TiffinTrails: Software Documentation

CSC-510: Software Engineering Project 2

Version 1.0.0

Authors:

Yash Dhavale
Shreeya Ranwadkar
Aditya Deshpande
Ravi Goparaju

Instructor: Professor Tim Menzies
Institution: North Carolina State University

Contents

1 Introduction **3**

2 Software Overview **4**
2.1 Project Structure 4
2.2 Purpose of Each Module 5
2.2.1 Backend Modules 5
2.2.2 Frontend Components 6
2.2.3 API Module 7

3 System Architecture and User Flow **7**
3.1 Application Flow 7
3.2 Future Milestones 8

4 Quick Start Guide **9**
4.1 Prerequisites 9
4.2 Installation 9
4.3 Running the Project 10
4.3.1 Step 1: Generate Synthetic Datasets 10
4.3.2 Step 2: Load and Clean Data 11
4.3.3 Step 3: Compute Delivery Metrics 11
4.3.4 Step 4: Calculate Efficiency Scores 11
4.3.5 Step 5: Start Backend API Server 11
4.3.6 Step 6: Start Frontend Development Server 11
4.4 Testing the Application 11

5 Detailed Documentation **12**
5.1 Backend Data Processing 12
5.1.1 data_generator.py 12
5.1.2 data_loader.py 13
5.1.3 delivery_metrics.py 13
5.1.4 efficiency_scoring.py 13
5.1.5 rescue_meals_integration.py 14
5.2 Backend API (Node.js/Express) 14
5.2.1 backend/server.js 14
5.2.2 backend/routes/home.js 15
5.2.3 backend/routes/cart.js 15
5.3 Frontend Components 16
5.3.1 App.jsx 16
5.3.2 Home.jsx 16
5.3.3 LoginChoice.jsx 16
5.3.4 CustomerLogin.jsx 17
5.3.5 Dashboard.jsx 17
5.3.6 Cart.jsx 17
5.4 API Endpoints 18
5.4.1 Node.js Backend (Express) 18
5.4.2 Python Flask API (Legacy) 19

6 Efficiency Scoring System **19**
6.1 Scoring Methodology 19
6.2 Reward System 20

7 Data Analysis and Visualization **20**
7.1 Statistical Analysis 20
7.2 Visualization 20

8 Testing **21**
8.1 Frontend Testing 21
8.2 Backend Testing 21
8.3 Integration Testing 21

9 Troubleshooting **22**

10 Release History **22**

11 Revision Control **23**

12 Conclusion **23**

13 Troubleshooting **24**

14 Release History **24**

15 References **25**

1 Introduction

TiffinTrails is a data-driven, reward-based platform designed to address food waste challenges in the restaurant and food delivery ecosystem. This project is part of the CSC-510 Software Engineering course and demonstrates modern software development practices including full-stack web development, data analysis, API design, and modular architecture.

Food waste and inefficient delivery remain major challenges for restaurants and food platforms, increasing costs and environmental impact. **TiffinTrails** tackles this through a data-driven, reward-based system that tracks delivery efficiency, waste reduction, and sustainability. Restaurants earn badges and improved scores for minimizing waste and offering rescue meals—discounted portions that prevent food loss—while customers enjoy fresher, eco-friendly choices. By combining analytics, incentives, and transparency, **TiffinTrails** promotes smarter, sustainable dining for all.

The platform consists of a web application where customers can browse partner restaurants, view their efficiency metrics, discover rescue meals at discounted prices, add items to their cart, and place orders. When customers purchase rescue meals, restaurants are rewarded with points and improved efficiency scores, creating a positive feedback loop that incentivizes sustainable practices.

Motivation

Traditional food delivery platforms focus primarily on convenience and speed, often overlooking the environmental and economic costs of food waste. **TiffinTrails** addresses this gap by:

- Providing transparency into restaurant efficiency metrics
- Creating incentives for restaurants to minimize waste through rescue meal programs
- Empowering customers to make sustainable choices while saving money
- Building a data-driven foundation for continuous improvement in delivery operations
- Enabling seamless order placement through cart functionality

Objectives

The primary objectives of this project are:

- To design and implement a full-stack web application with modular frontend and backend components
- To generate and process synthetic datasets that simulate real-world restaurant and delivery operations

- To compute quantitative efficiency scores for restaurants based on delivery performance and waste metrics
- To create a user-friendly interface for customers to discover restaurants and rescue meals
- To establish a reward system that incentivizes sustainable practices
- To implement cart functionality and order placement system
- To demonstrate reproducibility, clarity, and maintainability through strong documentation and testing

2 Software Overview

2.1 Project Structure

The folder structure below represents the modular organization of the project:

```
Proj2/

 data/                    # Dataset files (CSV, JSON)
    cleaned_master_dataset.csv
    Customer_Feedback.csv
    Delivery_Logs.csv
    Menu_Portions.csv
    rescue_meals.csv
    Restaurant_Metadata.csv
    vendor_delivery_metrics.csv
    vendor_efficiency_scores.csv


 src/                     # Source code
    data_generator.py    # Synthetic data generation
    data_loader.py       # Data loading and cleaning
    delivery_metrics.py  # Delivery performance metrics
    efficiency_scoring.py # Restaurant efficiency scoring
    rescue_meals_integration.py # Rescue meal data generation


    backend/             # Node.js backend API
       server.js        # Express server
       routes/          # API route handlers
          home.js      # Home routes
          cart.js      # Cart and order routes
       data/            # Data storage
          restaurant_points.json
```

```
        orders.json
    secrets/        # User credentials
        users.js


api/                  # Python Flask API
    app.py          # Flask REST API server


frontend/             # React frontend application
    src/
        App.jsx     # Main application component
        components/ # React components
            Home.jsx
            LoginChoice.jsx
            CustomerLogin.jsx
            Dashboard.jsx
            Cart.jsx
            RestaurantCard.jsx
            ImpactCard.jsx
            Navbar.jsx
        data/       # Static data
    package.json


analysis/             # Data analysis modules
    correlate_efficiency_waste.py


visualization/        # Visualization modules
     efficiency_waste_viz.py

README.md
SE_Proj2_Documentation.pdf
```

2.2 Purpose of Each Module

2.2.1 Backend Modules

- **data_generator.py:** Generates realistic synthetic datasets for restaurant metadata, menu portions, and waste data.
- **data_loader.py:** Loads, validates, and integrates all generated data into a single clean master dataset for analysis.
- **delivery_metrics.py:** Computes performance metrics such as average delivery time, on-time rate, average distance, and deliveries per day for each restaurant.

- **efficiency_scoring.py:** Merges metrics and metadata to compute an overall restaurant efficiency score using a weighted formula.
- **rescue_meals_integration.py:** Generates rescue meal data for restaurants, including meal names, original prices, discounted rescue prices, quantities, and expiration times.
- **backend/server.js:** Express.js server providing REST API endpoints for user authentication, order placement, and restaurant points management. Handles CORS and JSON parsing.
- **backend/routes/home.js:** Route handlers for restaurant data and impact statistics endpoints.
- **backend/routes/cart.js:** Route handlers for order placement, restaurant points retrieval, and order history. Manages restaurant points increment when rescue meals are ordered.
- **analysis/correlate_efficiency_waste.py:** Performs statistical analysis to correlate delivery efficiency with waste metrics, providing insights for restaurant improvement.
- **visualization/efficiency_waste_viz.py:** Creates visualizations and charts for efficiency and waste data analysis.

2.2.2 Frontend Components

- **Home.jsx:** Landing page displaying TiffinTrails mission, partner restaurants, and community impact statistics. Features a call-to-action to join the platform.
- **LoginChoice.jsx:** Login selection page allowing users to choose between customer or restaurant login. Restaurant login is marked as "Coming Soon" for future implementation.
- **CustomerLogin.jsx:** Customer authentication interface with email and password fields. Validates credentials against stored user data.
- **Dashboard.jsx:** Main customer dashboard displaying available restaurants with their menus, distance, on-time rate, efficiency scores, and rescue meal availability. Features cart icon navigation, browse and rescue meal views, and cart management integration.
- **Cart.jsx:** Cart page component displaying order items in a table format. Features quantity management, item removal, order summary with savings calculation, and order placement functionality. Shows success message after order placement and tracks rescue meal impact.
- **RestaurantCard.jsx:** Component rendering individual restaurant information cards with key metrics and action buttons.
- **ImpactCard.jsx:** Component displaying user environmental impact metrics.
- **Navbar.jsx:** Navigation component providing consistent navigation across the application.

2.2.3 API Module

- **api/app.py:** Flask REST API server providing endpoints for efficiency-waste correlation analysis, health checks, and data access. Uses Flask-CORS to enable cross-origin requests from the frontend.

3 System Architecture and User Flow

3.1 Application Flow

TiffinTrails follows a clear user journey:

1. **Home Page:** Users land on the homepage which showcases:
   - TiffinTrails mission and values
   - Partner restaurants and their cuisine types
   - Community impact statistics (meals rescued, waste prevented, active users)
   - Call-to-action button to join the platform
2. **Login Selection:** Users choose between customer or restaurant login. Currently, only customer login is implemented; restaurant login is designated for future milestones.
3. **Customer Authentication:** Customers enter their email and password credentials. The system validates against stored user data and authenticates valid users.
4. **Customer Dashboard:** After successful login, customers see:
   - Navigation bar with Browse, Rescue Meals, My Impact, and Community tabs
   - Cart icon in the top right corner showing number of items
   - List of available partner restaurants with:
     - Restaurant name and cuisine type
     - Distance from customer location
     - On-time delivery rate percentage
     - Efficiency score and sustainability metrics
     - Available rescue meals (if any)
   - Rescue meal sections showing discounted meals with expiration times
   - "Rescue" or "Rescue This Meal" buttons to add items to cart
5. **Cart Page:** When customers click the cart icon, they navigate to the cart page which displays:
   - Table of all cart items with columns for item name, restaurant, price, quantity, total, and actions
   - Quantity controls (+/- buttons) to modify item quantities
   - Remove item functionality

- Order summary sidebar showing subtotal, savings, and final total
- Impact points message indicating rescue meals in the order
- "Place Order" button to complete the purchase

6. **Order Placement:** When customers click "Place Order":
   - Order is sent to backend API (`POST /api/orders`)
   - Restaurant points are calculated and updated (10 points per rescue meal)
   - Order record is saved with timestamp, items, totals, and points earned
   - Success message is displayed: "Order Placed Successfully!"
   - Impact message shows number of rescue meals saved
   - Cart is automatically cleared
   - Option to continue shopping and return to dashboard
7. **Restaurant Rewards:** When customers place orders with rescue meals:
   - Restaurants receive points (10 points per rescue meal ordered)
   - Points are stored in `restaurant_points.json`
   - Order history is maintained in `orders.json`
   - Restaurant efficiency scores can be updated based on points earned
   - Positive feedback loop encourages waste reduction

3.2 Future Milestones

The upcoming development goals for TiffinTrails focus on expanding functionality, intelligence, and sustainability-driven features:

- **Restaurant Portal and Analytics Dashboard:** Build a dedicated restaurant dashboard to manage rescue meals, analyze delivery performance, and monitor waste reduction metrics. The dashboard will also feature visual analytics and performance trends.
- **Smart Insights and Recommendation Engine:** Develop a data-driven engine that provides actionable recommendations for optimizing delivery routes, minimizing waste, and improving overall operational efficiency.
- **Gamified Reward System:** Introduce an engaging reward mechanism for both customers and restaurants to promote sustainable practices, offering badges, points, and incentives for consistent eco-friendly participation.
- **Sustainability Analytics and API Integration:** Enhance sustainability tracking by integrating external APIs and environmental databases to calculate real-time carbon savings and food waste reduction metrics.
- **Integrated Payment System:** Implement a secure payment processing module for customer orders, supporting multiple payment methods and providing transaction history within the app.

4 Quick Start Guide

This section provides a concise walkthrough for new users to get TiffinTrails running locally.

4.1 Prerequisites

- Python 3.8 or higher
- Node.js 16.x or higher and npm
- Required Python libraries:

```
pandas

numpy

flask

flask-cors
```

- Required JavaScript libraries (managed via npm):

```
react

react-dom

react-router-dom

express

cors

tailwindcss
```

- Ensure that Git and pip are properly configured on your system.

4.2 Installation

Follow these steps to set up and deploy the software:

```
# Clone the repository
git clone https://github.com/YashDhavale/CSC-510-SE-Project.git
# Navigate to project directory
cd CSC-510-SE-Project/Proj2
# Install Python dependencies
pip install pandas numpy flask flask-cors
# Install backend dependencies
cd src/backend
npm install
# Install frontend dependencies
cd ../frontend
npm install
# Return to Proj2 directory
cd ../..
```

After installation, verify the environment by executing:

```
python --version
node --version
npm --version
pip list
```

4.3 Running the Project

The complete execution workflow involves multiple stages:

4.3.1 Step 1: Generate Synthetic Datasets

```
python src/data_generator.py
python src/rescue_meals_integration.py
```

This generates all necessary CSV files in the `data/` directory.

4.3.2 Step 2: Load and Clean Data

```
python src/data_loader.py
```

This integrates all datasets into a cleaned master dataset.

4.3.3 Step 3: Compute Delivery Metrics

```
python src/delivery_metrics.py
```

This calculates vendor delivery performance metrics.

4.3.4 Step 4: Calculate Efficiency Scores

```
python src/efficiency_scoring.py
```

This computes overall restaurant efficiency scores.

4.3.5 Step 5: Start Backend API Server

```
cd src/backend

node server.js
```

The Express API server will start on `http://localhost:5000`.

4.3.6 Step 6: Start Frontend Development Server

In a new terminal window:

```
cd src/frontend

npm start
```

The React application will start on `http://localhost:3000` and automatically open in your browser.

4.4 Testing the Application

1. Navigate to `http://localhost:3000` in your browser
2. You should see the TiffinTrails homepage
3. Click "Join Us Today" or navigate to the login page

4. Select "Customer" login
5. Use test credentials (see `src/backend/secrets/users.js`) to log in
6. Explore the dashboard with restaurant listings and metrics
7. Click on rescue meals or browse restaurants to add items to cart
8. Click the cart icon in the top right corner
9. Review your order in the cart page table
10. Click "Place Order" to complete the purchase
11. Verify the success message and order confirmation

5 Detailed Documentation

This section provides comprehensive explanations of each source file, their major functions, parameters, and intended outputs.

5.1 Backend Data Processing

5.1.1 data_generator.py

Purpose: Generates synthetic datasets that simulate real-world restaurant operations.

Key Functions:

- `generate_waste_data(n)` – Creates weekly food waste dataset with timestamps and quantities
- `generate_restaurant_metadata()` – Builds restaurant profiles with names, cuisines, locations, and contact information
- `generate_customer_feedback()` – Simulates customer review data with ratings and comments
- `generate_menu_portions()` – Generates menu item data with portion sizes and prices
- `generate_delivery_logs()` – Produces delivery activity logs with timestamps, distances, and status

Usage:

```
from src.data_generator import main

main()  # generates all datasets under /data
```

Each generated dataset is saved as a CSV file under the `data/` directory and can be reused across modules.

5.1.2 data_loader.py

Purpose: Loads, validates, and integrates datasets into a unified structure.

Key Functions:

- `load_csv(name)` – Loads CSV files into pandas DataFrames with error handling
- `basic_clean(df)` – Cleans string values, removes duplicates, and handles missing data
- `integrate_all()` – Merges all datasets into one unified table using appropriate join keys

Usage:

```
from src.data_loader import integrate_all

merged_df = integrate_all()
```

This module ensures dataset integrity before analytical processing.

5.1.3 delivery_metrics.py

Purpose: Processes delivery data to calculate vendor KPIs.

Computed Metrics:

- Average delivery time per restaurant
- On-time delivery rate (percentage of deliveries within estimated time)
- Average delivery distance
- Deliveries per day (average)

Usage:

```
from src.delivery_metrics import compute_delivery_metrics

compute_delivery_metrics("data/Delivery_Logs.csv",
                         "data/vendor_delivery_metrics.csv")
```

5.1.4 efficiency_scoring.py

Purpose: Combines delivery metrics with restaurant metadata to compute a normalized efficiency score.

Formula:

Efficiency Score = 0.4(on_time_rate) + 0.3(1-norm_delivery_time) + 0.2(1-norm_distance) + 0.1(norm_deliveries_per_day)

Where normalization is performed relative to the dataset min/max values.

Usage:

```
from src.efficiency_scoring import compute_efficiency_scores

compute_efficiency_scores("data/vendor_delivery_metrics.csv",
                          "data/Restaurant_Metadata.csv",
                          "data/vendor_efficiency_scores.csv")
```

This computation allows the evaluation of delivery efficiency in a quantitative, comparable form.

5.1.5 rescue_meals_integration.py

Purpose: Generates rescue meal data for restaurants with surplus food.

Key Data Fields:

- Restaurant name
- Meal name
- Original price
- Rescue price (discounted)
- Available quantity
- Expiration time (hours until expiration)

Usage:

```
python src/rescue_meals_integration.py
```

Generates both CSV and JSON formats for easy integration with API and frontend.

5.2 Backend API (Node.js/Express)

5.2.1 backend/server.js

Purpose: Express.js server providing REST API endpoints and middleware configuration.

Key Features:

- CORS configuration for cross-origin requests
- JSON body parsing middleware
- User authentication endpoints (`POST /login`, `POST /register`)

- Route integration for home and cart modules
- Port configuration (default: 5000)

Usage:

```
cd src/backend

node server.js
```

The server initializes restaurant points and orders data files if they don't exist.

5.2.2 backend/routes/home.js

Purpose: Route handlers for restaurant data and impact statistics.

Endpoints:

- `GET /restaurants` – Returns restaurant metadata from CSV files
- `GET /impact` – Returns community impact statistics including total meals rescued, waste prevented, and active users

5.2.3 backend/routes/cart.js

Purpose: Route handlers for order placement, restaurant points management, and order history.

Key Functions:

- Initializes restaurant points file with default values (0 points for all restaurants)
- Initializes orders file as empty array
- Processes order placement and calculates restaurant points

Endpoints:

- `POST /api/orders` – Places a new order
  - Request body: `{items: Array, totals: Object}`
  - Validates order data
  - Calculates points earned (10 points per rescue meal)
  - Updates restaurant points in JSON file
  - Creates order record with ID, timestamp, items, totals, and points earned
  - Saves order to orders.json
  - Returns success response with order details
- `GET /api/restaurant-points` – Retrieves current restaurant points
  - Returns JSON object with restaurant names as keys and points as values

- `GET /api/orders` – Retrieves all order history
  - Returns array of all orders with full details

Restaurant Points System:

- Each rescue meal ordered earns the restaurant 10 points
- Points are cumulative and persist across sessions
- Points are stored in `data/restaurant_points.json`
- Points can be used to calculate restaurant rankings and rewards

5.3 Frontend Components

5.3.1 App.jsx

Purpose: Main application component managing routing and state.
Key Features:

- State management for current page and user authentication
- Navigation between Home, Login Choice, Customer Login, and Dashboard
- User login handling and session management
- User data passing to Dashboard component

5.3.2 Home.jsx

Purpose: Landing page component showcasing platform mission and features.
Key Sections:

- Hero section with mission statement
- Feature highlights (Save Planet, Save Money, Support Local)
- Partner restaurants grid
- Community impact statistics
- Call-to-action button

5.3.3 LoginChoice.jsx

Purpose: Allows users to select their account type.
Features:

- Customer login option (active)
- Restaurant login option (marked as "Coming Soon")
- Feature lists for each account type
- Navigation back to home page

5.3.4 CustomerLogin.jsx

Purpose: Customer authentication interface.
Features:

- Email and password input fields
- Form validation
- Credential verification against user database via API
- Error handling for invalid credentials
- Navigation to dashboard upon successful login

5.3.5 Dashboard.jsx

Purpose: Main customer interface displaying restaurant information and cart management.
Key Features:

- Navigation bar with multiple views:
  - Browse – List of all restaurants
  - Rescue Meals – Filtered view of available rescue meals
  - My Impact – User's environmental impact statistics
  - Community – Community-wide statistics
- Cart icon in header with item count badge
- Restaurant listings with menu information, distance metrics, on-time rate percentage, efficiency score, and rescue meal availability
- Add to cart functionality for rescue meals
- Cart state management
- Navigation to cart page
- Favorite restaurants functionality
- Search and filter by cuisine type

5.3.6 Cart.jsx

Purpose: Cart page component for order review and placement.
Key Features:

- **Order Items Table:** Displays all cart items with columns for item name (with rescue meal badge indicator), restaurant name, price (showing original and rescue prices for rescue meals), quantity controls (increase/decrease buttons), total price per item, and remove item button

- **Order Summary Sidebar:** Shows subtotal calculation, total savings display (for rescue meals), final total, and impact points message showing rescue meal count
- **Quantity Management:** Increase/decrease quantity buttons with minimum quantity of 1 enforced and real-time total recalculation
- **Order Placement:** "Place Order" button with API call to `POST /api/orders`, loading state during order processing, success message display, automatic cart clearing after successful order, and continue shopping option
- **Empty Cart State:** Message when cart is empty with button to return to browse page
- **Success State:** Confirmation message "Order Placed Successfully!", impact message showing rescue meals saved, and option to continue shopping

5.4 API Endpoints

5.4.1 Node.js Backend (Express)

- `POST /login` – Customer authentication
  - Request body: `{email: string, password: string}`
  - Returns: `{success: boolean, user: Object}`
- `POST /register` – Customer registration
  - Request body: `{name: string, email: string, password: string}`
  - Returns: `{success: boolean, message: string}`
- `POST /api/orders` – Place order
  - Request body: `{items: Array, totals: Object}`
  - Returns: `{success: boolean, order: Object, message: string}`
  - Updates restaurant points automatically
- `GET /api/restaurant-points` – Get restaurant points
  - Returns: `{success: boolean, points: Object}`
- `GET /api/orders` – Get order history
  - Returns: `{success: boolean, orders: Array}`
- `GET /restaurants` – Get restaurant data
  - Returns: Array of restaurant objects from CSV
- `GET /impact` – Get impact statistics

  - Returns: `{mealsRescued: number, wastePreventedTons: number, communityImpact: number}`

- `GET /` – Root endpoint
  - Returns: "Backend is running successfully!"

5.4.2 Python Flask API (Legacy)

- `GET /api/efficiency-waste-correlation` – Returns correlation and regression analysis results between delivery efficiency and waste metrics
- `GET /api/health` – Health check endpoint for API status
- `GET /` – Root endpoint with API information and available endpoints

Response Format:
The efficiency-waste correlation endpoint returns JSON with:

- Correlation coefficients (Pearson and Spearman)
- Regression model coefficients and R^2 scores
- Summary of strong correlations
- Number of restaurants analyzed

6 Efficiency Scoring System

6.1 Scoring Methodology

The efficiency score is a composite metric that evaluates restaurant performance across multiple dimensions:

- **On-Time Rate (40%):** Weighted highest as it directly impacts customer satisfaction
- **Delivery Time (30%):** Normalized inverse of average delivery time (faster = higher score)
- **Distance Efficiency (20%):** Normalized inverse of average delivery distance (shorter = higher score)
- **Delivery Volume (10%):** Normalized deliveries per day (higher = better, indicating operational capacity)

6.2 Reward System

When customers place orders, especially for rescue meals:

1. Restaurants receive **points** (10 points per rescue meal ordered)
2. Points are stored in `restaurant_points.json` and persist across sessions
3. Restaurant efficiency scores can be **incremented** based on:
    - Number of rescue meals sold
    - Points accumulated
    - Customer feedback ratings
    - Waste reduction metrics
4. Higher efficiency scores improve restaurant visibility and ranking in the platform
5. Orders are tracked in `orders.json` with full details including order ID, timestamp, items ordered, totals and savings, and points earned per restaurant

This creates a positive feedback loop: restaurants that actively reduce waste receive better scores, improved visibility, and more customer orders.

7 Data Analysis and Visualization

7.1 Statistical Analysis

The `analysis/correlate_efficiency_waste.py` module performs:

- Pearson correlation analysis between efficiency metrics and waste data
- Spearman rank correlation for non-parametric relationships
- Linear regression modeling to predict waste based on efficiency metrics
- Summary statistics identifying strong correlations for actionable insights

7.2 Visualization

The `visualization/efficiency_waste_viz.py` module generates:

- Scatter plots showing efficiency vs. waste relationships
- Time series charts for waste trends
- Bar charts comparing restaurant efficiency scores
- Heatmaps for correlation matrices

8 Testing

8.1 Frontend Testing

The frontend includes unit tests using Jest and React Testing Library:

- `CustomerLogin.test.jsx` – Tests login form validation and submission
- `HomePage.test.jsx` – Tests homepage rendering and navigation
- `LoginChoice.test.jsx` – Tests login choice selection

Run tests with:

```
cd src/frontend

npm test
```

8.2 Backend Testing

Backend modules can be tested by:

1. Verifying data generation produces expected CSV files
2. Checking data loading produces clean, integrated datasets
3. Validating metric calculations against known values
4. Testing API endpoints with sample requests:
    - Test order placement with sample cart data
    - Verify restaurant points are incremented correctly
    - Check order history is saved properly
    - Validate authentication endpoints

8.3 Integration Testing

Test the complete order flow:

1. Log in as a customer
2. Add rescue meals to cart
3. Navigate to cart page
4. Verify cart items display correctly
5. Place an order
6. Verify success message appears
7. Check restaurant points are updated
8. Verify order appears in order history

9 Troubleshooting

- **ModuleNotFoundError:**

  Ensure all dependencies are installed via

  `pip install pandas numpy flask flask-cors` and `npm install` in backend and frontend directories.

- **FileNotFoundError:**

  Run `data_generator.py` and `rescue_meals_integration.py` before executing downstream modules.

- **Empty Dataset:**

  Check that the data directory is correctly generated and accessible. Verify file paths in configuration.

- **Frontend Build Errors:**

  Ensure Node.js and npm are properly installed. Delete `node_modules` and run `npm install` again.

- **API Connection Errors:**

  Verify the Express API server is running on port 5000. Check CORS configuration if frontend cannot connect.

- **Port Already in Use:**

  If port 3000 (React) or 5000 (Express) is occupied, change ports in configuration or stop the conflicting process.

- **Cart Not Updating:**

  Verify cart state is properly managed in Dashboard component. Check browser console for errors.

- **Order Placement Fails:**

  Ensure backend server is running and `data/` directory exists with write permissions. Check that `restaurant_points.json` and `orders.json` are initialized.

If persistent issues occur, verify your Python and Node.js environments, check file paths, and consult GitHub issues for support.

10 Release History

- **v1.0.0 (Initial Release)**

  Complete modular pipeline for generating, cleaning, and analyzing synthetic data for restaurant efficiency scoring. Full-stack web application with React frontend and Express backend. Customer login, restaurant browsing, rescue meal integration, cart functionality, and order placement system. Restaurant points tracking and order history. Statistical analysis and visualization modules.

Future versions may include:

- Restaurant login and dashboard for managing rescue meals
- Real-time order tracking and status updates
- Payment processing integration
- Enhanced visualization dashboards with interactive charts
- Mobile application support
- Real-time notifications for rescue meal availability
- Advanced analytics dashboard for restaurants
- Customer order history and impact tracking
- Integration with real-world payment and delivery systems

11 Revision Control

Documentation and source code are managed under GitHub version control.

All revisions, commits, and updates are publicly visible in:
https://github.com/YashDhavale/CSC-510-SE-Project

Each commit is associated with a descriptive message that records bug fixes, enhancements, or documentation improvements. Version control ensures traceability and collaborative reliability.

12 Conclusion

TiffinTrails integrates data engineering, analytics, and web development to create a comprehensive platform for reducing food waste and promoting sustainable dining practices.

The modular architecture spans backend data processing, REST API services (both Node.js/Express and Python/Flask), and modern frontend interfaces, following professional software engineering standards. The reward-based system creates incentives for restaurants to minimize waste while providing customers with eco-friendly, affordable meal options through an intuitive cart and order placement system.

The cart functionality enables seamless order management with quantity controls, item removal, and real-time price calculations. The order placement system automatically tracks restaurant points, creating a transparent and rewarding experience for both customers and restaurants. This implementation demonstrates the team's ability to design structured, traceable, and extensible software, combining automation, analytics, and user experience design for better data-driven decisions and environmental impact.

13 Troubleshooting

This section outlines common issues that may occur during setup or execution, along with their corresponding solutions.

- **Issue:** "ModuleNotFoundError" or missing library during execution. **Solution:** Ensure all dependencies are installed. Run:

```
pip install -r requirements.txt
cd src/frontend
npm install
```

- **Issue:** Frontend fails to start with an error like "Port already in use". **Solution:** Kill the process using that port or change it in `package.json`:

```
npx kill-port 3000
npm start
```

- **Issue:** Backend not serving data or CSV not found. **Solution:** Verify that data files (e.g., `Restaurant_Metadata.csv`, `Delivery_Metrics.csv`) exist in the correct directory and paths match in the script.

- **Issue:** Blank or unresponsive frontend after starting the server. **Solution:** Clear the browser cache or ensure the backend Flask server is running before starting the React frontend.

- **Issue:** Git merge conflicts when pushing updates. **Solution:** Use:

```
git pull origin main --rebase
git push origin main
```

  and resolve conflicts manually if needed.

14 Release History

Version	Description
v1.0.0	Initial project release including backend computation modules, React-based frontend, and core data integration for restaurant efficiency and waste analysis.
v1.1.0 (Planned)	Add restaurant portal dashboard with analytics and interactive UI components. Implement API for real-time data retrieval and visualization.
v1.2.0 (Planned)	Integrate Smart Insights engine, gamified reward system, and payment gateway. Expand sustainability tracking through external API connections.

15 References

- Python Software Foundation. Python Language Reference, version 3.10.
- https://pandas.pydata.org/
- https://numpy.org/
- https://reactjs.org/
- https://expressjs.com/
- https://flask.palletsprojects.com/
- https://tailwindcss.com/
- https://jestjs.io/