GreenChain Insights - Final Project Report

Project Changes from Original Proposal

Due to the time needed to complete the bulk of our project, we decided that it was in our best interest to not include the category-specific dashboard page as suggested in our original project proposal. We felt that our current landing page implementation provided enough information to the user about each specific category and thus saw the category-specific views to be unnecessary and distracting.

To replace the category-specific dashboard, we added a Carbon Insights page as a feature that looked deeper into the carbon impact that we had. We felt that this was a better use of our resources as it dived deeper into each user's data instead of the dataset we had on hand. As a tool, the Carbon Insights change felt better. We will explain in detail about this change in the "Functionality Additions and Removals" section.

Outside of those changes, we were fortunately able to follow our project proposal to a tee and implement several key functional components that make our application what it is.

Project Achievements and Limitations

The project was well in the areas of acting like a personal carbon footprint tracker for users. We felt like the personal carbon footprint simulation was an excellent feature that most people would use. We intended to have more people aware of the emissions they were releasing and this accomplished exactly that with being able to add, update, and delete orders while seeing the exact changes it made to the emissions data. As far as the data analysis of categories goes, I feel like we did a decent job of showing which categories are at risk given the data we had.

For the scope of this project, we definitely felt like we were limited by the data we were able to find online. Supply Chain data with the right NAICS codes is pretty hard to come by, and thus, we were limited to a few data sets. We would want to gather more data that is more recent so that the application's data would be more representative of the real world.

Data Source and Schema Changes

We made no direct modifications to the three original data sources we referenced in our initial reports: **GHG Emission Factors by Industry**, **Supply Chain Operational Data**, and **EC1700BASIC**. Instead, we transformed them to fit more naturally into our relational schema. First, to facilitate user-level demonstrations of our "Explore Carbon Footprint" module and to test authentication flows, we generated a mock **Users** CSV containing pseudonymous usernames, emails and hashed passwords. This new relation underpins our SSO-style login and enables full CRUD operations on personal scenario entries without altering the integrity of the original emissions or supply-chain datasets.

Second, we leveraged the small, 30-row **EC1700BASIC** file to derive a true **Category** table (exported as category.csv) with columns **Category_ID**, **Category_Name** and **NAICS_Code**. By doing so, we created a clear bridge between our **Industries** table (from the emission-factors CSV) and our **Orders**/**Shipping_Details** tables (from the operational data). This normalization step was necessary to avoid repeated NAICS codes scattered across order records, to simplify joins for both dashboard filters and advanced queries, and to improve overall query performance and referential integrity.

Together, these schema adaptations, adding a dedicated **Users** relation and factoring out **Category** as an explicit dimension, greatly enhance our application's modularity and maintainability. The mock user data allows us to demonstrate personalised emissions scenarios end-to-end, while the new Category linkage ensures that our industry-level analytics, scenario planner inputs and interactive visualisations all draw from a consistent, well-structured relational backbone.

ER Diagram and Table Implementation Changes

We made two main changes to our original database design:

1. We choose to remove Customer_Scenario, a proposed entity, opting instead to model it as a view.

This entity was initially introduced to reflect the interactive simulation aspect (users managing specific customer cases), but we realized that no extra attributes were needed to support this entity and therefore it was better modeled as a view. Consequently, in our final schema design we treat the link between users and customers as a derived relationship which proves more suitable for our application needs.

2. We changed the relationship between our Users and Customers entities from a 1:1 relationship to a many-to-many relationship.

Making this relationship less restrictive supports proper normalization since neither table is forced to depend on the other's keys. This allowed us to cover scenarios our application might face in real usage, where one user might oversee several customer profiles or multiple users might share access to a single customer.

Functionality Additions and Removals

Our application introduces the Carbon Insights route, which is responsible for providing personalized carbon impact analytics for each user based on their purchasing history. When this route is called, the server uses a stored procedure called GetUserCarbonInsights, which efficiently aggregates the user's order data in the MySQL database. Specifically, it computes two sets of insights: a breakdown of emissions by product category (showing how much each category contributed to the user's total emissions) and a temporal analysis of monthly spending and associated carbon emissions.

The process ensures that users receive both a category-based view (e.g., "Food purchases contributed to a majority of your emissions") and a time-based view (e.g., "Your emissions in March were lower than in February"). To achieve this efficiently, the backend uses multiple SQL joins between the Orders, Category, and Industries tables, and groups the results by category name and by month, respectively. If a user has no purchases, the route safely returns empty insights. Overall, this route enables users to visually track their environmental impact over time and identify which areas of their consumption have the largest carbon footprint, helping to drive behavior change toward more sustainable choices.

We believe that the Carbon Insights portion of our project provides the user with more of a visual representation of their footprint versus a standard list of purchases. As we mentioned before, we felt like this would impact users more by showing them their emissions from their own orders instead of the general information about the categories that we initially intended to make.

Advanced Database Program Implementation

Our implementation of advanced database features — including transactions, stored procedures, triggers, and constraints — significantly enhanced the functionality, reliability, and performance of our GreenChain Insights application.

Transactions were critical when performing multi-step operations, such as deleting a user transaction. For example, when a user deleted a purchase, we used a transaction to first remove any dependent shipping details and then remove the corresponding order, ensuring that the database stayed consistent even if an error occurred midway. This prevented partial deletions and guaranteed atomicity.

We used a stored procedure to generate personalized carbon insights for each user. By predefining this complex set of JOIN and aggregation queries inside the database and using proper indexing when needed, we reduced the amount of data transferred between the server and database, improved response times, and centralized critical business logic to ensure consistent calculations across different parts of the app.

Triggers played a major role in automatically maintaining user emissions data. For example, after every new order insertion, update, or deletion, our triggers recalculated the user's Total_Emissions and Monthly_Emissions fields in the Users table. This allowed the application to instantly reflect updated emissions values without requiring expensive recalculations every time the dashboard or scenario planner was loaded.

Finally, constraints like primary keys, foreign keys, and type checks ensured the referential integrity between entities such as Users, Orders, Categories, and Industries. This prevented invalid data — like linking an order to a non-existent category — and kept the application stable even under heavy usage or when introducing new features like user-created transactions.

Overall, our implementation of these advanced database features provides our application with more than just a list of categories on a screen, but rather an interactive and robust application that can be used for a variety of reasons.

Technical Challenges

Team Member 1 (Ismail M.)

Technical Challenge: Implementing the Customer Scenario

Probably the biggest challenge I faced on this project was trying to figure out how the Customer Scenario should look and making sure that it integrated with the back end. Implementing the adding functionality was pretty easy, but once we got to updating and deleting users' information, I started to run into issues. After spending some time looking at the issue, I realized it wouldn't allow me to delete some entries and for some new users, there would already be information there (not the intended way this was supposed to work). Because I had less experience on the backend, where the issue might be arising from, I communicated with Shashank (our backend lead) and the rest of the team. We realized that there was some issues with how the foreign keys were being used. To be exact, the issue was that the User_ID was being treated as the Customer_ID within the orders table and so the application was adding orders that had the matching customer_id (even though the user didn't add anything). To fix it, we just made sure that the user_id would be 1 + max(customer_id)

Even though I was more front-end focused, I tried to learn more about how the back end was interacting with the databases and why there was issues. I learnt a lot about how to debug with stuff that was in GCP and learnt a lot more about JavaScript backends.

Team Member 2 (Ashley W.)

Technical challenge: Designing an intuitive user navigation system across complex data interactions

One of the major challenges I faced in our project was designing an intuitive and accessible navigation flow, for our GreenChain website, that would smoothly guide users through complex tasks while building a clear and compelling data story. This user experience had to serve both technically skilled users and those with less background knowledge in emission data. To achieve this, I had to balance data richness with simplicity, ensuring users could draw meaningful insights without feeling overwhelmed.

Specifically with the customer scenario functionality, it was important that users understood both what factors they were changing and how those changes affected the emission trends and outcomes. I also made specific design decisions to support clarity. These included enforcing consistency within the navigation structure, with things like visible breadcrumb trails, and within visualizations, with things like uniform styles, color schemes, and buttons. This reinforced familiarity even as the users moved between very different types of tasks. Additionally, I included small things to help guide first-time users, such as suggested industries below the search bar making exploration easy and intuitive.

Overall, the goal was to create a navigation system that made complex environmental data approachable, empowering users to explore real-world impact scenarios with both confidence and clarity.

Team Member 3 (Shashank B.)

One of the significant technical challenges I faced during our project was implementing and testing the connection to our GCP SQL instance. The goal was to ensure our application could reliably read from and write to the cloud database, supporting critical features like user authentication, purchase history tracking, and emissions calculations. From a technical standpoint, while we had designed our database schema and API endpoints, I initially struggled to establish a stable and secure connection between our backend server and the GCP SQL instance. To overcome this, I set up a connection pool using the mysql2 library, configured private IP access for security, and carefully tuned the connection parameters (like connectionLimit and queueLimit) to ensure scalability. I also wrote and tested dedicated API routes for basic queries, inserts, updates, and deletions to validate that the connection behaved correctly under different load scenarios. Throughout the process, I communicated updates frequently with the group (good or bad!) and stress tested the connection both from the backend and by running controlled load tests on the database and checking for intended updates and deletes. Overall, this experience helped me learn how to tackle ambiguous challenges step-by-step with good communication, proactive testing, and a focus on incremental progress.

Team Member 4 (Ananth H.)

During development, we encountered a persistent data-integration challenge: our three raw CSV sources featured inconsistent field formats, missing or mismatched NAICS codes, and irregular ordering that made bulk loading into the relational schema error-prone, particularly as the supply-chain dataset combined date formats and referenced NAICS codes absent from the emission-factors file when we imported them in, leading to violations and flawed dashboard metrics. To resolve this, I created a Pandas-based ETL script that took in each CSV, standardized dates and numeric types, filled or flagged missing codes, generated a deduplicated `category.csv` with surrogate Category_IDs, enforced strict typing for financial fields, and exported clean, ordered files for MySQL's bulk loader. This ultimately helped in eliminating schema mismatches. This reinforced a schema-first data-ingestion philosophy: by defining our SQL tables in advance, we caught inconsistencies early, and by automating cleanup with Python-based tools I was familiar with like Pandas, we established a reproducible pipeline that we now version-control, test against our relational model and document thoroughly to guarantee clean data for future updates.

Other Changes from Original Proposal

Aside from the ones we mentioned above about the Carbon Insights and Schema changes, we didn't make any other changes.

Future Work

In future phases, we would like to implement a real-time ingestion pipeline by leveraging Kafka and a lightweight Node.js ETL to stream live shipping updates and refreshed emission factors directly into warehouses. Concurrently, we would expand our emissions modelling beyond CO_2e per USD by integrating additional Scope 1 and Scope 2 factors, automating the ingestion of multi-year USEEIO releases and sector-specific life-cycle analyses via scheduled Airflow workflows. To unlock deeper insight, we will introduce predictive analytics (deploying ARIMA or LSTM models for time-series forecasting of emissions trends) and embed statistical anomaly detection (e.g. Isolation Forest) to flag atypical carbon intensities in shipments. Finally, a geospatial visualization layer (using Mapbox GL or Leaflet) to render shipment routes, regional heat-maps of emissions and delivery-risk overlays would make an excellent addition to best visualize supply chains in tandem with our data.

On the infrastructure side, we would partition our large fact tables by date or region and evaluate columnar storage engines (e.g. ClickHouse or Redshift) alongside a Redis caching tier to serve high-frequency queries with minimal latency. We'll also expose a robust API ecosystem (with REST/GraphQL endpoints and OpenAPI documentation) and develop connectors for popular ERP and ESG platforms to synchronise scenario data externally. To foster collaboration and maintain auditability, we would also like to add multi-user scenario sharing, versioned snapshots of simulations and an event-sourced audit trail to track every modification in users' carbon footprint analyses.

Division of Labor and Team Management

Throughout the final implementation, we adhered closely to our original task assignments: Ismail remained the lead front-end architect, crafting the React components and D3 visualizations; Ashley oversaw UI/UX design and integrated the API layers into the interface; Shashank engineered the core backend services, data pipelines and database schemas; and Ananth served as the full-stack bridge, connecting front and back-end modules and refining data models to ensure functionality. Throughout, we each reviewed one another's pull requests, coordinated via Git feature branches and frequent meetings, and collectively validated our components. Along the way, we all made sure that - despite our assigned role - we delved into what other team members were doing to get a full grasp of our application and how it can impact our users.

Early on, we ran into server-access issues: SSH keys and permissions weren't correctly configured on our GCP instances, which prevented seamless code pushes and database migrations. These setbacks stalled front-end/back-end integration until the professor helped us during office hours to set up roles and shared SSH credentials as well as guide us through GCP networking. Once the environment was stable, we resumed collaborative development, resolving merge conflicts, updating our CI/CD pipeline, and finalizing load-testing scripts. In the end, not only did we complete all the tasks we originally outlined, but the troubleshooting process also strengthened our team communication and our ability to manage cloud infrastructure under deadline pressure.

Conclusion

Overall, our GreenChain Insights project came together really well. We were able to build a solid, working application that gives users a better picture of their personal carbon footprint and what changes they can make to their purchasing habits to reduce it. From setting up cloud databases and API routes to designing user-friendly features and components, we learned a lot through this project. We are truly proud of what we accomplished through GreenChain Insights and excited for the user potential of this product.