# INFSCI 2710 - Database Management Systems

## Final Project Design Document Template

* **Title page**
  + Your team (for each teammate):
    - Jiani Cheng
    - [JIC281@pitt.edu](mailto:JIC281@pitt.edu)
    - Yue Li
    - [YUL478@pitt.edu](mailto:YUL478@pitt.edu)
    - Zhuoju Li
    - [ZHL309@pitt.edu](mailto:ZHL309@pitt.edu)
  + Course name: INFSCI 2710 - Database Management Systems
  + Instructor: Dmitriy Babichenko
  + Date: 4/21/2024

* **Table of contents**

1. Introduction/Abstract
2. E-R Model
3. Business rules
4. Architecture Design
5. API Design
6. Testing Plan
7. **Introduction/Abstract**

* A brief 1-2 paragraph project description:

GreenStep is an innovative application designed to revolutionize the hiking gear industry by promoting and integrating sustainable and eco-friendly practices. The platform aims to cater to the growing market of environmentally conscious consumers by offering a wide range of high-quality, sustainable hiking products. By leveraging advanced technology and sustainable business practices, GreenStep seeks to minimize the ecological footprint of outdoor activities, promoting a greener approach to hiking.

* Description of your target audience:

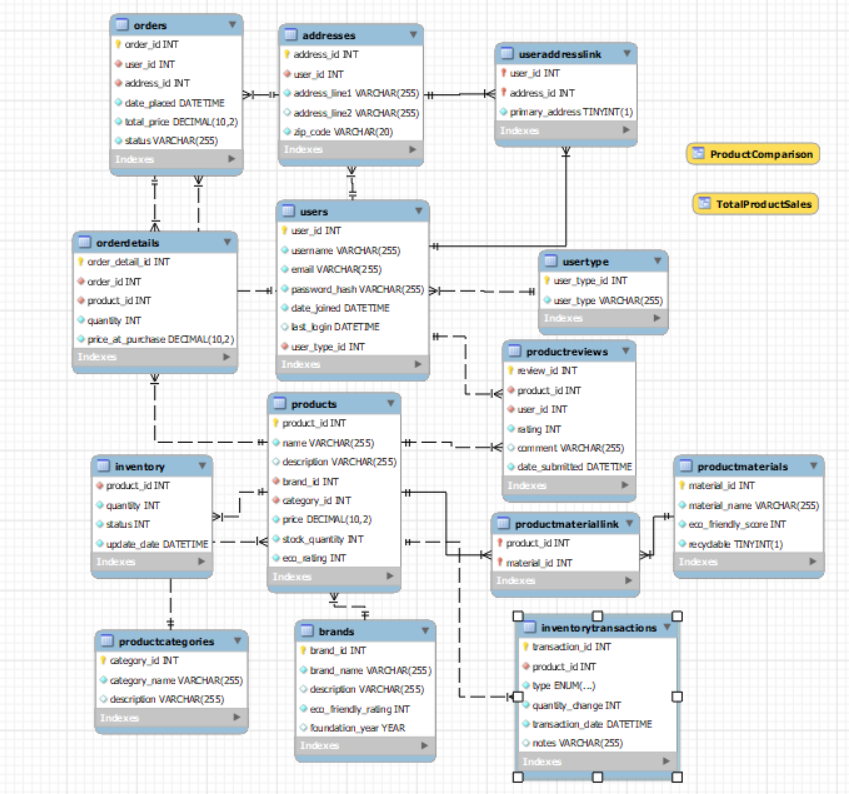
This design document is intended for multiple audiences within the GreenStep project team.

* Purpose and benefits of your database:

The primary purpose of this database is to efficiently manage user data, product information, and transaction details in a way that supports scalability, security, and robust data retrieval. The database will enable dynamic product listing, user authentication and management, inventory control, and environmental impact assessments. This structured approach not only enhances user experience but also supports GreenStep’s mission of promoting sustainability in outdoor activities by providing detailed insights into the eco-friendliness of products.

1. **E-R Model**

* You may export and include the diagram using the MySQL “reverse engineer” feature



1. **Business rules**Use the table below to list all business rules and link them to E-R Model cardinalities. For many-to-many relationships, make sure that your E-R model contains appropriate junction tables. You still need to specify cardinalities of each relationship. Use the first line in the table as an example:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity 1** | **Entity 2** | **Cardinality on Entity 1 side** | **Cardinality on Entity 2 side** | **Business Rule(s)** |
| Users | UserType | 1 | 0..\* | Each user has one user type, but each user type can have multiple users. |
| Users | UserAddressLink | 1..\* | 1 | Each user can have multiple address links, but each link is associated with one user. |
| UserAddressLink | Addresses | 1 | 1..\* | Each address link points to one address, but each address can have multiple links. |
| Products | Brands | 1 | 1..\* | Each product is made by one brand, but a brand can manufacture multiple products. |
| Products | ProductCategories | 1 | 1..\* | Each product belongs to one category, but each category can include multiple products. |
| Products | ProductReviews | 1..\* | 1 | Each product can have multiple reviews, but each review pertains to one product only. |
| Users | Orders | 1..\* | 1 | Each user can place multiple orders, but each order is placed by one user. |
| Orders | OrderDetails | 1 | 1..\* | Each order has multiple order details, but each detail pertains to one order. |
| Products | Inventory | 1 | 1 | Each product has associated inventory details, with each inventory entry corresponding to one product. |
| Inventory | InventoryTransactions | 1..\* | 1 | Each inventory record can have multiple transactions, but each transaction affects only one inventory record. |
| Products | ProductMaterials | 1..\* | 1..\* | Products are linked to multiple materials, and each material can be linked to multiple products via ProductMaterialLink. |
| ProductMaterials | ProductMaterialLink | 1..\* | 1 | Each material can be part of multiple links, but each link is associated with one material only. |

1. **Architecture Design**

* System Architecture:

The GreenStep system architecture is designed to support scalability, security, and high availability, consisting of three main layers:

* Front-End: Built with HTML, CSS, and JavaScript, ensuring responsive design across devices. It interfaces with the back-end through RESTful APIs.
* Back-End: Implemented in Python, handling all server-side logic, session management, and database interactions.
* Database: Utilizes MySQL for robust data management and complex query execution.
* System Interaction:

User requests are processed through API calls from the front-end to the back-end, which then interacts with the MySQL database to retrieve or update data, subsequently returning it to the front-end.

* Technology Stack:
* Front-End: HTML5, CSS3, JavaScript, Bootstrap.
* Back-End: Python.
* Database: MySQL.
* Tools: Git.

1. **API Design**

* API Endpoints:

GreenStep's backend exposes several RESTful API endpoints that facilitate operations related to orders, users, and products. Each endpoint supports various operations by handling specific HTTP methods (GET, POST, PUT, DELETE).

#### Orders API

* **GET /orders**
  + **Purpose:** Retrieves all orders from the database.
  + **Parameters:** None
  + **Success Response:** JSON array of order objects.
  + **Error Response:** **404 Not Found** if no orders are available.
* **POST /orders**
  + **Purpose:** Creates a new order.
  + **Required Parameters:** **address\_id**, **date\_placed**, **status**, **total\_price**, **user\_id**.
  + **Success Response:** **{ "message": "Order added successfully" }**, status **201 Created**.
  + **Error Response:** **400 Bad Request** if any required parameter is missing.
* **PUT /orders**
  + **Purpose:** Updates an existing order specified by **order\_id**.
  + **Required Parameters:** **order\_id**, **address\_id**, **date\_placed**, **status**, **total\_price**, **user\_id**.
  + **Success Response:** **{ "message": "Order updated successfully" }**, status **200 OK**.
  + **Error Response:** **400 Bad Request** if any required parameter is missing, **404 Not Found** if no order is found.
* **DELETE /orders**
  + **Purpose:** Deletes an order specified by **order\_id**.
  + **Required Parameter:** **order\_id**.
  + **Success Response:** **{ "message": "Order deleted successfully" }**, status **200 OK**.
  + **Error Response:** **404 Not Found** if no order is found.

#### Users API

* **GET /users**
  + **Purpose:** Retrieves all users from the database.
  + **Parameters:** None
  + **Success Response:** JSON array of user objects.
  + **Error Response:** **404 Not Found** if no users are found.
* **POST /users**
  + **Purpose:** Registers a new user.
  + **Required Parameters:** **username**, **email**, **password\_hash**, **date\_joined**, **user\_type\_id**.
  + **Optional Parameter:** **last\_login**.
  + **Success Response:** **{ "message": "User added successfully" }**, status **201 Created**.
  + **Error Response:** **400 Bad Request** if any required parameter is missing.
* **PUT /users/{user\_id}**
  + **Purpose:** Updates an existing user specified by **user\_id**.
  + **Required Parameters:** **username**, **email**, **password\_hash**, **date\_joined**, **user\_type\_id**, **user\_id**.
  + **Optional Parameter:** **last\_login**.
  + **Success Response:** **{ "message": "User updated successfully" }**, status **200 OK**.
  + **Error Response:** **400 Bad Request** if any required parameter is missing, **404 Not Found** if no user is found.
* **DELETE /users/{user\_id}**
  + **Purpose:** Deletes a user specified by **user\_id**.
  + **Required Parameter:** **user\_id**.
  + **Success Response:** **{ "message": "User deleted successfully" }**, status **200 OK**.
  + **Error Response:** **404 Not Found** if no user is found.

#### Products API

* **GET /products**
  + **Purpose:** Retrieves all products from the database.
  + **Parameters:** None
  + **Success Response:** JSON array of product objects.
  + **Error Response:** **404 Not Found** if no products are found.
* **POST /products**
  + **Purpose:** Adds a new product to the inventory.
  + **Required Parameters:** **name**, **brand\_id**, **category\_id**, **price**, **stock\_quantity**, **eco\_rating**.
  + **Optional Parameter:** **description**.
  + **Success Response:** **{ "message": "Product added successfully" }**, status **201 Created**.
  + **Error Response:** **400 Bad Request** if any required parameter is missing.
* **PUT /products/{product\_id}**
  + **Purpose:** Updates an existing product specified by **product\_id**.
  + **Required Parameters:** **name**, **brand\_id**, **category\_id**, **price**, **stock\_quantity**, **eco\_rating**, **product\_id**.
  + **Optional Parameter:** **description**.
  + **Success Response:** **{ "message": "Product updated successfully" }**, status **200 OK**.
  + **Error Response:** **400 Bad Request** if any required parameter is missing, **404 Not Found** if no product is found.
* **DELETE /products/{product\_id}**
  + **Purpose:** Deletes a product specified by **product\_id**.
  + **Required Parameter:** **product\_id**.
  + **Success Response:** **{ "message": "Product deleted successfully" }**, status **200 OK**.
  + **Error Response:** **404 Not Found** if no product is found.

1. **Test Case**

Platform: postman

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成