Software Design Document

**Greenshoes**

Development team:

Boxin Yang

Nongxu Li

Kunye Tang

Distribution clause:

This document was created as a deliverable for the Spring 2025 CSCi 6235 course at the George Washington University, 1918 F Street, NW, Washington, DC 20052. Distribution is unrestricted provided the source is referenced.

**REVISION HISTORY**

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Date** | **Affected Pages** | **Explanation of Change** |
| 0.1 | 04/11 | 1-8 | First edition |
| 0.2 | 04/16 | 1-15 | Adhere to client requests on 04/11 |
|  |  |  |  |
|  |  |  |  |

TABLE OF CONTENTS

[Scope 1](#_Toc130663781)

[Overview 1](#_Toc130663782)

[Theoretical background and definition of terms 1](#_Toc130663783)

[Software Architecture 1](#_Toc130663784)

[Software Bill Of Materials (SBOM) 1](#_Toc130663785)

[File Structure 1](#_Toc130663786)

[Class structure 1](#_Toc130663787)

[Data Design 1](#_Toc130663788)

[Object structure 1](#_Toc130663789)

[Database Model <if any> 1](#_Toc130663790)

[Inter-Process Communication <if any> 2](#_Toc130663791)

[Software Functionality 2](#_Toc130663792)

[Use Cases 2](#_Toc130663793)

[Function execution 2](#_Toc130663794)

# Scope

# Greenshoes is a web application designed to provide users with an interactive and seamless shopping experience for high-quality footwear. The system integrates dynamic product listings, user authentication, and order management in a scalable and secure architecture.

# Overview

## The Greenshoes application is a multi-tier web platform built using modern technologies like Next.js for the frontend, Express.js for the backend, and PostgreSQL for database management. The platform supports various functionalities, including product browsing, user registration, and order processing, with a focus on performance and user experience.

## Theoretical background and definition of terms

# Next.js: A React-based framework enabling server-side rendering (SSR) and static site generation (SSG).

# Express.js: A minimalist web framework for Node.js to build robust Application Programming Interface (APIs).

# PostgreSQL: An open-source relational database for secure and efficient data management.

# NextAuth.js: A library for implementing secure user authentication in Next.js.

# Software Architecture

The Greenshoes architecture employs a modular design with clear separation of concerns:

* **Frontend:** Built with Next.js to handle SSR/SSG and user interface.
* **Backend:** An Express.js server managing APIs and connecting to the database.
* **Database:** PostgreSQL storing normalized tables for users, products, orders, and more.

A diagram of a program

AI-generated content may be incorrect.

## Software Bill Of Materials (SBOM)

|  |  |  |  |
| --- | --- | --- | --- |
| **Component Name** | **Version** | **Purpose** | **Source URL** |
| next | 15.2.3 | Enables SSR and SSG | https://nextjs.org/ |
| react | 18.0.0 | Core UI library | https://react.dev/ |
| express | 4.17.1 | API creation and middleware management | https://expressjs.com/ |
| pg | 8.7.1 | PostgreSQL database client for Node.js | https://node-postgres.com/ |
| next-auth | 4.22.1 | Simplified authentication for Next.js | https://next-auth.js.org/ |
| dotenv | 16.4.7 | Environment variable management | https://www.npmjs.com/package/dotenv |

## File Structure

## greenshoes/

## ├── components/ # Reusable frontend components

## ├── db/ # Database connection and schema files

## ├── pages/ # Frontend page structure

## ├── public/ # Static assets like images and favicon

## ├── routes/ # Express API routes

## ├── styles/ # CSS files

## ├── utils/ # Utility functions

## ├── .env # Environment variables

## ├── package.json # Project dependencies

## ├── README.md # Project overview and setup instructions

## └── server.js # Entry point for the backend server

## Class structure

# A diagram of a computer program AI-generated content may be incorrect.

**Components Overview:**

1. **CartUtils**
   * Provides utility methods for managing the shopping cart.
   * Methods:
     + addToCart(product, quantity): Adds a product to the cart with the specified quantity.
     + removeFromCart(productId): Removes a product from the cart using its ID.
2. **ExpressServer**
   * Manages the server functionality and routes requests to appropriate components.
   * Methods:
     + connectDatabase(): Establishes a connection to the database.
     + useMiddleware(): Configures middleware for request handling.
     + routeRequests(): Directs incoming requests to corresponding modules.
     + startServer(): Starts the web server.
3. **ProductManagement**
   * Handles operations related to product data.
   * Methods:
     + getAllProducts(): Fetches a list of all products.
     + getProductById(id): Retrieves details of a specific product by its ID.
     + createProduct(product): Creates a new product.
     + updateProduct(product): Updates an existing product.
     + deleteProduct(id): Deletes a product by its ID.
4. **AuthenticationHandler**
   * Manages user authentication and session handling.
   * Methods:
     + authorize(credentials): Validates user credentials and returns user data.
     + jwtCallback(token, user): Handles JSON Web Token (JWT) operations.
     + sessionCallback(session, token): Manages session-token synchronization.
5. **UserManagement**
   * Manages user-related operations, such as account creation and updates.
   * Methods:
     + signup(userData): Creates a new user account.
     + signin(credentials): Authenticates a user using their credentials.
     + updateProfile(userData): Updates user profile information.
6. **CheckoutAndOrders**
   * Handles the checkout process and order management.
   * Methods:
     + checkoutOrder(orderData): Processes an order and initiates checkout.
     + getOrder(orderId): Retrieves details of a specific order.
7. **DatabasePool**
   * Provides an interface for database operations.
   * Properties:
     + pool: Represents the database connection pool.
   * Methods:
     + connectDB(): Establishes a connection to the database.
     + query(sql, params): Executes SQL queries with parameters.
8. **Layout**
   * Supports rendering of user interface elements.
   * Methods:
     + renderHeader(): Renders the page header.
     + renderFooter(): Renders the page footer.
     + wrapPage(content): Wraps the main content within a layout structure.

# Data Design

## Object structure

## Critical in-memory data structures:

## **Cart Object**: Tracks product IDs, quantities, and prices during a session.

## **Session Token:** Stores authentication details for logged-in users.

## Database Model

A screenshot of a computer

AI-generated content may be incorrect.

**Schema Overview**

1. **Users**:  
   Stores information about customers, including their email, hashed password, full name, contact details, address, and payment information (securely encrypted).
   * **Primary Key**: id (UUID).
   * Includes fields for tracking the last 4 digits of the credit card and storing encrypted payment details for security.
2. **Admins**:  
   A separate table for administrative users who manage the platform but are not allowed to shop.
   * **Primary Key**: id (UUID).
   * Stores admin credentials securely.
3. **Products**:  
   Contains details about the products (shoes) available for purchase.
   * **Primary Key**: id (UUID).
   * Includes fields for product name, description, price, stock quantity, and category (e.g., running, casual, formal).
   * Tracks creation and update timestamps for inventory management.
4. **Product Images**:  
   Stores image URLs for products to enhance the user shopping experience.
   * **Primary Key**: id (UUID).
   * References the products table to associate images with specific products.
   * Allows marking an image as the primary display image.
5. **Orders**:  
   Tracks customer orders, including total price, order status (e.g., pending, paid, shipped), and timestamps.
   * **Primary Key**: id (UUID).
   * References the users table to associate orders with customers.
6. **Order Items**:  
   Details the products included in each order, including quantity and snapshot of the product price at the time of the order.
   * **Primary Key**: id (UUID).
   * References both orders and products tables to link specific products to orders.
7. **User Order History**:  
   Maintains a historical record of user orders for easy tracking and review.
   * **Primary Key**: id (UUID).
   * References the orders table and users table to log each user's order lifecycle.
   * Stores the status of each order for historical reference.
8. **Enums**:  
   Predefined enumerations for key fields to ensure data integrity:
   * **Order Status**: Defines possible statuses for orders (pending, paid, shipped, delivered, canceled).
   * **Shoe Categories**: Specifies the categories of shoes (running, casual, formal, hiking, sports).

## Inter-Process Communication

# Ports used:

# **Frontend to Backend:** HTTP/HTTPS on port 3000.

# **Backend to Database:** PostgreSQL default port 5432.

# Software Functionality

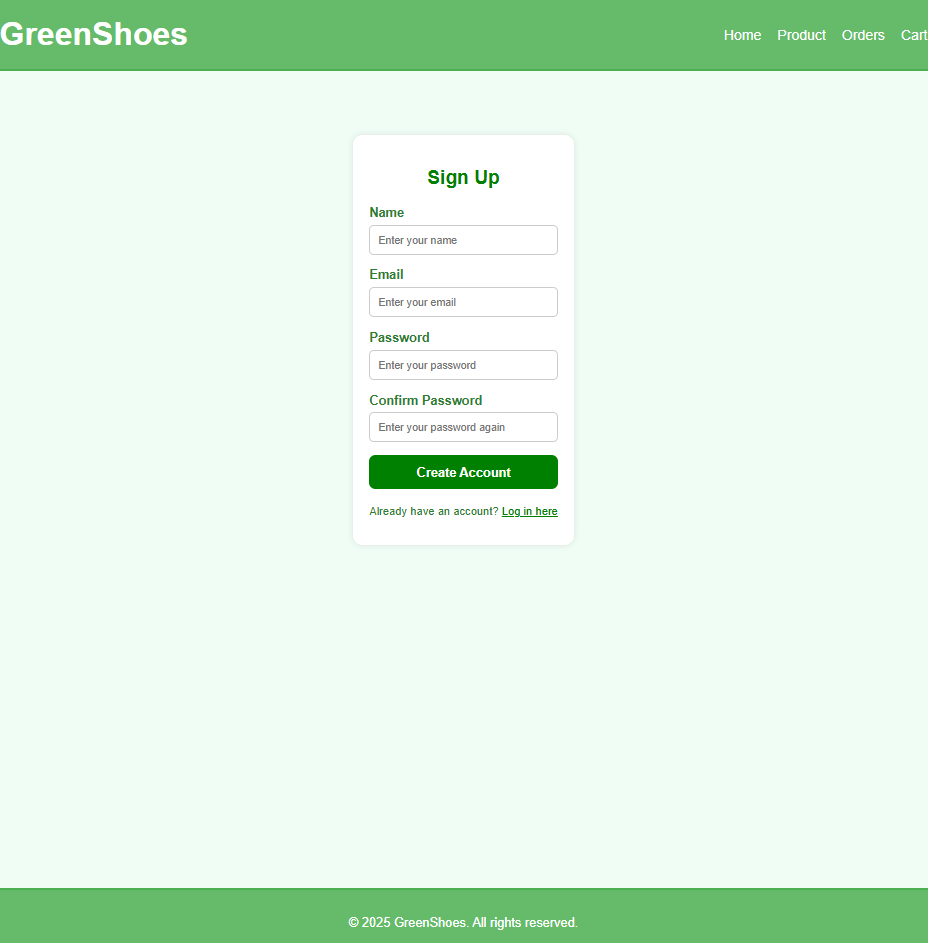
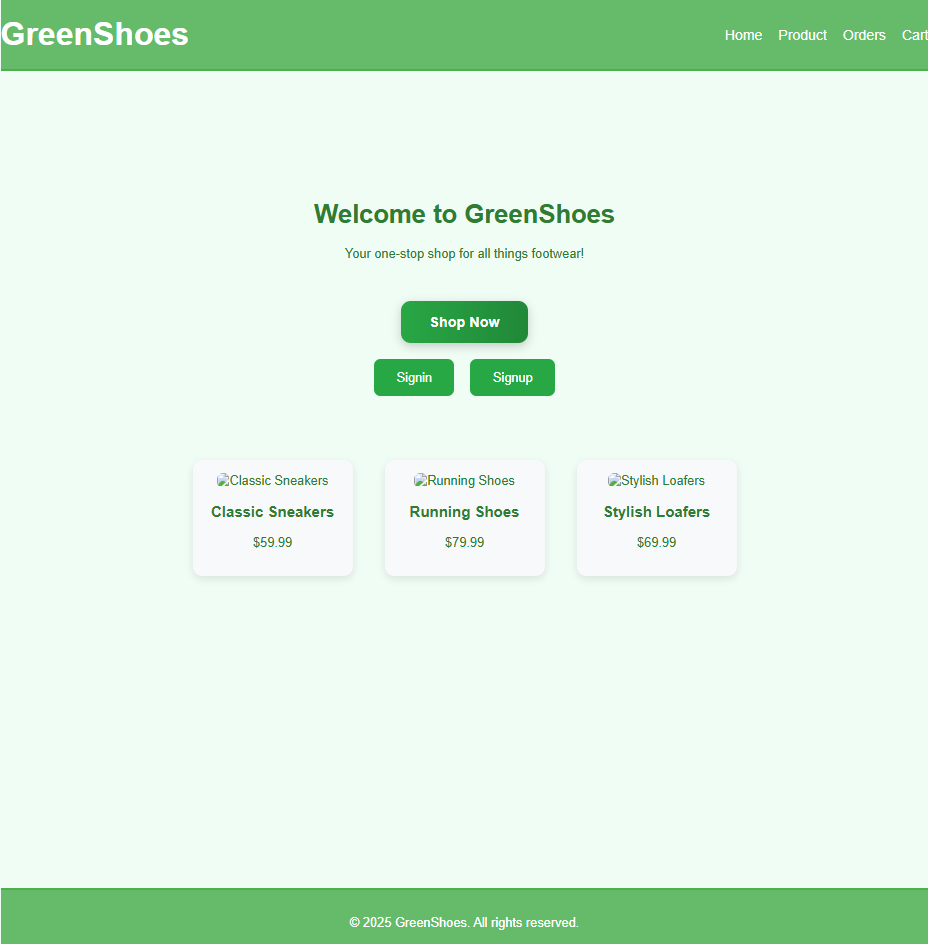
## Use Cases

## **User Registration:**

## A new user registers on the platform by providing their email and password.

## The backend validates the input and creates a new user in the database.

## The user receives a confirmation message on successful registration.

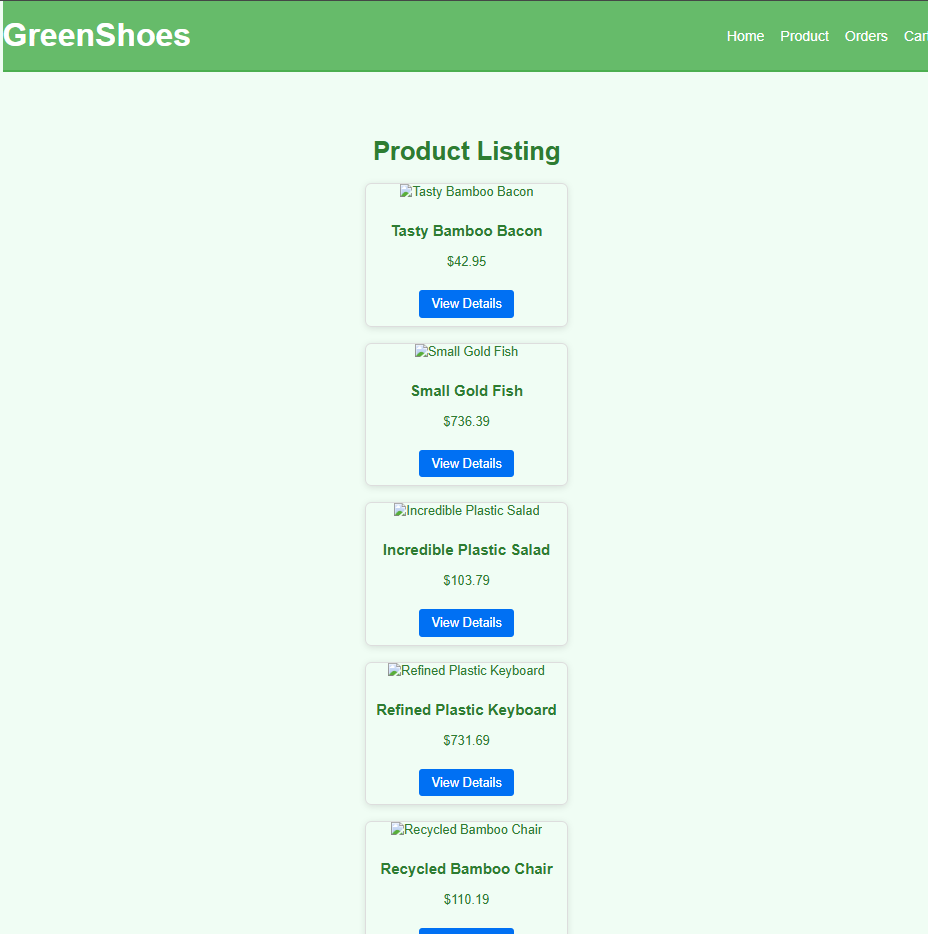


Type in the information

## **Product Browsing:**

## Users navigate the product listing page.

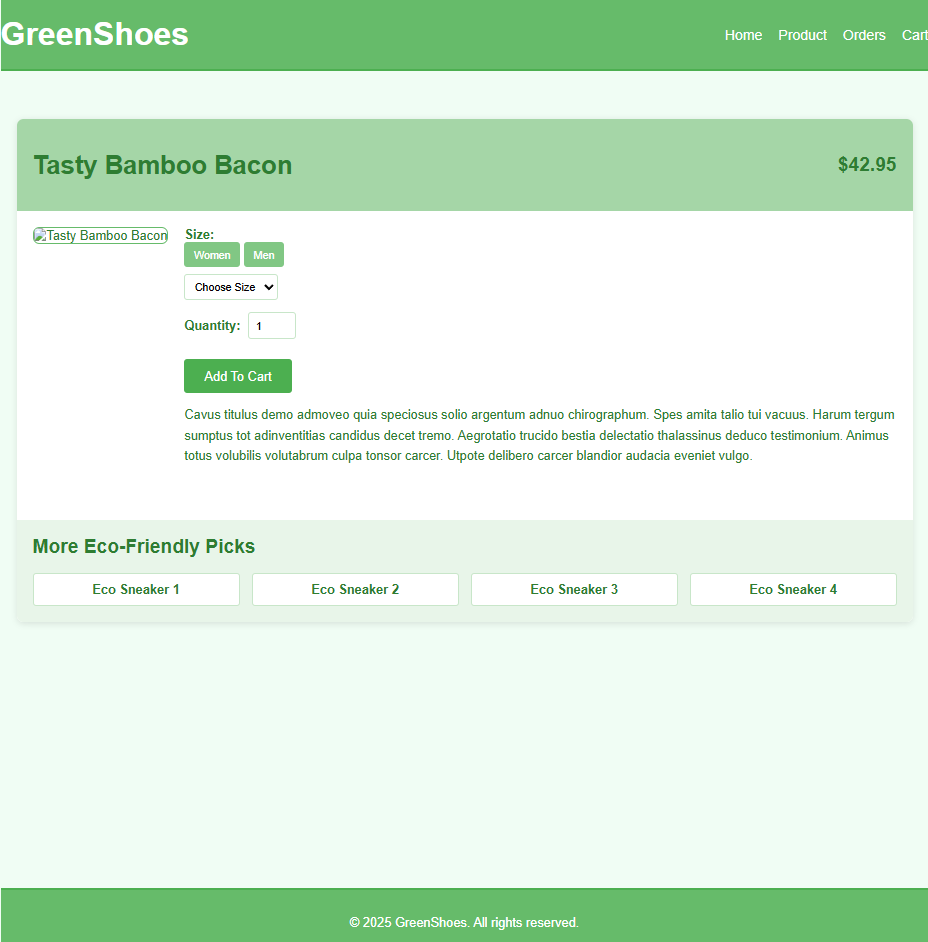
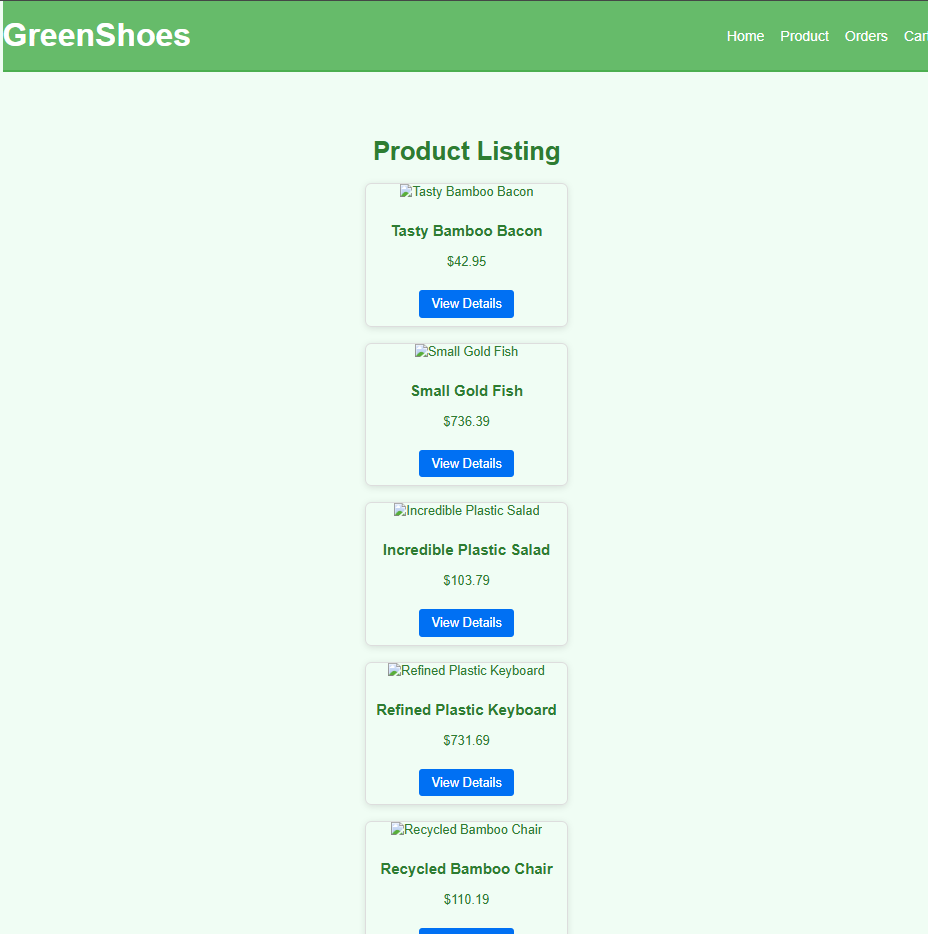
## The frontend fetches product data from the backend and displays it dynamically.

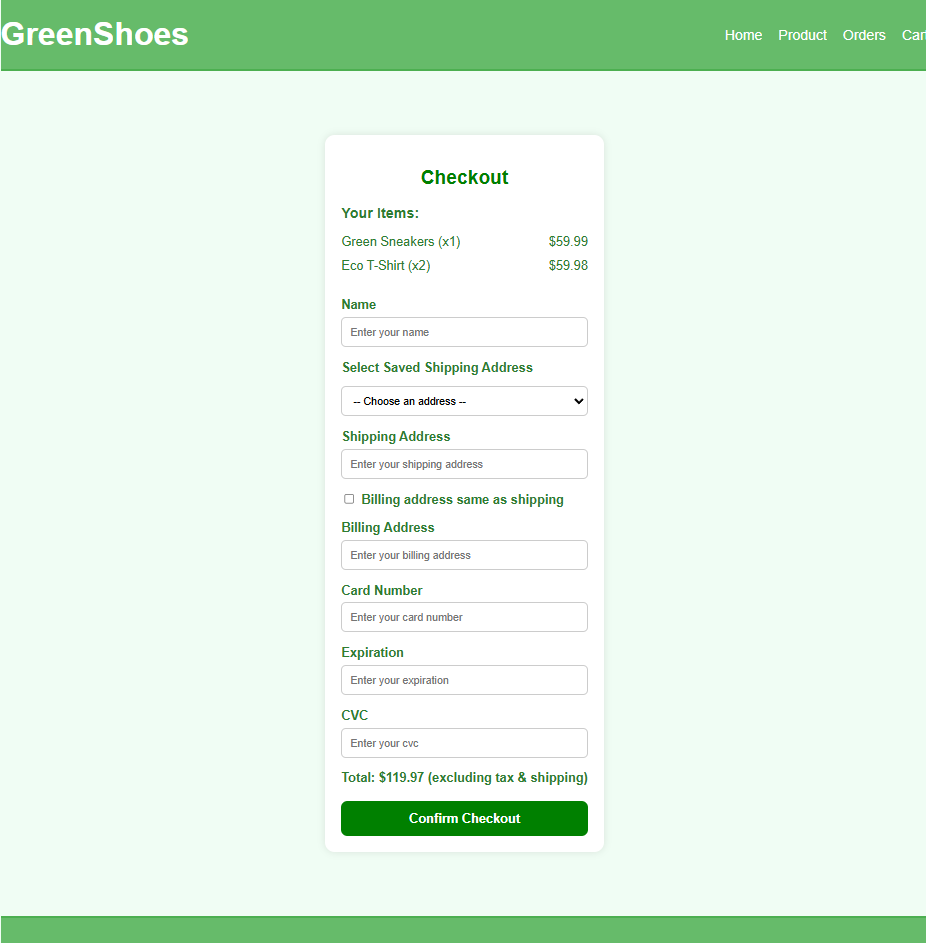
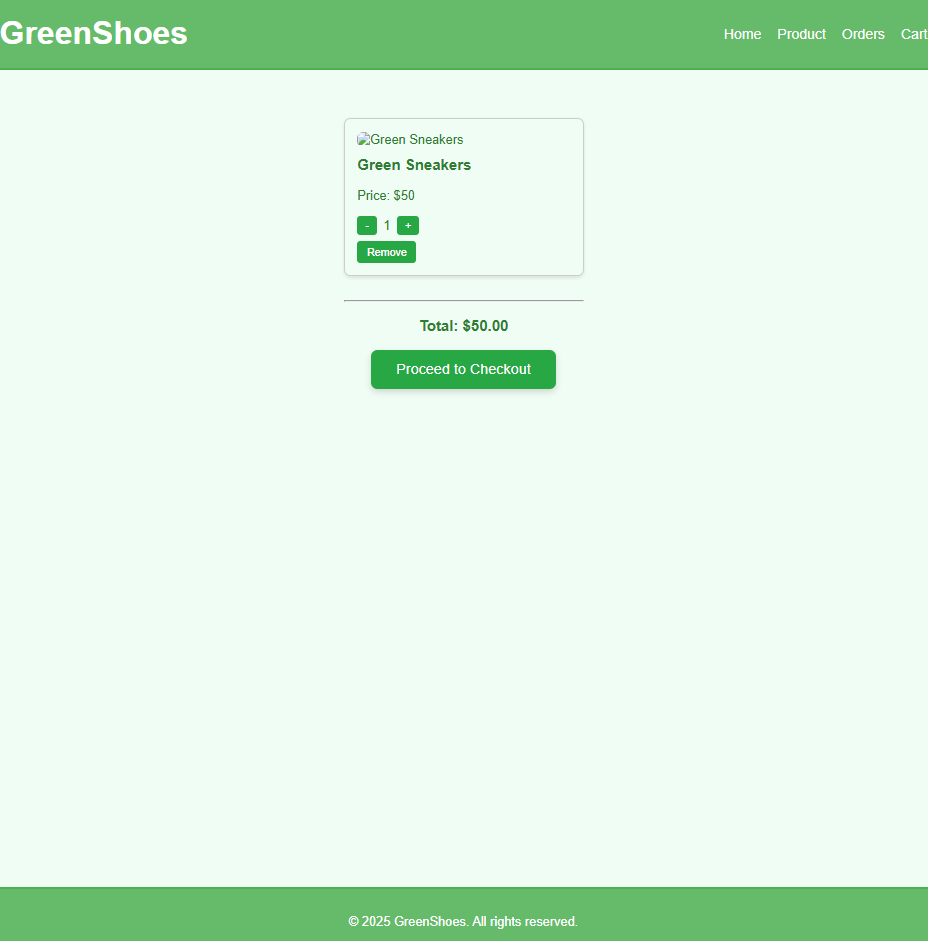


## **Add to Cart:**

## A user selects a product and quantity to add to their cart.

## The backend updates the user's cart session data.



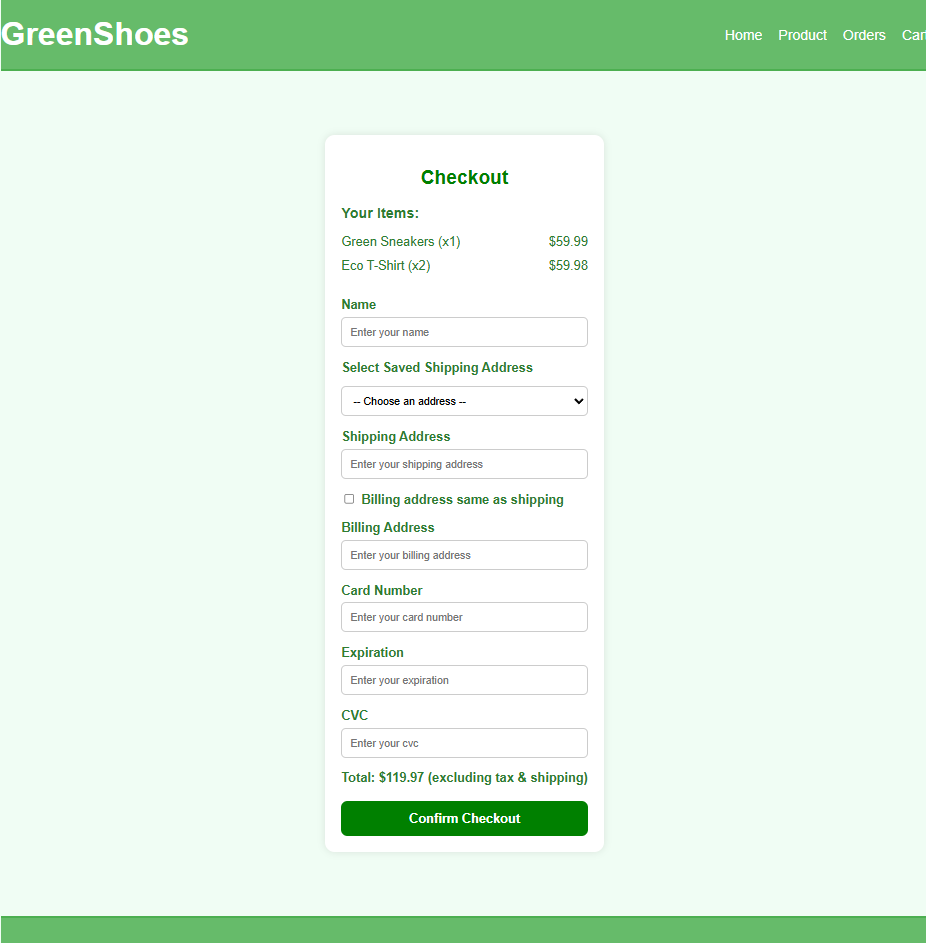


## **Checkout:**

## The user reviews their cart and provides payment and shipping details.

## The backend processes the payment and creates an order in the database.

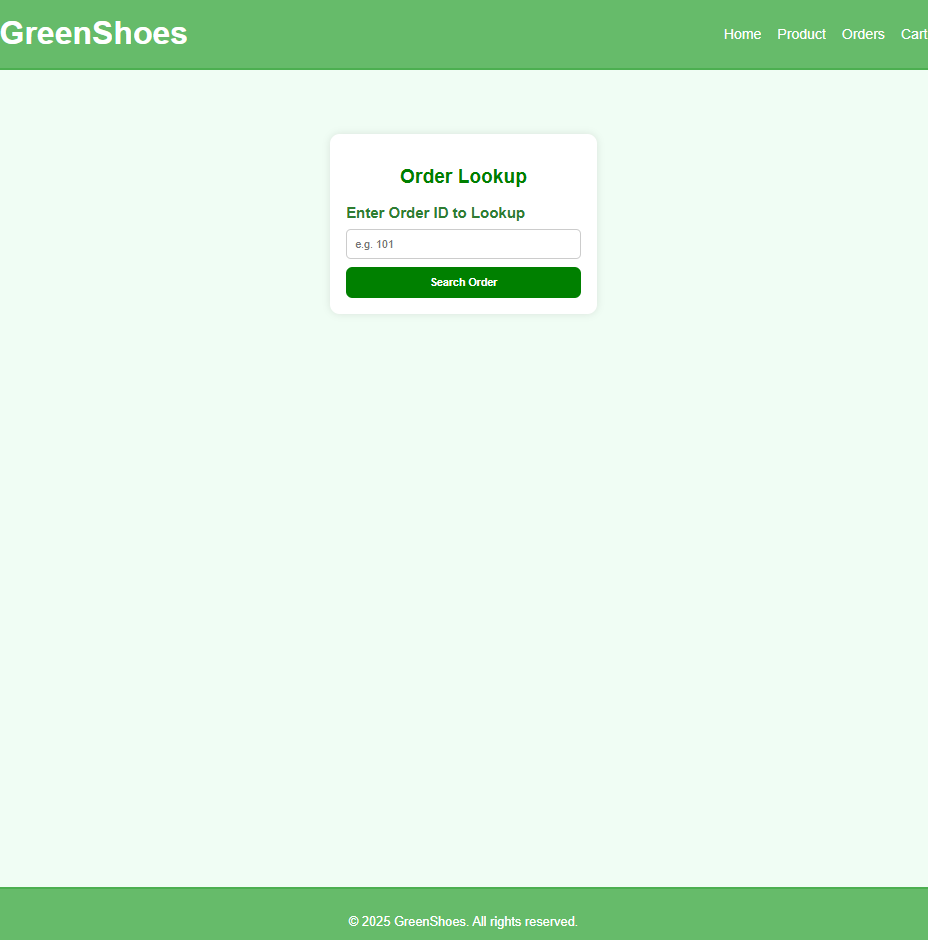
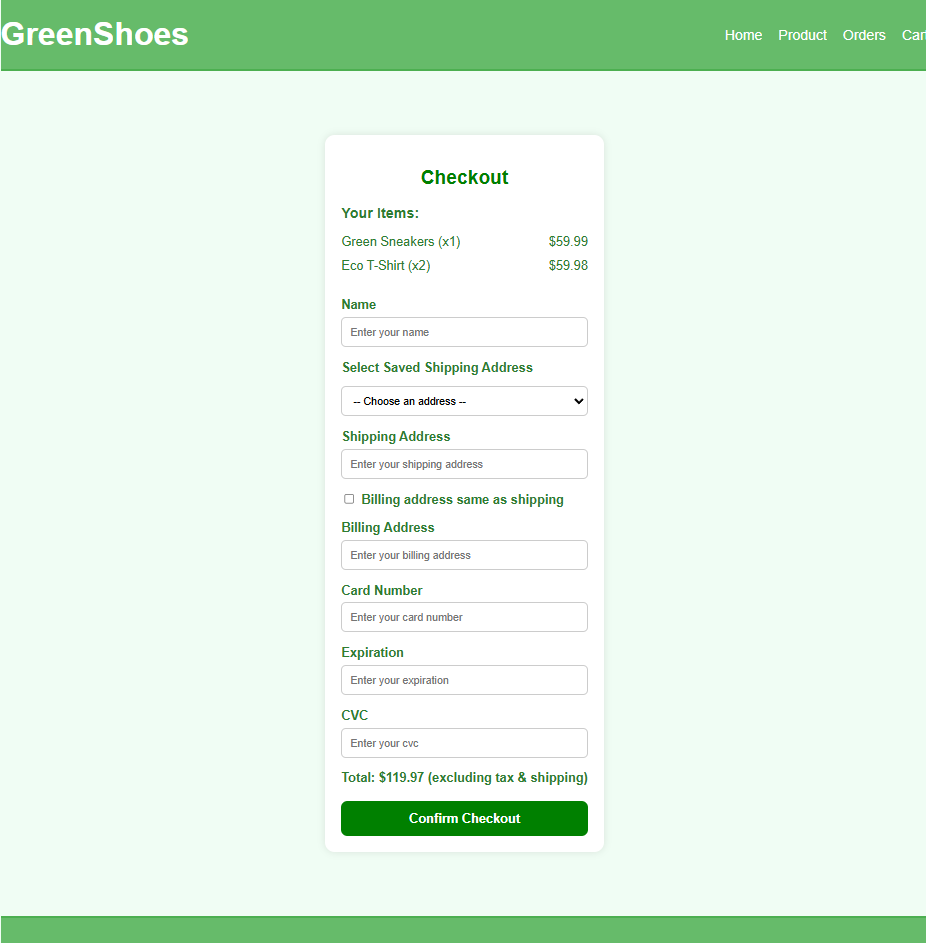
## The user receives a confirmation email with order details.



## **Order History:**

## A logged-in user navigates to their order history page.

## The backend retrieves and displays the user's past orders.



## Function execution

1. **User Registration Execution:**
   * **File:** pages/api/auth/register.js
   * **Workflow:**
     1. Receive user details from the frontend.
     2. Hash the password using bcrypt.
     3. Insert the user into the database.
2. **Product Browsing Execution:**
   * **File:** pages/api/products.js
   * **Workflow:**
     1. Query the database for all available products.
     2. Return the data to the frontend for rendering.
3. **Add to Cart Execution:**
   * **File:** utils/cartUtils.js
   * **Workflow:**
     1. Add or update the product in the user's cart session.
     2. Persist the updated cart in the database if the user is logged in.
4. **Checkout Execution:**
   * **File:** pages/api/orders.js
   * **Workflow:**
     1. Validate the cart and payment details.
     2. Deduct product stock in the database.
     3. Create an order entry and send a confirmation email.
5. **Order History Execution:**
   * **File:** pages/api/orders/history.js
   * **Workflow:**
     1. Fetch the user ID from the session.
     2. Query the database for all orders associated with the user.
     3. Return the data to the frontend for display.