# Software Design Specification Document

DecalPro

Team Members: Nick Daniel, Elijah Pearce, Anthony Corona, Jake Valdez

# System Description

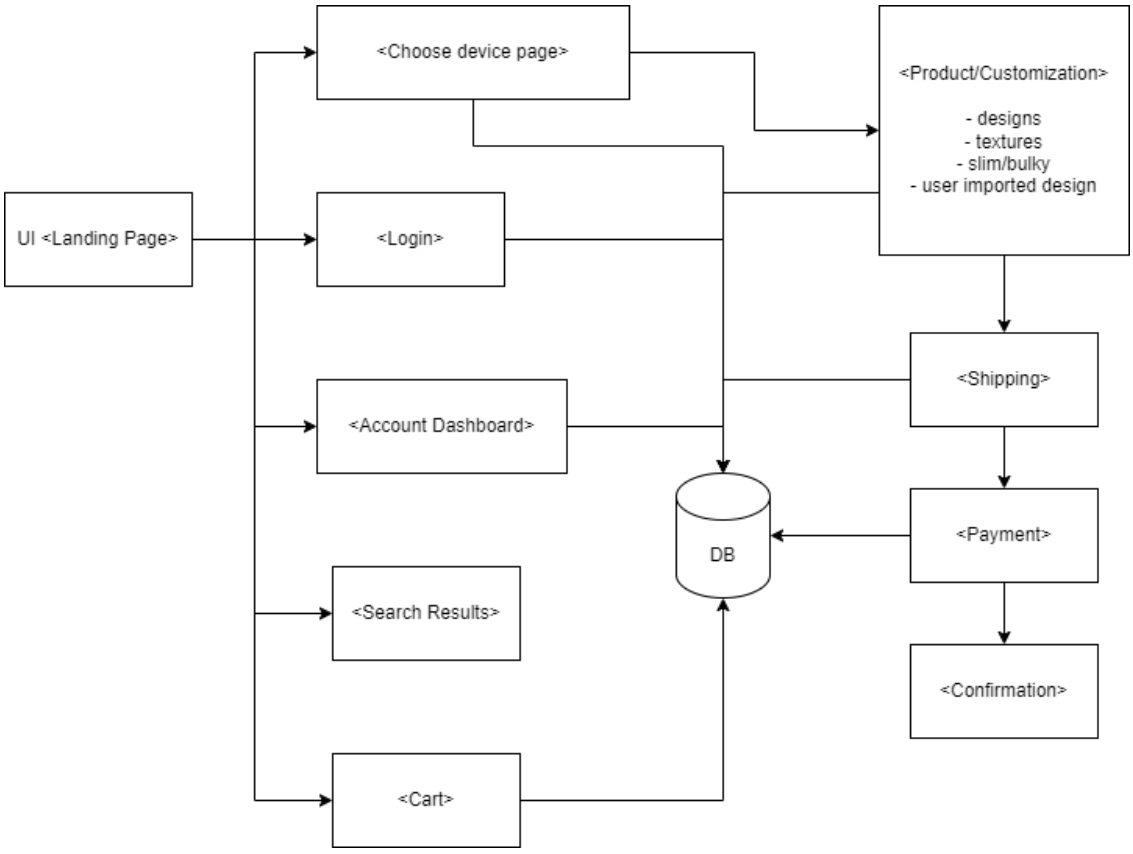
## Brief Overview of System

The DecalPro website is an e-commerce platform dedicated to selling a variety of decal products for electronic devices. It provides a user-friendly interface for customers to browse, select, and purchase decals for laptops, smartphones, tablets, gaming consoles, and other electronic devices. The system also includes features such as user registration, order management, payment processing, and product reviews.

# Software Architecture Overview

This section will contain a Software Architecture Diagram alongside a UML Class Diagram. The main elements of the website and how they interact are shown in the software architecture diagram. It gives a quick overview of the system's architecture. The essential classes, their attributes, and operations will be shown in the UML class diagram, providing a thorough understanding of the internal organization of the product.

## Software Architecture Diagram



## Landing Page

The homepage of the website. On this page, the user has links to login/register, to the user's own cart, the search bar, the gallery, and the choose device page.

## Choose Device Page

The Choose Device Page link is available on the landing page. It is a page that has all of the devices that we provide cases for. When you click on a device on the page, it will link to a results page that contains links for all of the products available for said device.

## Login Page

This is a page that the user is redirected to from the landing page. On the Login page, the user may enter their username/email and password or register an account for the website.

## Account Dashboard

The account dashboard is the central hub for managing a user’s account. From here, they can verify saved information, whether that be saved addresses or payment methods. The management of other account specific information (username, email, password, etc.) can also be accessed/changed through here.

## Search Results Page

The result page is the page that the user will be linked to after they use the search bar or the Choose Device Page. The page will contain all of the available products that fit the search criteria of the user's search input. If no product can be found that matches the search, “No result” will be displayed on the page.

## Cart Page

This page is used to record and show the user what they plan on purchasing as well as the accumulated price of all the items they have purchased. The User may add or take away products from their cart as they want.

## DB

This is the main database for the entire system. It will store the data required for user accounts, payments, product information, carts of users, and items. It is important to recognize, this database can only be accessed through the backend.

## Products / Customization Page

This page holds a list of the products and allows the user to change the designs, textures, and import designs for the products.

### Designs

This field is dedicated to showing the customer basic colors or default designs so that they can choose one for any case. This page can be accessed from the Product/Customization page.

### Textures

This field is dedicated to showing the customer all the available textures and that they can choose one for any case. This page can be accessed from the Product/Customization page.

### Slim/Bulky

This field is dedicated to showing the customer all the available slim and bulky cases and that they can choose one for any device.  This page can be accessed from the Product/Customization page.

### User-imported design

This field is dedicated to allowing the customer to import a JPEG picture so that they can put it on any device. This page can be accessed from the Product/Customization page.

## Shipping Page

This page allows the user to choose the desired address that they want the product delivered to.

## Payment Page

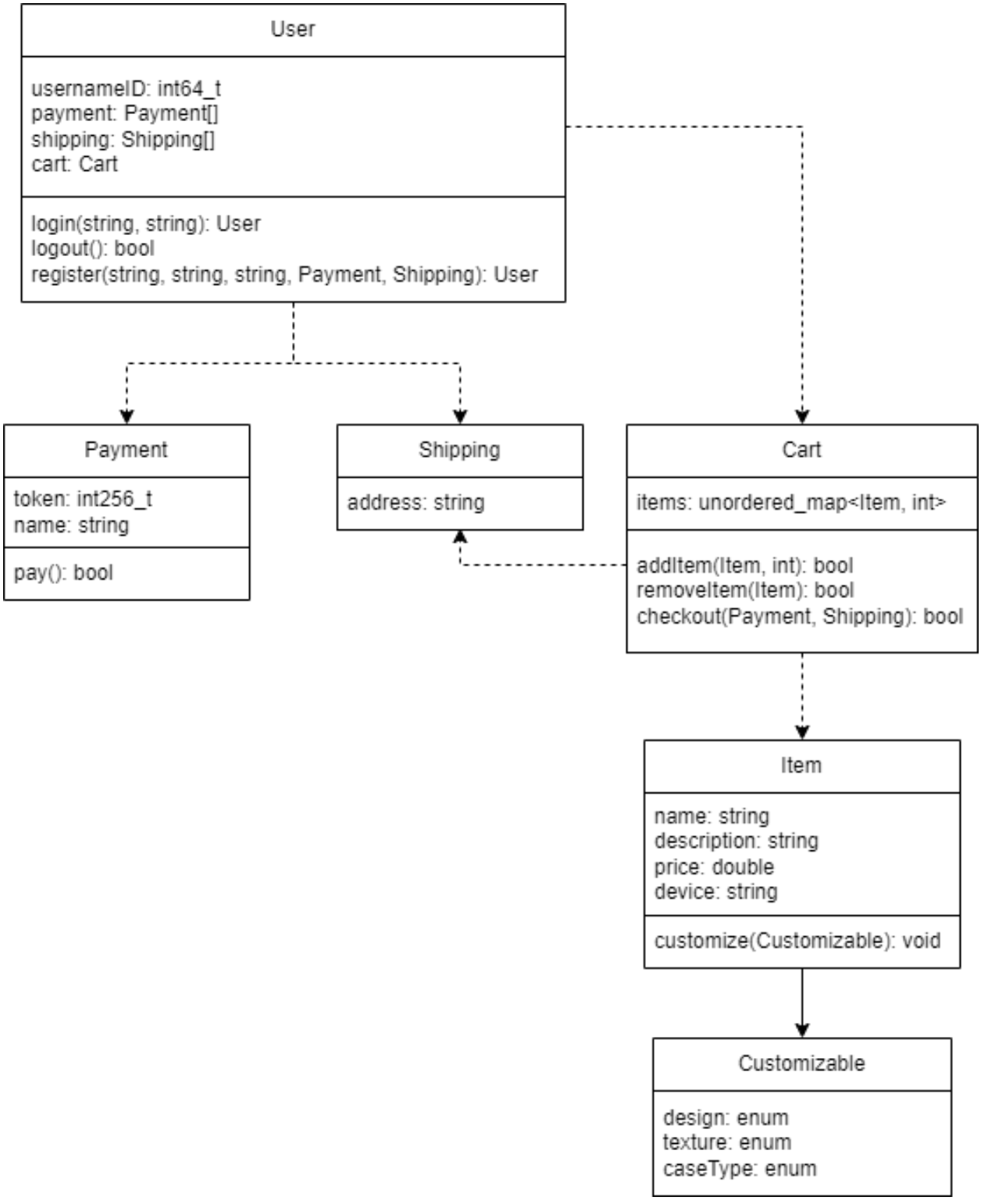
This is the page that allows the user to enter their payment information into the page in order to purchase the products.

## Confirmation Page

This page shows the user all the products they purchased, the desired shipping address, the payment method, and the total price in order to make sure that everything is okay by the user.

# UML Class Diagram

The UML Class Diagram below provides a visual representation of the major classes and their relationships within the backend.



## User

The purpose of the user class is to initialize new users and allow them to log in to, log out of, and register accounts. This class is also utilized to manage the activity of a particular user. Their activity can span from modifying their cart, login credentials, payment options, or shipping options. This class comes equipped with the data necessary to store a user's login in a safe way. A user's username will be stored in a string whereas the password will be sent as a salted hashed from the user to allow for the safe storing of passwords. In the event a data leak occurs, the user's true password won't be exposed. Amongst the login credentials, there are instances of the following classes: Payment, Shipping, and Cart. Payment and Shipping instances are stored in a list to allow for multiple options from the user while the Cart instance exists as a single instance for every User. Function wise, this class can login, logout, and register a user.

## Payment

This class, or more so struct, is a collection of data required to make a purchase. Our service only allows payments through tokens. Therefore, this class contains the data required for such a purchase to be made. This is made into a class to allow the possibility of changing payment methods in the future. The user class will also store a list of payments, allowing the user to choose a preferred payment at checkout. The name field allows for the user to identify their preferred payment.

## Shipping

Just like the payment class/struct, this collection of data will be the necessary information to ship products to users after a purchase has been made. The user class stores a list of this class to allow for multiple shipping addresses to be stored and chosen at checkout.

## Cart

The cart class will be the primary class used to store the items a user has inside their cart. It can be used as a way to organize all their potential purchases before checkout. The only data contained in this class is a map of items a user has selected. A map is used for quick lookup times and the ability to increase/decrease quantity of a particular item. Amongst this, there are functions for adding and removing items from this block of data as well as a function to finally checkout which returns if the checkout was successful or not. As every user is equipped with the potential to purchase, this class will be a member of every user.

## Item

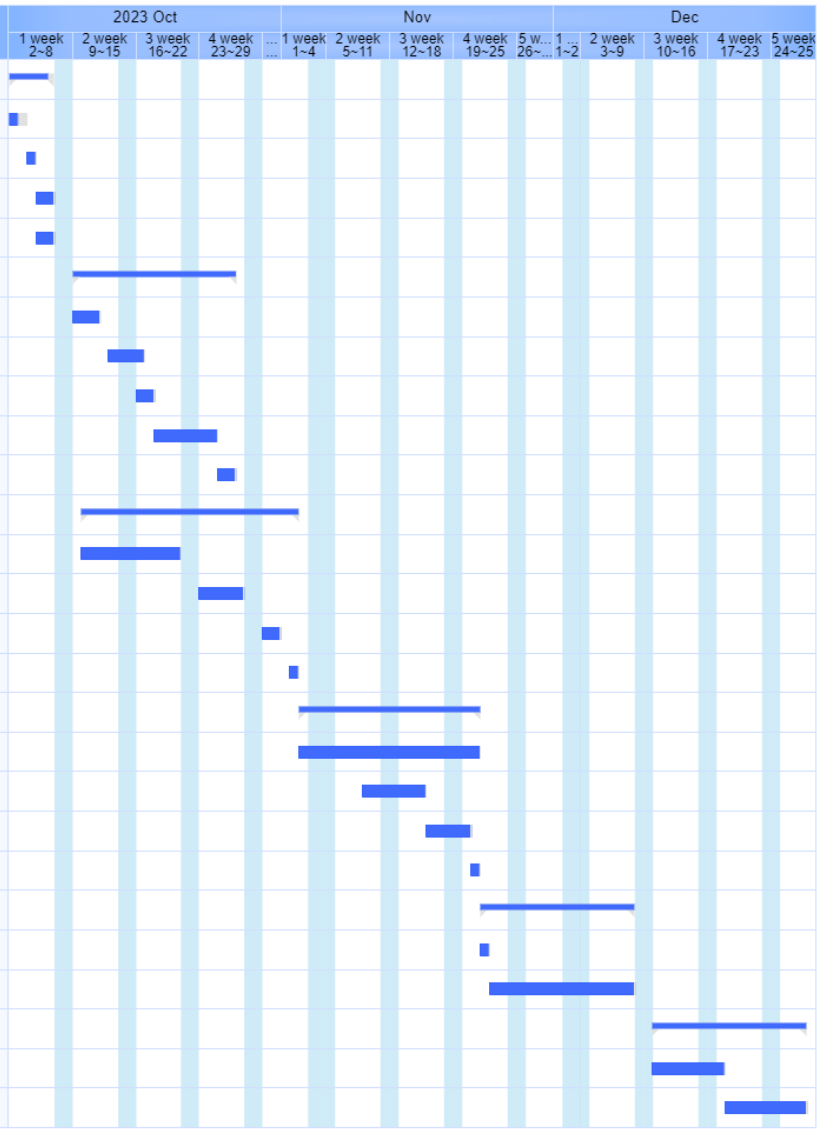
This class, similar to a struct, contains the data that every item listed on the website will need. This data includes the name, description, price, and device type. On top of this, this class inherits the class Customizable as every Item we provide is considered customizable. This class also contains a customize function in which a Customizable class is passed. A function is used here to ensure that the requested customization is actually possible with the specific item calling it.

## Customizable

This class allows for the further customization of specific items. As every item we sell allows for customization, this class will be inherited by the class Item. The possible types of customization include design, texture, and case type specific choices. These values are stored as enums as specific options for each. Inheritance is also used here to allow for the modularity object orientation provides; this modularity allows for the ease of changing/adding customizable features to items.

# Development Plan and Timeline

| ID | Task Name | Start | Finish | Duration | Resources |
|---|---|---|---|---|---|
| 1 | **Analysis** | 10-02-23 | 10-06-23 | 5.0 d. | Anthony, Nick |
| 2 | Requirement Meeting | 10-02-23 | 10-03-23 | 2.0 d. | Anthony, Nick |
| 3 | Communicate with client | 10-04-23 | 10-04-23 | 1.0 d. | Anthony, Nick |
| 4 | Document System | 10-05-23 | 10-06-23 | 2.0 d. | Anthony, Nick |
| 5 | Analysis finish | 10-05-23 | 10-06-23 | 2.0 d. | Anthony, Nick |
| 6 | **Design** | 10-09-23 | 10-26-23 | 14.0 d. | Jake, Anthony |
| 7 | Desing Database | 10-09-23 | 10-11-23 | 3.0 d. | Jake, Anthony |
| 8 | Software Design | 10-13-23 | 10-16-23 | 2.0 d. | Jake, Anthony |
| 9 | Interface Design | 10-16-23 | 10-17-23 | 2.0 d. | Jake, Anthony |
| 10 | Create Design Specifications | 10-18-23 | 10-24-23 | 5.0 d. | Jake, Anthony |
| 11 | Design Finish | 10-25-23 | 10-26-23 | 2.0 d. | Jake, Anthony |
| 12 | **Development** | 10-10-23 | 11-02-23 | 18.0 d. | Elijah, Nick |
| 13 | Develop system module | 10-10-23 | 10-20-23 | 9.0 d. | Elijah, Nick |
| 14 | Intergrate system module | 10-23-23 | 10-27-23 | 5.0 d. | Elijah, Nick |
| 15 | Perform initial testing | 10-30-23 | 10-31-23 | 2.0 d. | Elijah, Nick |
| 16 | Development Finish | 11-02-23 | 11-02-23 | 1.0 d. | Elijah, Nick |
| 17 | **Testing** | 11-03-23 | 11-21-23 | 14.0 d. | Anthony, Jake, Elijah, Nick |
| 18 | Perform system testing | 11-03-23 | 11-21-23 | 5 | Anthony, Jake, Elijah, Nick |
| 19 | Document issues found | 11-09-23 | 11-15-23 | 5.0 d. | Anthony, Jake, Elijah, Nick |
| 20 | Correct issues found | 11-16-23 | 11-20-23 | 3.0 d. | Anthony, Jake, Elijah, Nick |
| 21 | Testing Finish | 11-21-23 | 11-21-23 | 1.0 d. | Anthony, Jake, Elijah, Nick |
| 22 | **Implementation** | 11-22-23 | 12-08-23 | 13.0 d. | Anthony, Jake |
| 23 | Onsite Installation | 11-22-23 | 11-22-23 | 1.0 d. | Anthony, Jake |
| 24 | Support plan for the system | 11-23-23 | 12-08-23 | 12.0 d. | Anthony, Jake |
| 25 | **Compleation** | 12-11-23 | 12-27-23 | 13.0 d. | Nick, Elijah |
| 26 | System mantainance | 12-11-23 | 12-18-23 | 6.0 d. | Nick, Elijah |
| 27 | Evaluation | 12-19-23 | 12-27-23 | 7.0 d. | Nick, Elijah |




Planned start date: September 18, 2023.
Planned completion date: December 27, 2023.
Total design period: 101 days.

# Verification Test Plan for DecalPro

## **Feature:** User Accounts

### Unit Testing

Test Objective: Verify the correctness of individual components and functions related to user accounts.

**Test 1: Test User Registration**

*Description: Verify that users can successfully register an account on the website.*

**Testing Procedure:**

- Enter valid user information (username, email, password) and register.
- Verify the user is redirected to the account dashboard.
- Check if user data is stored correctly in the database.

**Test 2: Test User Login**

*Description: Verify that registered users can log in to their accounts.*

**Testing Procedure:**

- Enter valid username and password and log in.
- Verify the user is redirected to the account dashboard.
- Check for successful authentication.

### Functional Testing

Test Objective: Validate the functionality of user login and account data across the website.

**Test 1: Test User Login Flow**

*Description: Test the end-to-end user login process.*

**Testing Procedure:**

- Travel to the login page.
- Enter valid credentials and log in.
- Verify that the user is taken to the account dashboard.
- Check for proper error handling with invalid credentials.

**Test 2: Test Account Data Update**

*Description: Verify that users can accurately update/view their account information.*

**Testing Procedure:**

- Verify the current user information stored is correct.
- Navigate to the account settings.
- Update profile information (username, email, password, etc.).
- Save the changes.
- Verify that the updated information is reflected properly on the account dashboard.

## System Testing

Test Objective: Ensure the system as a whole integrates the login and account data functionality properly.

**Test 1: Test User Account and Ordering**

*Description: Verify that user account data is correctly linked to order management.*

**Testing Procedure:**

- Log in to the user account.
- Place an order for a decal product.
- Ensure the payment is properly verified and the order is placed.

**Test 2: Test Concurrent User Logins**

*Description: Assess the system's ability to handle multiple user logins simultaneously.*

**Testing Procedure:**

- Access several user accounts simultaneously from different devices.
- Ensure that each user is in a separate, safe session.
- Make sure that no user's actions interfere with another user's session.

**Test 3: Test Cross-Browser and Cross-Device Compatibility**

*Description: Ensure that the login and account data feature works consistently on different browsers and devices.*

**Testing Procedure:**

- Access the website and login from various browsers (e.g., Chrome, Firefox, Safari).
- Log in from different devices (e.g., laptop, smartphone, tablet).
- Verify that the user experience is consistent across all platforms.

# **Feature:** Customizability of Items

## Unit Testing

Test Objective: Verify the correctness and functionality of individual components related to customizing items on the DecalPro website.

**Test 1: Test Customization Options**

*Description: Verify that customization options are available and accurately stated for each item.*

**Testing Procedure:**

- Choose a product.
- Check to make sure any customization options—such as color, size, and design—are visible and work as intended.
- Try different combinations of customisation options to ensure they apply as intended.

**Test 2: Test Item Customization Validation**

*Description: Verify the restrictions and limitations of customizing an item.*

**Testing Procedure:**

- Try personalizing a product with invalid options.
- Ensure that customisation failures are handled appropriately and that the user is informed of them.
- Check to see if restrictions are enforced by the system, such as character limits for custom text.

## Functional Testing

Test Objective: Validate the end-to-end functionality of item customization within the DecalPro website.

**Test 1: Test Customization Process**

*Description: Test the process of customizing an item from selection to finalization.*

**Testing Procedure:**

- Select a decal product.
- Customize the item with available options.
- Check to make sure customization is simple to use and that choices are appropriately displayed.
- Verify that customers may add their customized item to their shopping cart.

**Test 2: Test Integration with Shopping Cart**

*Description: Ensure that customized items are correctly integrated into the shopping cart and order management.*

**Testing Procedure:**

- Add several customized items to the cart.
- Check to see if the items with the selected customizations show up in the cart.
- Verify that customers can complete their checkout.

**Test 3: Test Customization Persistence**

*Description: Verify that customized items are retained during the shopping session.*

**Testing Procedure:**

- After customizing an item, add it to the cart.
- Navigate away from the cart page and return back.
- Verify that the item in the cart is still applied with the customisation options selected.

## System Testing

Test Objective: Evaluate the performance and user experience of item customization across the entire DecalPro website.

**Test 1: Test Concurrent Customization**

*Description: Assess the system's ability to handle multiple users customizing items simultaneously.*

**Testing Procedure:**

- Have items customized and added to carts by several people at once.
- Verify that no customization affects the others.
- Ensure all of the user's customizations are appropriately shown.

**Test 2: End-to-End Customization and Checkout Process**

*Description: Evaluate the seamless execution of the customization, cart management, and checkout process, ensuring that the user experience is smooth and error-free.*

**Testing Procedure:**

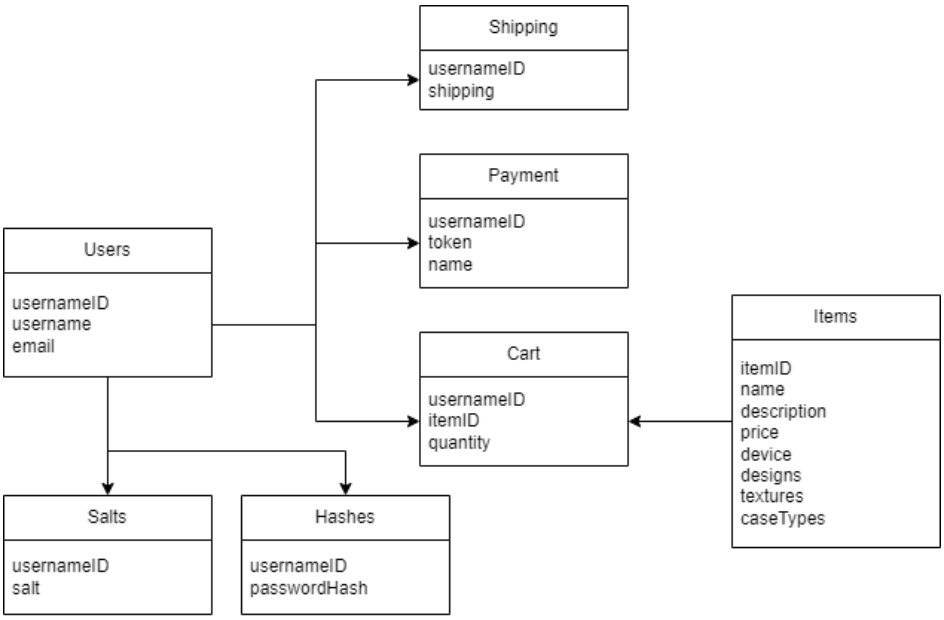
- Customize a product, add to cart, and verify items.
- Continue shopping, add more items, and check cart updates.
- Proceed to checkout, review order, and confirm details.
- Place the order and verify the order was properly placed.

# Data Management Strategy

## SQL vs NoSQL

We chose to use SQL to implement our database because it is structured which is appropriate since all of our data is dependent on each other. Alternatively, we could have used non-SQL to create our database. This would have been cheaper and easier to implement as we would not have to deal with tables and organizing the data. However, the organization that SQL provides is important to us as the data we are storing is highly relational.

## Entity Relationship Diagram




**Data Dictionary**

usernameID: Unique identifier for users
username: Username of users
salt: Random data to be fed as additional input to hash function
passwordHash: Salted and hashed version of passwords
email: Email of users
token: A token used for secure payments
name (Payment): Name for the payment token
quantity: The quantity of an item in a user's cart
itemID: Unique identifier for items
name (Items): Name of an item
description: Description of an item
price: Price of an item
device: The device an item was made for
designs: Available designs for an item
textures: Available textures for an item
caseTypes: Available case types for an item

# Design Decisions

## One Database vs Multiple

Recognizing that the data we are storing is relational, we split the data into entities that cover various aspects of the system. As shown in the diagram above, the entities are split by the groupings the website requires. We chose to have a singular database with the tables displayed above. The reasoning behind this comes from the fact that we don't have a lot of tables and found it more logical to implement as such. While it could've been possible to split up the database into multiple databases, with the intention of security, we found there wouldn't be much to gain as we have already taken measures to safely store sensitive data.

## Security

Starting off, passwords aren't stored in plain-text in this database; rather, they are stored as salted + hashed passwords that are hashed server side. This method of salting and hashing makes a data leak useless to an attacker. Alongside, payments are securely done through tokenization, reducing the consequences of a database leak. Storing data with the mindset of security allows us to take the more natural approach of using a single database. Another important factor is that access to this database is purely done through the backend; therefore, even if we were to split up the data into separate databases, it would require similar abuse of API to even leak any data.

## Storing Dynamic Amounts of Data

As mentioned throughout this document, there will be times where dynamic amounts of data need to be stored. To achieve such, the shipping, payment, and cart tables are designed with each entry being a single object. To add multiple shipping addresses, multiple payment methods, or multiple items in a cart, the same usernameID will be used for new entries. This enables a practical way to store dynamic amounts of data for each user.