.querySelectorAll('#wrapper section').forEach(section => {
        section.style.display = 'none';
    });
}

function showAllSections() {
    document.querySelectorAll('#wrapper section').forEach(section => {
        section.style.display = '';
    });
}
```

Description

This script loads the Google Charts library and manages the visibility of sections on the webpage.

Functionality

1. **Google Charts Loading**:
   - **Load Library**: Uses `google.charts.load` to load the current version of the `corechart` and `bar` packages.
   - **Callback**: Sets a callback function (`hideAllSections`) to be called once the library is loaded.
2. **Section Visibility Management**:
   - **Hide All Sections**:
     - **Function**: `hideAllSections`
     - **Action**: Selects all sections within the `#wrapper` element and sets their display style to `none`, effectively hiding them.
   - **Show All Sections**:
     - **Function**: `showAllSections`
     - **Action**: Selects all sections within the `#wrapper` element and resets their display style to the default, making them visible.

2.3.2.3. Section for Generation, Consumption & Excess

```
<section id="one" class="wrapper spotlight style1">
    <div class="inner">
        <a href="#" class="image2">
            <div id="piechart" style="width: 300px; height: 300px;"></div>
        </a>
        <div class="content">
            <h2 class="major">Generation, Consumption & Excess</h2>
            <p>This pie chart provides a visual representation of energy
generation and consumption, highlighting
                the proportion of energy that is consumed versus the amount
```

```
generated. The green segment represents
                the generated energy, while the red segment illustrates the
consumed energy. If the amount of the
                generated energy is greater than the consumed another segment is
shown in orange. The excess energy
                is being sold to the main grid.</p>
            <a href="#" class="special">Learn more</a>
        </div>
    </div>
</section>
```

Description

This HTML section provides a visual representation and description of energy generation, consumption, and excess energy being sold to the main grid.

Components

1. **Section Wrapper**:
   - **ID**: `one`
   - **Class**: `wrapper spotlight style1`
2. **Inner Container**:
   - **Class**: `inner`
3. **Image Container**:
   - **Class**: `image2`
   - **Chart**: Contains a `div` with ID `piechart` sized 300px by 300px, intended to display a pie chart.
4. **Content**:
   - **Header**: `<h2 class="major">Generation, Consumption & Excess</h2>`
   - **Paragraph**: Describes the pie chart segments:
     - Green segment: Generated energy.
     - Red segment: Consumed energy.
     - Orange segment: Excess energy sold to the main grid.
   - **Link**: `<a href="#" class="special">Learn more</a>`

```
function generationComsumptionChart(totalGeneration, totalConsumption) {
    var data;
    var colors;


    if (totalGeneration <= totalConsumption) {
        data = google.visualization.arrayToDataTable([
            ['Task', 'Consumption & Generation (kWh)'],
            ['Generation', totalGeneration],
            ['Consumption', totalConsumption]
        ]);
        colors = ['green', 'red'];
    } else {
        data = google.visualization.arrayToDataTable([
            ['Task', 'Consumption & Generation (kWh)'],
```

```
            ['Generation', totalConsumption],
            ["Sold to Main Grid", totalGeneration - totalConsumption],
            ['Consumption', totalConsumption]
        ]);
        colors = ['green', 'orange', 'red'];
    }

    var options = {
        legend: 'none',
        pieSliceText: 'label',
        backgroundColor: 'transparent',
        chartArea: {
            left: 0,
            top: 0,
            width: '99%',
            height: '99%'
        },
        fontSize: 12,
        colors: colors
    };

    var chart = new
google.visualization.PieChart(document.getElementById('piechart'));
    chart.draw(data, options);
}
```

Description

This function creates a pie chart to visually represent the energy generation, consumption, and any excess energy sold to the main grid.

Parameters:

- `totalGeneration`: The total amount of energy generated.
- `totalConsumption`: The total amount of energy consumed.

Functionality

1. **Data Preparation**:
    - If `totalGeneration` is less than or equal to `totalConsumption`, the data array will include:
        - `Generation`: `totalGeneration`
        - `Consumption`: `totalConsumption`
        - Colors used: `green` for Generation and `red` for Consumption.
    - If `totalGeneration` is greater than `totalConsumption`, the data array will include:
        - `Generation`: `totalConsumption`
        - `Sold to Main Grid`: `totalGeneration - totalConsumption`
        - `Consumption`: `totalConsumption`
        - Colors used: `green` for Generation, `orange` for Sold to Main Grid, and `red` for Consumption.
2. **Chart Options**:
    - `legend`: Set to `none`.

- `pieSliceText`: Set to `label`.
- `backgroundColor`: Set to `transparent`.
- `chartArea`: Configured to cover 99% of the width and height of the container, with no margin (`left: 0`, `top: 0`).
- `fontSize`: Set to 12.
- `colors`: Set to the `colors` array defined based on the data.

3. **Chart Creation**:
   - Creates a new `google.visualization.PieChart` object with the container element identified by the ID `piechart`.
   - Draws the chart with the prepared data and options.

2.3.2.4. Section for Energy Sources

```
<section id="two" class="wrapper alt spotlight style2">
    <div class="inner">
        <a href="#" class="image2">
            <div id="piechart2" style="width: 300px; height: 300px;"></div>
        </a>
        <div class="content">
            <h2 class="major">Energy Sources</h2>
            <p>This chart focuses on the role and importance of the energy mix.
The consumption from the previous
                pie chart is split to show the contribution of the different
energy sources. Energy produced using
                PV is visualized in green, and energy from the main grid is in
orange. If the energy produced using
                PV isn’t sufficient to cover the whole consumption energy from the
main grid is being used. </p>
            <a href="#" class="special">Learn more</a>
        </div>
    </div>
</section>
```

Description

This section of the webpage highlights the sources of energy used and their contributions to the overall consumption.

Components

1. **Container**:
   - **Section**: Identified by the ID `two`, with classes `wrapper alt spotlight style2`.
   - **Inner Div**: Contains the main content of the section.
2. **Image Container**:
   - **Anchor Tag**: Wraps the `div` that will hold the pie chart.
   - **Div**: Identified by the ID `piechart2`, with a fixed size of 300px by 300px, which will render the pie chart.
3. **Content**:

- **Heading**: An `h2` tag with the class `major` displaying the title "Energy Sources".
- **Paragraph**:
  - Describes the purpose of the chart, which is to show the breakdown of energy consumption from different sources.
  - Explains that the pie chart will split the consumption data to display the contribution of energy produced using photovoltaic (PV) systems (in green) and energy sourced from the main grid (in orange).
- **Anchor Tag**: A link with the class `special` for further information labeled "Learn more".

```
function generationComsumptionChart2(totalGeneration, totalConsumption) {
    if (totalGeneration > totalConsumption) {
        totalConsumption = totalGeneration;
    }

    var data = google.visualization.arrayToDataTable([
        ['Task', 'Total Consumption (kWh)'],
        ['PV Generation', totalGeneration],
        ['Main Grid', totalConsumption - totalGeneration]
    ]);

    var options = {
        legend: 'none',
        pieSliceText: 'label',
        backgroundColor: 'transparent',
        chartArea: {
            left: 0,
            top: 0,
            width: '99%',
            height: '99%'
        },
        fontSize: 12,
        colors: ['green', 'darkorange']
    };

    var chart = new
google.visualization.PieChart(document.getElementById('piechart2'));
    chart.draw(data, options);
}
```

Description

This function generates a pie chart that illustrates the contributions of photovoltaic (PV) generation and main grid energy to the total energy consumption.

Parameters:

- `totalGeneration`: The total amount of energy generated by PV systems.
- `totalConsumption`: The total amount of energy consumed.

Functionality

1. **Data Preparation**:
    - If `totalGeneration` exceeds `totalConsumption`, adjust `totalConsumption` to match `totalGeneration` to ensure accurate representation.
    - Create an array `data` with the following entries:
        - `PV Generation`: The amount of energy generated by PV systems (`totalGeneration`).
        - `Main Grid`: The difference between `totalConsumption` and `totalGeneration`.
2. **Chart Options**:
    - **`legend`**: Set to `none` to hide the legend.
    - **`pieSliceText`**: Set to `label` to display the labels on the pie slices.
    - **`backgroundColor`**: Set to `transparent` to make the chart background transparent.
    - **`chartArea`**: Configured to cover 99% of the width and height of the container, with no margin (`left: 0`, `top: 0`).
    - **`fontSize`**: Set to 12 for the chart text.
    - **`colors`**: Set to `['green', 'darkorange']` to differentiate between PV generation and main grid energy.
3. **Chart Creation**:
    - Create a new `google.visualization.PieChart` object with the container element identified by the ID `piechart2`.
    - Draw the chart with the prepared data and options.

2.3.2.5. Section for Investment & Profit

```
<section id="three" class="wrapper spotlight style3">
    <div class="inner">
        <a href="#" class="image2">
            <div id="piechart3" style="width: 300px; height: 300px;"></div>
        </a>
        <div class="content">
            <h2 class="major">Investment & Profit</h2>
            <p>This chart illustrates the relationship between initial investment
and profit over the entire
                duration of the project. If the profit earned does not cover the
initial investment, the chart shows
                the profit earned (calculated as multiplication of generation and
cost) and the net investment
                (initial investment minus profit). If the profit is greater than
the initial investment, the chart
                displays the net profit, which is calculated as profit minus the
initial investment.</p>
            <a href="#" class="special">Learn more</a>
        </div>
    </div>
</section>
```

Description

This HTML section presents a pie chart that visualizes the relationship between initial investment and profit over the entire duration of the project.

Components

1. **Section Container**:
   - **ID**: `three`
   - **Classes**: `wrapper spotlight style3`
2. **Inner Container**:
   - **Class**: `inner`
3. **Image Link**:
   - **Class**: `image2`
   - **Content**: A `div` with an ID `piechart3` and fixed dimensions of 300x300 pixels. This is where the Google Charts pie chart will be rendered.
4. **Content Container**:
   - **Class**: `content`
   - **Header**: `<h2>` element with the class `major` and text "Investment & Profit".
   - **Paragraph**: A description of the pie chart, explaining that the chart illustrates the relationship between initial investment and profit. The paragraph specifies that if the profit earned does not cover the initial investment, it will show both the profit earned and the net investment. If the profit exceeds the initial investment, the chart will display the net profit.
   - **Learn More Link**: An anchor element with the class `special` and text "Learn more".

```
function investmentProfitChart(initialInvestment, profitEarned) {
    var data;
    var colors;

    if (initialInvestment > profitEarned) {
        data = google.visualization.arrayToDataTable([
            ['Task', 'Cost ($)'],
            ['Net Initial Investement', initialInvestment - profitEarned],
            ['Profit Earned', profitEarned]
        ]);
        colors = ['blue', 'darkgreen'];
    } else {
        data = google.visualization.arrayToDataTable([
            ['Task', 'Cost ($)'],
            ['Net Profit Earned', profitEarned - initialInvestment]
        ]);
        colors = ['darkgreen'];
    }

    var options = {
        legend: 'none',
        pieSliceText: 'label',
        backgroundColor: 'transparent',
        chartArea: {
            left: 0,
            top: 0,
```

```
            width: '99%',
            height: '99%'
        },
        fontSize: 12,
        colors: colors
    };

    var chart = new
google.visualization.PieChart(document.getElementById('piechart3'));
    chart.draw(data, options);
}
```

Description

This function creates and renders a pie chart using Google Charts to visualize the relationship between initial investment and profit earned.

Functionality

1. **Variables**:
    - **Data**: The data for the pie chart.
    - **Colors**: The colors for the pie chart slices.
2. **Data and Colors Initialization**:
    - **Condition**: If `initialInvestment` is greater than `profitEarned`, the data array contains:
        - **Net Initial Investment**: Difference between `initialInvestment` and `profitEarned`.
        - **Profit Earned**: Value of `profitEarned`.
        - **Colors**: 'blue' and 'darkgreen'.
    - **Else**: If `profitEarned` is greater than `initialInvestment`, the data array contains:
        - **Net Profit Earned**: Difference between `profitEarned` and `initialInvestment`.
        - **Colors**: 'darkgreen'.
3. **Options**:
    - **Legend**: Set to 'none'.
    - **Pie Slice Text**: Set to 'label'.
    - **Background Color**: Set to 'transparent'.
    - **Chart Area**: Defines the position and size of the chart area.
        - **Left**: 0.
        - **Top**: 0.
        - **Width**: '99%'.
        - **Height**: '99%'.
    - **Font Size**: Set to 12.
    - **Colors**: Uses the colors initialized based on the condition.
4. **Chart Rendering**:
    - **Chart Type**: `google.visualization.PieChart`.
    - **Element**: The pie chart is rendered inside the HTML element with the ID `piechart3`.
    - **Data**: Uses the data array initialized earlier.
    - **Options**: Uses the options defined earlier.

2.3.2.6. Section for Cost Analysis

```
<section id="four" class="wrapper alt style1">
    <div class="inner">
        <section class="features">
            <article>
                <a href="#" class="image">
                    <div id="columnchart" style="width: 100%; height: 300px;">
</div>
                </a>
                <div class="content">
                    <h3 class="major">Cost Analysis</h3>
                    <p>This bar diagram visualizes the breakdown of energy costs
into three key components:
                        Total Consumption Cost: Represented by the red bar, this
shows the overall cost of energy
                        consumed. This bar is the sum of the green and the orange
bar. In scenarios where generation
                        exceeds consumption, such as during the summer, this value
is equal to the generation cost.
                        Generation Cost: The green bar indicates the cost
associated with energy produced by the
                        photovoltaic (PV) system. This segment highlights the
contribution of renewable energy to
                        the overall energy mix. Main Grid Cost: Shown by the
orange bar, this represents the cost of
                        energy sourced from the main grid. This is utilized when
the energy produced by the PV
                        system is insufficient to meet total consumption. This
chart effectively demonstrates the
                        cost dynamics between total energy consumption, PV
generation, and main grid supply,
                        emphasizing the cost efficiency and reliance on different
energy sources.</p>

                    <a href="#" class="special">Learn more</a>
                </div>
            </article>
        </section>
    </div>
</section>
```

Description

This section presents a bar chart that visualizes the breakdown of energy costs, highlighting the relationship between total consumption cost, generation cost, and main grid cost.

Components

1. **Wrapper Section**:
    - **ID**: `four`.

    - **Classes**: `wrapper`, `alt`, `style1`.
2. **Inner Div**:
    - Contains the main content for the section.
3. **Features Section**:
    - Wraps individual articles.
4. **Article**:
    - **Image Link**:
        - **ID**: `columnchart`.
        - **Style**: `width: 100%; height: 300px;`.
    - **Content Div**:
        - **Heading**: `Cost Analysis`.
        - **Description Paragraph**:
            - **Total Consumption Cost**: Represented by the red bar, showing the overall cost of energy consumed.
            - **Generation Cost**: The green bar indicates the cost of energy produced by the photovoltaic (PV) system.
            - **Main Grid Cost**: The orange bar represents the cost of energy sourced from the main grid.
        - **Special Link**: Link to learn more about the cost analysis.

Functionality

The section effectively demonstrates the cost dynamics between total energy consumption, PV generation, and main grid supply, emphasizing the cost efficiency and reliance on different energy sources.

```
function drawColumnChart(consumptionCost, generationCost) {
    if (consumptionCost < generationCost) {
        generationCost = consumptionCost;
    }

    var data = google.visualization.arrayToDataTable([
        ['Type', 'Amount', { role: 'style' }],
        ['Total Consumption Cost', consumptionCost, 'red'],
        ['Generation Cost', generationCost, 'green'],
        ['Main Grid Cost', consumptionCost - generationCost, 'orange']
    ]);

    var options = {
        legend: 'none',
        backgroundColor: 'transparent',
        chartArea: {
            width: '80%',
            height: '80%'
        }
    };

    var chart = new
google.visualization.ColumnChart(document.getElementById('columnchart'));
```

```
    chart.draw(data, options);
}
```

Description This function generates a column chart visualizing the breakdown of energy costs.

Functionality

1. **Data Handling**:
    - **Condition**: Ensures that the generation cost does not exceed the total consumption cost.
    - **Data Population**:
        - Represents the total consumption cost, generation cost, and main grid cost.
        - Colors: Red for total consumption cost, green for generation cost, and orange for main grid cost.
2. **Chart Options**:
    - **Legend**: None
    - **Background Color**: Transparent
    - **Chart Area**: Dimensions for width and height set to 80%
3. **Chart Drawing**:
    - **Column Chart**: Initializes a new `ColumnChart` object targeting the `columnchart` div.
    - **Draw Method**: Renders the chart with the specified data and options.

2.4. **Roof Calculator**

2.4.1. **Backend**

2.4.1.1. Function: `roof_calculator`

```
@main.route("/roof_calculator")
def roof_calculator():
    return flask.render_template(
        "roof_calculator.html", img_width=IMG_WIDTH, img_height=IMG_HEIGHT
    )
```

Route: `/roof_calculator`

Methods: `GET`

Description

The `roof_calculator` function renders the `roof_calculator.html` template when the `/roof_calculator` endpoint is accessed via a GET request.

Functionality Breakdown

1. **Route Definition**:
    - The function is mapped to the `/roof_calculator` URL route, which will trigger the function when accessed.

2. **Rendering Template**:
    - The function uses Flask's `render_template` method to serve the `roof_calculator.html` template.
    - It passes the image dimensions `img_width` and `img_height` as context variables to the template.
    - This allows the web application to present the roof calculator interface defined within the `roof_calculator.html` template file to the user.

2.4.1.2. Function: `get_image`

```
@main.route("/get_image", methods=["POST"])
def get_image():
    data = flask.request.get_json()
    address = data.get("address")
    geocode_result = gmaps.geocode(address)
    if not geocode_result:
        return flask.jsonify({"error": "Place not found"})

    location = geocode_result[0]["geometry"]["location"]
    image_url = generate_static_map_url(location["lat"], location["lng"])

    file_path = os.path.join(DATA_DIR, "{}.json".format(geocode_result[0]
["place_id"]))
    roofs = []
    if os.path.exists(file_path):
        with open(file_path, "r") as f:
            roofs = json.load(f)

    flask.session["place_id"] = geocode_result[0]["place_id"]
    flask.session["address"] = address
    flask.session["sqm_per_pixel"] = (
        156543033.92 * math.cos(location["lat"] * math.pi / 180) / math.pow(2,
ZOOM)
    )

    return flask.jsonify(
        {
            "image_url": image_url,
            "width": IMG_WIDTH,
            "height": IMG_HEIGHT,
            "roofs": roofs,
        }
    )
```

Route: `/get_image`

Methods: `POST`

Description

The `get_image` function retrieves the satellite image of a given address, along with roof data if available. It processes JSON data from a POST request and returns the image URL, image dimensions, and roof data as a JSON response.

Functionality Breakdown

1. **Extracting and Validating Input Data**:
    - **Data Extraction**: Retrieves the JSON data from the request, extracting the address.
    - **Geocoding**: Uses the Google Maps API to geocode the address.
    - **Validation**: Checks if the geocoding result is empty and returns an error if the place is not found.
2. **Generating Image URL**:
    - **Location**: Extracts the latitude and longitude from the geocoding result.
    - **URL Generation**: Calls the `generate_static_map_url` function to create a URL for the satellite image of the given location.
3. **Loading Roof Data**:
    - **File Path**: Constructs the file path for storing roof data based on the place ID.
    - **Roof Data**: Loads roof data from the file if it exists.
4. **Session Variables**:
    - **Place ID**: Stores the place ID in the session.
    - **Address**: Stores the address in the session.
    - **Square Meters per Pixel**: Calculates the square meters per pixel and stores it in the session.
5. **Returning the Results**:
    - Returns a JSON response containing:
        - `image_url`: The URL of the satellite image.
        - `width`: The width of the image.
        - `height`: The height of the image.
        - `roofs`: The roof data if available.

2.4.1.3. Function: `calculate`

```
@main.route("/calculate", methods=["POST"])
def calculate():
    data = flask.request.json
    roofs = data.get("roofs")
    obstacles = data.get("obstacles", [])
    sqm_per_pixel = flask.session.get("sqm_per_pixel")

    if not flask.session.get("place_id"):
        return (
            flask.jsonify({"error": "Missing address"}),
            400,
        )

    if not roofs:
        return (
            flask.jsonify({"error": "Missing roofs data"}),
            400,
        )
```

```
    file_path = os.path.join(DATA_DIR, "
{}.json".format(flask.session["place_id"]))
    estimated_rectangles = []

    for roof in roofs:
        polygon_coords = roof["coordinates"]
        shapely_polygon = shapely.geometry.Polygon(polygon_coords)
        area_sqm = shapely_polygon.area / sqm_per_pixel
        roof["area_sqm"] = area_sqm

        obstacle_polygons = [
            shapely.geometry.Polygon(obstacle["coordinates"]) for obstacle in
obstacles
        ]

        rect_width = 1 * math.sqrt(sqm_per_pixel)  # Define the width of the
rectangles
        rect_height = 1.6 * math.sqrt(
            sqm_per_pixel
        )  # Define the height of the rectangles

        best_rectangles = fill_polygon_with_rectangles(
            shapely_polygon, obstacle_polygons, rect_width, rect_height
        )

        roof_kWp = 0.33 * len(best_rectangles)
        roof["roof_kWp"] = roof_kWp

        estimated_rectangles.extend(
            {
                "coordinates": list(rectangle.exterior.coords),
                "roof_polygon_id": roof["id"],
            }
            for rectangle in best_rectangles
        )

    with open(file_path, "w") as f:
        json.dump(roofs + obstacles, f, indent=2)

    return flask.jsonify({"estimated_rectangles": estimated_rectangles, "roofs":
roofs})
```

Route: `/calculate`

Methods: `POST`

Description

The `calculate` function processes the roof and obstacle data to estimate the potential solar power generation. It calculates the area of each roof, the number of suitable rectangles for solar panels, and returns the results as a JSON response.

Functionality Breakdown

1. **Extracting and Validating Input Data**:
    - **Data Extraction**: Retrieves the JSON data from the request, extracting roofs and obstacles.
    - **Validation**:
        - Checks if the session contains the place ID and square meters per pixel.
        - Returns errors if either is missing or if the roofs data is missing.
2. **Loading and Processing Roof Data**:
    - **File Path**: Constructs the file path for storing roof data based on the place ID.
    - **Initializing Variables**: Initializes a list for estimated rectangles.
3. **Calculating Roof Areas and Solar Potential**:
    - **Polygon Conversion**: Converts roof coordinates to Shapely polygons.
    - **Area Calculation**: Calculates the area of each roof in square meters.
    - **Obstacle Polygons**: Converts obstacle coordinates to Shapely polygons.
4. **Defining Rectangle Dimensions**:
    - **Rectangle Dimensions**: Defines the width and height of the rectangles based on square meters per pixel.
5. **Finding Best Rectangles**:
    - **Best Rectangles**: Calls the `fill_polygon_with_rectangles` function to find the best fitting rectangles within the roof polygons, avoiding obstacles.
    - **Calculating Power**: Calculates the potential power generation for each roof based on the number of rectangles.
6. **Storing and Returning Results**:
    - **Store Results**: Saves the roofs and obstacles data to a JSON file.
    - **Return Results**: Returns a JSON response containing the estimated rectangles and updated roof data.

2.4.1.4. Function: `fill_polygon_with_rectangles`

```
def fill_polygon_with_rectangles(polygon, obstacles, rect_width, rect_height):
    def create_rectangle(x, y, rect_width, rect_height, angle=0):
        rectangle = shapely.geometry.box(x, y, x + rect_width, y + rect_height)
        return shapely.affinity.rotate(rectangle, angle, origin="centroid")

    def rotate_to_edge_angle(polygon):
        edges = []
        coords = list(polygon.exterior.coords)
        for i in range(len(coords) - 1):
            p1, p2 = coords[i], coords[i + 1]
            angle = math.degrees(math.atan2(p2[1] - p1[1], p2[0] - p1[0]))
            edges.append((p1, p2, angle))
        return edges

    best_rectangles = []
    max_rectangle_count = 0

    polygon_edges = rotate_to_edge_angle(polygon)
```

```
    for edge in polygon_edges:
        p1, p2, angle = edge
        aligned_rectangles = []
        placed_rectangles = []

        minx, miny, maxx, maxy = polygon.bounds
        y = miny
        while y < maxy:
            x = minx
            while x < maxx:
                rectangle = create_rectangle(x, y, rect_width, rect_height, angle)
                if (
                    polygon.contains(rectangle)
                    and not any(
                        rectangle.intersects(obstacle) for obstacle in obstacles
                    )
                    and not any(
                        rectangle.intersects(placed) for placed in
placed_rectangles
                    )
                ):
                    aligned_rectangles.append(rectangle)
                    placed_rectangles.append(rectangle)
                x += rect_width
            y += rect_height

        if len(aligned_rectangles) > max_rectangle_count:
            best_rectangles = aligned_rectangles
            max_rectangle_count = len(aligned_rectangles)

    return best_rectangles
```

Description

The `fill_polygon_with_rectangles` function attempts to fit as many rectangles as possible within a given polygon (roof area) while avoiding obstacles. It aligns the rectangles to the edges of the polygon and returns the best fitting configuration.

Parameters:

- `polygon`: The Shapely polygon representing the roof area.
- `obstacles`: A list of Shapely polygons representing obstacles within the roof area.
- `rect_width`: The width of the rectangles to fit within the polygon.
- `rect_height`: The height of the rectangles to fit within the polygon.

Functionality Breakdown

1. **Rectangle Creation**:
    - `create_rectangle`: A nested function that creates a rectangle at given coordinates, optionally rotated by a specified angle.
2. **Edge Angle Calculation**:

- `rotate_to_edge_angle`: A nested function that calculates the angles of the edges of the polygon. This helps in aligning the rectangles along these edges.

3. **Finding the Best Rectangles**:
   - **Iterating Edges**: For each edge of the polygon, calculates the best placement of rectangles aligned to that edge's angle.
   - **Rectangle Placement**: Places rectangles within the polygon bounds, ensuring they do not intersect with obstacles or other placed rectangles.
   - **Maximizing Fit**: Keeps track of the configuration with the maximum number of rectangles that fit within the polygon.
4. **Returning Results**:
   - Returns the configuration of rectangles that fits the most within the polygon without intersecting obstacles.

2.4.2. **Frontend**

2.4.2.1. Instructions Section

```
<h3>Instructions</h3>
<ol class="no-gap-list">
    <li>Enter the address or the latitude and longitude of the location you
        want to calculate the roof for.</li>
    <li>
        Click on "Get Sallite Image" button.
    </li>
    <li>To select a roof section, follow these steps:
        <ol>
            <li>Click on the <img class="instructions-image"
                    src="{{ url_for('static', filename='images/polygon-
marker.png') }}"
                    alt> icon
                located in the top left corner of the map.</li>
            <li>Click the corners of the desired roof section</li>
        </ol>
    </li>
    <li>
        (Optional) To delete a roof section, follow these steps:
        <ol>
            <li>Click on the <img class="instructions-image"
                    src="{{ url_for('static', filename='images/trash-marker.png')
}}"
                    alt> icon
                located in the top left corner of the map.</li>
            <li>Click on the roof section you want to delete.</li>
            <li>Click "Save".</li>
        </li>
    </ol>
    <li>Fill in the details for the roof section in the table
        below.</li>
    <li>Click on "Calculate" button to calculate the solar power potential,
```

```
        and wait for the solar panels to appear in black on the drawn roof
        areas.</li>
</div>
```

Description

This HTML block provides step-by-step instructions for users to calculate the roof area and solar power potential.

Components

1. **Instructions Header**:
    - `<h3>`: Displays the header "Instructions".
2. **Ordered List**:
    - `<ol>`: Contains the main steps for the instructions.
    - `class="no-gap-list"`: Removes gaps between list items.
3. **List Items**:
    - **Step 1**: Enter the address or latitude and longitude.
    - **Step 2**: Click the "Get Satellite Image" button.
    - **Step 3**: Select a roof section by:
        - Clicking on the polygon marker icon.
        - Clicking the corners of the desired roof section.
    - **Step 4 (Optional)**: Delete a roof section by:
        - Clicking on the trash marker icon.
        - Clicking on the roof section to delete.
        - Clicking "Save".
    - **Step 5**: Fill in the details for the roof section in the table below.
    - **Step 6**: Click the "Calculate" button to calculate the solar power potential and wait for the solar panels to appear in black on the drawn roof areas.
4. **Images**:
    - **Polygon Marker Icon**: Displayed as part of the instructions for selecting a roof section.
    - **Trash Marker Icon**: Displayed as part of the instructions for deleting a roof section.

2.4.2.2. Address Input and Satellite Image Request

```
<div class="row">
    <div class="col-6 col-12">
        <ul class="actions fit">
            <li>
                <div class="col-6 col-12">
                    <h3 for="address">Address</h3>
                    <input type="text" name="address" id="address"
                        placeholder="Enter Address">
                </div>
            </li>
        </ul>
        <ul class="col-6 actions">
```

```
            <li><button id="get-image" class="button">Get Satellite
                    Image</button></li>
        </ul>
    </div>
</div>
```

Description

This section provides an input field for the user to enter an address and a button to fetch the satellite image for the entered address.

Components

1. **Row Div**:
    - Container div with classes `row`, `col-6`, and `col-12`.
2. **Actions List**:
    - **First List Item**:
        - **Address Label**:
            - `<h3>` element for the address label.
        - **Address Input**:
            - Input field with `type="text"`, `name="address"`, `id="address"`, and a placeholder "Enter Address".
    - **Second List Item**:
        - **Button**:
            - Button element with `id="get-image"` and class `button`.
            - Button text: "Get Satellite Image".

```
document.getElementById('get-image').addEventListener('click', function () {
    const address = document.getElementById('address').value;
    fetch('/get_image', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({ "address": address }),
    })
        .then(response => response.json())
        .then(data => {
            if (data.error) {
                document.getElementById('error-message').style.display = 'block';
                document.getElementById('map-container').style.display = 'none';
                document.getElementById('polygon-list-container').style.display =
'none';
                document.getElementById('polygon-list').innerHTML = '';
            } else {
                document.getElementById('error-message').style.display = 'none';
```

```
                document.getElementById('map-container').style.display = 'block';
                initializeMap(data.image_url, data.width, data.height);
                if (data.roofs && data.roofs.length > 0) {
                    document.getElementById('polygon-list-
container').style.display = 'block';
                    data.roofs.forEach(polygon => {
                        addPolygonToMap(polygon);
                    });
                }
            }
        });
});
```

Description

This script adds an event listener to the "Get Satellite Image" button. When clicked, it fetches the satellite image for the entered address and displays it on the map.

Components

1. **Event Listener**:
   - Adds a click event listener to the button with `id="get-image"`.
2. **Fetch Address**:
   - Retrieves the value from the input field with `id="address"`.
3. **Fetch Request**:
   - Sends a POST request to the `/get_image` endpoint with the address as JSON payload.
4. **Response Handling**:
   - **Error Handling**:
     - If the response contains an error, displays the error message and hides the map and polygon list containers.
     - Clears the polygon list.
   - **Success Handling**:
     - If the response is successful, hides the error message and displays the map container.
     - Calls the `initializeMap` function with the received image URL, width, and height.
     - If there are roofs in the response, displays the polygon list container and adds each polygon to the map using the `addPolygonToMap` function.

2.4.2.3. Initializing and Managing the Map

```
<div id="map-container">
    <div id="map" style="width: {{ img_width }}px; height: {{ img_height }}px;">
</div>
</div>
```

Description

This section describes the button element used to trigger calculations in the application.

Components

1. **Button Element**:
    - **ID**: Sets the unique identifier for the button as "calculate".
    - **Class**: Applies the "button" CSS class for styling.
    - **Text**: Displays the text "Calculate" on the button.

```
let map;
let drawnItems;

function initializeMap(imageUrl, imgWidth, imgHeight) {
    if (map) {
        map.remove();
    }

    const bounds = [[0, 0], [imgHeight, imgWidth]];

    map = L.map('map', {
        center: [imgHeight / 2, imgWidth / 2],
        zoom: 0,
        crs: L.CRS.Simple,
        zoomControl: true,
        scrollWheelZoom: true,
        doubleClickZoom: true,
        boxZoom: true,
        keyboard: true,
        maxBounds: bounds,
        maxBoundsViscosity: 1.0
    });

    L.imageOverlay(imageUrl, bounds).addTo(map);
    map.fitBounds(bounds);

    if (drawnItems) {
        drawnItems.clearLayers();
    }

    drawnItems = new L.FeatureGroup();
    map.addLayer(drawnItems);

    const drawControl = new L.Control.Draw({
        draw: {
            polygon: true,
            rectangle: true,
            circle: false,
            marker: false,
            circlemarker: false,
        },
        edit: {
            featureGroup: drawnItems
        }
```

```
    });
    map.addControl(drawControl);

    document.getElementById('polygon-list').innerHTML = '';

    map.on('draw:created', function (e) {
        const layer = e.layer;
        const polygonId = generateUniquePolygonId();

        drawnItems.addLayer(layer);
        addPolygonItem(polygonId, layer);
    });

    map.on('draw:deleted', function (e) {
        e.layers.eachLayer(function (layer) {
            const polygonId = layer._leaflet_id;
            const row = document.querySelector(`tr[data-layer-
id='${polygonId}']`);
            if (row) {
                row.remove();
            }
        });
        togglePolygonListContainer();
    });
}
```

Description

This section describes the functionality to initialize a map, draw polygons, and handle polygon events.

Components

1. **Global Variables**:
   - `map`: Holds the map instance.
   - `drawnItems`: Holds the drawn items on the map.
2. **Function**: `initializeMap`:
   - **Parameters**:
     - `imageUrl`: URL of the image to be displayed on the map.
     - `imgWidth`: Width of the image.
     - `imgHeight`: Height of the image.
   - **Functionality**
     - Removes the existing map if it exists.
     - Sets the bounds for the map based on image dimensions.
     - Initializes the map with specified settings (center, zoom, CRS, controls).
     - Adds the image overlay to the map.
     - Clears any existing drawn items and initializes a new feature group for drawn items.
     - Adds drawing controls to the map, enabling polygon and rectangle drawing.
     - Clears the polygon list in the HTML.
3. **Event Listener**: `draw:created`:
   - Triggered when a new polygon is created.

- Adds the new layer to the map and associates it with a unique polygon ID.
- Calls `addPolygonItem` to handle the new polygon.

4. **Event Listener**: `draw:deleted`:
   - Triggered when a polygon is deleted.
   - Removes the corresponding row in the polygon list for each deleted layer.
   - Calls `togglePolygonListContainer` to update the visibility of the polygon list container.

2.4.2.3.1. Add Polygon to Map

```
function addPolygonToMap(polygon) {
    const latlngs = polygon.coordinates.map(coord => [coord[0], coord[1]]);
    const layer = L.polygon(latlngs).addTo(drawnItems);
    const polygonId = generateUniquePolygonId();
    addPolygonItem(polygonId, layer);

    // Set additional properties (name, roofType, tilt, azimuth)
    const polygonItem = document.getElementById(polygonId);
    polygonItem.querySelector('input[type="text"]').value = polygon.name;
    polygonItem.querySelector('select.roof-type').value = polygon.roofType;
    polygonItem.querySelector('select.tilt').value = polygon.tilt;
    polygonItem.querySelector('select.azimuth').value = polygon.azimuth;
    polygonItem.querySelector('select.feature-type').value = polygon.featureType;

    document.getElementById('polygon-list-container').style.display = 'block';
}
```

Function: `addPolygonToMap(polygon)`

Description This function adds a given polygon to the map and updates the polygon list with its properties.

Functionality

1. **Coordinate Mapping**:
   - Converts polygon coordinates to a format suitable for Leaflet.
2. **Layer Creation**:
   - Creates a Leaflet polygon layer using the mapped coordinates and adds it to the `drawnItems` layer group.
3. **Unique ID Generation**:
   - Generates a unique ID for the polygon.
4. **Polygon Item Addition**:
   - Calls a function to add the polygon item to the list using the unique ID and the created layer.
5. **Set Polygon Properties**:
   - Sets additional properties for the polygon, including name, roof type, tilt, azimuth, and feature type.
6. **Update Polygon List Visibility**:
   - Ensures the polygon list container is displayed.

2.4.2.3.2. Generate Unique ID

```
let polygonCount = 0;

function generateUniquePolygonId() {
    const timestamp = new Date().getTime();
    return `polygon-${timestamp}-${polygonCount++}`;
}
```

Variables:

- `polygonCount`: Initializes a counter to keep track of the number of polygons created.

Function: `generateUniquePolygonId()`

Description Generates a unique identifier for each polygon.

Functionality

1. **Timestamp Generation**:
    - Retrieves the current timestamp.
2. **Unique ID Formation**:
    - Combines the timestamp and the polygon count to form a unique ID.
3. **Increment Counter**:
    - Increments the polygon count for the next ID generation.

2.4.2.3.3. Add Polygon to Polygon List Container

```
function addPolygonItem(polygonId, layer) {
    const polygonList = document.getElementById('polygon-list');
    const row = document.createElement('tr');
    row.id = polygonId;

    // Set a custom data attribute to link the HTML element with the Leaflet layer
    row.setAttribute('data-layer-id', layer._leaflet_id);

    const visibilityCell = document.createElement('td');
    const visibilityCheckboxContainer = document.createElement('div');
    const visibilityCheckbox = document.createElement('input');
    visibilityCheckbox.type = 'checkbox';
    visibilityCheckbox.id = `visibility-${polygonId}`;
    visibilityCheckbox.name = `visibility-${polygonId}`;
    visibilityCheckbox.checked = true;
    visibilityCheckbox.onchange = function () {
        if (this.checked) {
            map.addLayer(layer);
        } else {
            map.removeLayer(layer);
```

```
        }
    };
    const visibilityLabel = document.createElement('label');
    visibilityLabel.setAttribute('for', `visibility-${polygonId}`);
    visibilityLabel.className = 'visibility-checkbox-label';
    visibilityCheckboxContainer.appendChild(visibilityCheckbox);
    visibilityCheckboxContainer.appendChild(visibilityLabel);
    visibilityCell.appendChild(visibilityCheckboxContainer);

    const nameCell = document.createElement('td');
    const input = document.createElement('input');
    input.type = 'text';
    input.value = polygonId;
    input.onchange = function () {
        const existingTooltip = layer.getTooltip() ?
layer.getTooltip().getContent().split('<br>')[1] : "";
        layer.bindTooltip("Name: " + this.value + "<br>" +
existingTooltip).openTooltip();
    };
    nameCell.appendChild(input);

    const featureTypeCell = document.createElement('td');
    const featureTypeSelect = document.createElement('select');
    featureTypeSelect.className = 'feature-type';
    const roofOption = document.createElement('option');
    roofOption.value = 'roof';
    roofOption.text = 'Roof';
    featureTypeSelect.appendChild(roofOption);
    const obstacleOption = document.createElement('option');
    obstacleOption.value = 'obstacle';
    obstacleOption.text = 'Obstacle';
    featureTypeSelect.appendChild(obstacleOption);
    featureTypeCell.appendChild(featureTypeSelect);

    const roofTypeCell = document.createElement('td');
    const roofTypeSelect = document.createElement('select');
    roofTypeSelect.className = 'roof-type';
    const flatOption = document.createElement('option');
    flatOption.value = 'flat';
    flatOption.text = 'Flat';
    roofTypeSelect.appendChild(flatOption);
    const tiltOption = document.createElement('option');
    tiltOption.value = 'tilt';
    tiltOption.text = 'Tilt';
    roofTypeSelect.appendChild(tiltOption);
    roofTypeCell.appendChild(roofTypeSelect);

    const tiltCell = document.createElement('td');
    const tiltSelect = document.createElement('select');
    tiltSelect.className = 'tilt';
    for (let i = 0; i <= 50; i++) {
        const option = document.createElement('option');
        option.value = i;
        option.text = i;
```

```
            if (i === 30) {
                option.selected = true;
            }
            tiltSelect.appendChild(option);
        }
        tiltCell.appendChild(tiltSelect);

        const azimuthCell = document.createElement('td');
        const azimuthSelect = document.createElement('select');
        azimuthSelect.className = 'azimuth';
        for (let i = 0; i < 360; i++) {
            const option = document.createElement('option');
            option.value = i;
            option.text = i;
            azimuthSelect.appendChild(option);
        }
        azimuthCell.appendChild(azimuthSelect);

        row.appendChild(visibilityCell);
        row.appendChild(nameCell);
        row.appendChild(featureTypeCell);
        row.appendChild(roofTypeCell);
        row.appendChild(tiltCell);
        row.appendChild(azimuthCell);

        polygonList.appendChild(row);

        // Show the polygon list container
        document.getElementById('polygon-list-container').style.display = 'block';

        // Initial tooltip with just the name
        layer.bindTooltip("Name: " + polygonId).openTooltip();
    }
```

Function: `addPolygonItem(polygonId, layer)`

Description Adds a polygon item to the polygon list in the UI and links it with the corresponding Leaflet layer.

Functionality

1. **Element Creation and Initialization**:
    - Creates a table row element to represent the polygon.
    - Sets a custom data attribute to link the HTML element with the Leaflet layer.
2. **Visibility Checkbox**:
    - Creates a checkbox to control the visibility of the polygon.
    - Attaches an event listener to the checkbox to show/hide the polygon on the map.
3. **Name Input**:
    - Creates an input field for the polygon name.
    - Attaches an event listener to update the polygon's tooltip with the new name.
4. **Feature Type Selector**:
    - Creates a dropdown to select the feature type (roof or obstacle).

5. **Roof Type Selector**:
    - Creates a dropdown to select the roof type (flat or tilt).
6. **Tilt Selector**:
    - Creates a dropdown to select the tilt value (0-50).
7. **Azimuth Selector**:
    - Creates a dropdown to select the azimuth value (0-360).
8. **Appending Elements**:
    - Appends the created elements to the table row.
    - Appends the row to the polygon list in the UI.
9. **Display Update**:
    - Shows the polygon list container.
    - Sets an initial tooltip for the polygon with its name.

2.4.2.4. Polygon List Container

```
<div id="polygon-list-container" style="display: none;">
    <h3>Selected Roof Areas</h3>
    <div class="table-wrapper">
        <table class="alt" id="polygon-table">
            <thead>
                <tr>
                    <th>Visibility</th>
                    <th>Name</th>
                    <th>Feature Type</th>
                    <th>Roof Type</th>
                    <th>Tilt</th>
                    <th>Azimuth</th>
                </tr>
            </thead>
            <tbody id="polygon-list">
                <!-- Polygon items will be added here -->
            </tbody>
        </table>
    </div>
</div>
```

Description

This section describes the HTML structure for displaying a list of selected roof areas.

Components

1. **Container**:
    - **Element**: `<div id="polygon-list-container" style="display: none;">`
        - **Description** Container for the selected roof areas, initially hidden.
2. **Heading**:
    - **Element**: `<h3>Selected Roof Areas</h3>`
        - **Description** Heading for the selected roof areas section.

3. **Table Wrapper**:
    - **Element**: `<div class="table-wrapper">`
        - **Description** Wrapper for the table to ensure proper styling and scrolling.
4. **Table**:
    - **Element**: `<table class="alt" id="polygon-table">`
        - **Description** Table element with class `alt` and ID `polygon-table` for listing selected roof areas.
5. **Table Header**:
    - **Element**: `<thead>`
        - **Description** Table header containing column titles.
        - **Columns**:
            - **Visibility**: Column for visibility checkbox.
            - **Name**: Column for the name of the polygon.
            - **Feature Type**: Column for selecting feature type (e.g., roof, obstacle).
            - **Roof Type**: Column for selecting roof type (e.g., flat, tilt).
            - **Tilt**: Column for selecting tilt angle.
            - **Azimuth**: Column for selecting azimuth angle.
6. **Table Body**:
    - **Element**: `<tbody id="polygon-list">`
        - **Description** Table body where polygon items will be dynamically added.
        - **Comment**: Placeholder comment indicating where polygon items will be inserted.

```
function togglePolygonListContainer() {
    const polygonList = document.getElementById('polygon-list');
    const polygonListContainer = document.getElementById('polygon-list-
container');
    if (polygonList.children.length === 0) {
        polygonListContainer.style.display = 'none';
    } else {
        polygonListContainer.style.display = 'block';
    }
}
```

Description

This function toggles the visibility of the polygon list container based on the presence of children in the polygon list.

Components

1. **Function Declaration**:
    - **Function Name**: `togglePolygonListContainer`
        - **Purpose**: To show or hide the polygon list container based on the number of children in the polygon list.
2. **DOM Elements**:
    - **polygonList**:

        - **Method**: `document.getElementById('polygon-list')`
        - **Description** Retrieves the element with the ID `polygon-list`, which contains the list of polygons.
    - **polygonListContainer**:
        - **Method**: `document.getElementById('polygon-list-container')`
        - **Description** Retrieves the element with the ID `polygon-list-container`, which contains the entire polygon list section.
3. **Condition Check**:
    - **Condition**: `if (polygonList.children.length === 0)`
        - **Description** Checks if the polygon list has no children.
        - **Action**: If true, sets the display style of `polygonListContainer` to `'none'`, hiding it.
        - **Else Action**: Otherwise, sets the display style of `polygonListContainer` to `'block'`, showing it.

2.4.2.5. Calculate Button

```
<button id="calculate" class="button">Calculate</button>
```

HTML Structure: Calculate Button

Description This section defines the HTML structure for the calculate button which initiates the calculation process.

Components

- **Button Container**:
    - `<ul class="actions fir">`: A list container for actions.
    - `<li>`: A list item containing the calculate button.
        - `<button id="calculate" class="button">Calculate</button>`: The button element with an ID of `calculate` and a class of `button`.

```
document.getElementById('calculate').addEventListener('click', function () {
    const address = document.getElementById('address').value;
    if (!address) {
        alert("Please enter an address first.");
        return;
    }

    const roofs = [];
    const obstacles = [];
    const roofs_to_leaflet_ids = new Object();

    document.querySelectorAll('#polygon-list tr').forEach(item => {
        const polygonId = item.id;
        const name = item.querySelector('input[type="text"]').value;
        const roofType = item.querySelector('select.roof-type').value;
```

```
        const tilt = item.querySelector('select.tilt').value;
        const azimuth = item.querySelector('select.azimuth').value;
        const featureType = item.querySelector('select.feature-type').value;

        const layerId = parseInt(item.getAttribute('data-layer-id'), 10);
        const layer = drawnItems.getLayers().find(layer => layer._leaflet_id ===
layerId);
        if (layer && layer._latlngs) {
            const coordinates = layer.getLatLngs()[0].map(latlng => [latlng.lat,
latlng.lng]);
            roofs_to_leaflet_ids[polygonId] = layerId;

            if (featureType === 'roof') {
                roofs.push({
                    id: polygonId,
                    name,
                    roofType,
                    tilt,
                    azimuth,
                    featureType,
                    coordinates
                });
            } else {
                obstacles.push({
                    id: polygonId,
                    name,
                    roofType,
                    tilt,
                    azimuth,
                    featureType,
                    coordinates
                });
            }
        }
    });

    fetch('/calculate', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({
            address,
            roofs,
            obstacles
        })
    })
        .then(response => response.json())
        .then(data => {
            if (data.error) {
                alert("Error: " + data.error);
            } else {
                // Clear existing rectangles
                drawnItems.eachLayer(function (layer) {
```

```
                    if (layer.options.color === 'black') {
                        drawnItems.removeLayer(layer);
                    }
                });

                // Add estimated rectangles to the map
                data.estimated_rectangles.forEach(rectangleData => {
                    const rectangle = L.polygon(rectangleData.coordinates, {
color: 'black' });
                    drawnItems.addLayer(rectangle);
                });

                // Update tooltips with area and roof_kWp for each roof polygon
                data.roofs.forEach(polygon => {
                    console.log(drawnItems.getLayers())
                    const layer = drawnItems.getLayers().find(layer =>
layer._leaflet_id === parseInt(roofs_to_leaflet_ids[polygon.id], 10));
                    const area = polygon.area_sqm;
                    const roof_kWp = polygon.roof_kWp;
                    layer.bindTooltip("Name: " + polygon.name + "<br>Area: " +
area.toFixed(2) + " sqm<br>Power: " + roof_kWp.toFixed(2) + " kWp").openTooltip();
                });
            }
        });
});
```

Function: `document.getElementById('calculate').addEventListener('click', function () { ... });`

Description Handles the click event for the calculate button, collects roof and obstacle data, sends it to the server for calculation, and updates the map with the results.

Functionality

1. **Event Listener**:
    - **Calculate Button**: Adds a click event listener to the calculate button.
2. **Address Validation**:
    - **Validation**: Checks if the address input is not empty. If empty, shows an alert and returns.
3. **Data Collection**:
    - **Initialization**: Initializes arrays for roofs and obstacles, and an object to map roof IDs to Leaflet layer IDs.
    - **Polygon List Iteration**: Iterates over each row in the polygon list to collect data.
        - **Data Extraction**: Extracts polygon data (ID, name, roof type, tilt, azimuth, feature type, coordinates) and categorizes it as either a roof or an obstacle.
4. **API Request**:
    - **Request**: Sends a POST request to the `/calculate` endpoint with the address, roofs, and obstacles data in JSON format.
    - **Response Handling**:
        - **Error Handling**: Displays an alert if there's an error.
        - **Success Handling**:

- Clears existing black rectangles from the map.
- Adds new estimated rectangles to the map.
- Updates tooltips for each roof polygon with area and roof kWp information.

5. **Tooltip Update**:
    - **Loop Through Data**: Iterates over the returned roof data to update tooltips with area and power information.

---

3. **Step-by-step instructions to re-create your prototype (e.g. see project descriptions at Hackster.io)**

1. Install Anaconda
    - Follow the instructions on the Anaconda website to install Anaconda on your system.
2. Create Anaconda environment and activate it
    - Open a terminal or command prompt and run the following commands:

```
conda create -n tc-energy python=3.11
conda activate tc-energy
```

3. Install Requirements:
    - Ensure you are in your project's directory, then run:

```
pip install -r requirements.txt
```

4. Get Google Maps API key
    - Follow the instructions on the Google Developers website to obtain your API key.
5. Set Google Maps API key as environment variable
    - Local:
        - Operating System
            - Linux / MacOS:

```
echo "export GOOGLE_API_KEY='your_google_api_key'" >>
~/.zshrc
source ~/.zshrc
echo $GOOGLE_API_KEY
```

            - Windows:

```
setx GOOGLE_API_KEY "your_google_api_key"
echo %GOOGLE_API_KEY%
```

        - Uncomment `app.run(debug=True)` in `main.py`
        - Run `main.py`
    - Online Deployment:
        - Open `app-temp.yaml`
        - Modify the following line:

```
GOOGLE_API_KEY: 'your_google_api_key'
```

- Rename the file to `app.yaml`
- Uncomment `app.run(host="0.0.0.0", port=8080)` in `main.py`
- Use Google Cloud web hosting for online deployment.

---

4. **Link to an online code repository (e.g. GitHub, GitLab, BitBucket) is mandatory**

Code: https://github.com/Maqarios/TC-Energy-Communities

Online Deployment: https://energy-422810.ey.r.appspot.com/