



B9IS109 - Web Development for Information Systems

Cricket Equipment Store

Lecturer Name: Dinesh Kanojiya

Submitted By:

Sravani Agatamudi – 20030994

Gautham Askani – 20025842

Purna Teja Reddy - 20025460

Table of Contents

1. Introduction	4
2. Research and Planning	4
Main Goal:	4
Objectives:	4
Steps Taken:	4
3. Choice of Framework and Technologies	5
3.1. Frontend	5
3.2. Backend	5
3.3. Database	6
3.4. Hosting	6
4. UX Design	7
4.1. Flow Diagram	7
4.2. Login Page	8
4.3. Registration Page	9
4.4. Product Page	10
4.5 Single Product Page	10
4.6 Search Feature	12
4.7 Cart and Order Page	13
5. Web Services	14
Key Features	14
Product Search and Filters:	14
Image Gallery:	15
Add to Cart and Buy Now:	16
API DESIGN	16
6. Security Threats and Measures	22
6.1. Potential Threats	22
6.2 Mitigation Strategies	22
7. Links	22
GitHub	22
Deployment:	22
Presentation Video	22
8. References	23

Figure Table of Contents

Figure 1 Cricket Web Application Flow Diagram	7
Figure 2 Login page Layout	8
Figure 3 Registration Page.....	9
Figure 4 Registration Failed.....	10
Figure 5 Product Page	10
Figure 6 Single Product page.....	11
Figure 7 Search Bar	11
Figure 8 Search Result Page	12
Figure 9 Result page.....	12
Figure 10 Cart Page	13
Figure 11 Order Page.....	13
Figure 12 Orders Placed	14
Figure 13 Product search code.....	15
Figure 14 Image Gallery	15
Figure 15 Add to Cart and Buy now	16
Figure 16 Search Functionality Route	16
Figure 17 Login Route	17
Figure 18 Cart Functioning Route	18
Figure 19 MongoDB Connection	20
Figure 20 Mongoddb Structure	21

1. Introduction

The Cricket Equipment Store is a modern and user-friendly platform designed to make it easy for cricket fans to shop for cricket gear. It offers a simple interface where users can browse a variety of products, search for items instantly, and enjoy a smooth shopping experience. The website is built using advanced technologies for fast performance, strong security, and the ability to handle many users.

It works well on desktops, tablets, and phones, so users can shop anytime, anywhere. The site also has features like personalized dashboards to track orders, favourites, and more. With secure payment options, live search, and multiple product images, the store is designed to meet all your cricket needs. Whether you're a player, coach, or cricket enthusiast, this platform makes shopping for cricket equipment easy and enjoyable while keeping you updated with the latest trends in the sport.

2. Research and Planning

Main Goal:

To build a fast, scalable, and easy-to-use website that works well on both mobile phones and computers. The design is modular, making it easy to maintain and expand in the future.

Objectives:

- Focus on the key needs of cricket lovers.
- Make the website simple to navigate and enjoyable to use.
- Fix common issues like slow speed and confusing layouts seen in other websites.
- Create a login page for users.
- Add features like search, cart, ordering.

Steps Taken:

- Looked at similar websites to learn what works and what doesn't.
- Picked the best tools and frameworks for speed, ease of use, and compatibility.
- Planned the backend services and created clear routes for the website to communicate with the database.
- Ensuring the website is user-friendly, fast, and ready for future upgrades.

3. Choice of Framework and Technologies

3.1. Frontend : React.js

Why React?

- **Framework:** React with TypeScript for building scalable and type-safe user interfaces.
 - React: A JavaScript library used for creating dynamic and interactive user interfaces. React's component-based architecture helps in building reusable UI components, making the code modular and maintainable.
 - TypeScript: Provides static type checking, enhancing the development experience by catching type-related errors early in the development process.
- **Styling:** Tailwind CSS for rapid and customizable UI development.
 - Tailwind CSS is a utility-first CSS framework that allows developers to build designs quickly by applying utility classes directly in the HTML or JSX. This approach ensures flexibility and control over styling.
- **Build Tool:** Vite for fast development and optimized builds.
 - Vite is a build tool that provides faster builds and a more efficient development environment compared to traditional bundlers. It significantly improves the development experience by pre-bundling dependencies and offering faster hot module replacement.
- **Components:** Modular React components for reusability and maintainability.
 - Components like a product card or user profile are designed as reusable building blocks, reducing redundancy and increasing maintainability across the app.

3.2. Backend : Python Flask

Why FLASK?

- **Language:** Python leveraging its robust ecosystem and libraries.
 - Python is a versatile programming language with a vast library ecosystem. It's well-suited for backend development due to its simplicity and rich feature set for handling tasks such as data processing, API development, and user authentication.
- **Framework:** Flask for developing RESTful APIs efficiently.
 - Flask is a lightweight Python framework for building APIs. It is known for its simplicity and flexibility, making it ideal for developing web services. Flask

allows easy handling of HTTP requests and routing, providing an efficient way to create RESTful APIs.

- **API Standards:** Follows REST conventions for consistency and interoperability.
 - RESTful APIs enable the seamless exchange of data between the frontend and backend. For instance, an endpoint like `/products/{id}` is used to retrieve specific product details, ensuring standardized interaction between client and server.

3.3. Database

Why MONGODB?

- **Database Management System:** MongoDB for flexible and scalable data storage.
 - MongoDB is a NoSQL database that uses a document-oriented data model. It allows storing data in JSON-like documents, which is ideal for applications with varying data types. MongoDB's flexibility enables efficient handling of complex and dynamic data.
- **Features:** Scalable, flexible data models with support for large datasets and high read/write throughput.
 - ACID compliance is partially supported in MongoDB, ensuring reliable transactions.
 - Indexes are used for faster queries, particularly for searches and filtering in large datasets.
- **Integration:** MongoEngine for seamless integration with Flask.
 - MongoEngine is a Python library that provides an interface to interact with MongoDB, allowing for smooth integration between Flask and the database.

3.4. Hosting

- **Environment:** Dockerized for consistent deployment across environments.
 - Docker is used to containerize the application, ensuring that it runs consistently across different environments (development, staging, production). Docker containers encapsulate all dependencies, making the app easier to deploy, scale, and manage.
- **Platform:** AWS (Amazon Web Services) for cloud-based infrastructure and hosting.
 - AWS provides scalable cloud infrastructure for hosting web applications. It includes services like EC2 for virtual servers, S3 for storage, and RDS for

managed database services, providing a reliable and flexible platform for hosting the backend and database.

- **CI/CD:** Automated Continuous Integration/Continuous Deployment pipelines for seamless updates.
 - CI/CD pipelines automate the process of code testing, building, and deployment. This ensures fast, reliable delivery of new features and bug fixes, and reduces manual intervention during the release process.

4. UX Design

4.1. Flow Diagram

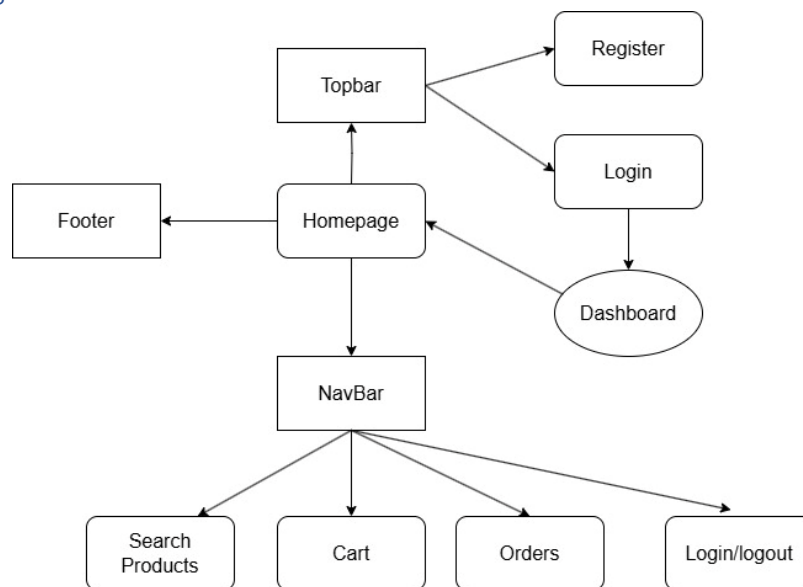
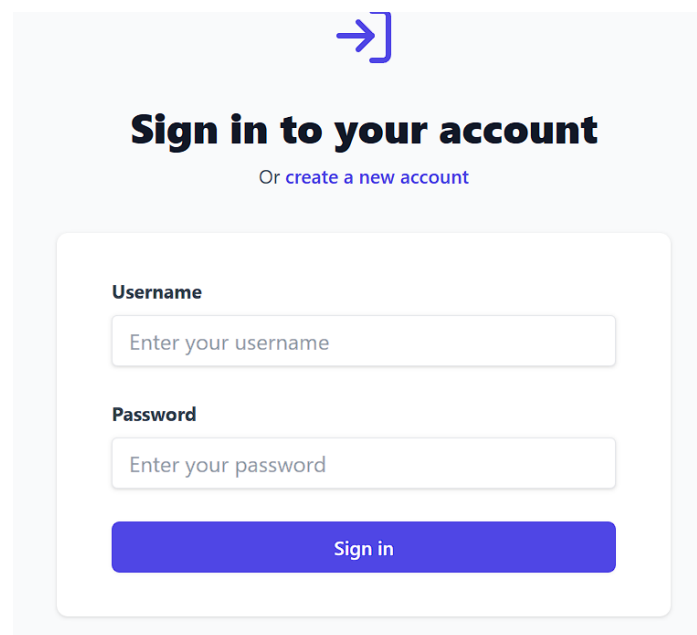


Figure 1 Cricket Web Application Flow Diagram

4.2. Login Page

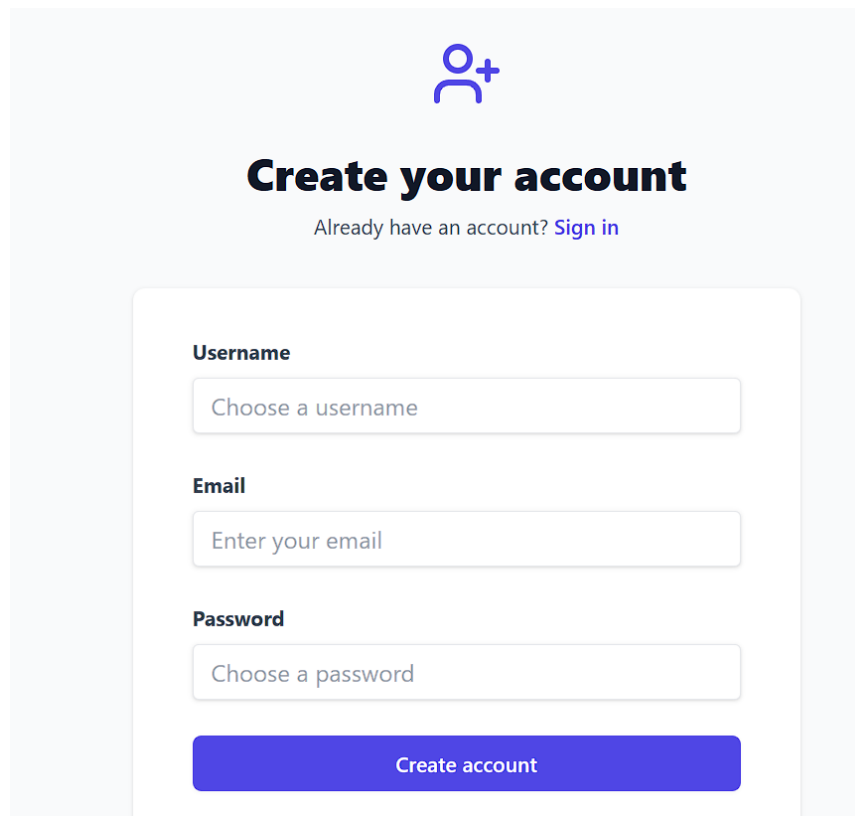


The image shows a login page layout. At the top, there is a blue icon of a right-pointing arrow followed by a closing square bracket. Below this is the heading "Sign in to your account" in bold black text. Underneath the heading is a link "Or [create a new account](#)" in blue text. The main form is a white rounded rectangle containing two input fields. The first is labeled "Username" and has the placeholder text "Enter your username". The second is labeled "Password" and has the placeholder text "Enter your password". Below these fields is a solid blue button with the text "Sign in" in white.

Figure 2 Login page Layout

The Login Page provides a clean and straightforward design for users to access their accounts or create a new one. At the top of the page, a clear heading *"Sign in to your account"* welcomes users and guides them to either sign in or register. Below the heading, the page features two input fields: *Username* and *Password*, where users can enter their login credentials. Each input field includes placeholder text to guide the user, such as *"Enter your username"* and *"Enter your password"*. Once the credentials are entered, users can click the Sign in button, which stands out with its contrasting color, ensuring that the action is easy to find. Additionally, for new users, a link to *create a new account* is provided below the sign-in section, encouraging registration. The layout is simple, user-friendly, and designed for easy navigation, making it convenient for users to either log in or sign up.

4.3. Registration Page



The registration form is centered on a light gray background. At the top, there is a purple icon of a person with a plus sign. Below the icon, the title "Create your account" is displayed in a bold, black font. Underneath the title, a link "Sign in" is provided for users who already have an account. The form itself is a white rounded rectangle containing three input fields: "Username" with the placeholder "Choose a username", "Email" with the placeholder "Enter your email", and "Password" with the placeholder "Choose a password". A prominent blue button labeled "Create account" is positioned at the bottom of the form.

Figure 3 Registration Page

On this page new user can register by creating an account with username and password. All the fields on the login form are mandatory, and the form uses validations to ensure the information is accurate. If the login credentials are incorrect, the user will receive an error message prompting them to verify their username and password. Upon successful login, the system will authenticate the user and grant access to their account, redirecting them to their dashboard. For new users, a link is provided to navigate to the registration page where they can create an account.

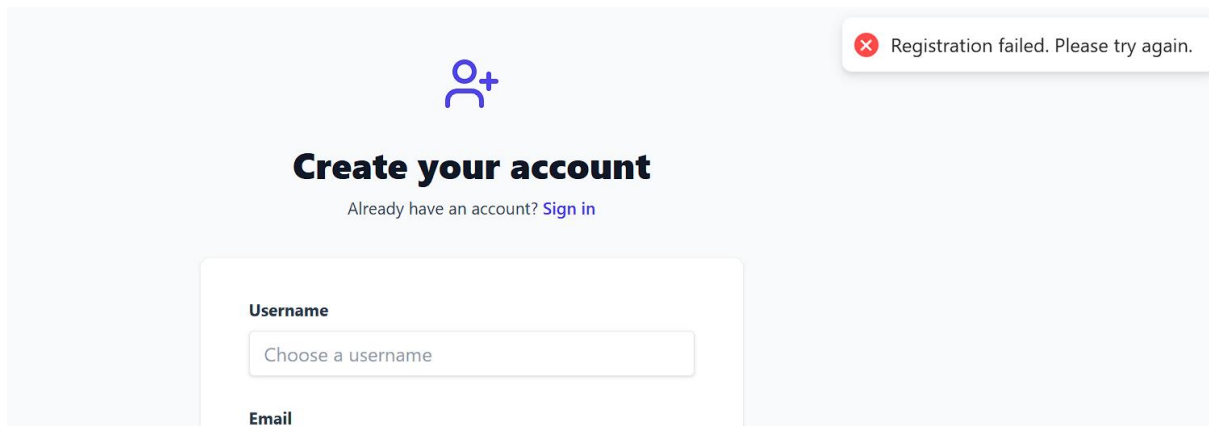


Figure 4 Registration Failed

4.4. Product Page

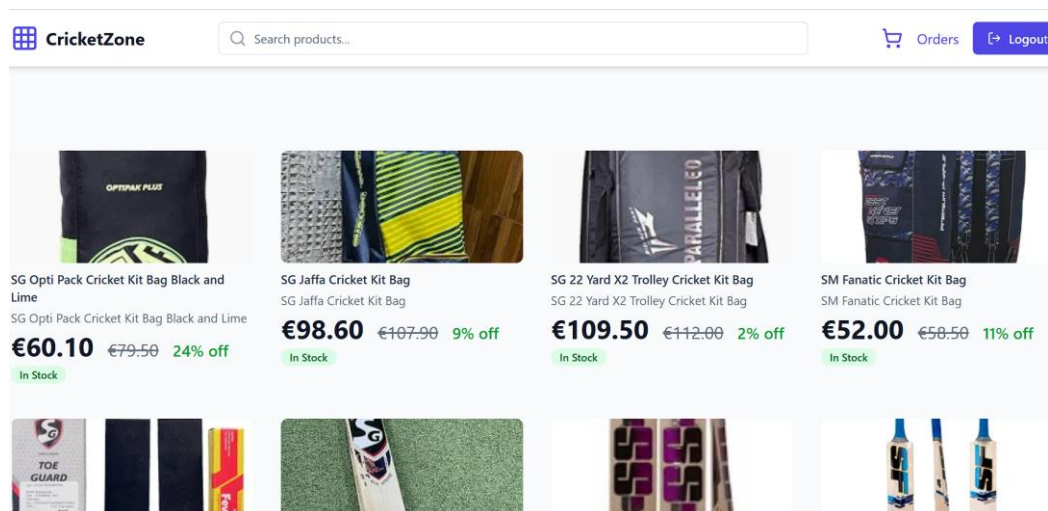


Figure 5 Product Page

On this page, the user is welcomed with a clean and user-friendly interface that allows easy navigation across the platform. It includes a well-organized navigation bar (Navbar) that provides easy access to key sections such as products, cart, orders, and logout. Displays product details with images, names, descriptions, and prices, while also indicating whether the product is in stock or not. And allows users to view multiple product images, with navigation buttons for easy browsing

4.5 Single Product Page

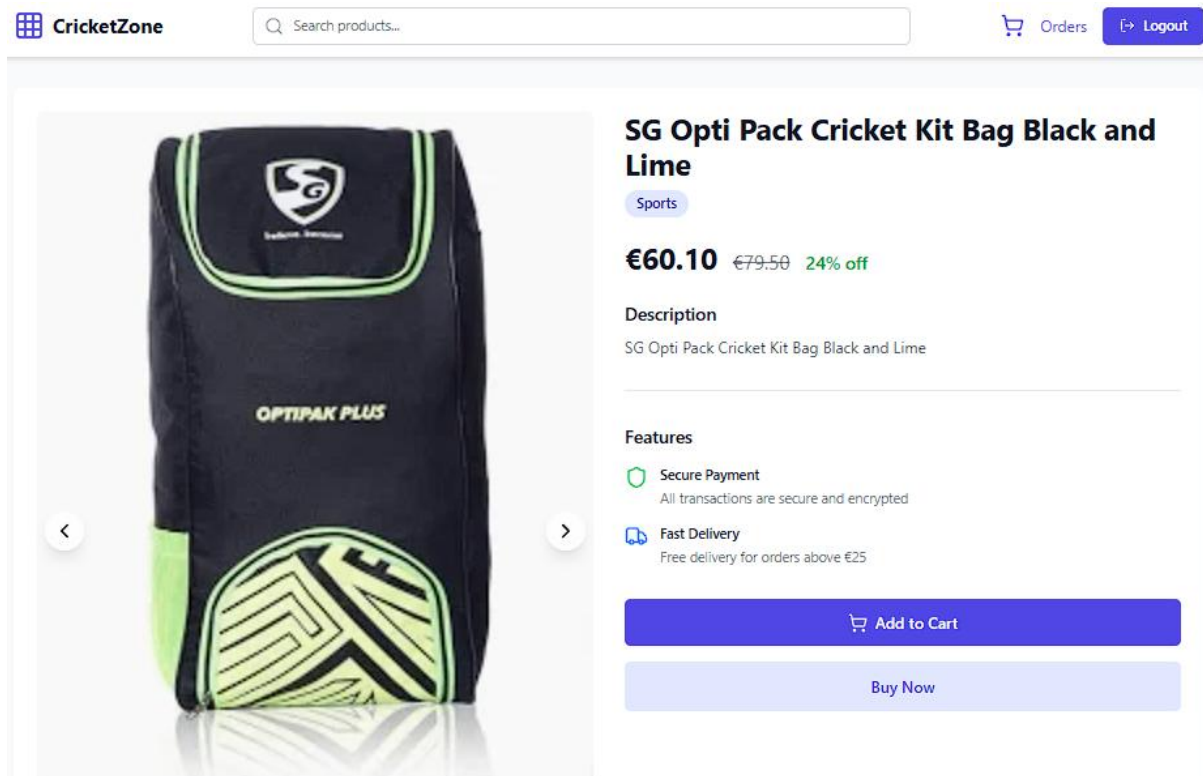


Figure 6 Single Product page

On this page, The User is provided with all details of the cricket product. The user can browse through all the products related to cricket like shirts, bats, gloves, kit bags, and guard pack.

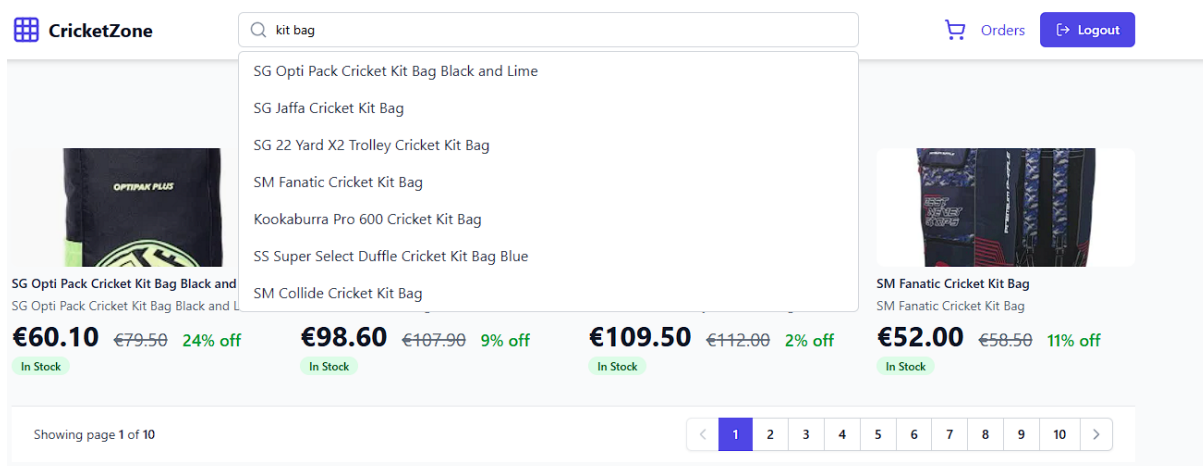


Figure 7 Search Bar

In the above figure, the user can search for their products and also see the relevant product data like types, models, and colours.

4.6 Search Feature

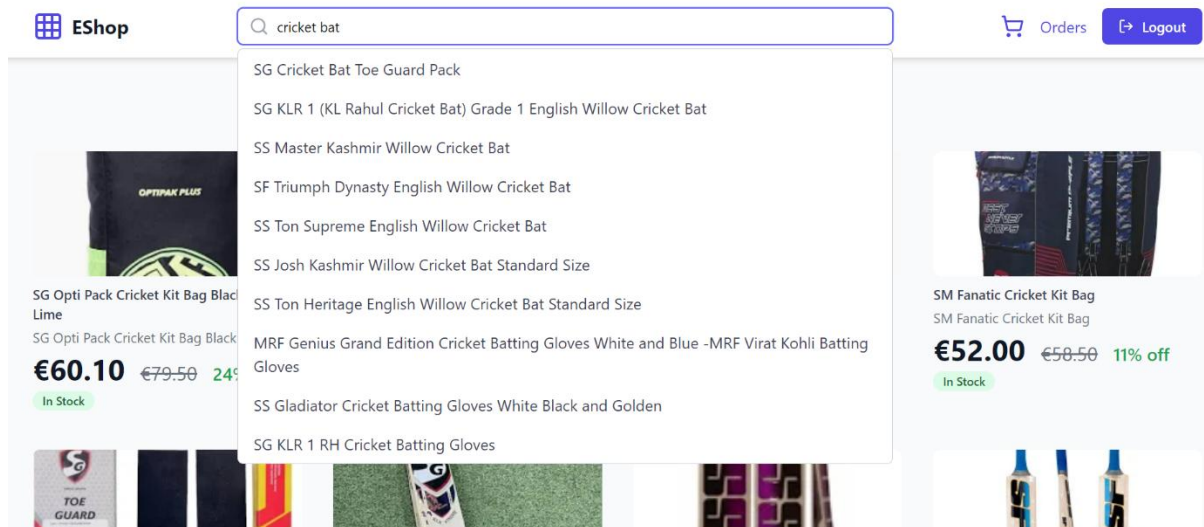


Figure 8 Search Result Page

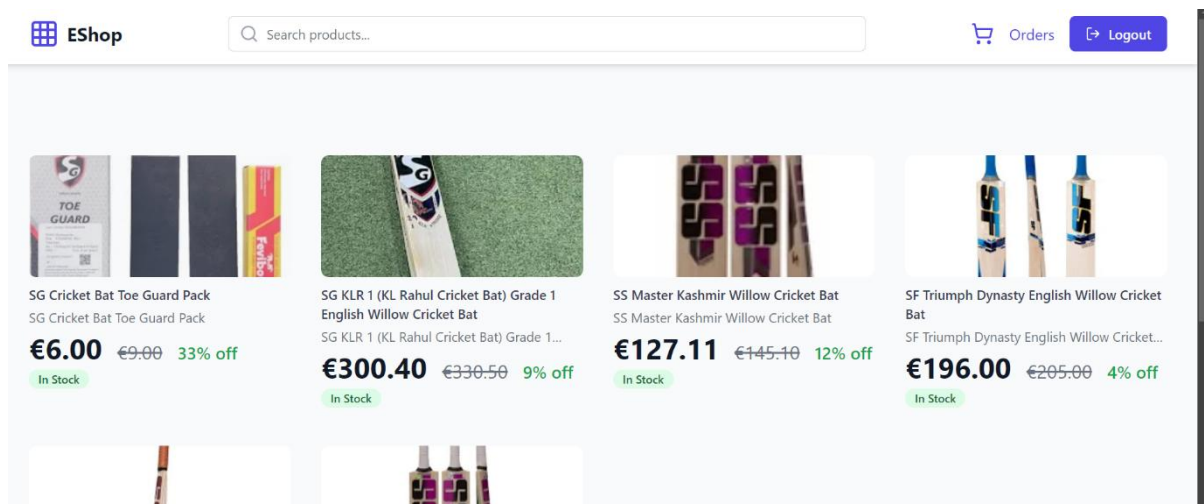


Figure 9 Result page

4.7 Cart and Order Page

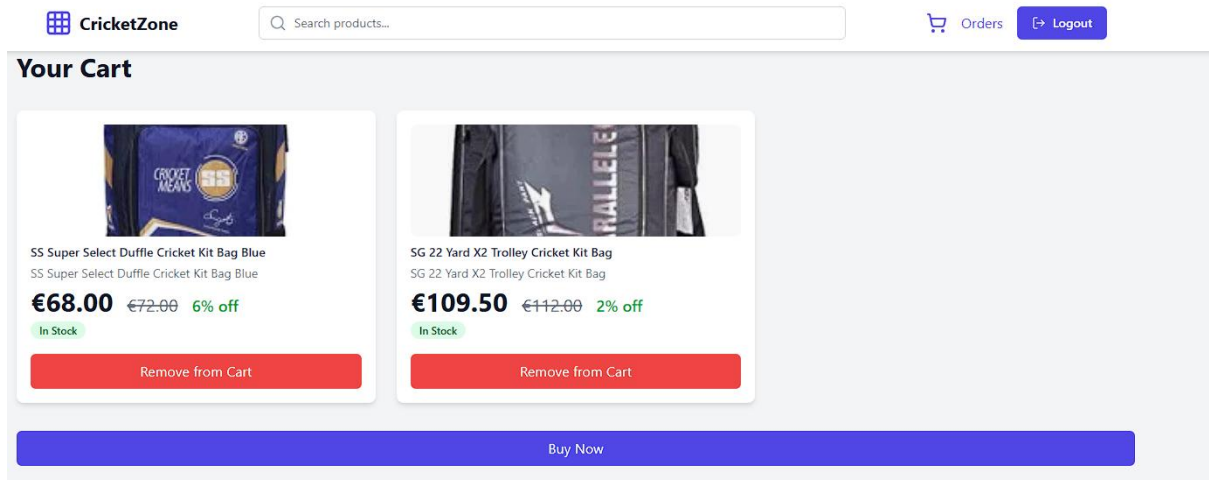


Figure 10 Cart Page

Place Your Order

Your Products

SS Super Select Duffle Cricket Kit Bag Blue	₹72.00
SS Super Select Duffle Cricket Kit Bag Blue	
SG 22 Yard X2 Trolley Cricket Kit Bag	₹112.00
SG 22 Yard X2 Trolley Cricket Kit Bag	

Figure 11 Order Page

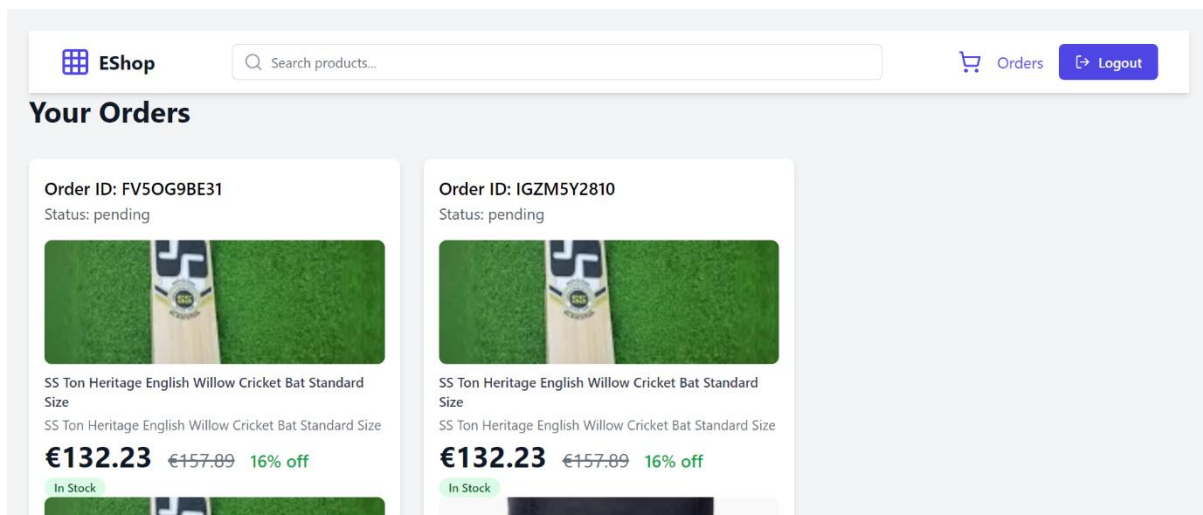


Figure 12 Orders Placed

5. Web Services

Key Features

Product Search and Filters:

```
rc > components > layout > SearchBar.tsx > SearchBar
1  import React, { useState, useEffect } from 'react';
2  import { Search } from 'lucide-react';
3  import { useSearchStore } from '../store/searchStore';
4  import { searchProducts } from '../services/api'; // Import the search API func
5  import { Link } from 'react-router-dom';
6
7  export const SearchBar: React.FC = () => {
8    const { setSearchQuery } = useSearchStore();
9    const [localQuery, setLocalQuery] = useState('');
10   const [searchResults, setSearchResults] = useState<any[]>([]); // State for sear
11
12   const handleSearch = async (e: React.FormEvent) => {
13     e.preventDefault();
14     setSearchQuery(localQuery);
15   };
16
17   useEffect(() => {
18     const fetchSearchResults = async () => {
19       if (localQuery) {
20         try {
21           const response = await searchProducts(localQuery); // Call the search AP
22           setSearchResults(response.data.products);
23         } catch (error) {
24           console.error('Failed to fetch search results', error);
25         }
26       } else {
27         setSearchResults([]); // Clear results if the query is empty
28       }
29     };
30   });
31 }
```

Figure 13 Product search code

This feature enables users to search for products in real-time. As users type, results dynamically update, showing product names, descriptions, and availability.

Image Gallery:

```
src > components > product > ImageGallery.tsx > ImageGallery > previousImage
8
9  export const ImageGallery: React.FC<ImageGalleryProps> = ({ images, productName
10    const [currentImage, setCurrentImage] = useState(0);
11
12    const nextImage = () => {
13      setCurrentImage((prev) => (prev + 1) % images.length);
14    };
15
16    const previousImage = () => {
17      setCurrentImage((prev) => (prev - 1 + images.length) % images.length);
18    };
19
20    return (
21      <div className="relative">
22        <div className="aspect-w-1 aspect-h-1 w-full overflow-hidden rounded-lg bg
23          <img
24            // src={`${API_URL}${images[currentImage].replace('./app/', '')}`}
25            src={`${images[currentImage]}`}
26            alt={`${productName} - Image ${currentImage + 1}`}
27
28            className="h-full w-full object-cover object-center"
29          />
30        </div>
31
32        {/* Navigation buttons */}
33        <button
34          onClick={previousImage}
```

Figure 14 Image Gallery

This component provides a user-friendly way to browse product images, complete with navigation buttons for viewing multiple pictures.

Add to Cart and Buy Now:

```
components > product > ProductActions.tsx > ProductActions > handleAddToCart
export const ProductActions: React.FC<ProductActionsProps> = ({ isAvailable, productId }) => {

  const handleAddToCart = async () => {
    if (isAvailable) {
      try {
        await addToCart(productId);
        toast.success('Product added to cart!');
      } catch (error) {
        toast.error('Failed to add product to cart.');
```

Figure 15 Add to Cart and Buy now

The file handles these actions, allowing users to easily add items to their cart or proceed directly to checkout.

API DESIGN

This web application uses a RESTful API to connect the frontend with the backend. The backend has endpoints that the frontend uses to get and send data, which is then shown to the user.

1.Search functionality

GET: To fetch products based on text typed.

```
1  from flask import Blueprint, request, jsonify
2  from app.services.search_service import SearchService
3
4  search_bp = Blueprint('search', __name__)
5
6  @search_bp.route('/search', methods=['GET'])
7  def search():
8      search_term = request.args.get('q', '', type=str) # Get the search term from query parameters
9      products = SearchService.search_products(search_term)
10     return jsonify({
11         "status": "success",
12         "data": {
13             "products": [product.serialize() for product in products] # Serialize each product
14         },
15         "message": None
16     }), 200
```

Figure 16 Search Functionality Route

2.User Authentication

POST: Sends email and password and return JWTtoken to authenticate a user

```
1 from flask import Blueprint, request, jsonify
2 from app.services.auth_service import AuthService
3
4 auth_bp = Blueprint('auth', __name__)
5
6 @auth_bp.route('/register', methods=['POST'])
7 def register():
8     data = request.get_json()
9     response = AuthService.register(data['username'], data['email'], data['password'])
10    try:
11        if isinstance(response, dict): # Check if it's an error message
12            return jsonify({
13                "status": "error",
14                "data": {},
15                "message": response['message']
16            }), 400
17        return jsonify({
18            "status": "success",
19            "data": response.serialize(),
20            "message": None
21        }), 201
22    except Exception as e:
23        return jsonify({
24            "status": "error",
25            "data": {},
26            "message": str(e)
27        }), 500
28
29 @auth_bp.route('/login', methods=['POST'])
30 def login():
31     data = request.get_json()
32     response = AuthService.login(data['username'], data['password'])
33     if 'access_token' in response[0]:
34         return jsonify({
35             "status": "success",
36             "data": {
37                 "access_token": response[0]['access_token']
38             },
39             "message": None
40         }), 200
41     return jsonify({
42         "status": "error",
43         "data": {},
44         "message": response['message']
45     }), 401
```

Figure 17 Login Route

3. Cart Functionality

POST: Adds item to cart

GET: Retrieve and displays cart

DELETE: delete products from cart

```
1  from flask import Blueprint, request, jsonify
2  from app.services.cart_service import CartService
3  from app.models.product import Product # Ensure Product model is imported
4  from app.middleware import jwt_required_with_user # Import the custom middleware
5
6  cart_bp = Blueprint('cart', __name__)
7
8  @cart_bp.route('/cart', methods=['POST'])
9  @jwt_required_with_user
10 def add_to_cart():
11     current_user = request.user # Access the user from the request context
12     data = request.get_json()
13     product_id = data['product_id']
14     product = Product.objects(id=product_id).first()
15     if product:
16         CartService.add_to_cart(current_user.id, product)
17         return jsonify({
18             "status": "success",
19             "data": {
20                 "message": "Product added to cart"
21             },
22             "message": None
23         }), 201
24     return jsonify({
25         "status": "error",
26         "data": {},
27         "message": "Product not found"
28     }), 404
29
```

Figure 18 Cart Functioning Route

```

29 @cart_bp.route('/cart', methods=['GET'])
30 @jwt_required_with_user
31 def get_cart():
32     current_user = request.user # Access the user from the request context
33     cart_data = CartService.get_cart(current_user.id)
34     return jsonify({
35         "status": "success",
36         "data": {
37             "cart": cart_data # Return serialized product data
38         },
39         "message": None
40     }), 200
41
42 @cart_bp.route('/cart', methods=['DELETE'])
43 @jwt_required_with_user
44 def remove_from_cart():
45     current_user = request.user # Access the user from the request context
46     data = request.get_json()
47     product_id = data['product_id']
48     product = Product.objects(id=product_id).first()
49     if product:
50         CartService.remove_from_cart(current_user.id, product)
51     return jsonify({
52         "status": "success",
53         "data": {
54             "message": "Product removed from cart"
55         },
56         "message": None
57     }), 200

```

MONGODB CONNECTION:

```
1  import pandas as pd
2  from mongoengine import connect
3  import sys;sys.path.append('./')
4  from app.models.product import Product
5  from dotenv import load_dotenv
6  import os
7
8  load_dotenv()
9  connect('sports_ecommerce', host=os.getenv('MONGO_URI'))
10
11
12  def update_products_from_csv(csv_file):
13
14      df = pd.read_csv(csv_file)
15
16      for index, row in df.iterrows():
17
18          product_name = row['Product Name']
19          price = row['Old Price']
20          special_price = row['Special Price']
21          product_description = row['Product Name']
22          category = 'Sports'
23          is_available = True
24
25          product_urls = [row["Images"]]
26
27          product = Product.objects(product_name=product_name).first()
28
29          if product:
30
31              product.price = price
32              product.special_price = special_price
33              product.product_description = product_description
34              product.category = category
35              product.is_available = is_available
36              product.product_urls = product_urls
37              product.save()
38              print(f"Updated product: {product_name}")
39          else:
40
41              new_product = Product(
42                  product_name=product_name,
43                  price=price,
44                  special_price=special_price,
45                  product_description=product_description,
46                  category=category,
47                  is_available=is_available,
48                  product_urls=product_urls
49              )
50              new_product.save()
51              print(f"Created new product: {product_name}")
52
53  if __name__ == "__main__":
54      update_products_from_csv('/Users/jaya.patibandla/Downloads/Sports_Ecommerce_Product_Data_updated.csv')
```

Figure 19 MongoDB Connection

MONGODB Structure :

sports_ecommerce.product

STORAGE SIZE: 32KB

LOGICAL DATA SIZE: 27.93KB

TOTAL DOCUMENTS: 65

INDEXES TOTAL SIZE: 20KB

Find

Indexes

Schema Anti-Patterns

Aggregation

Search Indexes

Generate queries from natural language in Compass

INSERT DOCUMENT

Filter

Type a query: { field: 'value' }

Reset

Apply

Options

QUERY RESULTS: 1-20 OF MANY

```

_id: ObjectId('675d0f1317b377e9829fa8e2')
product_name: "SG Opti Pack Cricket Kit Bag Black and Lime"
price: 79.5
special_price: 60.1
product_description: "SG Opti Pack Cricket Kit Bag Black and Lime"
category: "Sports"
is_available: true
product_urls: Array (1)

```

```

_id: ObjectId('675d0f1317b377e9829fa8e3')
product_name: "SG Jaffa Cricket Kit Bag"
price: 107.9
special_price: 98.6
product_description: "SG Jaffa Cricket Kit Bag"

```

PREVIOUS

1-20 of many results

NEXT

sports_ecommerce

LOGICAL DATA SIZE: 33.12KB

STORAGE SIZE: 176KB

INDEX SIZE: 272KB

TOTAL COLLECTIONS: 5

CREATE COLLECTION

Collection Name	Documents	Logical Data Size	Avg Document Size	Storage Size	Indexes	Index Size	Avg Index Size
address	18	2.43KB	139B	36KB	1	36KB	36KB
cart	5	440B	88B	36KB	1	36KB	36KB
order	4	586B	147B	36KB	2	72KB	36KB
product	65	27.93KB	441B	32KB	1	20KB	20KB
user	14	1.75KB	129B	36KB	3	108KB	36KB

Figure 20 Mongodb Structure

6. Security Threats and Measures

6.1. Potential Threats

1. **Cross-Site Scripting (XSS):** Exploitation via malicious script injections.
2. **SQL Injection:** Unauthorized database access through unsanitized inputs.
3. **Data Breaches:** Risk of exposing user data due to insufficient encryption.

6.2 Mitigation Strategies

- **Input Validation:** Strict validation rules to prevent malicious inputs.
- **Use of Parameterized Queries:** Ensures secure interactions with the database.
- **Data Encryption:** Encrypt sensitive data both in transit and at rest.

7. Links:

GitHub : We used GitHub to store our backend and frontend code . GitHub helps us keep track of changes every time we update the code. After saving everything on GitHub, we used AWS to make our website live.

Link: <https://github.com/purnatejaR/CricketStore.git>

Deployment:

We used AWS to deploy both the frontend and backend of the website because it is easy to use and supports automatic updates. When we make changes to the code, we upload them to GitHub, and AWS automatically updates the live website with the new changes.

Link:

<https://sports-commerce-frontend-service.30dbgrwh4g67y.us-east-1.cs.amazonlightsail.com/login>

Presentation Video has been uploaded to the following link

Video Link: <https://youtu.be/cchm3LbLvKE>

8. References

- Flask, 2024. *Flask Documentation*. [online] Available at: <https://flask.palletsprojects.com/>.
- React, 2024. *React - A JavaScript library for building user interfaces*. [online] Available at: <https://react.dev/>.
- MongoDB, 2024. *MongoDB Documentation*. [online] Available at: <https://www.mongodb.com/docs/>.
- RESTful API, 2024. *REST API Design - Best Practices and Standards*. [online] Available at: <https://restfulapi.net/>.
- AWS, 2024. *Amazon Web Services Documentation*. [online] Available at: <https://aws.amazon.com/documentation/>.
- Docker, 2024. *Docker Documentation - Empowering app development for individuals and teams*. [online] Available at: <https://docs.docker.com/>.
- Flask-SQLAlchemy, 2024. *Flask-SQLAlchemy Documentation*. [online] Available at: <https://flask-sqlalchemy.palletsprojects.com/en/3.0.x/>