**SVKM’s NMIMS**

**Mukesh Patel School of Technology Management & Engineering (Mumbai Campus)**

**Computer Engineering Department (BTI Sem VIII)**

**Database Management System**

**Project Report**

|  |  |  |
| --- | --- | --- |
| Program | BTI Computer Engineering | |
| Semester | VIII | |
| Name of the Project: | Online Sports Gear Store | |
|  | | |
| Details of Project Members |  |  |
| Batch | Roll No. | Name |
| D2 | C182 | Jeeval Shah |
| D2 | C173 | Sachi Mane |
| D2 | C161 | Chahak Daga |
| Date of Submission: 06/04/25 | | |

**Contribution of each project Members:**

|  |  |  |
| --- | --- | --- |
| Roll No. | Name | Contribution |
| C182 | Jeeval Shah | EER, Normalisation, Complex Queries, Backend |
| C173 | Sachi Mane | EER, Schema Reduction, Frontend |
| C161 | Chahak Daga | EER, Schema Reduction, Frontend |

**Rubrics for the Project evaluation:**

|  |  |
| --- | --- |
| First phase of evaluation:  Innovative Ideas (5 Marks)  Design and Partial implementation (5 Marks) | 7 marks (3M for EER, 4M for 15 tables, 10 rows in each , DDL command implemented, should include all constraints , all types of attributes, ISA and disjoint should be present) |
| Final phase of evaluation (Implementation, presentation and viva, Self-Learning and Learning Beyond classroom) | 8 marks (Complex queries, nested queries, aggregate, implementation, normalised database) |

**Project Report**

**Online Sports Gear Store**

**by**

**Jeeval Shah, C182**

**Sachi Mane, C173**

**Chahak Daga, C161**

**Course: DBMS**

**AY: 2024-25**

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **Sr no.** | **Topic** | **Page no.** |
| **1** | Storyline | 1 |
| **2** | Components of Database Design | 2 |
| **3** | Entity Relationship Diagram | 3 |
| **4** | Relational Model | 4 – 5 |
| **5** | Normalization | 6 – 8 |
| **6** | SQL Queries | 9 – 16 |
| **7** | Project Demonstration | 17 – 22 |
| **8** | Self-learning beyond classroom | 23 |
| **9** | Learning from the project | 24 |
| **10** | Challenges faced | 25 |
| **11** | Conclusion | 26 |

**I. Storyline**

In today’s digitally connected world, the demand for niche e-commerce platforms is rising rapidly. This project envisions an Online Sports Gear Store that brings together buyers and sellers across India onto a unified digital platform.

The core idea is to provide customers with easy access to high-quality sports gear, ranging from winter trekking equipment to professional athletics gear & enabling small and large-scale sellers to reach a wider audience. The business model is marketplace-based, where multiple sellers manage their own storefronts under the umbrella of the main platform.

To efficiently run this online store, a robust and scalable relational database is essential. The system must capture and manage critical information across various components of the business including:

1. Users (Customers and Sellers): Their details and login credentials.
2. Sellers and their Stores: Each seller runs a store that operates in a specific city and state. Stores are uniquely identified and connected to the seller.
3. Products: Each store offers a variety of sports gear products. Products belong to different categories (like Winter Gear, Water Sports, Fitness Equipment, etc.) and come with details like price, description, and available quantity.
4. Orders and Payments: Customers can place orders which are linked to specific stores and products. Payment details including card information, expiry dates, and payment status must be securely managed.
5. Categories: Products are grouped under categories to enable efficient browsing and filtering.
6. Shopping Cart: Customers can add items to their cart before purchase.
7. Order Status: The system should track whether an order is placed, packed, shipped, or delivered.

This storyline forms the foundation of the database design. We aim is to mimic real-world complexities while ensuring normalized data storage, referential integrity, and easy retrieval of information through SQL queries.

Through this project, we not only simulate the backend of a real e-commerce business but also learn how structured data helps streamline operations, enhance user experience, and support decision-making.

**II. Components of Database Design**

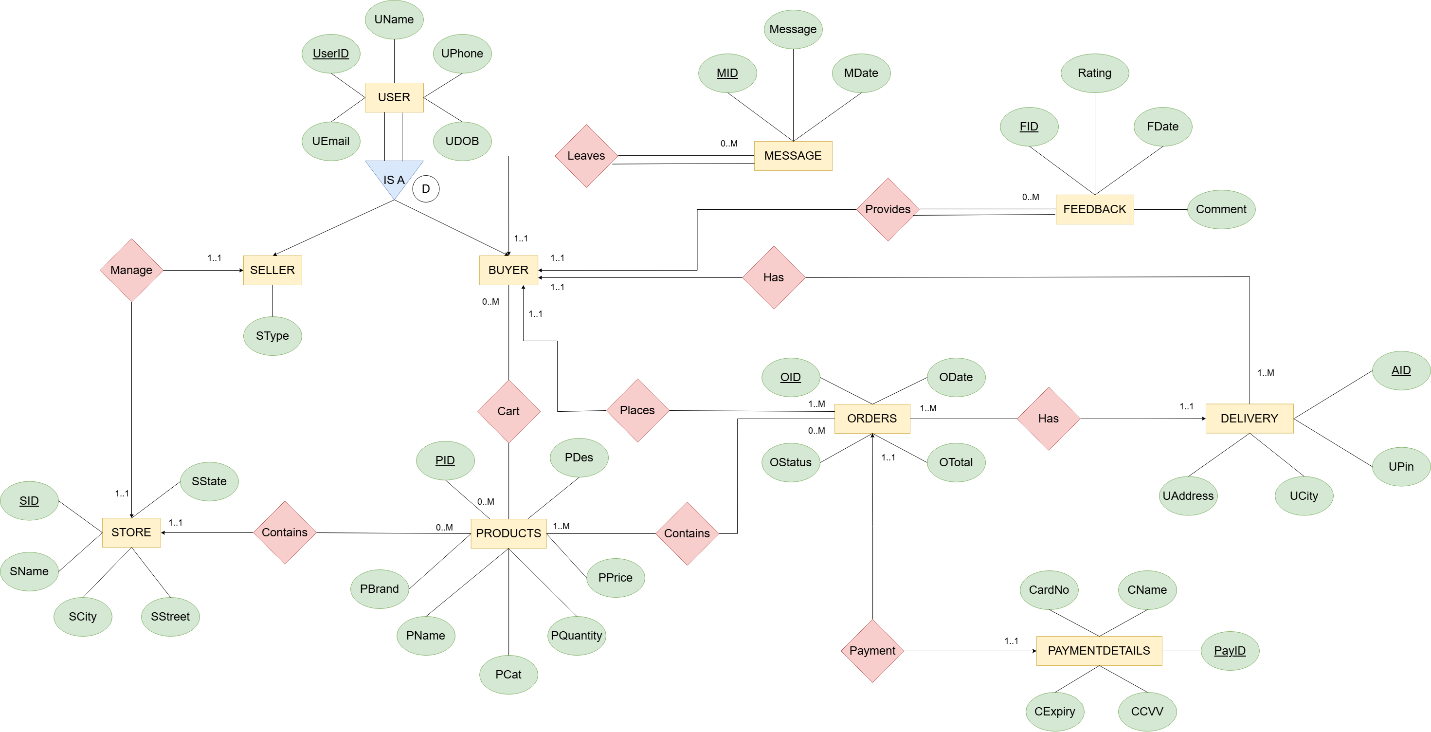
Entity Set With Attributes

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Primary Attributes | Non – Primary Attributes |
| USER | Strong | UserID | UName, UPhone, UDOB, UEmail |
| BUYER | Strong | UserID | UName, UPhone, UDOB, UEmail |
| SELLER | Strong | UserID | UName, UPhone, UDOB, UEmail, SType |
| STORE | Strong | SID | SName, SStreet, SCity, SState |
| PRODUCT | Strong | PID | PName, PBrand, PDes, PPrice, PQuantity, PCat |
| ORDERS | Strong | OID | ODate, OTotal, OStatus |
| DELIVERY | Strong | AID | UPin, UCity, UAddress |
| PAYMENTDETAILS | Strong | PayID | CardNo, CName, CCVV, CExpiry |
| MESSAGE | Strong | MID | MDate, Message |
| FEEDBACK | Strong | FID | Comment, Rating, FDate |

Relationship Set

|  |  |  |  |
| --- | --- | --- | --- |
| Relationship | Between | Cardinality | Participation |
| IS A | USER -> BUYER, SELLER | 1 to 1 | Disjoint Total Participation |
| Manage | SELLER - STORE | 1 to 1 | Partial Participation |
| Cart | BUYER - PRODUCT | M to M | Partial Participation |
| Contains | PRODUCT - STORE | M to 1 | Total Participation for Product |
| Places | BUYER – ORDER | 1 to M | Total Participation for ORDER |
| Has | BUYER – DELIVERY | 1 to M | Partial Participation |
| Has | ORDERS – DELIVERY | M to 1 | Partial Participation |
| Contains | PRODUCT - ORDER | M to M | Partial Participation |
| Payment | PAYMENTDETAILS – ORDER | 1 to 1 | Total Participation for both entities |
| Leaves | BUYER - MESSAGE | 1 to M | Total Participation for MESSAGE |
| Provides | BUYER - FEEDBACK | 1 to M | Total Participation for FEEDBACK |

**III. Entity Relationship Diagram**



**IV. Relational Model**

Using the above relational & entity sets, we have 7 Strong Entities whose attributes will be as per the table.

To resolve the relations, one – by – one we: -

1. IS A – Since, it is a Disjoin Total Participation we only take its children (BUYER & SELLER) into account.
2. Manage – It is a 1 to 1 partial participation relationship between SELLER & STORE. Hence, they receive each other’s primary key as foreign key.
3. Cart – A Many to Many Relationship between BUYER & PRODUCT. Hence, it has its own separate table with a composite primary key (PID, UserID) & attribute Quantity
4. Contains – It is a 1 to Many Relationship between STORE & PRODUCT with PRODUCT’s Total participation. Hence, PRODUCT takes SID as foreign key.
5. Places - It is a 1 to Many relationship between BUYER & ORDER. Hence, ORDER takes UserID as foreign key.
6. Has – It is a 1 to Many Relationship between BUYER & DELIVERY. Hence, DELIVERY gets UserID from BUYER as foreign key.
7. Has – It is a Many to 1 relationship between ORDERS & DELIVERY. Hence, ORDERS receives AID as foreign key.
8. Contains – It is a Many to Many Relationship between ORDERS & PRODUCTS. Hence, we create a separate table & it also has Quantity.
9. Payment – It is a Total participation 1 to 1 Relationship between ORDERS & PAYMENTDETAILS. Hence, we add the primary keys of each other as foreign keys.
10. Leaves – It is a 1 to M Relationship between BUYER & MESSAGE with MESSAGE in Total Participation. Hence, we add BUYER’s UserID as foreign key.
11. Provides – It is a 1 to M Relationship between BUYER & FEEDBACK with FEEDBACK in Total Participation. Hence, we add BUYER’s UserID as foreign key.

After doing all this, we get:

BUYER(UserID, UName, UPhone, UEmail, UDOB)

SELLER(UserID, UName, UPhone, UEmail, UDOB, SType, SID)

STORE(SID, SState, SCity, SStreet, SName, UserID)

Cart(UserID, PID, Quantity)

PRODUCT(PID, PDes, PPrice, PName, PBrand, PQuantity, PCat, SID)

ORDERS(OID, ODate, OStatus, OTotal, AID, UserID, PayID)

Contains(PID, OID, Quantity)

PAYMETNDETAILS(PayID, CardNo, CName, CExpiry, CCVV, OID)

DELIVERY(AID, UPin, Ucity, UAddress, UserID)

MESSAGE(MID, MDate, Message, UserID)

FEEDBACK(FID, FDate, Comment, Rating, UserID)

**V. Normalization**

**First Normal Form (1NF)** - Elimination of Multi – Valued Attributes:

There are no multi-valued attributes in any of the given tables. Hence, they meet 1NF requirements

BUYER(UserID, UName, UPhone, UEmail, UDOB)

SELLER(UserID, UName, UPhone, UEmail, UDOB, SType, SID)

STORE(SID, SState, SCity, SStreet, SName, UserID)

Cart(UserID, PID, Quantity)

PRODUCT(PID, PDes, PPrice, PName, PBrand, PQuantity, PCat, SID)

ORDERS(OID, ODate, OStatus, OTotal, AID, UserID, PayID)

Contains(PID, OID, Quantity)

PAYMETNDETAILS(PayID, CardNo, CName, CExpiry, CCVV, OID)

DELIVERY(AID, UPin, Ucity, UAddress, UserID)

MESSAGE(MID, MDate, Message, UserID)

FEEDBACK(FID, FDate, Comment, Rating, UserID)

**Second Normal Form (2NF) –** Elimination of partial dependencies

BUYER(UserID, UName, UPhone, UEmail, UDOB) – It is already in 2NF since there is a single priamry key (UserID)

SELLER(UserID, UName, UPhone, UEmail, UDOB, SType, SID) – It is already in 2NF since there is a single primary key (UserID)

STORE(SID, SState, SCity, SStreet, SName, UserID) - It is already in 2NF since there is a single primary key (SID)

Cart(UserID, PID, Quantity) – It is in 2NF with a composite primary key (SID, UserID)

PRODUCT(PID, PDes, PPrice, PName, PBrand, PQuantity, PCat, SID) - The primary key is PID, and all other attributes are fully functionally dependent on it.

ORDERS(OID, ODate, OStatus, OTotal, AID, UserID, PayID) – The primary key is OID, and all other attributes are fully functionally dependent on it.

Contains(PID, OID, Quantity) – It is in 2NF with a composite primary key (PID, OID)

PAYMETNDETAILS(PayID, CardNo, CName, CExpiry, CCVV, OID) – The primary key is PayID, and all other attributes are fully functionally dependent on it.

DELIVERY(AID, UPin, Ucity, UAddress, UserID) – The primary key is AID, and all other attributes are fully functionally dependent on it.

MESSAGE(MID, MDate, Message, UserID) – The primary key is MID, and all other attributes are fully functionally dependent on it.

FEEDBACK(FID, FDate, Comment, Rating, UserID) – The primary key is FID, and all other attributes are fully functionally dependent on it.

Hence, the above tables are in 2NF

**Third Normal Form (3NF) –** Eliminating Transitive Dependencies

BUYER(UserID, UName, UPhone, UEmail, UDOB) – No non-prime attribute depends on another non-prime attribute — 3NF is satisfied.

SELLER(UserID, UName, UPhone, UEmail, UDOB, SType, SID) – No non-prime attribute depends on another non-prime attribute — 3NF is satisfied.

STORE(SID, SState, SCity, SStreet, SName, UserID) - No non-prime attribute depends on another non-prime attribute — 3NF is satisfied.

Cart(UserID, PID, Quantity) – No transitive dependencies — 3NF is satisfied

PRODUCT(PID, PDes, PPrice, PName, PBrand, PQuantity, PCat, SID) - No transitive dependencies — 3NF is satisfied

ORDERS(OID, ODate, OStatus, OTotal, AID, UserID, PayID) – No transitive dependencies — 3NF is satisfied.

Contains(PID, OID, Quantity) – No transitive dependencies — 3NF is satisfied

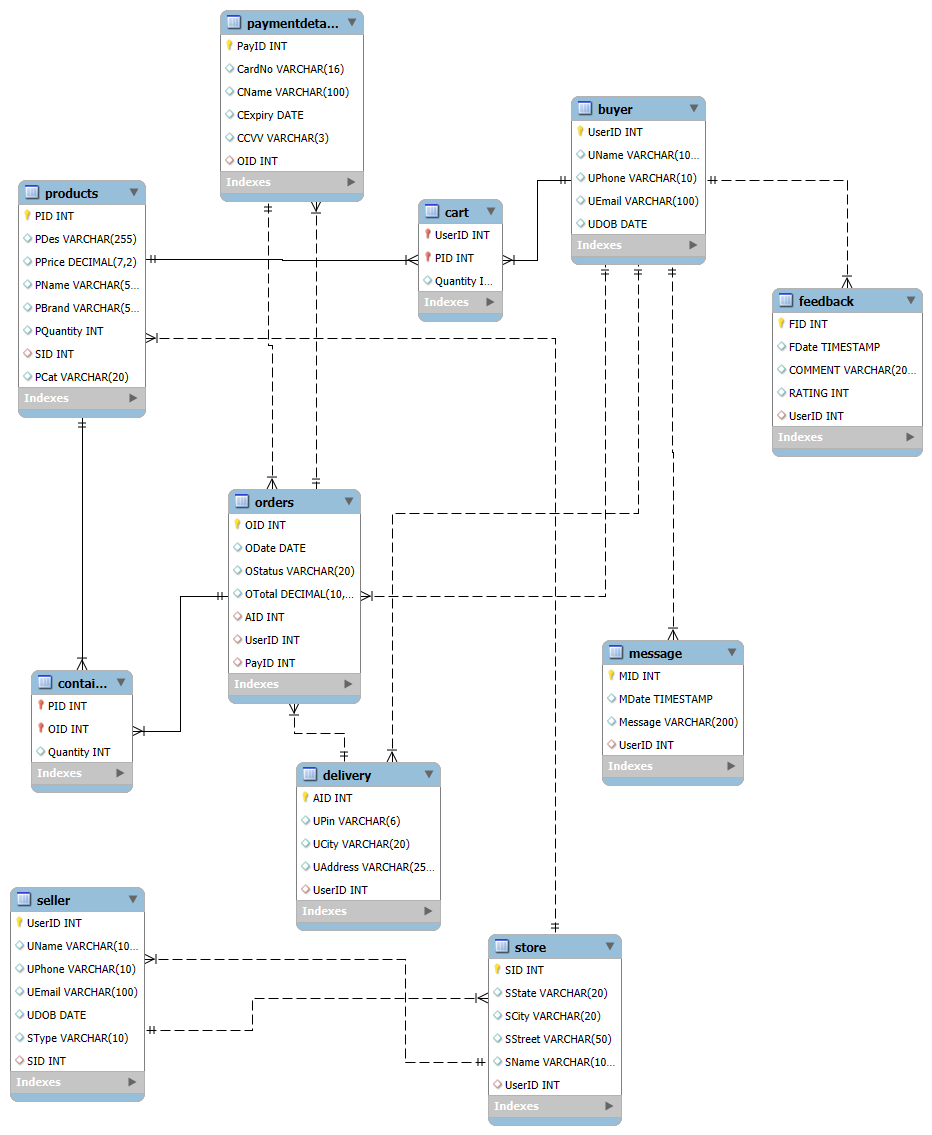
PAYMETNDETAILS(PayID, CardNo, CName, CExpiry, CCVV, OID) – No transitive dependencies — 3NF is satisfied

DELIVERY(AID, UPin, Ucity, UAddress, UserID) – No transitive dependencies — 3NF is satisfied

MESSAGE(MID, MDate, Message, UserID) – No transitive dependencies — 3NF is satisfied

FEEDBACK(FID, FDate, Comment, Rating, UserID) – No transitive dependencies — 3NF is satisfied

Hence, the above tables have been normalized till 3NF



Schema Representation of the database

**VI. SQL Queries**

Creation of the tables – <https://github.com/JeevalShah/Online-Sports-Gear-Store>

Insertion of Tuples – <https://github.com/JeevalShah/Online-Sports-Gear-Store>

USE DBMS;

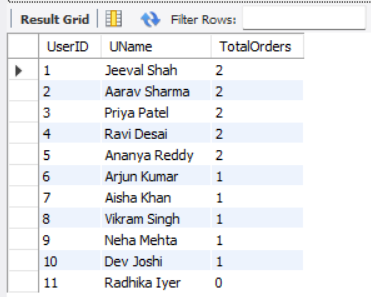
-- 1. Total number of orders per buyer

SELECT b.UserID, b.UName, COUNT(o.OID) AS TotalOrders

FROM BUYER b

LEFT OUTER JOIN ORDERS o ON b.UserID = o.UserID

GROUP BY b.UserID, b.UName;



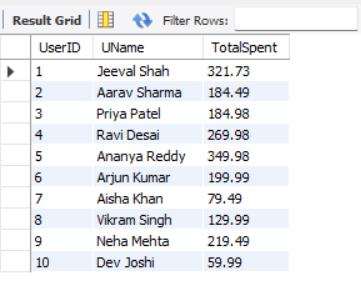
-- 2. Total amount spent by each buyer

SELECT b.UserID, b.UName, SUM(o.OTotal) AS TotalSpent

FROM BUYER b

NATURAL JOIN ORDERS o

GROUP BY b.UserID, b.UName;

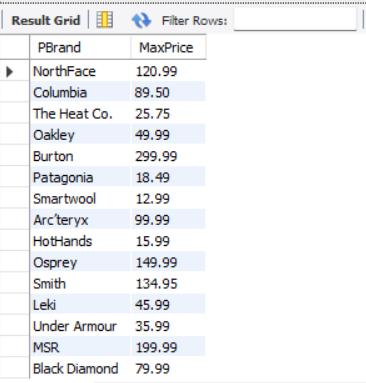


-- 3. Most expensive product per brand

SELECT PBrand, MAX(PPrice) AS MaxPrice

FROM PRODUCT

GROUP BY PBrand;



-- 4. Products sold more than 2 times

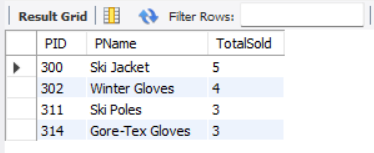
SELECT p.PID, p.PName, SUM(c.Quantity) AS TotalSold

FROM PRODUCT p

JOIN CONTAINS c ON p.PID = c.PID

GROUP BY p.PID, p.PName

HAVING SUM(c.Quantity) > 2;



-- 5. Sellers and their product counts

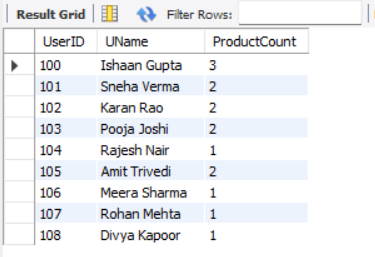
SELECT s.UserID, s.UName, COUNT(p.PID) AS ProductCount

FROM (SELLER s

JOIN STORE st ON s.UserID = st.UserID)

JOIN PRODUCT p ON st.SID = p.SID

GROUP BY s.UserID, s.UName;



-- 6. Top 5 most purchased products

SELECT p.PID, p.PName, SUM(c.Quantity) AS PurchasedQty

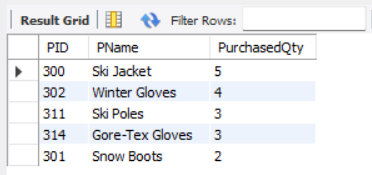
FROM PRODUCT p

JOIN CONTAINS c ON p.PID = c.PID

GROUP BY p.PID, p.PName

ORDER BY PurchasedQty DESC

LIMIT 5;

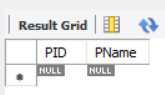


-- 7. Out of stock products

SELECT PID, PName

FROM PRODUCT

WHERE PQuantity = 0;



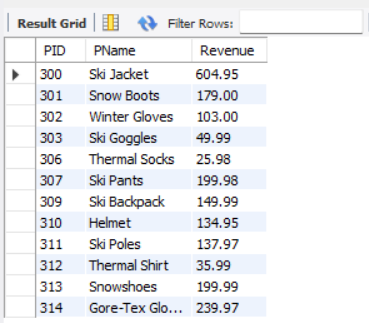
-- 8. Revenue per product

SELECT p.PID, p.PName, SUM(c.Quantity \* p.PPrice) AS Revenue

FROM PRODUCT p

JOIN CONTAINS c ON p.PID = c.PID

GROUP BY p.PID, p.PName;



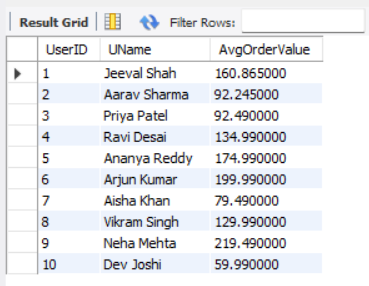
-- 9. Average order value per buyer

SELECT b.UserID, b.UName, AVG(o.OTotal) AS AvgOrderValue

FROM BUYER b

JOIN ORDERS o ON b.UserID = o.UserID

GROUP BY b.UserID, b.UName;



-- 10. Products in more than 3 orders

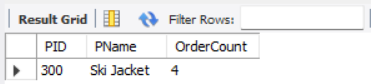
SELECT c.PID, p.PName, COUNT(DISTINCT c.OID) AS OrderCount

FROM CONTAINS c

JOIN PRODUCT p ON c.PID = p.PID

GROUP BY c.PID, p.PName

HAVING COUNT(DISTINCT c.OID) > 3;



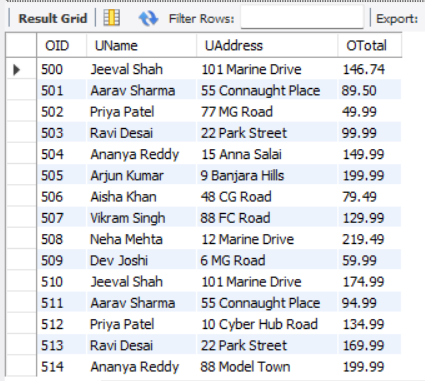
-- 11. Orders with buyer and address

SELECT o.OID, b.UName, d.UAddress, o.OTotal

FROM ORDERS o

JOIN BUYER b ON o.UserID = b.UserID

JOIN DELIVERY d ON o.AID = d.AID;



-- 12. Sellers without products

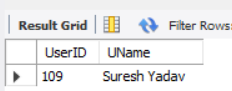
SELECT s.UserID, s.UName

FROM SELLER s

LEFT JOIN STORE st ON s.UserID = st.UserID

LEFT JOIN PRODUCT p ON st.SID = p.SID

WHERE p.PID IS NULL;



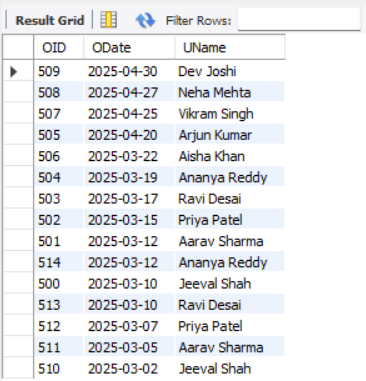
-- 13. Recent Orders and Buyers

SELECT o.OID, o.ODate, b.UName

FROM ORDERS o

JOIN BUYER b ON o.UserID = b.UserID

ORDER BY o.ODate DESC;

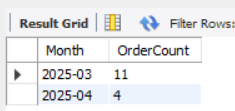


-- 14. Orders placed each month

SELECT DATE\_FORMAT(ODate, '%Y-%m') AS Month, COUNT(OID) AS OrderCount

FROM ORDERS

GROUP BY DATE\_FORMAT(ODate, '%Y-%m');



-- 15. Revenue by Each Store

SELECT s.SID, s.SName, SUM(p.PPrice \* c.Quantity) AS Revenue

FROM STORE s

JOIN PRODUCT p ON s.SID = p.SID

JOIN CONTAINS c ON p.PID = c.PID

GROUP BY s.SID, s.SName;

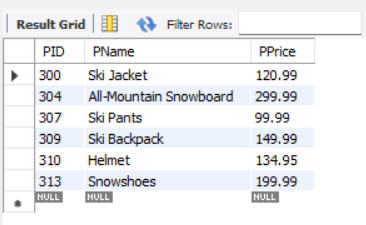


-- 16. Products above average price

SELECT PID, PName, PPrice

FROM PRODUCT

WHERE PPrice > (SELECT AVG(PPrice) FROM PRODUCT);

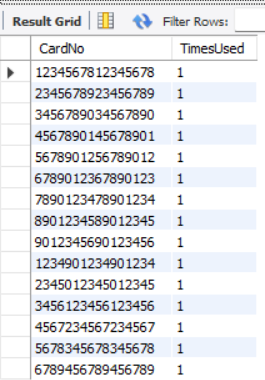


-- 17. Payment methods used and count

SELECT CardNo, COUNT(\*) AS TimesUsed

FROM PAYMENTDETAILS

GROUP BY CardNo;



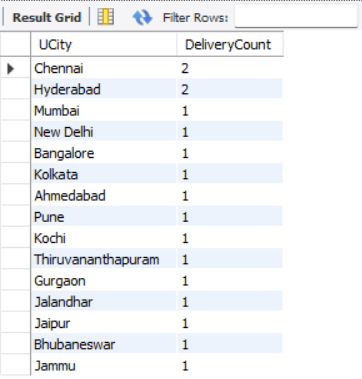
-- 18. Count of Deliveries to Cities

SELECT UCity, COUNT(\*) AS DeliveryCount

FROM DELIVERY

GROUP BY UCity

ORDER BY DeliveryCount DESC;



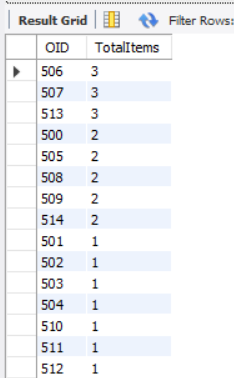
-- 19. Quantity of items per order

SELECT OID, SUM(Quantity) AS TotalItems

FROM CONTAINS

GROUP BY OID

ORDER BY TotalItems DESC;



-- 20. Value of Items in Buyer's Cart

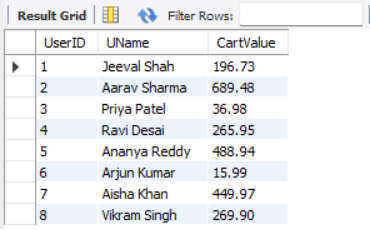
SELECT b.UserID, b.UName, SUM(p.PPrice \* c.Quantity) AS CartValue

FROM BUYER b

JOIN CART c ON b.UserID = c.UserID

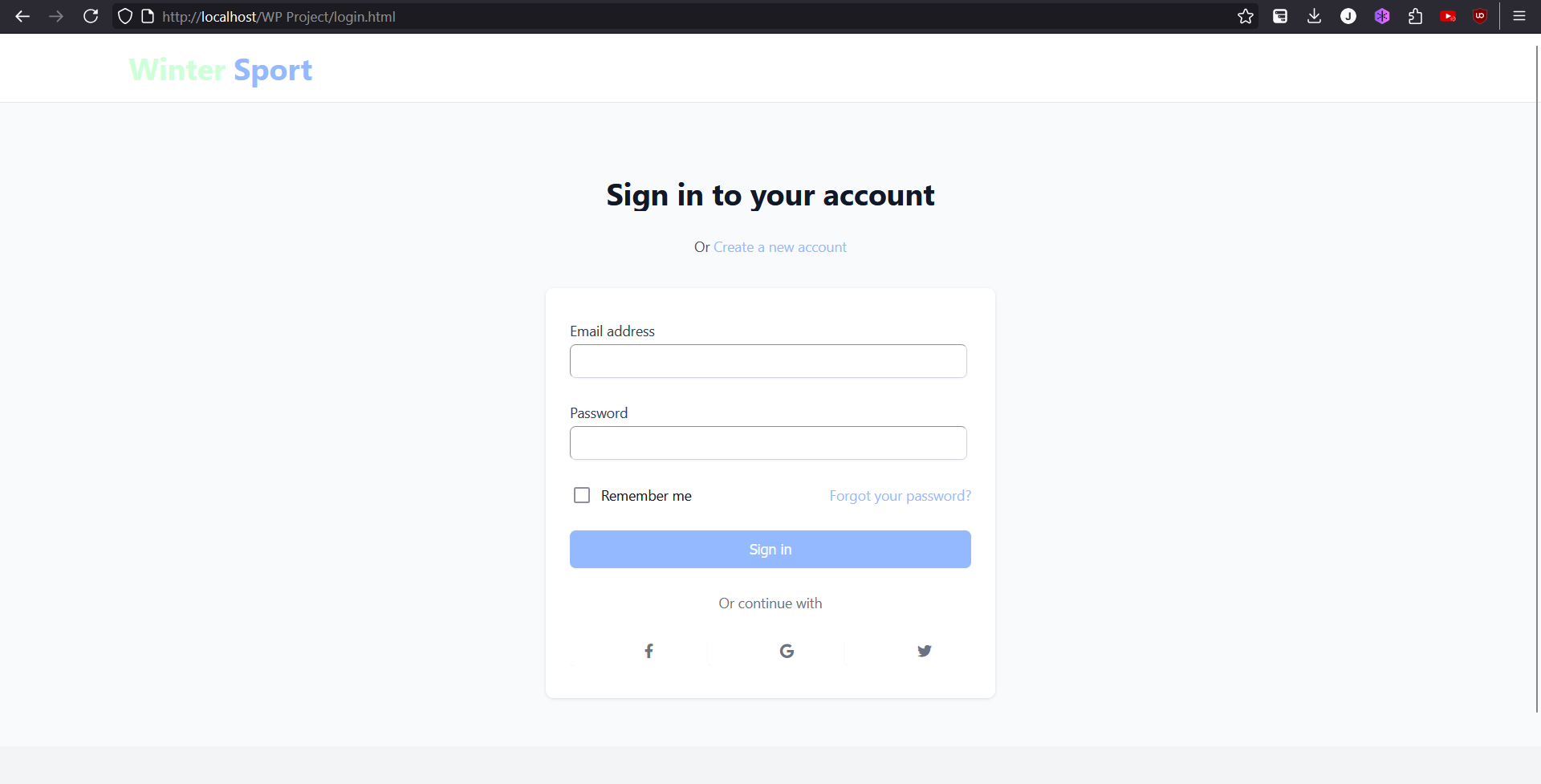
JOIN PRODUCT p ON c.PID = p.PID

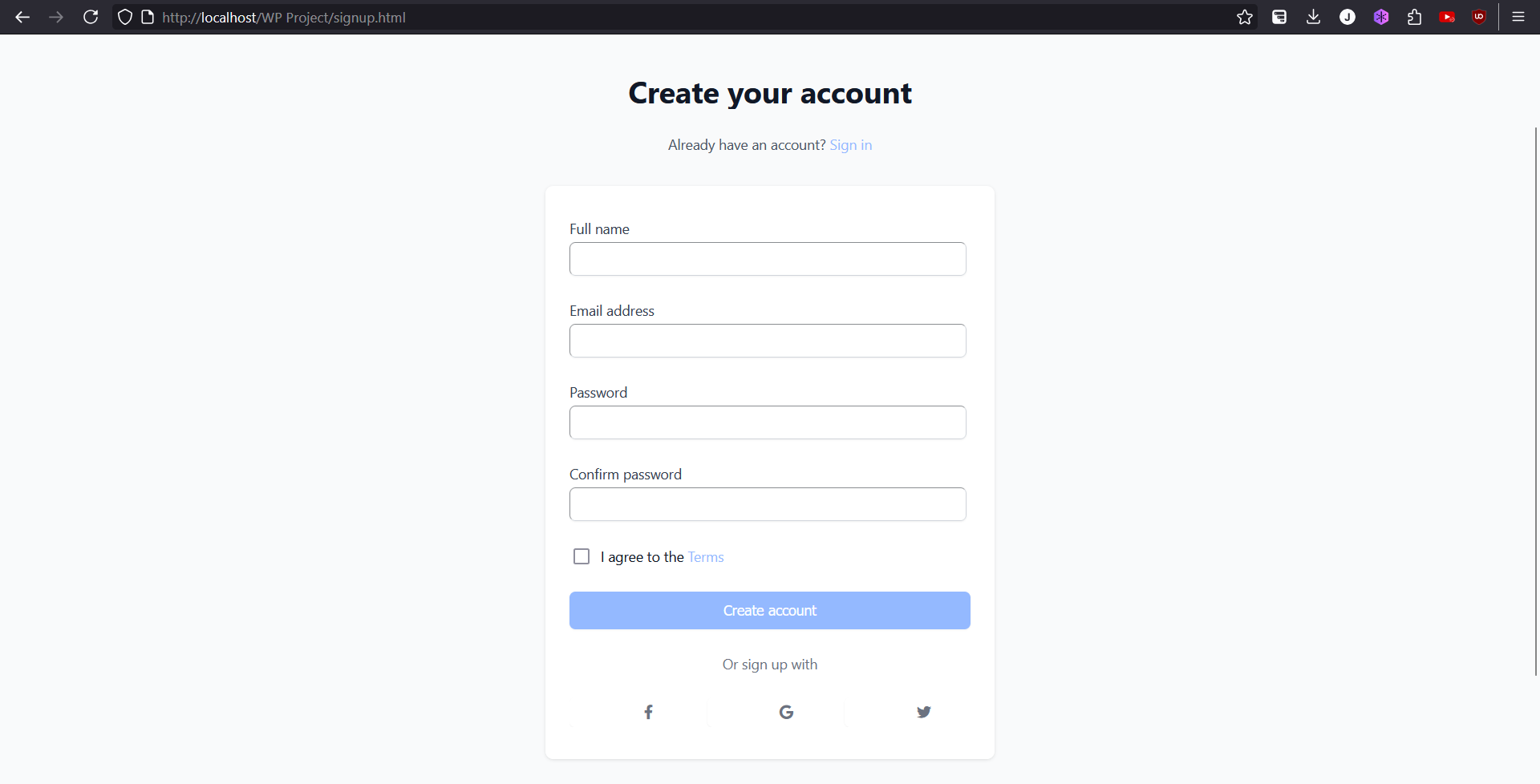
GROUP BY b.UserID, b.UName;

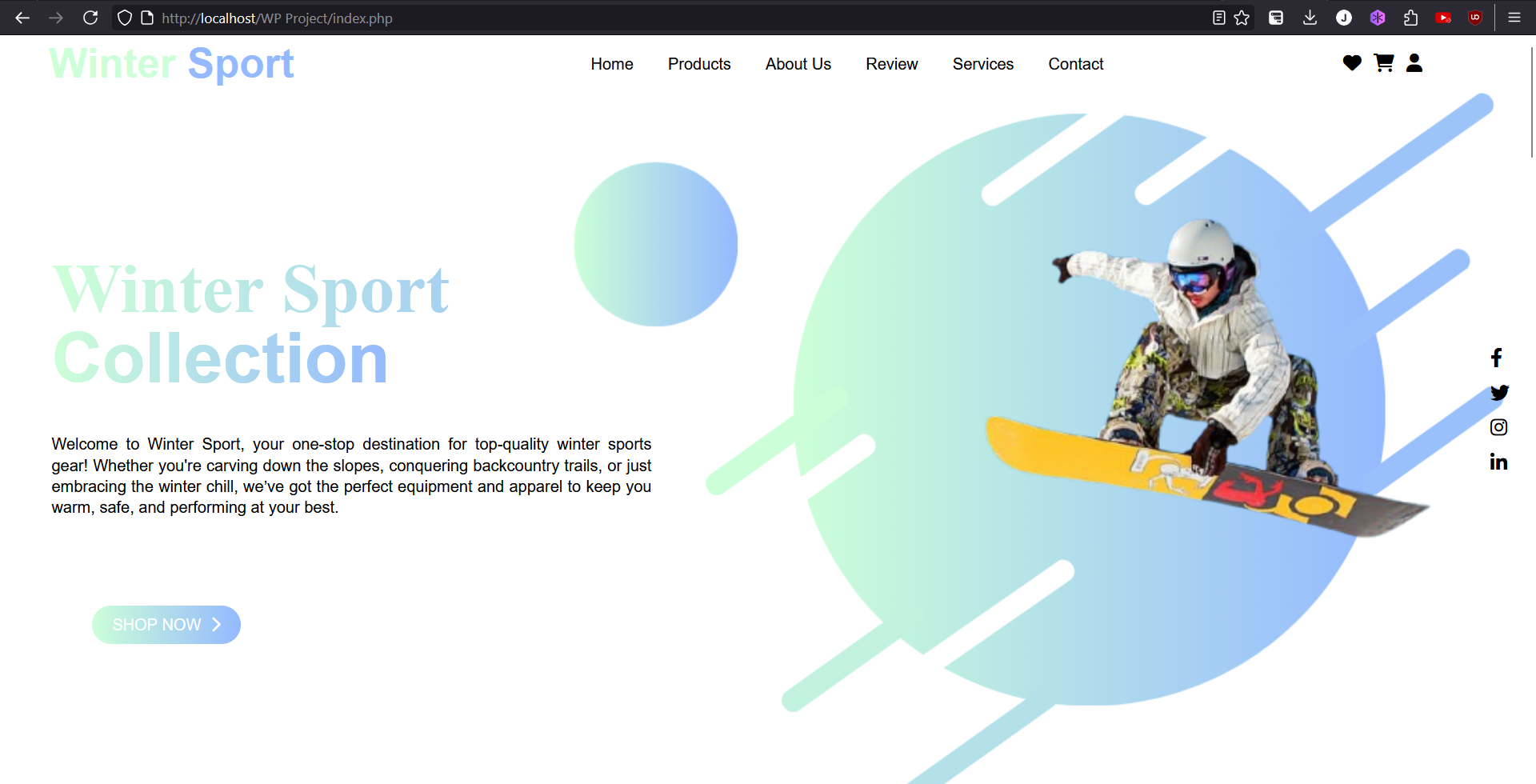


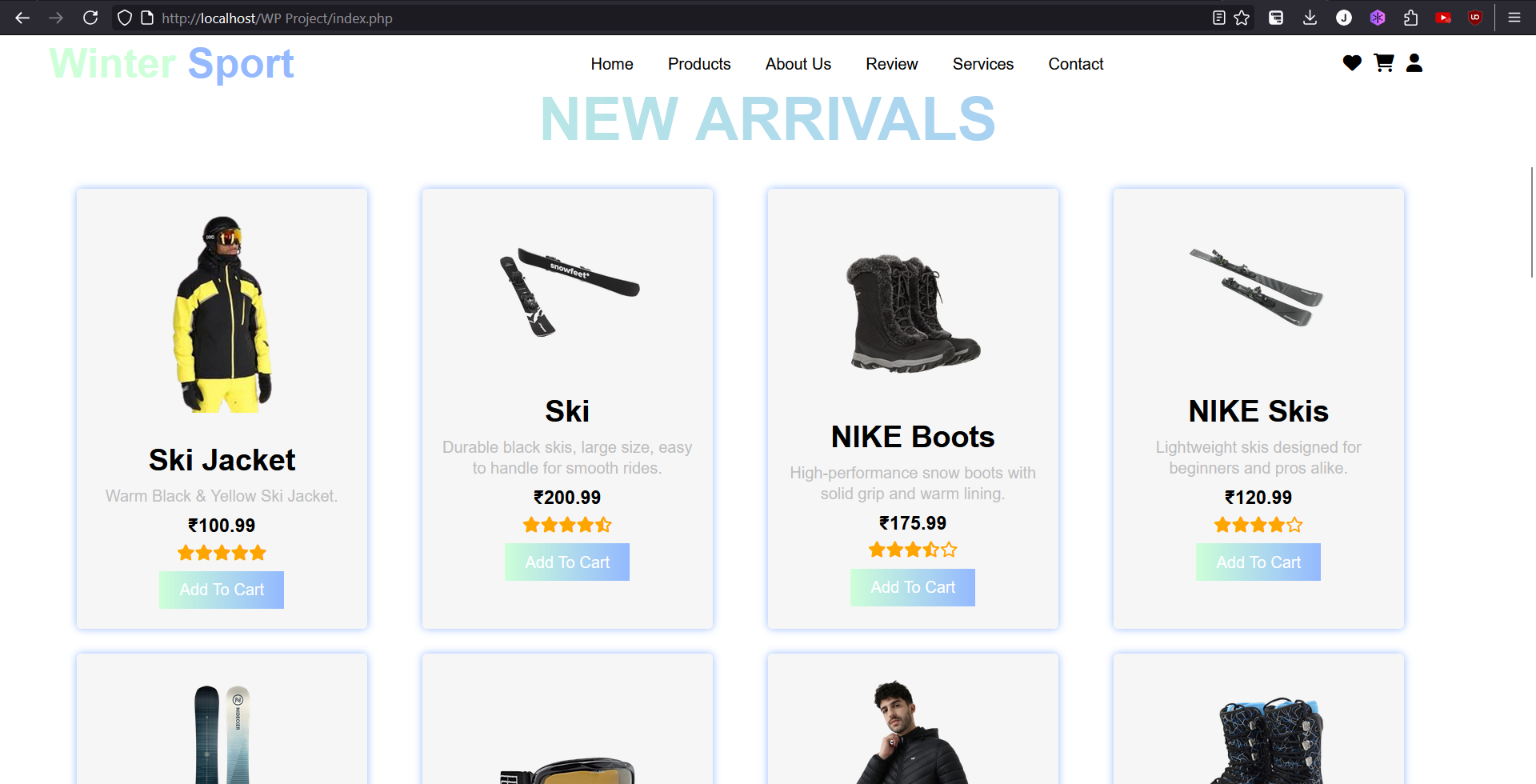
**VI. Project demonstration**

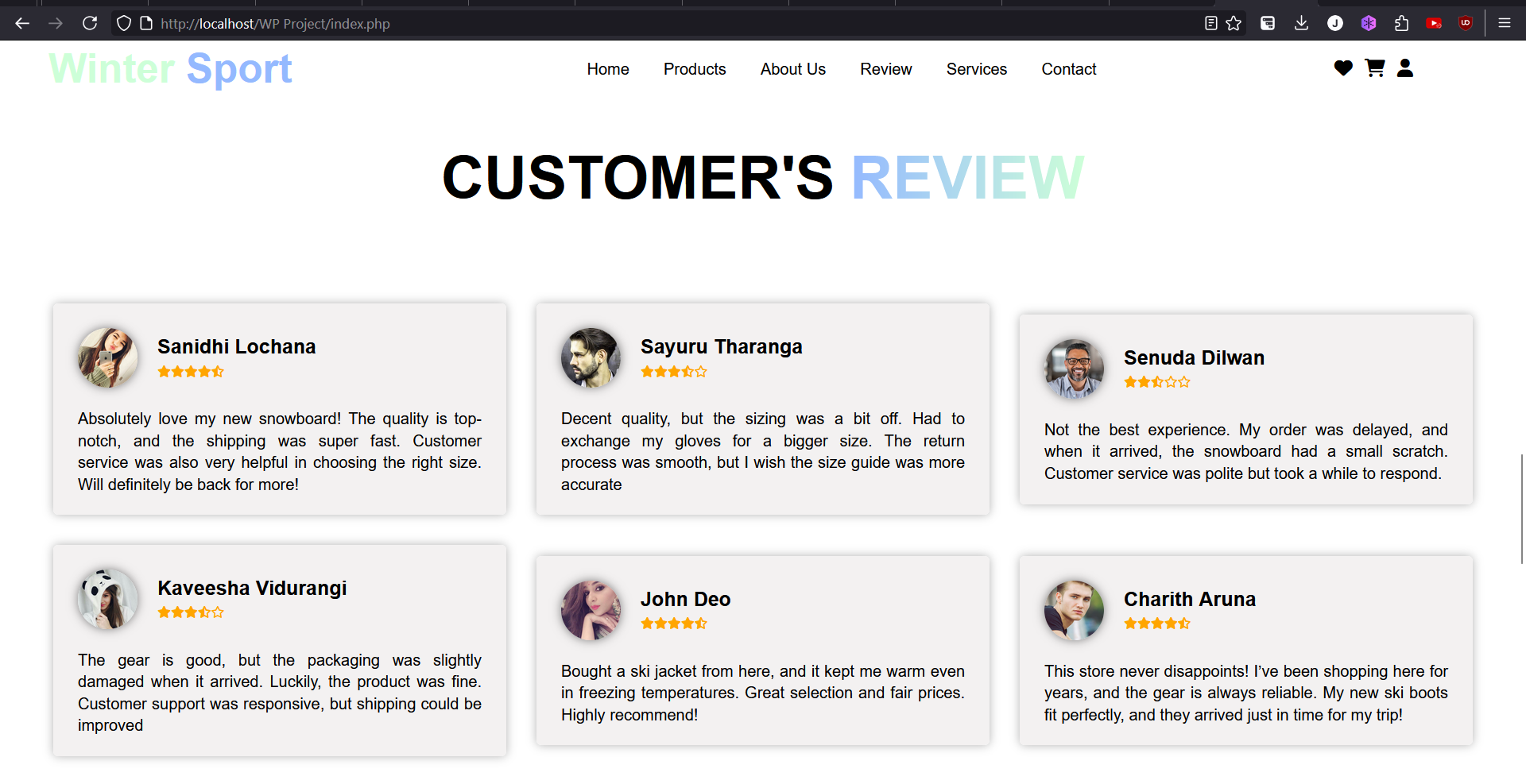
Project Demonstration: GitHub Link: <https://github.com/JeevalShah/WP-Project>

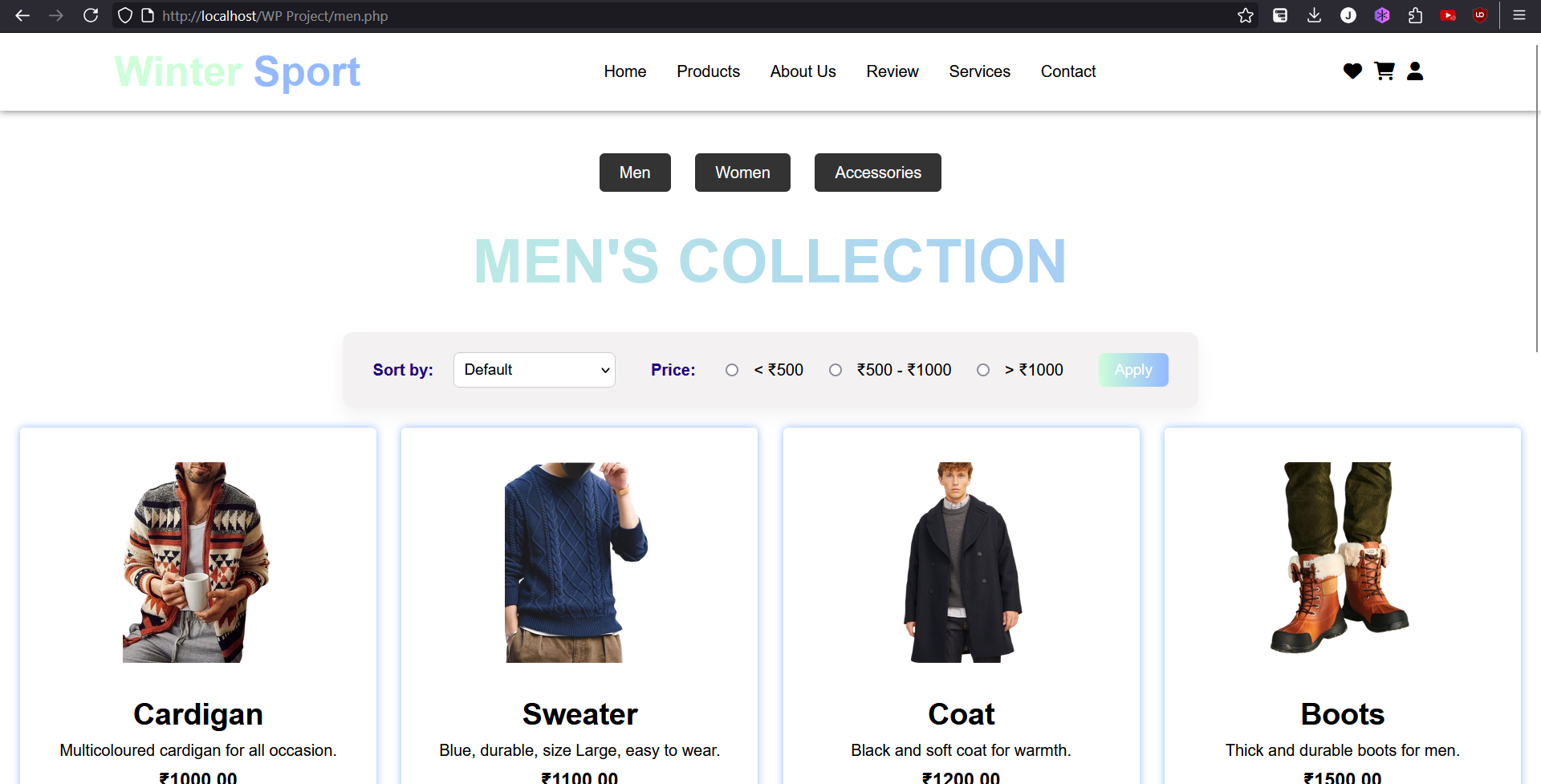


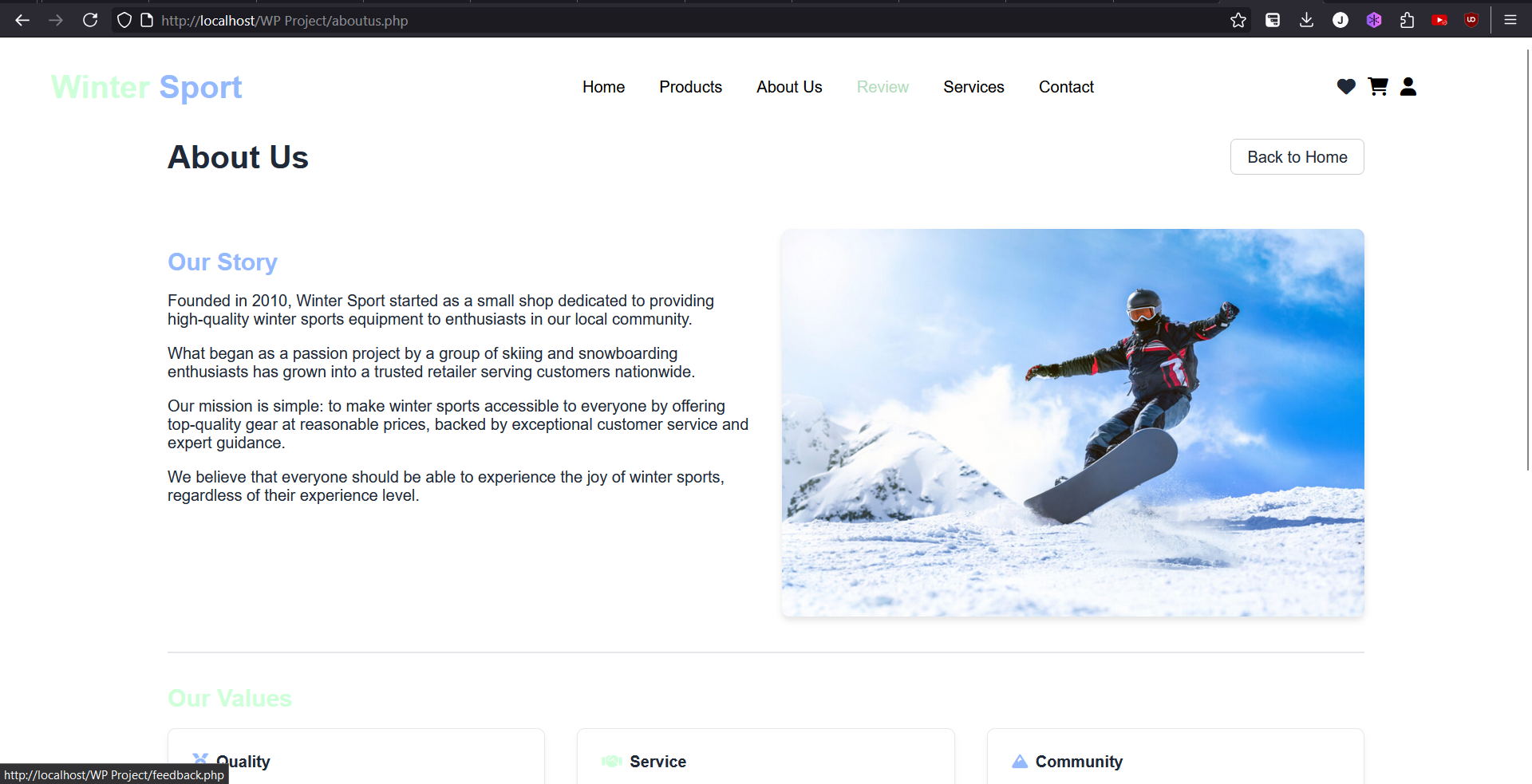


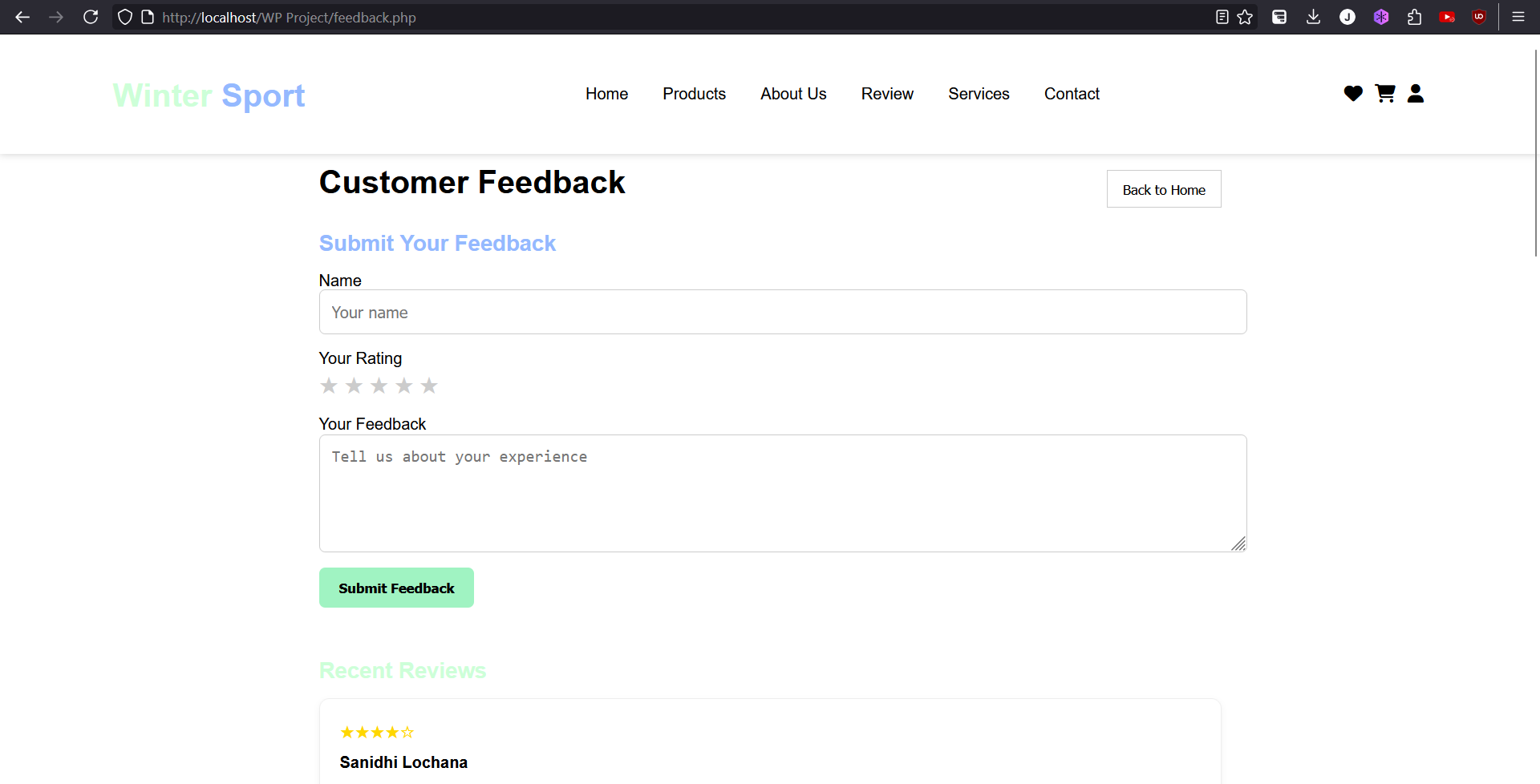


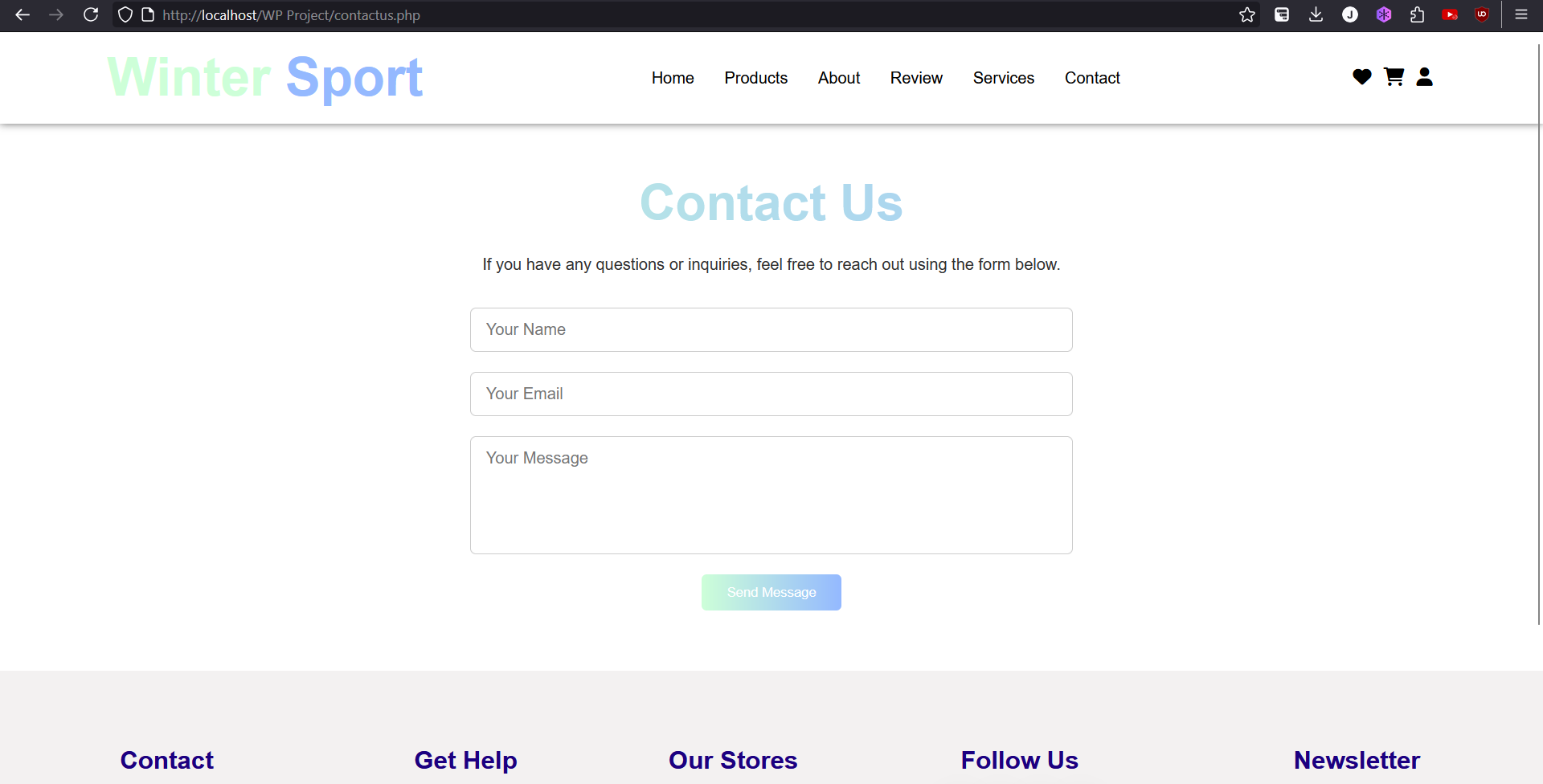


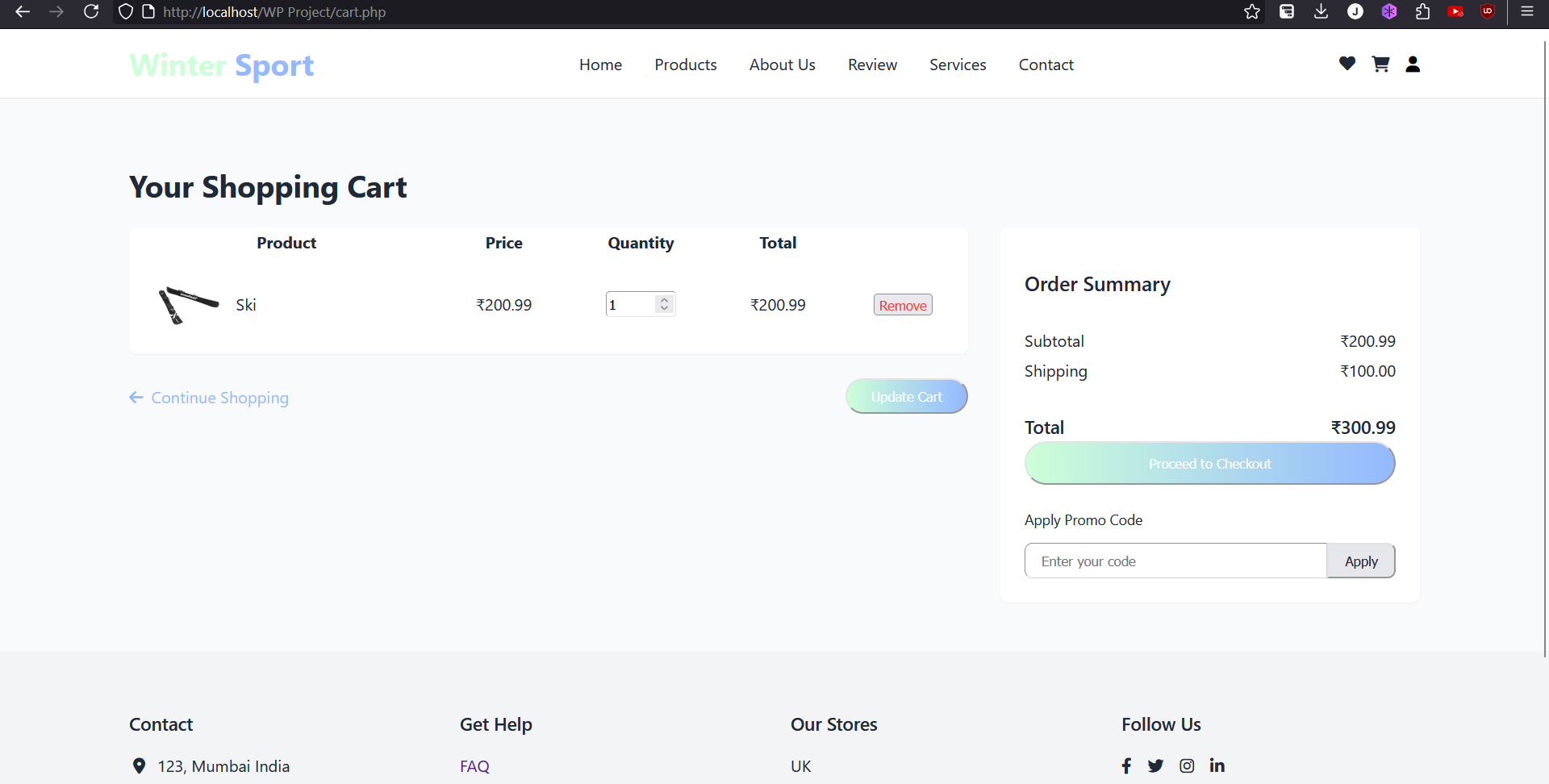


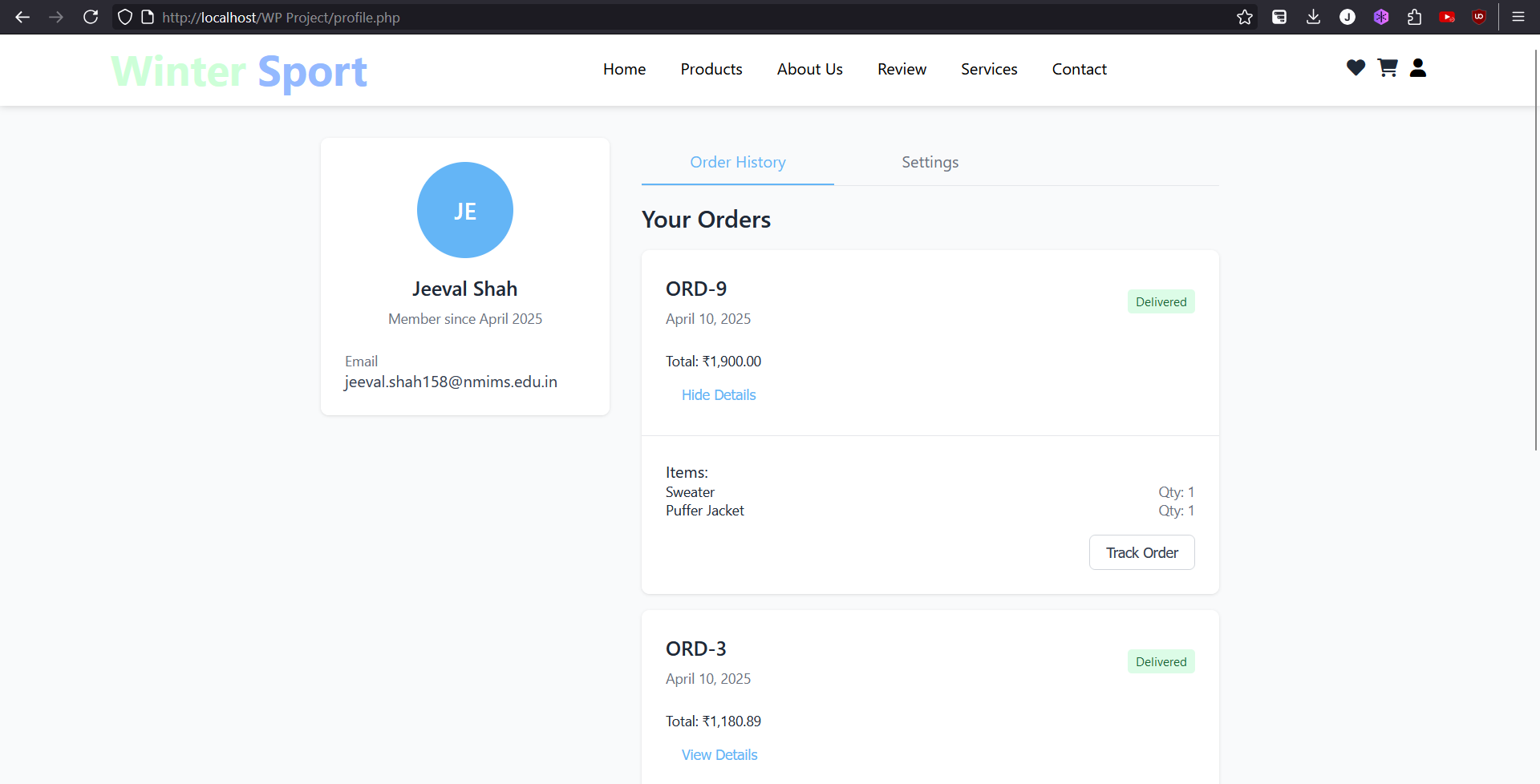


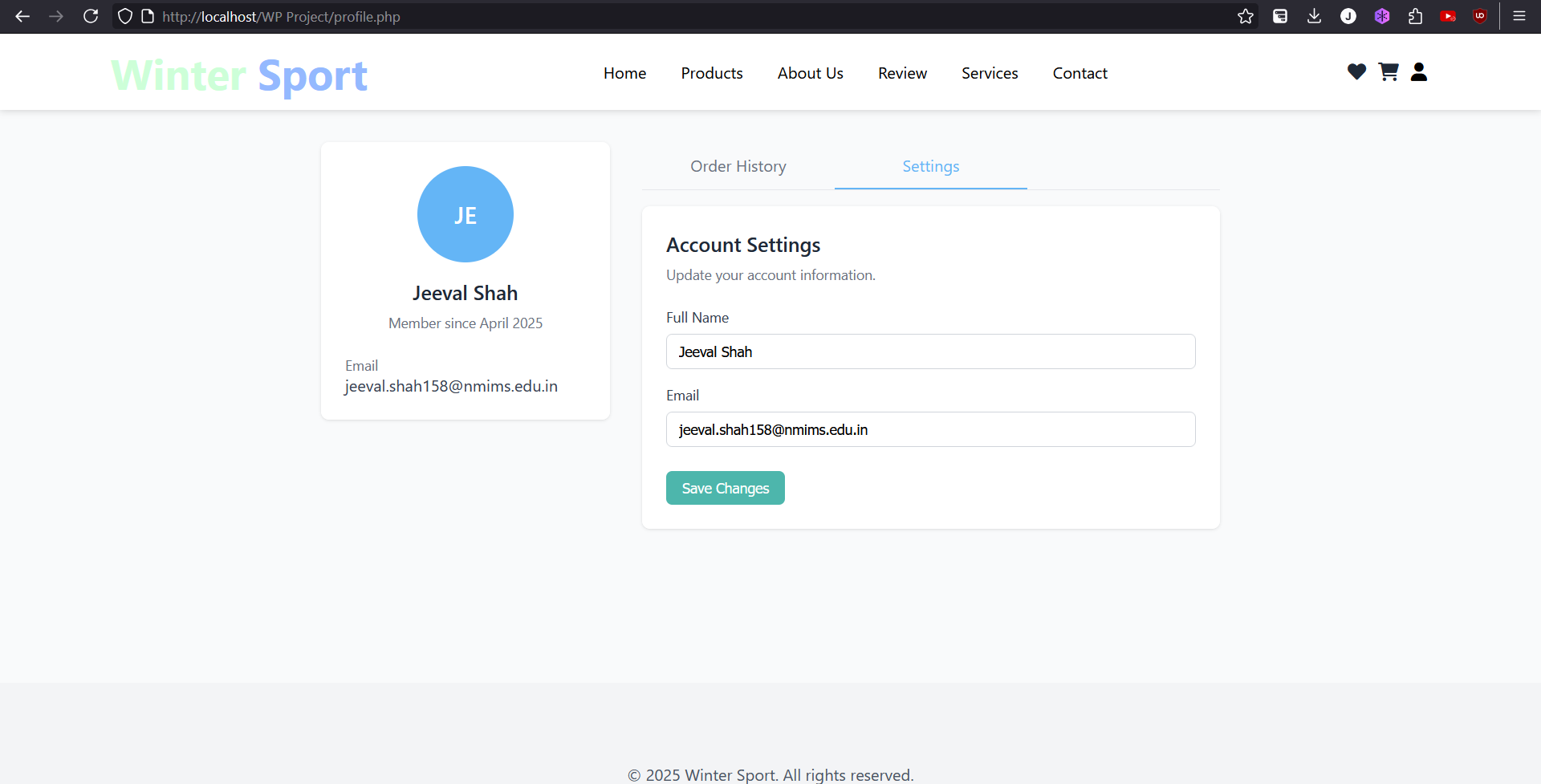












**VII. Self -Learning beyond classroom**

While working on the DBMS mini-project, I went beyond classroom teachings to explore several new concepts and tools. I learned how to design an efficient ER diagram and convert it into a normalized relational schema, ensuring data consistency and avoiding redundancy.

I also learned how to write complex SQL queries, including nested subqueries, aggregate functions, and joins to retrieve meaningful insights from the database.

Additionally, I experimented with frontend-database integration using tools like PHP and MySQL Workbench, which was not covered in class but added a practical edge to the project.

This process of self-learning allowed me to better understand real-world database applications and strengthened my ability to work independently, debug efficiently, and approach problems with a logical mindset.

**VIII. Learning from the Project**

This project significantly enhanced my understanding of how database systems work in real-world applications. I gained hands-on experience in designing ER diagrams, normalizing data, and converting conceptual designs into structured relational schemas.

It also improved my skills in SQL query writing, including the use of joins, aggregate functions, subqueries, and constraints to manage and retrieve data efficiently. Implementing triggers and stored procedures gave me practical insight into how business logic can be embedded directly within the database.

Beyond technical skills, this project helped me improve my problem-solving abilities, collaboration skills, and attention to detail while debugging. I also learned the importance of planning a clear schema early in the development process to avoid complications later.

Overall, the project bridged the gap between theoretical concepts learned in class and their practical implementation, preparing me better for industry-level database work.

**IX. Challenges Faced**

During the course of this project, we encountered several challenges. One of the initial hurdles was in the designing phase of the ER diagram — ensuring all entities, relationships, and attributes were captured correctly and logically without redundancy.

Another challenge was in normalizing the database without losing any critical information. Striking a balance between reducing data redundancy and maintaining query performance took time and several iterations.

We also faced difficulties in writing complex SQL queries, especially involving nested subqueries and multi-table joins. Additionally, implementing constraints and triggers required a deep understanding of database rules to ensure data integrity without restricting necessary operations.

On the frontend side, one of the major difficulties was in connecting the user interface with the database. Ensuring smooth data flow from form inputs to the backend required proper handling of requests, responses, and SQL execution. Creating intuitive UI/UX components, validating user inputs, and displaying results or errors clearly was also time-consuming. Styling the frontend to make it user-friendly while keeping the code manageable added another layer of challenge.

Finally, coordinating as a team, especially while managing frontend-backend integration and version control, required effective communication and regular testing to avoid conflicts and ensure everything worked smoothly.

**X. Conclusion**

This DBMS project has been a valuable learning experience, blending theory with real-world implementation. Our key takeaways include:

* A solid understanding of database design and normalization principles
* Proficiency in writing efficient and optimized SQL queries
* The importance of planning and organizing data logically
* Improved teamwork, communication, and debugging skills

Overall, the project gave us a deeper appreciation of how structured data management forms the backbone of most software systems and how crucial database design is to building scalable applications.