Software Design Document

**Greenshoes**

Development team:

Boxin Yang

Nongxu Li

Kunye Tang

Distribution clause:

This document was created as a deliverable for the Fall 2024 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** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

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 APIs.

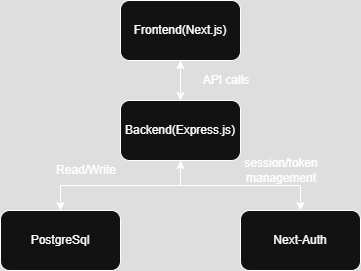
# PostgreSQL: An open-source relational database for secure and efficient data management.

# 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.



## 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.

# 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.

## 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.

## **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.