Software Design Specifications

for
Carbon Footprint Calculator - Version 1.0

Prepared by: Dhanush Reddy - se22ucse216, Sohith Paturi - se22ucse325, Sai Charan Reddy - se22ucse159, Ala Madhu Sudhan Reddy - se22ucse020, Durga Rushi Masagoni - se22ucse088, Sahith Potu - se22ucse213.

Date: April 9, 2025

Document Information

Title: Carbon Footprint Calculator - Software Design Document

Project Manager: Narthkannai Ma'am

Document Version No: 1.0

Document Version Date: April 9, 2025

Prepared By: Dhanush Reddy - se22ucse216, Sohith Paturi - se22ucse325, Sai Charan Reddy - se22ucse159, Ala Madhu Sudhan Reddy - se22ucse020, Durga Rushi Masagoni - se22ucse088, Sahith Potu - se22ucse213.

Preparation Date: April 9, 2025

Version History

Ver. No	Ver. Date	Revised By	Description	Filename
1.0	April 9, 2025	Team CFC	Initial version	SDD_Carbon_Footprint_Calculator.docx

Table of Contents

This document includes the following sections:

1 Introduction

The Carbon Footprint Calculator is a software application developed using Python and Flask. Its main purpose is to help users estimate their carbon emissions based on their daily activities such as travel, energy usage, food habits, and waste generation. The tool provides an easy way for users to understand their environmental impact and get suggestions on how to reduce it.

This Software Design Document (SDD) explains the overall design of the system, including its structure, components, data flow, and user interaction. It serves as a guide for developers, testers, and professors to understand how the system works and how it meets the project requirements.

The focus of the design is to keep the system simple, accurate, and user-friendly, while also promoting awareness about sustainable living.

1.1 Purpose

The purpose of this Software Design Document is to describe the architecture, components, and design decisions behind the Carbon Footprint Calculator. This document provides a clear understanding of how the system is structured and how it fulfills the required functionalities such as carbon emission estimation, user input handling, and suggestion generation.

Intended Audience

This document is intended for:

Developers

Testers

Project Mentors/Faculty

Future Contributors

1.2 Scope

The Carbon Footprint Calculator is a Python-Flask-based application designed to help users estimate their carbon emissions based on inputs like transportation habits, energy usage, food consumption, and waste generation. The application calculates the total emissions and provides simple, personalized suggestions to reduce them.

This document covers the software design of the system, including use cases, logical structure, data flow, and user interaction. It defines how the system components work together to meet the requirements, and serves as a foundation for development, testing, and future enhancements.

Goals and Objectives

- To provide an easy-to-use platform for individuals to calculate their carbon footprint.
- To promote environmental awareness and encourage more sustainable lifestyle choices.
- To offer clear, personalized recommendations for reducing carbon emissions.
- To track and display carbon output in an informative and visual format.

Benefits

- Environmental Awareness
- Personalized Feedback
- User-Friendly Interface
- Data-Driven Insights

Overview

This Software Design Document provides a detailed description of the design and structure of the Carbon Footprint Calculator. It includes system architecture, key modules, data flow, use cases, and interface designs. The document ensures that all stakeholders have a clear understanding of how the system is built and how it functions, supporting consistent development, testing, and maintenance efforts.

1.3 Definitions, Acronyms, and Abbreviations

CFC - Carbon Footprint Calculator
CO2e - Carbon Dioxide Equivalent
API - Application Programming Interface
UI - User Interface

1.4 References

1. Statement of Work – Carbon Footprint Calculator
2. IPCC Emission Factor Guidelines
3. IEEE 1016-2009

2 Use Case View

The use case diagram illustrates the primary interactions between the system and its actors: User, Admin, and Database. It defines the major functionalities provided by the Carbon Footprint Calculator and how each actor engages with the system. The diagram helps visualize user roles, system boundaries, and core operations supported by the application.

Actors

User: A general user who uses the system to calculate and track their carbon footprint.

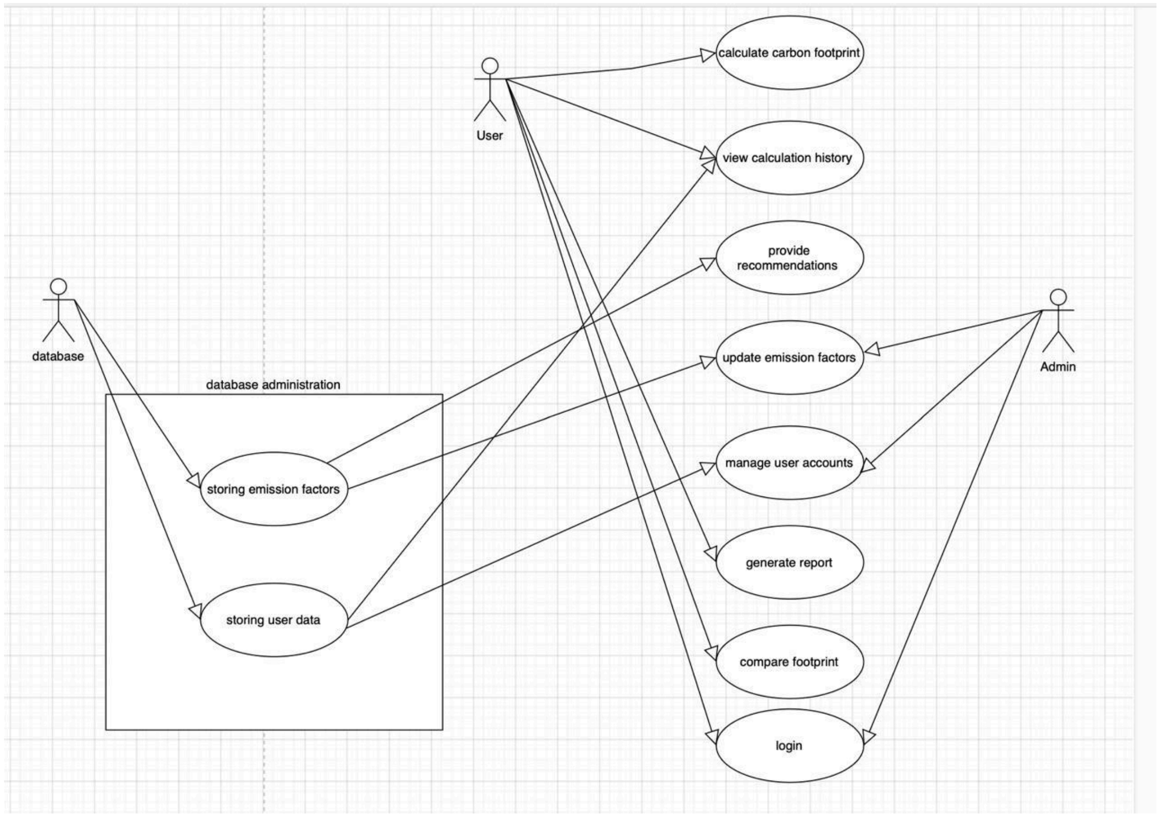
Admin: Responsible for managing system data, emission factors, and user accounts.

Database: Represents the data storage system for user inputs, calculation history, and emission factor data.

Key Use Cases

Use Case	Description
Calculate Carbon Footprint	Users input daily activity data to get their estimated carbon emissions.
View Calculation History	Users can view their previous footprint calculation records.
Provide Recommendations	System suggests personalized tips to reduce carbon footprint.
Update Emission Factors	Admin updates values used for calculating emissions (e.g., for electricity, travel).
Manage User Accounts	Admin can add, update, or remove user accounts.
Generate Report	Users or Admin can generate summary reports of carbon emissions.
Compare Footprint	Users can compare their emissions over time or with averages.

Use Case	Description
Login	Secure access point for both Users and Admins.
Storing Emission Factors	The system stores updated emission data in the database.
Storing User Data	User inputs and histories are saved to the database for future use.



3 Design Overview

The Carbon Footprint Calculator is designed as a modular, Python-based application using the Flask framework. It follows a **three-layer architecture** that includes the presentation layer (frontend), application logic (Flask routes and services), and data storage (SQLite/PostgreSQL database).

The primary goal of the system is to provide an easy-to-use platform for users to input daily activity data and receive accurate carbon footprint estimates. The system also offers personalized recommendations and visual feedback to encourage eco-friendly habits.

The system is designed with simplicity and clarity in mind, ensuring smooth user interaction and easy extensibility. Flask handles routing and logic, while HTML templates with embedded Python render user interfaces.

3.1 Design Goals and Constraints

Design Goals

Simplicity and Usability
The application should be easy to use, with a clean and minimal user interface that guides users step-by-step through data input and result interpretation.

Accuracy of Calculations
Carbon footprint estimates must be based on reliable and up-to-date emission factor datasets (e.g., from IPCC or government sources).

Modular and Maintainable Code
The design should follow modular principles, enabling easy debugging, maintenance, and potential future upgrades (e.g., more categories, advanced analytics).

Responsiveness and Feedback
The system should provide immediate feedback to users after data submission, including a breakdown of emissions and clear suggestions.

Data Persistence
Users' activity history and previous results must be saved in the database for progress tracking and comparisons.

Design Constraints

Limited Development Time
The project is bound by academic deadlines, which limits the time available for feature expansion or advanced UI features.

Third-party Dataset Dependence
The accuracy of emission calculations depends on externally sourced emission factor data, which may change over time.

Basic UI/UX Scope
Due to the use of Flask templates, the front-end is kept simple, without frameworks like React or Angular for dynamic interactivity.

3.2 Design Assumptions

Design Assumptions

User Accuracy
It is assumed that users will provide honest and reasonably accurate data regarding their daily habits (e.g., kilometers traveled, electricity usage, dietary preferences, etc.).

Stable Dataset Availability
Emission factor datasets used for calculations are assumed to be publicly available, stable, and reliable during the project timeline.

Single User Access at a Time
The system is designed for single or limited user access at a time; concurrency and session management are kept minimal due to this scope.

Modern Web Browser Usage
It is assumed users will access the application via modern browsers (like Chrome or Firefox) that support standard HTML, CSS, and basic JavaScript.

No External API Rate Limits
If external APIs are used (e.g., for emission factor updates), it is assumed there won't be strict rate limits affecting the application's performance.

No Advanced Security Needs
The application does not handle sensitive or financial data, so it assumes that basic form-based login and password hashing are sufficient for the academic scope.

3.3 Significant Design Packages

Frontend

home.html – Landing page that allows users to navigate to input forms or view past records.

dashboard.html – Displays forms to input user data (e.g., transportation, energy, diet, waste).

result.html – Shows carbon footprint results with category-wise breakdowns and suggestions.

history.html – Visual representation of past carbon entries using basic graphs/charts.

Backend

app.py – Main entry point for the Flask application; initializes the app and configures routes.

routes.py – Defines endpoints for handling user actions like input submission, calculation, login, and history viewing.

calculator.py – Contains all the emission calculation logic using predefined formulas and emission factors.

recommendation_engine.py – Suggests personalized tips based on user's lifestyle and emission data.

auth.py – Manages basic user authentication for login and registration processes.

Database (Data Layer)

models.py – Defines database schemas using Flask SQLAlchemy (or raw SQL if preferred):

Stores user account details.

Stores activity data and CO_2e results.

Contains category-wise emission values used in calculations.

3.4 Dependent External Interfaces

External Interface Name	Module Using the Interface	Description and Usage
Emission Factor Dataset	calculator.py	Used to fetch or reference standard CO_2e values for transportation, energy, diet, and waste calculations.
Chart.js / Matplotlib (optional)	result.html / dashboard.html	Used to generate visual graphs representing users' carbon output by category or over time.
Flask Framework	Entire Backend (app.py, routes.py)	Core framework used to manage routing, rendering templates, and handling user requests.
Browser Environment	Frontend (HTML/CSS/JS)	Assumes standard browser support for rendering dynamic templates and interactive charts.

3.5 Implemented Application External Interfaces

Interface Name	Module Implementing the Interface	Functionality / Description
calculate	routes.py / calculator.py	Accepts user input and returns calculated carbon footprint based on emission factors.
recommendations	routes.py / recommendation_engine.py	Provides personalized suggestions based on the user’s emission profile.

Interface Name	Module Implementing the Interface	Functionality / Description
history	routes.py	Retrieves and displays the user’s previous calculation records from the database.
register & /login	auth.py	Handles user authentication including registration and login using hashed passwords.
admin/update-factors	routes.py (Admin section)	Allows admin to update emission factor values for calculations.

4 Logical View

Frontend

- Function: Collects input from users, displays calculated results, and presents recommendations and historical data.
- Technologies: HTML, CSS, optional Chart.js for visualizations.
- Key Components:
  - home.html, dashboard.html, result.html, history.html
  - User forms for inputting transport, energy, diet, and waste information

Backend

- Function: Handles business logic, user authentication, and coordination between input/output and the database.
- Technologies: Flask (Python), WTForms (optional), session handling
- Key Modules:
  - routes.py – Manages all user-facing and admin-facing endpoints
  - calculator.py – Contains logic to compute carbon footprint from input data
  - recommendation_engine.py – Determines and returns relevant sustainability tips
  - auth.py – Handles login, registration, and basic session management

Database

- Function: Stores all user data, input logs, carbon results, and emission factors.
- Technologies: SQLite or PostgreSQL, SQLAlchemy ORM (if used)

- Key Components:
    - models.py – Defines database tables and relationships
        - User, CarbonEntry, EmissionFactor, Recommendation
    - database.py – Initializes the database and manages connections

4.1 Design Model

User

- Attributes:
    - username: String – The unique identifier for the user.
    - email: String – Contact email of the user.
    - password: String – Encrypted password for login.
- Methods:
    - register() – Handles new user registration.
    - login() – Authenticates user access.
    - inputData() – Allows users to submit their lifestyle and activity data.

UserData

- Attributes:
    - transportation: float – Emissions from commuting and travel.
    - energyUse: float – Emissions from electricity/gas consumption.
    - diet: float – Emissions from food habits.

    - wasteHabits: float – Emissions from waste production and recycling.

- Purpose: Holds the environmental input data submitted by the user.

CarbonFootprintCalculator

- Methods:
    - calculateFootprint(data: UserData): float – Computes total carbon footprint from the user's lifestyle data.
    - getRecommendations(footprint: float): String[] – Returns an array of sustainability recommendations based on the footprint value.
- Purpose: Core logic engine of the system, bridging input and insights.

Recommendation

- Attributes:
    - type: String – Category of recommendation (e.g., Energy, Transport).
    - description: String – Detailed suggestion text.
- Methods:
    - getRecommendation(): String – Returns formatted suggestion based on input type or impact level.
- Purpose: Suggests personalized tips for reducing carbon footprint.

DataVisualization

- Methods:
    - generateGraph(data: UserData): void – Creates graphical representations of user input and emissions.

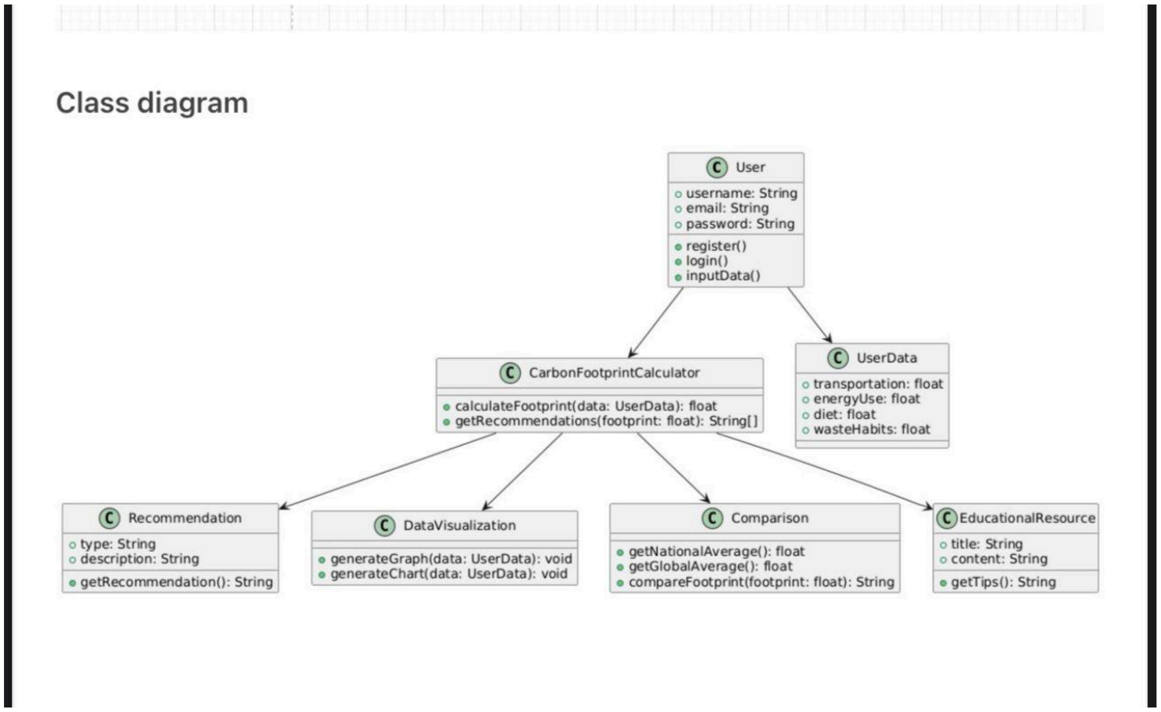
    - generateChart(data: UserData): void – Produces charts showing trends and insights.

- Purpose: Enhances user experience via visual feedback and historical comparison.

Comparison

- Methods:
    - getNationalAverage(): float – Retrieves average national footprint data.
    - getGlobalAverage(): float – Retrieves global footprint benchmark.
    - compareFootprint(footprint: float): String – Compares user's footprint to national/global stats and returns status summary.

- Purpose: Allows users to evaluate their performance relative to environmental norms.


Class diagram
User
username: String
email: String
password: String
register()
login()
inputData()
CarbonFootprintCalculator
calculateFootprint(data: UserData): float
getRecommendations(footprint: float): String[]
UserData
transportation: float
energyUse: float
diet: float
wasteHabits: float
Recommendation
type: String
description: String
getRecommendation(): String
DataVisualization
generateGraph(data: UserData): void
generateChart(data: UserData): void
Comparison
getNationalAverage(): float
getGlobalAverage(): float
compareFootprint(footprint: float): String
EducationalResource
title: String
content: String
getTips(): String

4.2 Use Case Realization

Use Case: Calculate Carbon Footprint

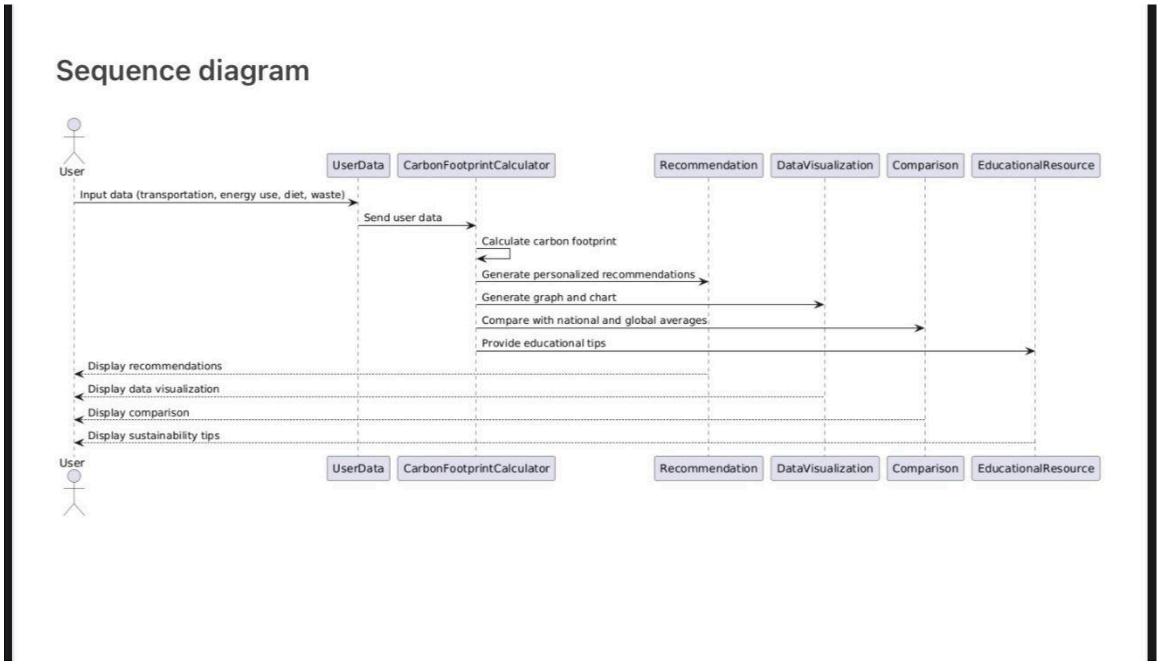
Actor: User
Modules Involved: UserData, CarbonFootprintCalculator, Recommendation, DataVisualization, Comparison, EducationalResource
This use case describes how the system processes user input to calculate carbon emissions, generate insights, and display personalized sustainability tips.

1. User Input
   The user inputs activity data such as:
   - Transportation (e.g., km traveled)
   - Energy use (e.g., electricity consumed)
   - Diet (e.g., vegetarian, non-veg, etc.)
   - Waste habits (e.g., recycling, disposal)
2. Data Transfer
   The input is captured into a UserData object and passed to the CarbonFootprintCalculator.
3. Carbon Calculation
   The CarbonFootprintCalculator processes the data using predefined emission factors and calculates the total carbon footprint (in CO_2e).
4. Generate Recommendations
   Based on the footprint result, the calculator invokes the Recommendation module to fetch relevant sustainability tips.
5. Visual Representation
   The DataVisualization module generates graphs and charts that visually represent the user's carbon emissions by category.
6. Comparison Logic
   The Comparison module retrieves national and global average emissions and compares them with the user’s footprint.

7. Educational Insights
   The EducationalResource module provides additional sustainability-related information and learning tips based on the user’s input.

8. Output to User
   The system displays:

   - Carbon footprint result
   - Personalized recommendations
   - Graphical charts
   - National/global comparisons
   - Educational content


Sequence diagram
User
UserData
CarbonFootprintCalculator
Recommendation
DataVisualization
Comparison
EducationalResource
Input data (transportation, energy use, diet, waste)
Send user data
Calculate carbon footprint
Generate personalized recommendations
Generate graph and chart
Compare with national and global averages
Provide educational tips
Display recommendations
Display data visualization
Display comparison
Display sustainability tips

5 Data View

The Carbon Footprint Calculator application uses a structured and modular data model to handle user inputs, computation, and presentation of results. The central entity is the

UserData class, which encapsulates the user’s input related to transportation, energy usage, diet, and waste habits. These inputs are represented as floating-point values and are crucial for calculating the user’s overall carbon footprint. This data is linked with a User class that stores authentication details such as username, email, and password, and handles registration and login functionality.

The CarbonFootprintCalculator class is responsible for processing the UserData and computing the user’s carbon emissions. It also interfaces with other supporting components. One such component is the Recommendation class, which provides tailored environmental suggestions based on the user's carbon footprint. These recommendations are categorized by type and include descriptive sustainability advice.

To present the results in a user-friendly format, the DataVisualization class generates graphs and charts from the input data. These visualizations allow users to easily understand their emission patterns. The Comparison class enhances this further by comparing the user's carbon footprint against national and global averages, providing context for their environmental impact. Finally, the EducationalResource class delivers informative content and sustainability tips that help users learn more about reducing their ecological footprint.

Together, these data models facilitate a smooth, end-to-end flow of data from user input to actionable insights, while ensuring clarity, modularity, and consistency across the frontend, backend, and visualization layers of the application.

5.1 Domain Model

The domain model of the Carbon Footprint Calculator app includes the main components that help users track and reduce their carbon emissions. At the core is the User, who provides login details and inputs personal data like travel, energy use, diet, and waste habits. This information is stored in the UserData class.

The CarbonFootprintCalculator uses this data to calculate the user's carbon footprint and provides personalized suggestions through the Recommendation class. To make the results easier to understand, the DataVisualization class creates graphs and charts.

The app also allows users to compare their footprint with national and global averages using the Comparison class. Finally, the EducationalResource class offers helpful tips and content to promote eco-friendly habits.

5.2 Data Model (Persistent Data View)

User

- Attributes:
  - user_id: Integer (Primary Key) – Unique identifier for each user
  - username: Varchar – User’s name
  - email: Varchar – User’s email address
  - password: Varchar – Encrypted password
- Relationships:
  - One-to-many relationship with CarbonEntry

CarbonEntry

- Attributes:
  - entry_id: Integer (Primary Key) – Unique entry ID
  - user_id: Integer (Foreign Key) – Linked to the User who submitted the entry
  - transportation: Float – Carbon emissions from transportation
  - energyUse: Float – Emissions from electricity or gas
  - diet: Float – Emissions from food consumption habits
  - wasteHabits: Float – Emissions from waste generation
  - total_emission: Float – Computed total CO_2e
  - timestamp: DateTime – Date and time of entry

Recommendation

- Attributes:
    - id: Integer (Primary Key) – Unique ID for each recommendation
    - type: Varchar – Category (e.g., Transport, Energy)
    - description: Text – The suggestion or tip content

Comparison

- Attributes:
    - id: Integer (Primary Key) – Unique ID
    - national_average: Float – Average national footprint
    - global_average: Float – Average global footprint

EducationalResource

- Attributes:
    - id: Integer (Primary Key) – Unique resource ID
    - title: Varchar – Topic or title of the resource
    - content: Text – Informative text related to sustainability

5.2.1 Data Dictionary

Name	Type	Description
user_id	Integer (PK)	Unique identifier for each user
username	Varchar	Name of the user
email	Varchar	Email used for login and notifications
password	Varchar	Encrypted password for authentication
entry_id	Integer (PK)	Unique ID for each carbon footprint entry
user_id	Integer (FK)	Foreign key linking the entry to a user
transportation	Float	Emissions from travel and transport activities
energyUse	Float	Emissions from electricity, LPG, and other household energy
diet	Float	Emissions based on food consumption habits
wasteHabits	Float	Emissions due to waste generation and management
total_emission	Float	Total calculated CO_2e for this entry
timestamp	DateTime	Time when the entry was submitted

Name	Type	Description
user_id	Integer (PK)	Unique identifier for each user
id (Recommendation)	Integer (PK)	Unique ID for each recommendation
type	Varchar	Category of the recommendation (e.g., Energy, Transport)
description	Text	Actual suggestion or advice text
id (Comparison)	Integer (PK)	Unique identifier for the comparison record
national_average	Float	National average CO_2e footprint value
global_average	Float	Global average CO_2e footprint value
id (EducationalResource)	Integer (PK)	Unique identifier for each educational tip
title	Varchar	Title of the sustainability topic
content	Text	Educational content or message

6 Exception Handling

The Carbon Footprint Calculator application includes exception handling mechanisms in both the backend and frontend to ensure smooth user experience and system stability. On the backend, try-except blocks are used in Flask routes to catch and manage common issues such as invalid inputs, database connection errors, and failed calculations. For example, if a user submits incomplete or invalid data, the system validates the input and returns a clear error message without crashing. Database-related issues, such as constraint violations or connection failures, are also caught and logged, ensuring the system can continue running even when errors occur.

In terms of user authentication, incorrect login attempts or unauthorized access are handled gracefully with appropriate feedback and redirection. On the frontend, HTML and JavaScript-based validation ensure that required fields are filled and inputs are within valid ranges before submission. If any issues are detected, user-friendly messages are displayed, prompting users to correct their input. Additionally, all critical exceptions are logged on the server for debugging purposes. In development mode, Flask's built-in debugger provides tracebacks, while in production, errors are handled silently with custom error pages shown to the user.

7 Configurable Parameters

Configuration Parameter Name	Definition and Usage	Dynamic?
EMISSION_FACTOR_SOURCE	Path or API endpoint used to fetch emission factor data	Yes
DEFAULT_UNITS	Measurement units for user input (e.g., km, kWh, kg)	Yes
CO2_THRESHOLD_ALERT	Emission value (in kg CO_2e) above which the system highlights high-impact results	Yes
SESSION_TIMEOUT	Duration of user login session before automatic logout	Yes
DATABASE_URI	Connection string for the app's database	No
RECOMMENDATION_LIMIT	Maximum number of tips shown to the user after each calculation	Yes
HISTORY_DISPLAY_LIMIT	Number of past entries shown to the user in the dashboard	Yes
ALLOW_FACTOR_EDIT	Boolean flag to allow admin access to update emission factors	Yes
MAX_INPUT_LENGTH	Character limit for each input field (used for validation and security)	Yes
ENABLE_COMPARISON_VIEW	Enables or disables the feature to compare user footprint with national/global averages	Yes

8.1 Availability

The Carbon Footprint Calculator is designed to be a lightweight, reliable application suitable for academic and local deployment. Since it is built using Flask, it can be hosted easily on local servers or lightweight cloud platforms. The system is expected to be available whenever the hosting server is running, with minimal downtime. During testing and demonstration, the application is run locally or over a LAN, ensuring availability for students, faculty, or evaluators.

Although the current scope does not require high-scale uptime or redundancy, the system can be extended for broader deployment by integrating with production-grade servers and databases. Basic health checks, logging, and exception handling help maintain consistent performance during usage. In future enhancements, features like auto-restart on crash or periodic backups could be added to improve availability further.

8.2 Security and Authorization

The Carbon Footprint Calculator implements basic security and authorization features to ensure that user data is protected and system access is controlled. User authentication is handled through a login system where credentials (email and password) are securely stored using hashing techniques. Only registered users can access the main features of the application such as data input, footprint calculation, and history tracking.

The system follows a role-based access approach with two main roles: **User** and **Admin**. Regular users can input data, view their results, and track progress, while administrators have additional access to update emission factors and manage system settings. Backend route protection ensures that sensitive operations like updating calculation parameters are only accessible to admins.

While advanced features such as two-factor authentication or OAuth integration are out of scope for this version, the current setup provides a secure foundation for a locally hosted academic project. All forms are also validated to prevent basic input tampering and injection attacks.

8.3 Load and Performance Implications

The Carbon Footprint Calculator is developed as a lightweight application using Flask, designed primarily for academic use and small-scale deployments. It is optimized for handling a limited number of users simultaneously, such as students and faculty accessing the tool during evaluations or demonstrations. Since it processes simple user inputs and performs basic calculations, the system does not demand high computational resources or complex server configurations.

All operations, including input handling, emission calculation, and database storage, are designed to execute quickly and efficiently. The use of server-side rendering with Jinja templates also reduces frontend complexity, further improving performance. Under typical use, the application is expected to deliver responses within a second or two, even with multiple data categories being processed.

While it is not currently intended for high-traffic public use, the application can be scaled in the future by upgrading to a more robust deployment environment, using asynchronous processing, and optimizing database queries. For the current scope, the system's performance is stable and reliable under normal academic load conditions.

8.4 Monitoring and Control

The Carbon Footprint Calculator includes basic monitoring and control mechanisms to ensure the system runs smoothly and can be maintained effectively. During development and testing, server logs are used to monitor key events such as user logins, data submissions, and error messages. These logs help identify issues quickly and support debugging.

Although advanced monitoring tools are not integrated due to the project's academic scope, the system is structured in a way that allows for easy extension. Health checks can be manually performed through test inputs, and the admin interface allows for reviewing and updating core system data like emission factors.

In future deployments, the application can include features such as user activity tracking, usage statistics, or automated log analysis to enhance visibility and control. For the current version, regular code reviews, test runs, and manual log inspection provide sufficient oversight and stability.