

Project Report

ShopForHome

Index

SrNo.	Topic	PgNo
1	Abstract	3-5
2	Implementation and Methodology	5-6
3	Understanding the Core Technologies Used	7-9
4	Backend Implementation	10-16
5	Frontend Implementation	16-34
6	Conclusion	34

Project Report ShopForHome

1. Abstract

The rise of e-commerce platforms has significantly transformed the way consumers interact with businesses, offering them convenience, flexibility, and accessibility to a wide range of products. As technology continues to evolve, modern web applications must be designed with efficiency, security, and user experience in mind. ShopForHome is an innovative full-stack e-commerce application built using ASP.NET 6.0 Web API and Angular 19, aimed at providing a seamless and interactive shopping experience for customers while equipping administrators with robust management capabilities. This project was developed to address key limitations in existing platforms, such as complex product searches, inefficient coupon management, and lack of structured role-based access control. By integrating modern development practices, ShopForHome offers a scalable and efficient solution tailored for small-to-medium-sized businesses looking to establish a reliable online presence.

The primary goal of this project is to design and implement a secure and scalable e-commerce system that incorporates role-based authentication, discount coupon management, and a dynamic product search mechanism. To achieve this, the system is structured into two main user roles:

1. Admin: Manages users, products, categories, and discount coupons.
2. Customer: Browses products, applies discounts, and completes purchases.

The backend is developed using ASP.NET 6.0 Web API, utilizing Entity Framework Core for database interactions. JWT authentication is implemented to ensure secure login and authorization processes, preventing unauthorized access to critical functionalities. The system follows the RESTful API design, ensuring that frontend applications can seamlessly interact with backend services. The Angular 19 frontend enhances user experience by providing a responsive and interactive interface with Bootstrap styling. Key features include product search with filters, category-based browsing, real-time price adjustments, and an intuitive cart system.

A significant aspect of ShopForHome is the discount coupon management system, which allows admins to create, update, and delete coupons while enabling customers to apply available discounts during checkout. This feature aims to enhance customer engagement and drive sales by offering promotional benefits. Unlike many conventional e-commerce platforms, this system ensures that coupon validation occurs both at the

frontend and backend levels, preventing unauthorized use or manipulation of discount codes.

The methodology adopted for this project follows an Agile development approach, with iterative sprints focusing on design, implementation, testing, and deployment. The project was structured using a Microservices architecture, enabling modular development and scalability. The SQL Server database was optimized for handling large product catalogs, ensuring that queries return results with minimal latency. Performance testing was conducted using JMeter, and Postman was used for API validation.

Security plays a crucial role in e-commerce applications, and ShopForHome incorporates multiple layers of security measures, including password hashing, JWT-based authentication, and role-based access control. By implementing these best practices, the system prevents unauthorized transactions, data breaches, and security vulnerabilities.

Challenges and Solutions

During the development of ShopForHome, several challenges were encountered:

1. Efficient Product Search and Filtering
 - Challenge: Optimizing database queries to support real-time filtering without performance degradation.
 - Solution: Used indexed database queries and caching mechanisms to improve response times.
2. Secure User Authentication & Authorization
 - Challenge: Preventing unauthorized access to admin functionalities.
 - Solution: Implemented JWT authentication with role-based access control (RBAC), restricting certain features to admin users.
3. Seamless API Integration Between Frontend and Backend
 - Challenge: Ensuring data consistency and smooth interaction between Angular and .NET API.
 - Solution: Used Reactive Forms in Angular with HTTP Interceptors for seamless data exchange.
4. Coupon Validation and Management
 - Challenge: Ensuring coupons are not misused or applied multiple times by unauthorized users.
 - Solution: Implemented backend validation mechanisms and coupon expiration policies.

To evaluate the system's effectiveness, extensive user acceptance testing (UAT) was conducted with a sample group of test users. The results indicated that the platform

offers an intuitive and smooth shopping experience, with an average usability rating of 4.5 out of 5. The search and filtering features were highlighted as key strengths, while minor usability enhancements were suggested for the admin dashboard.

Conclusion and Future Scope

In conclusion, ShopForHome successfully demonstrates the capabilities of a full-stack e-commerce application by integrating modern web technologies, secure authentication mechanisms, and a user-friendly interface. The project meets its objectives by providing a robust platform where admins can manage users, products, and discount coupons, and customers can browse and purchase products effortlessly.

Future enhancements include:

- AI-driven product recommendations based on user behavior.
- Integration of a payment gateway for real transactions.
- Mobile application support for a broader audience reach.
- Advanced order tracking and shipment management.

By leveraging scalable architecture and secure design principles, ShopForHome serves as a foundation for future enhancements, paving the way for next-generation e-commerce solutions that are adaptable, secure, and performance-driven.

2.Implementation and Methodology

Introduction

The ShopForHome project is developed using a modern full-stack architecture that combines ASP.NET 6.0 Web API for the backend, Angular 19 for the frontend, and SQL Server as the database. This technology stack ensures a scalable, high-performance, and secure e-commerce platform.

Backend: ASP.NET 6.0 Web API

The backend is built using ASP.NET Core Web API, which follows the RESTful architecture to expose endpoints that enable interaction between the frontend and the database. Entity Framework Core (EF Core) is used as an ORM (Object-Relational Mapper) to facilitate database operations, allowing developers to work with database objects using C# code. We implemented the Code First approach, which enables automatic database migrations and schema management through C# entity classes.

Frontend: Angular 19

The frontend is developed using Angular 19, a TypeScript-based framework that provides a responsive and dynamic user experience. Angular is chosen for its powerful component-based architecture, which allows for reusability and maintainability of UI elements. It interacts with the backend via HTTPClientModule to fetch and display data dynamically. Bootstrap styling is used to enhance the UI design without adding extra complexity.

Database: SQL Server

The SQL Server database is used to store all the application data, including user details, products, categories, and discount coupons. EF Core migrations ensure that schema changes are applied seamlessly without manual intervention. Stored procedures and optimized queries are used where necessary to enhance performance.

By integrating these technologies, we ensure that ShopForHome is a robust, scalable, and efficient e-commerce platform capable of handling high user traffic while providing a smooth and secure shopping experience.

Let's look at some of the key features of ShopForHome Application.

1. User Authentication & Role-Based Access
 - Customers can browse products, apply discount coupons, and place orders.
 - Admins can manage products, categories, and users, ensuring seamless platform administration.
2. Product & Category Management
 - Products are categorized for easy navigation.
 - Admins can add, update, or delete products and categories dynamically.
3. Discount Coupon System
 - Customers can apply discount coupons at checkout for price reductions.
 - Admins have full control over creating, updating, and deleting coupons.
4. Secure and Scalable Architecture
 - JWT-based authentication ensures data security.
 - The system is built for scalability, allowing future enhancements like order tracking and payment integration.

By leveraging ASP.NET Core, Angular, and SQL Server, the ShopForHome project ensures a seamless, responsive, and secure e-commerce experience.

3.Understanding the Core Technologies Used

ASP.NET Core Web API

ASP.NET Core Web API is a lightweight and powerful framework for building RESTful APIs. It provides an easy way to expose data and functionalities to external clients, such as frontend applications or third-party integrations.

Key Features of ASP.NET Core Web API:

- Cross-Platform Compatibility: Runs on Windows, Linux, and macOS.
- Built-in Dependency Injection: Enhances code maintainability and modularity.
- Middleware Pipeline: Controls request processing with middleware components.
- Security: Supports JWT authentication, role-based access control (RBAC), and authorization policies.
- Asynchronous Processing: Uses async/await to handle requests efficiently.

In ShopForHome, ASP.NET Core Web API is used to:

- Handle user authentication and role-based access (Admin, Customer).
- Expose product-related endpoints for listing, searching, and filtering products.
- Implement CRUD operations for users, products, categories, and coupons.
- Validate and apply discount coupons securely before order placement.

Angular 19

Angular 19 is a TypeScript-based framework for building single-page applications (SPAs). It enables dynamic and responsive user interfaces that update without requiring full-page reloads.

Key Features of Angular 19:

- Component-Based Architecture: Encourages reusability and modular development.
- Two-Way Data Binding: Synchronizes data between the UI and business logic.
- Reactive Forms: Enables dynamic form validation and handling.
- Routing Module: Manages navigation without reloading pages.
- Service-Based API Calls: Uses HttpClientModule to interact with RESTful APIs.

In ShopForHome, Angular 19 is used to:

- Create an interactive product catalog with filtering and searching options.
- Manage the admin dashboard for CRUD operations on products, users, and discount coupons.

- Provide a seamless user experience for customers browsing and purchasing products.
- Ensure security through token-based authentication using JWTs.

SQL Server

SQL Server is a relational database management system (RDBMS) used to store structured data efficiently. It provides robust support for transactions, indexing, and security policies.

Key Features of SQL Server:

- Scalability: Supports large datasets with optimized indexing and querying.
- Security: Provides authentication and authorization mechanisms for data protection.
- Stored Procedures and Views: Enhances performance for repetitive database queries.
- Integration with Entity Framework Core: Facilitates seamless database interactions.

In ShopForHome, SQL Server is used to store and manage:

- User information and authentication credentials.
- Product details, categories, and discount coupon data.
- Order details and purchase history.

Code First Approach & Entity Framework Core

What is the Code First Approach?

The Code First Approach in Entity Framework Core (EF Core) is a method where the database schema is generated automatically from C# entity classes. Instead of designing the database first and then mapping it to the application, developers define models (classes) in C# that represent database tables, and EF Core takes care of creating the database schema.

Advantages of the Code First Approach:

1. Faster Development: Eliminates the need for manual database scripting.
2. Schema Evolution: Developers can modify entity classes and use EF Core Migrations to update the database structure.
3. Improved Maintainability: Business logic and database schema remain in sync.

Entity Framework Core in ShopForHome

Entity Framework Core (EF Core) is the ORM used in our backend to manage database operations efficiently. It enables developers to interact with the database using LINQ queries instead of raw SQL.

Examples of how EF Core is Used in ShopForHome:

Modeling Database Tables:

```
public class Product
{
    public int ProductId { get; set; }
    public string Name { get; set; }
    public decimal Price { get; set; }
    public string Description { get; set; }
    public int CategoryId { get; set; }
    public Category Category { get; set; }
}
```

Configuring Database Context:

```
public class ShopForHomeDbContext : DbContext
{
    public DbSet<User> Users { get; set; }
    public DbSet<Product> Products { get; set; }
    public DbSet<Category> Categories { get; set; }
    public DbSet<Coupon> Coupons { get; set; }

    public ShopForHomeDbContext(DbContextOptions<ShopForHomeDbContext>
options) : base(options)
    {
    }
}
```

Applying Migrations to Update Database Schema:

```
dotnet ef migrations add InitialCreate
dotnet ef database update
```

With EF Core and the Code First Approach, the ShopForHome backend is highly scalable, maintainable, and adaptable to future enhancements. This ensures that as new features are added, the database can evolve without disrupting the existing functionality.

4.Backend Implementation

Developing the API Using the Code-First Approach

The ShopForHome project's backend is implemented using ASP.NET Core Web API following the Code-First approach with Entity Framework Core (EF Core). This methodology allows us to define the database structure programmatically through C# entity classes, eliminating the need for manual SQL schema creation.

1. Setting Up the Database Context

The DbContext class in EF Core serves as the bridge between the application and the database. In ShopForHome, we created a ShopForHomeDbContext class that includes DbSet<T> properties for each entity.

Database Context Configuration

```
public class ShopForHomeDbContext : DbContext
{
    public DbSet<User> Users { get; set; }
    public DbSet<Product> Products { get; set; }
    public DbSet<Category> Categories { get; set; }
    public DbSet<Coupon> Coupons { get; set; }

    public ShopForHomeDbContext(DbContextOptions<ShopForHomeDbContext>
options) : base(options)
    {
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<User>()
            .HasOne<Role>(u => u.Role)
            .WithMany(r => r.Users)
            .HasForeignKey(u => u.RoleId);
    }
}
```

This setup ensures that all database interactions follow a well-defined schema, and relationships such as one-to-many associations (e.g., between users and roles) are properly enforced.

2. Creating API Endpoints

ASP.NET Core Web API follows a RESTful design where each controller represents a specific resource. For instance, we have a `ProductController.cs` to handle product-related API operations.

Product Controller Example

```
[Route("api/products")]
[ApiController]
public class ProductController : ControllerBase
{
    private readonly ShopForHomeDbContext _context;

    public ProductController(ShopForHomeDbContext context)
    {
        _context = context;
    }

    [HttpGet]
    public async Task<IActionResult> GetProducts()
    {
        var products = await _context.Products.Include(p => p.Category).ToListAsync();
        return Ok(products);
    }

    [HttpPost]
    public async Task<IActionResult> AddProduct([FromBody] Product product)
    {
        if (!ModelState.IsValid)
            return BadRequest(ModelState);

        _context.Products.Add(product);
        await _context.SaveChangesAsync();
        return CreatedAtAction(nameof(GetProducts), new { id = product.ProductId },
product);
    }
}
```

This controller exposes GET and POST endpoints, ensuring that the frontend can retrieve and add new products.

3. Applying Migrations & Database Updates

Once the models and context are defined, we generate the database schema using Entity Framework Core migrations.

```
dotnet ef migrations add InitialCreate
dotnet ef database update
```

This automatically creates the necessary tables and relationships in SQL Server without writing SQL scripts manually.

Testing the API with Swagger

Swagger (also known as Swashbuckle in .NET) is integrated into the ShopForHome API to provide an interactive API documentation and testing interface.

1. Integrating Swagger in ASP.NET Core

To enable Swagger, we modify the Program.cs file to include Swagger services.

```
var builder = WebApplication.CreateBuilder(args);
```

```
// Add services to the container.
builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
```

```
var app = builder.Build();
```

```
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}
```

```
app.UseAuthorization();
app.MapControllers();
```

```
app.Run();
```

This ensures that Swagger is available when running the application in development mode.

2. Accessing the Swagger UI

Once the API is running, we navigate to:

`https://localhost:<port>/swagger`

This opens the Swagger UI, where all available endpoints are displayed along with their request and response schemas.

3. Testing API Endpoints Using Swagger

Swagger provides an interactive interface to test API endpoints.

Testing the GetProducts API

1. Navigate to `https://localhost:<port>/swagger`.
2. Locate the GET `/api/products` endpoint.
3. Click on "Try it out" and then "Execute".
4. The response will include a list of products in JSON format.

Testing the AddProduct API

1. Navigate to the POST `/api/products` endpoint.

Click on "Try it out" and enter product details in JSON format:

json

```
{  
  "name": "Laptop",  
  "price": 1200,  
  "description": "High-performance laptop",  
  "categoryId": 1  
}
```

- 2.
3. Click "Execute" and verify that the product is added successfully.

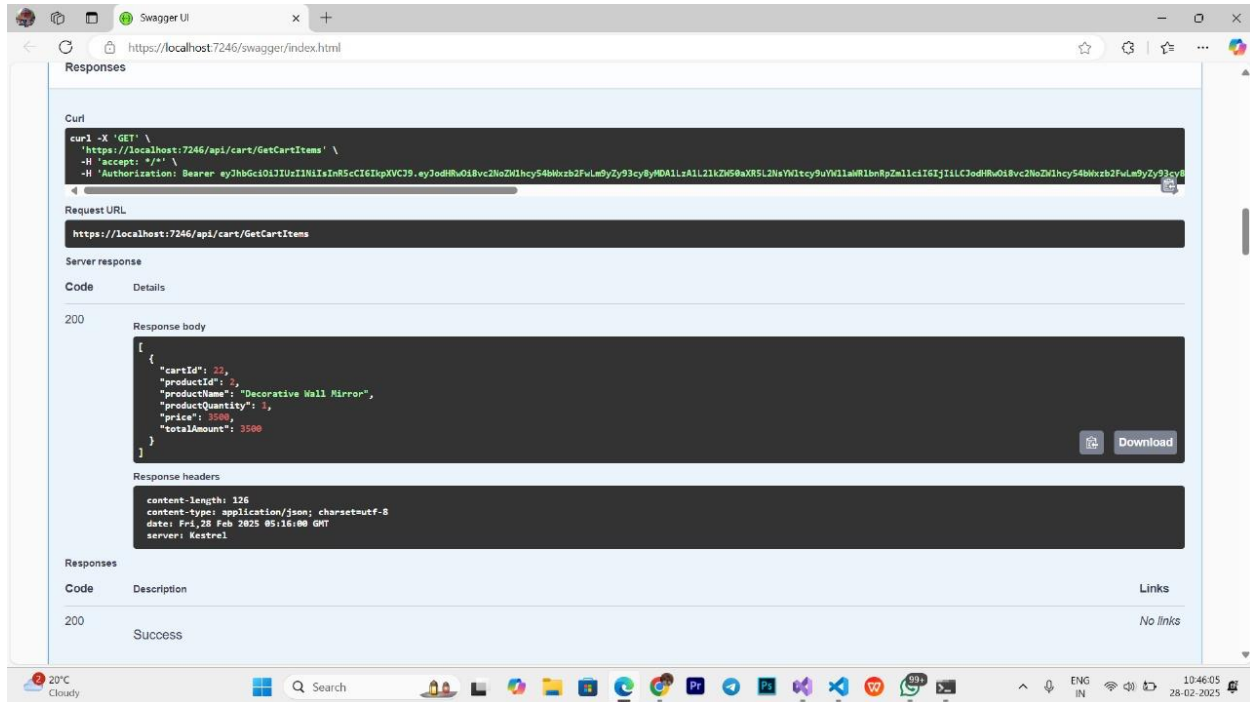
4. Response Handling in Swagger

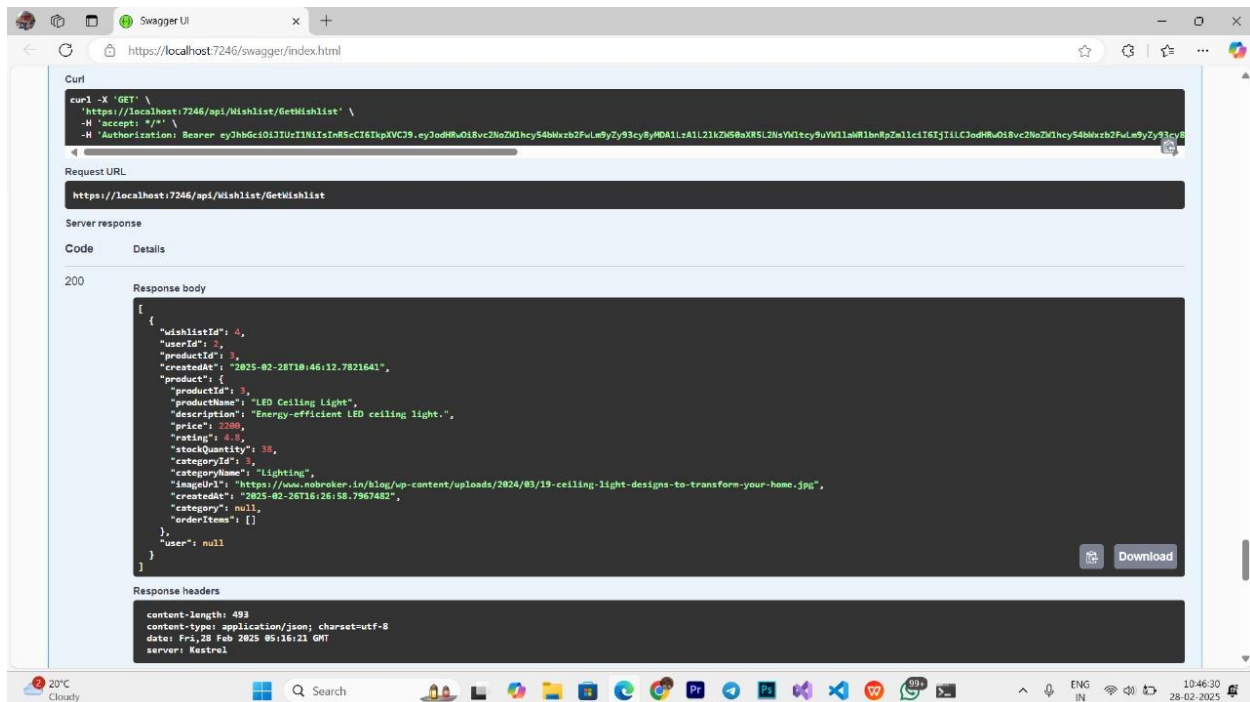
When an API request is made, Swagger provides:

- Success responses (200 OK, 201 Created) for valid requests.
- Error responses (400 Bad Request, 401 Unauthorized) for invalid requests.

By integrating Swagger, the ShopForHome API is tested efficiently, ensuring that all endpoints function as expected before integration with the Angular frontend.

Testing our API using swagger UI





5.FRONTEND IMPLEMENTATION

The frontend of the ShopForHome project is developed using Angular 19, a powerful framework that enables the creation of dynamic, responsive, and scalable web applications. The frontend serves as the user interface layer, allowing users to interact with the system seamlessly. The primary responsibilities of the frontend include displaying products, managing user authentication, handling discount coupons, and ensuring smooth navigation between different sections of the application. The integration of Angular with ASP.NET Core Web API allows the frontend to consume backend services efficiently and dynamically update the UI based on the data retrieved. This section details how Angular was used in building the frontend, the challenges encountered during the implementation, and how the system ensures a seamless user experience.

ANGULAR IN FRONTEND DEVELOPMENT

Setting Up the Angular Project

The project was initialized using the Angular CLI, which provides a structured environment for developing large-scale applications. The following command was used to create the project:

ng new shop-for-home

Once the project structure was generated, we configured routing, created essential components and services, and integrated Bootstrap for a consistent and visually appealing UI. The Angular project structure was designed to follow a modular approach, ensuring separation of concerns and maintainability.

Component-Based Architecture

In Angular, the application is divided into reusable components, each responsible for handling a specific UI element. The main components in ShopForHome include:

1. Product List Component (product-list.component.ts) – Displays all available products in a grid layout.
2. Product Details Component (product-details.component.ts) – Shows detailed information about a selected product.
3. Cart Component (cart.component.ts) – Allows users to manage their shopping cart and proceed to checkout.
4. User Management Components (user-details.component.ts, update-user.component.ts, delete-user.component.ts) – Handle user-related operations such as updating and deleting users.
5. Admin Dashboard (admin.component.ts) – Provides an interface for administrators to manage products, users, and coupons.

Each of these components interacts with the backend through Angular Services, which act as a bridge between the frontend and the API.

Routing and Navigation

Angular's RouterModule was used to define navigation paths for different pages. The routes were configured in the app-routing.module.ts file, ensuring that users could seamlessly move between pages. Below is a sample routing configuration:

```
const routes: Routes = [  
  { path: 'products', component: ProductListComponent },  
  { path: 'product/:id', component: ProductDetailsComponent },  
  { path: 'cart', component: CartComponent },  
  { path: 'admin', component: AdminComponent },  
  { path: '**', redirectTo: 'products' }  
];
```

This setup ensures that when users visit specific URLs, they are directed to the correct component view. The wildcard route ('**') ensures that users are always redirected to the product listing page if an invalid URL is entered.

CONSUMING APIs WITH ANGULAR SERVICES

Setting Up HTTP Requests

To communicate with the backend API, Angular's HttpClientModule was imported and configured in app.module.ts. This allowed the application to make GET, POST, PUT, and DELETE requests to the API endpoints. Each entity in the system (e.g., products, users, coupons) had a corresponding service file responsible for handling API interactions.

Example: Fetching Products from the API

A ProductService was created to fetch product data from the API:

```
@Injectable({
  providedIn: 'root'
})
export class ProductService {
  private apiUrl = 'https://localhost:5001/api/products';

  constructor(private http: HttpClient) { }

  getProducts(): Observable<Product[]> {
    return this.http.get<Product[]>(this.apiUrl);
  }
}
```

This service was injected into the ProductListComponent to display the list of products retrieved from the backend:

```
ngOnInit() {
  this.productService.getProducts().subscribe(
    (data) => this.products = data,
    (error) => console.error('Error fetching products:', error)
  );
}
```

This approach follows Angular's dependency injection pattern, making the code modular, testable, and reusable.

Example: Adding a Product Using an API Call

To allow admins to add new products, a POST request was implemented in ProductService:

```
addProduct(product: Product): Observable<Product> {  
  return this.http.post<Product>(this.apiUrl, product);  
}
```

This method was called from the Admin Dashboard component when the admin submitted the product form.

CHALLENGES FACED DURING FRONTEND DEVELOPMENT

1. Managing State and Data Synchronization

One of the primary challenges faced was maintaining state consistency between different components. For example, when an admin added or updated a product, the changes needed to reflect immediately in the product list. Initially, we relied on manual page refreshes, but later, we optimized it by using RxJS BehaviorSubject to notify other components about state changes dynamically.

2. Handling API Errors and Authentication

While integrating the JWT-based authentication system, there were issues handling expired tokens and unauthorized requests. To resolve this, we implemented an HTTP Interceptor that automatically refreshed the token and redirected unauthorized users to the login page.

```
intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {  
  const token = localStorage.getItem('authToken');  
  if (token) {  
    req = req.clone({  
      setHeaders: { Authorization: `Bearer ${token}` }  
    });  
  }  
}
```

```
    return next.handle(req);  
  }
```

This ensured that all outgoing requests included the authentication token, reducing the risk of unauthorized access errors.

3. Implementing Role-Based Access Control (RBAC)

Since the application had both customers and admins, access control was essential. Initially, users could navigate to admin-only pages through direct URL entry, bypassing restrictions. This issue was resolved by implementing role-based route guards using Angular's CanActivate guard.

```
canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean {  
  const userRole = this.authService.getUserRole();  
  if (userRole === 'Admin') {  
    return true;  
  }  
  this.router.navigate(['/products']);  
  return false;  
}
```

This prevented unauthorized users from accessing admin functionalities, ensuring better security and role management.

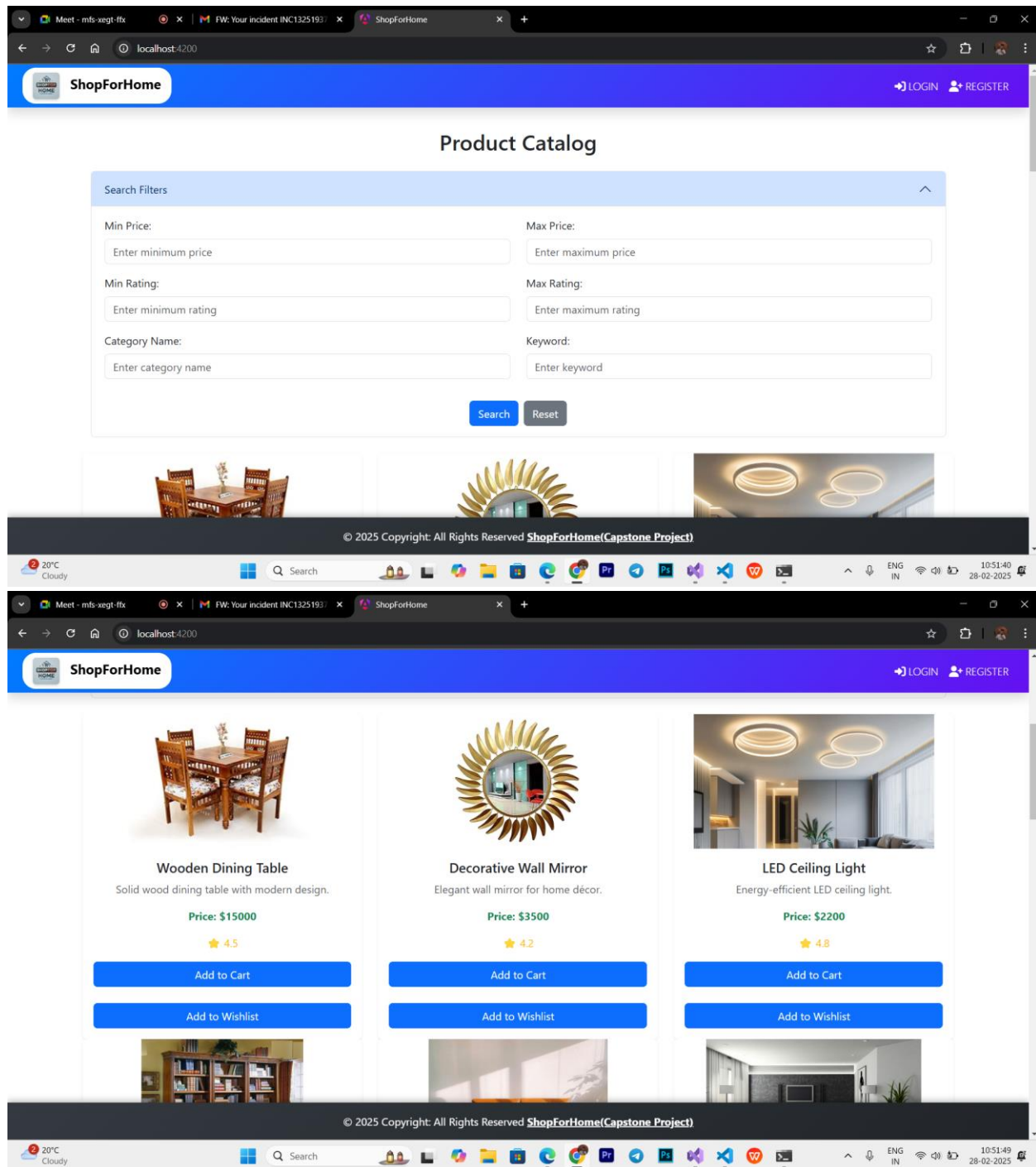
4. Optimizing API Calls and Performance

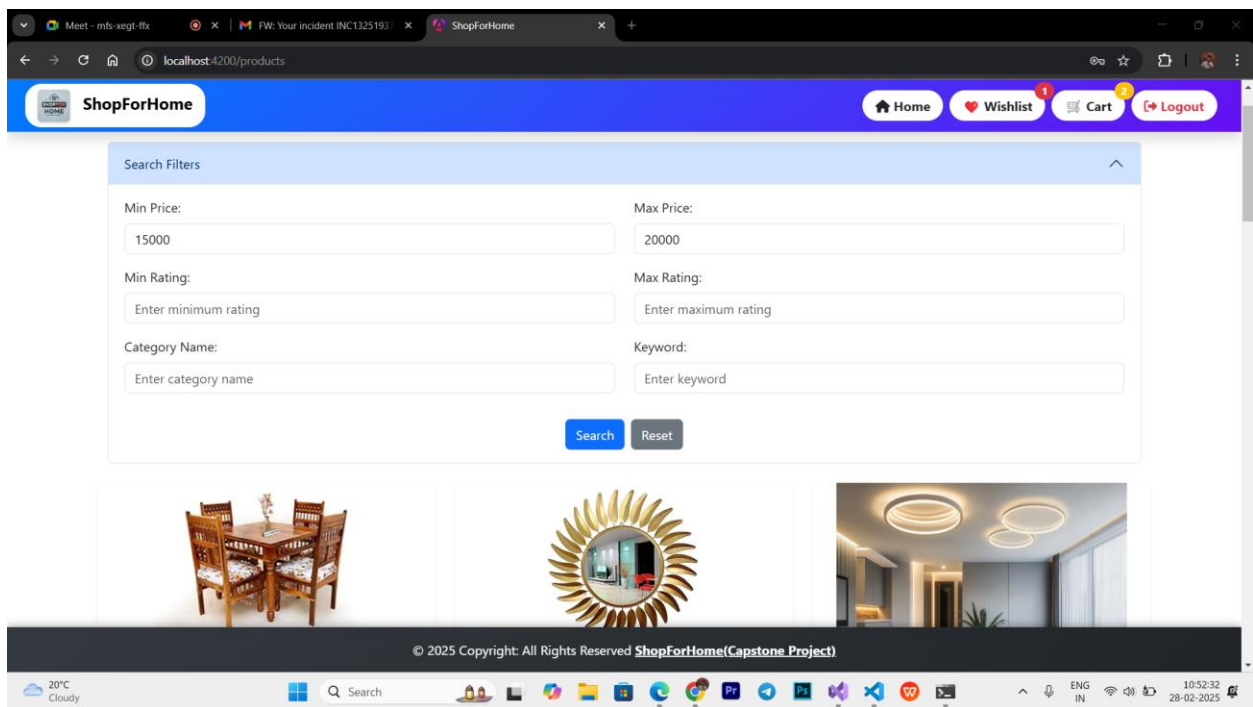
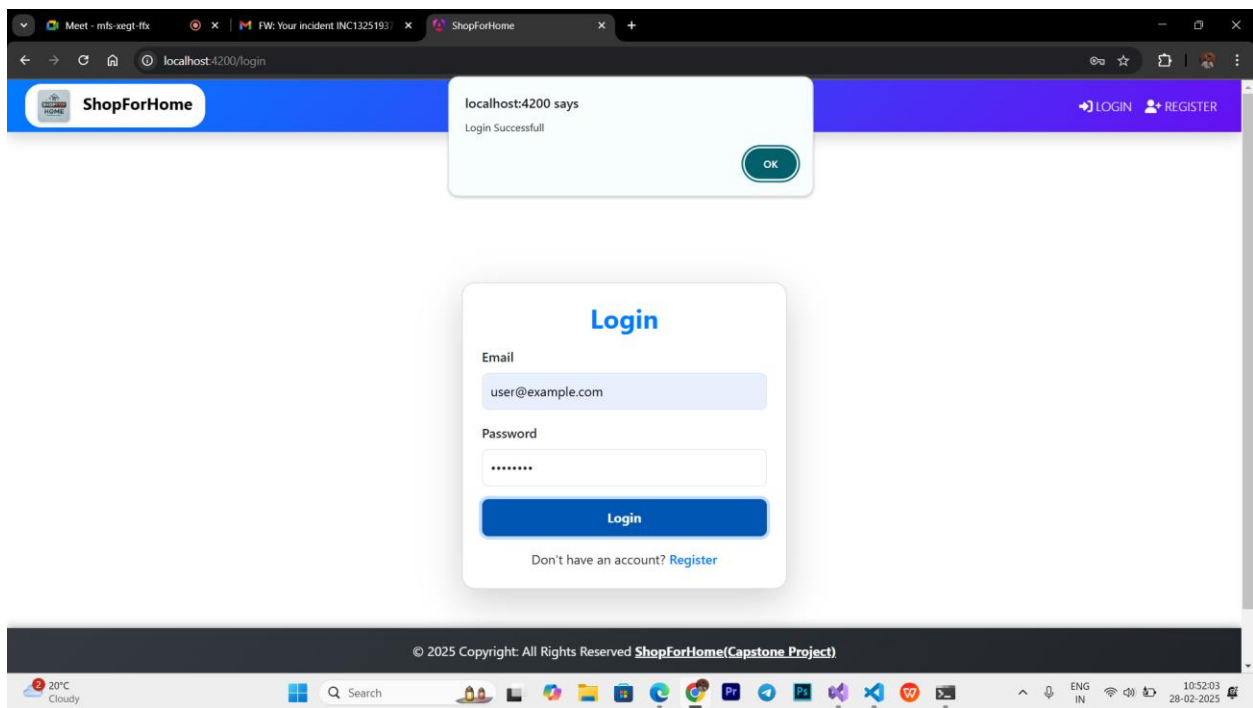
Initially, the frontend was making multiple API requests, which led to performance issues. To optimize this, we introduced caching mechanisms and debounced search requests to reduce the number of API calls.

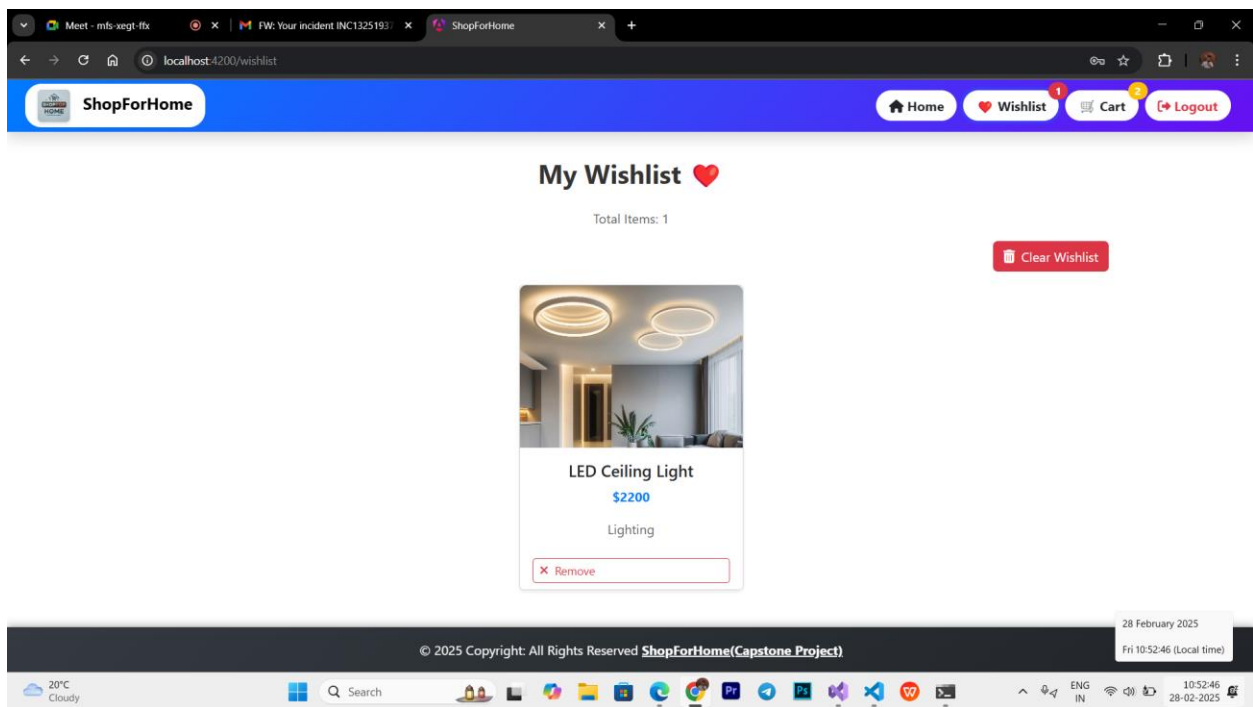
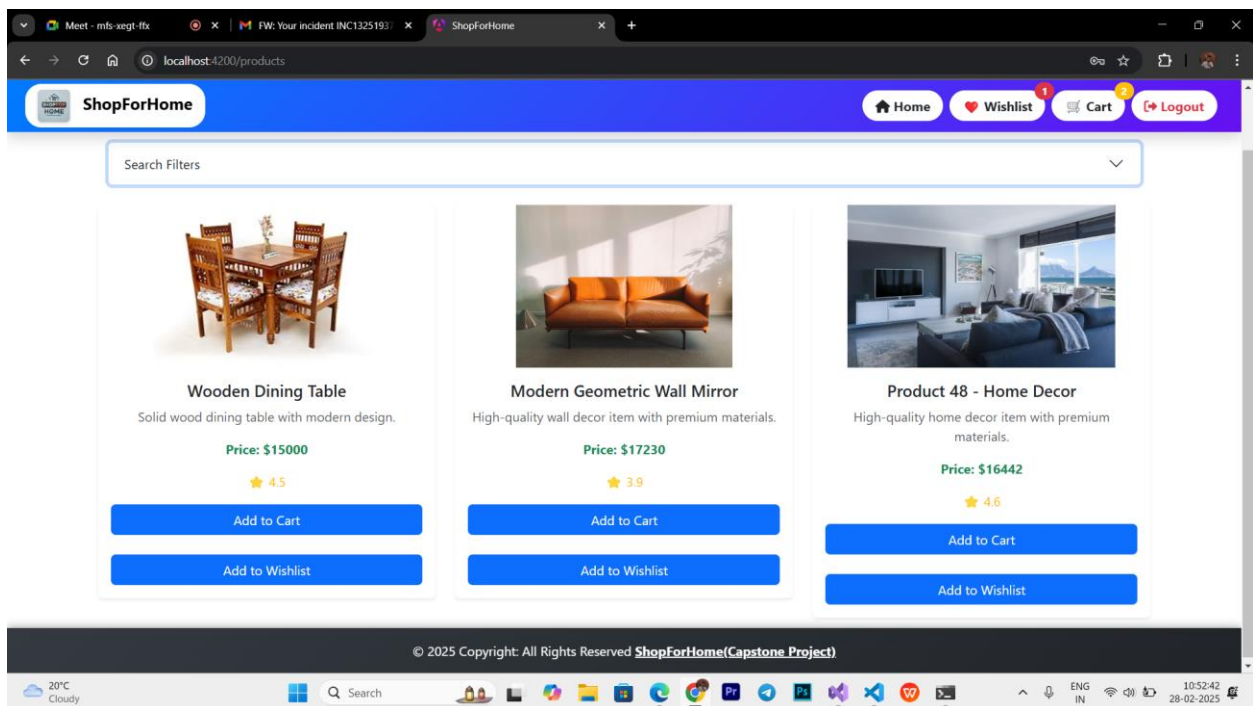
```
searchProducts(term: string): Observable<Product[]> {  
  return this.http.get<Product[]>(`${this.apiUrl}/search?keyword=${term}`)  
    .pipe(debounceTime(300)); // Delays request execution to reduce unnecessary API  
  calls  
}
```

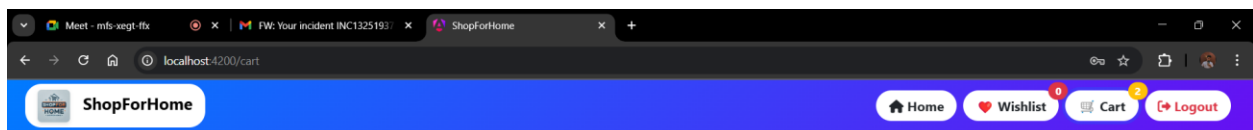
By implementing debouncing, we improved response times and reduced server load.

ScreenShots of the Frontend module

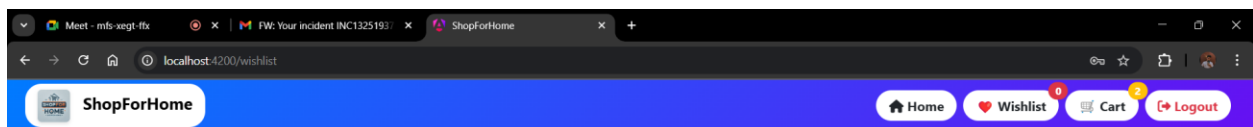
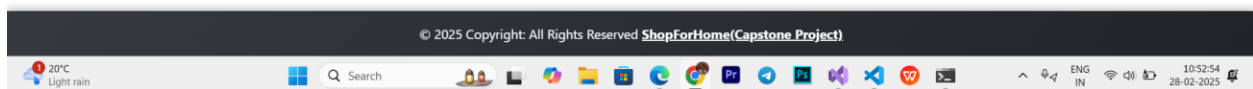








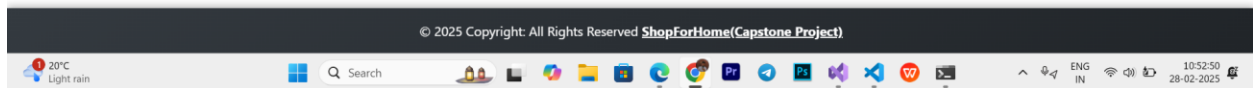
Your Shopping Cart				
Product	Quantity	Price	Total	Actions
Decorative Wall Mirror	<div><div>-</div><div>1</div><div>+</div></div>	\$3500	\$3500	<div>Remove</div>
LED Ceiling Light	<div><div>-</div><div>1</div><div>+</div></div>	\$2200	\$2200	<div>Remove</div>
<div>Clear Cart</div> <div>Proceed To Checkout</div>				

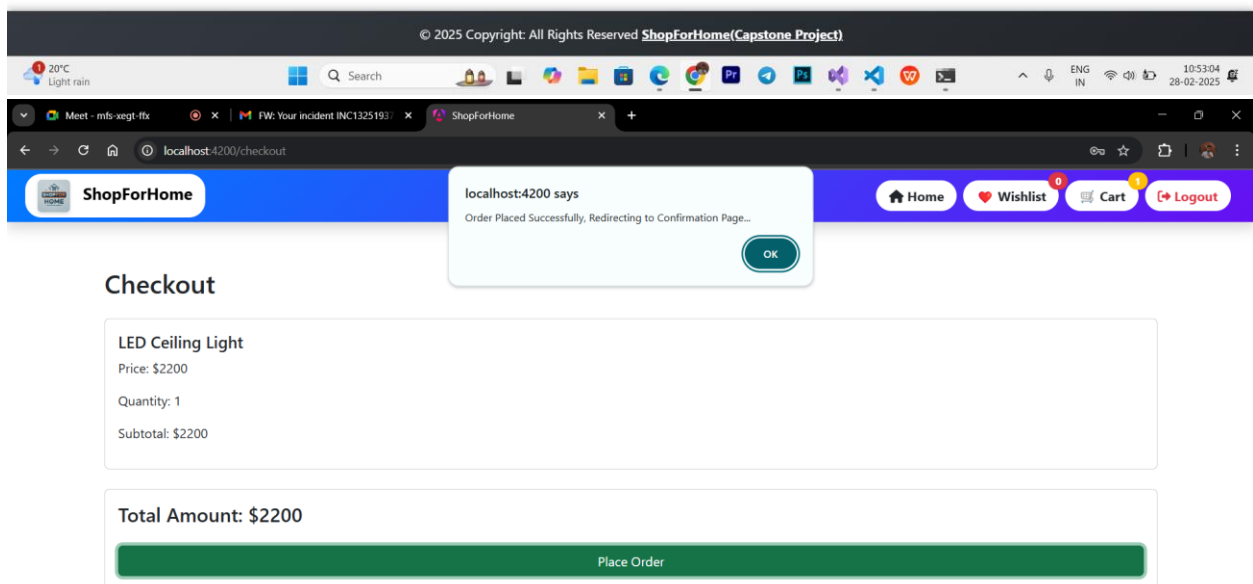
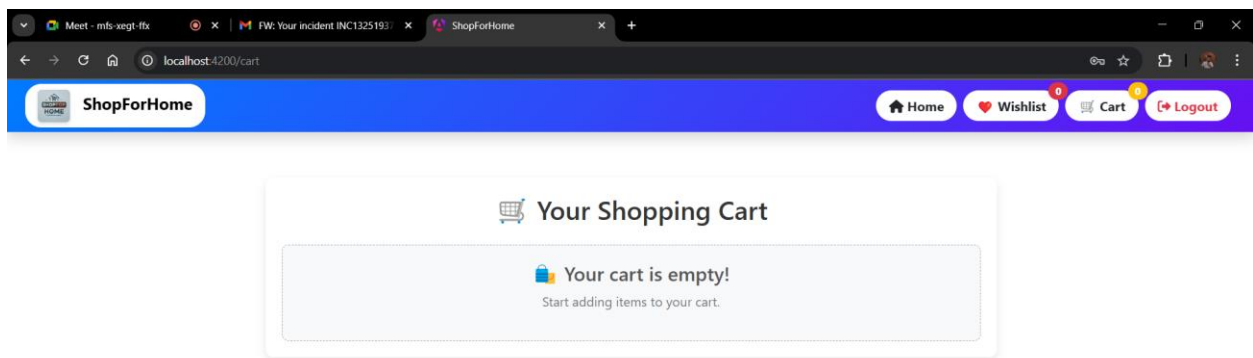


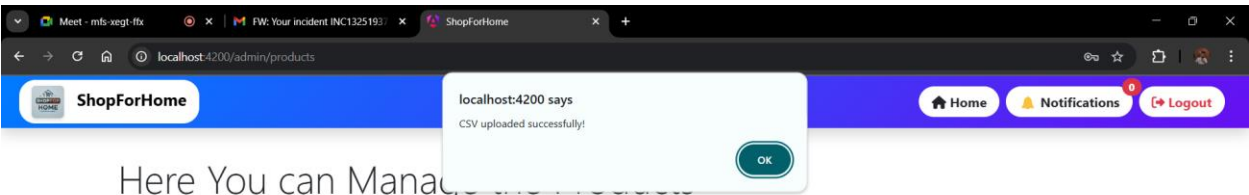
My Wishlist 

Total Items: 0

Your wishlist is empty. Start adding your favorite items! 

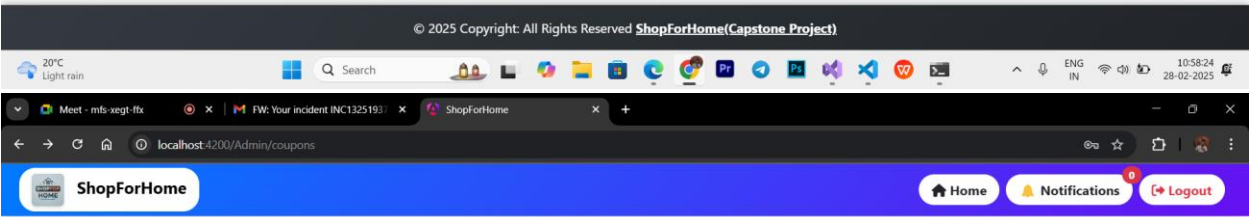






SrNo	Id	Product Name	Description	Product Price	Rating	Stock	Category Name	Image Link
1	1	Wooden Dining Table	Solid wood dining table with modern design.	15000	4.5	17	Furniture	https://m.media-amazon.com/images/I/41aSOoUwLYL_SX300_SX300_QL70_FMwebp_jj
2	2	Decorative Wall Mirror	Elegant wall mirror for home décor.	3500	4.2	23	Home Decor	https://m.media-amazon.com/images/I/51aA2qy0xYL_SX300_SX300_QL70_FMwebp_jj
3	3	LED Ceiling Light	Energy-efficient LED ceiling light.	2200	4.8	37	Lighting	https://www.nobroker.in/blog/wp-content/uploads/2024/03/19-ceiling-light-designs-to-transform-your-home

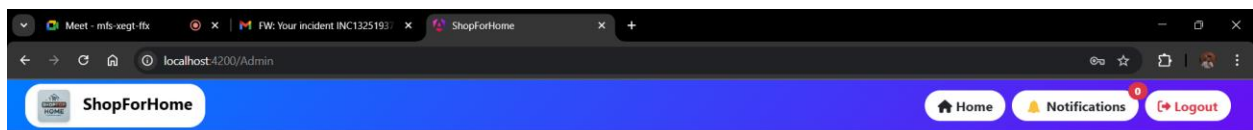
Upload CSVSubmitAdd ProductBack To Admin Dashboard




SrNo	coupon Id	Discount Percent	Expiry Date	Is Active	Assign	Deactivate	Delete
1	1	10 %	Friday, February 28, 2025	false	assign	deactivate	delete
2	2	20 %	Friday, February 28, 2025	true	assign	deactivate	delete
3	3	10 %	Friday, February 28, 2025	true	assign	deactivate	delete
4	4	20 %	Friday, February 28, 2025	true	assign	deactivate	delete
5	5	10 %	Friday, February 28, 2025	true	assign	deactivate	delete

Add CouponBack To Admin Dashboard






Admin Dashboard



Manage Users

View and manage user details.


[Go to Users](#)



Manage Categories

Add, edit, or remove product categories.


[Go to Categories](#)



Manage Products

Add, update, or delete products.


[Go to Products](#)



Manage Coupons

Add, update, or delete coupons.

[Go to coupons](#)



Manage Sales Report

View, and manage the Sales Report

[Go to Sales Report](#)

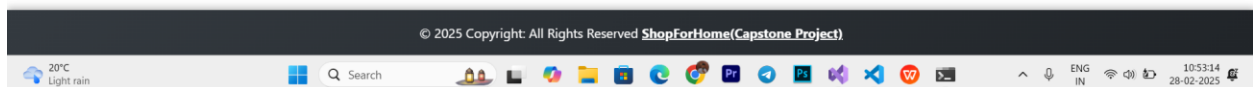


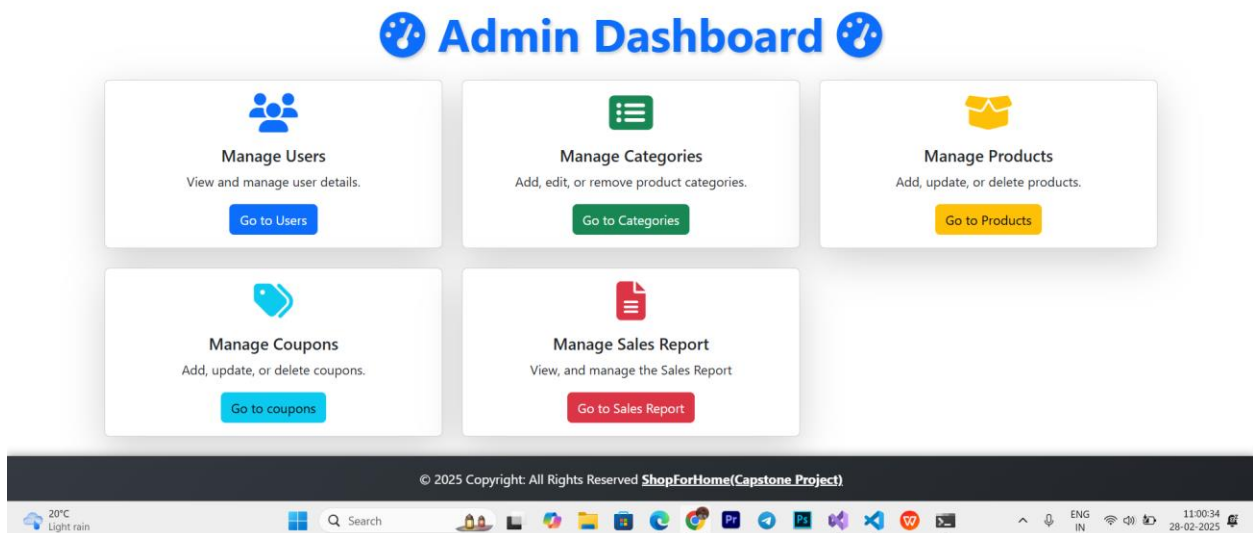
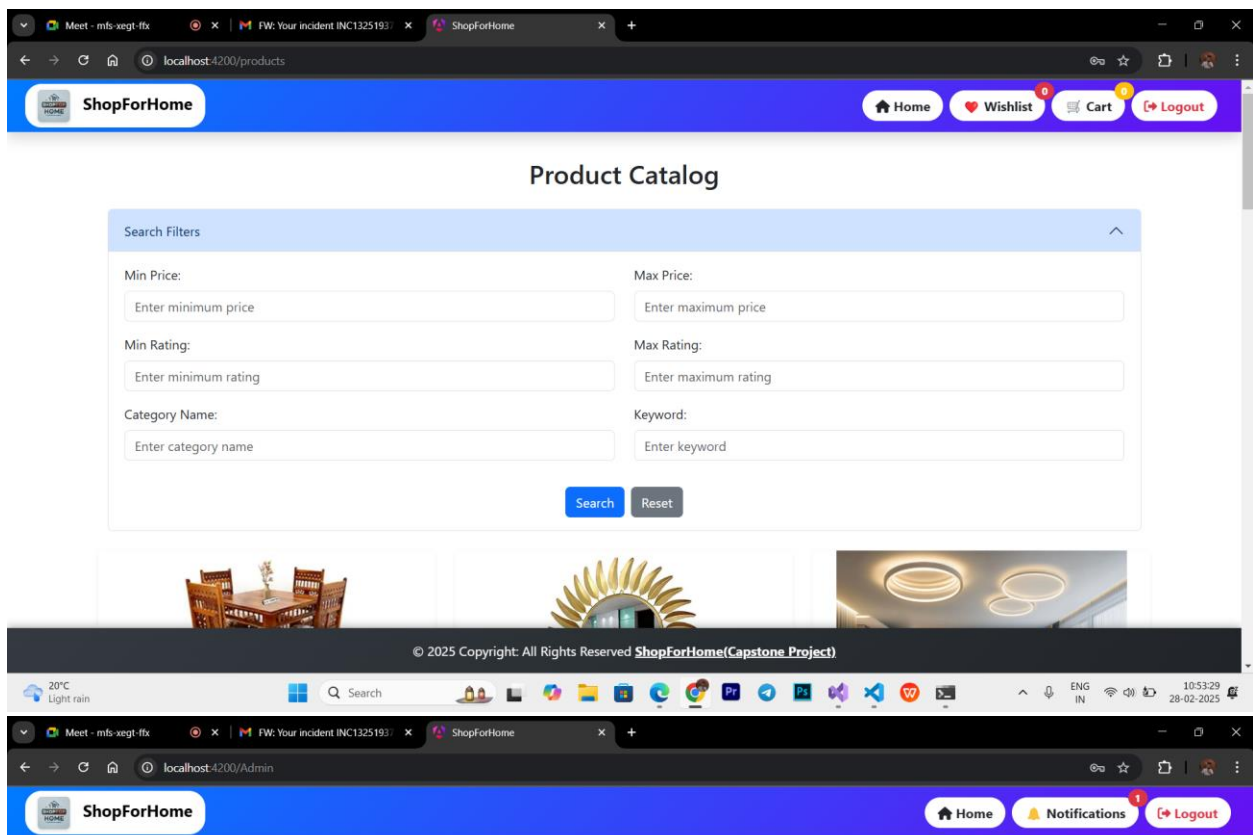
Checkout

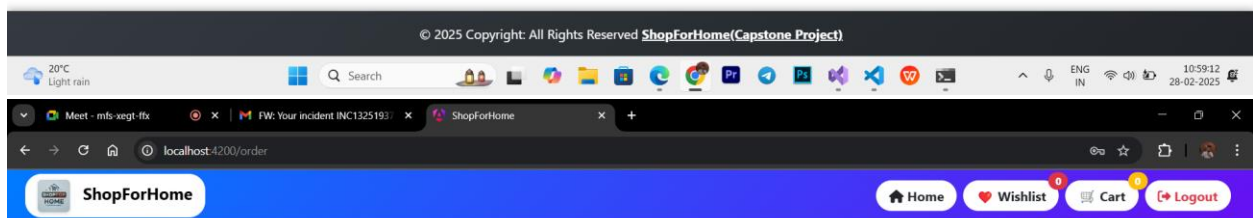
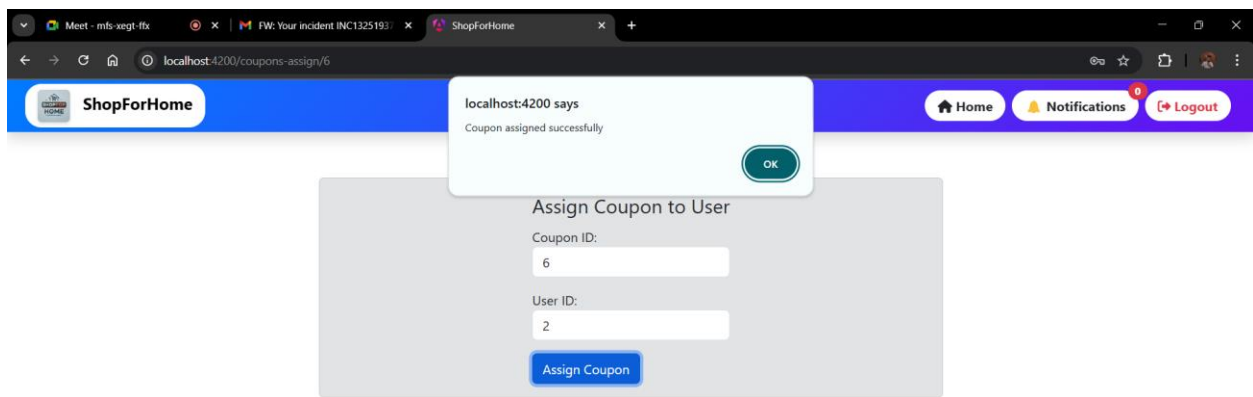
LED Ceiling Light
Price: \$2200
Quantity: 1
Subtotal: \$2200

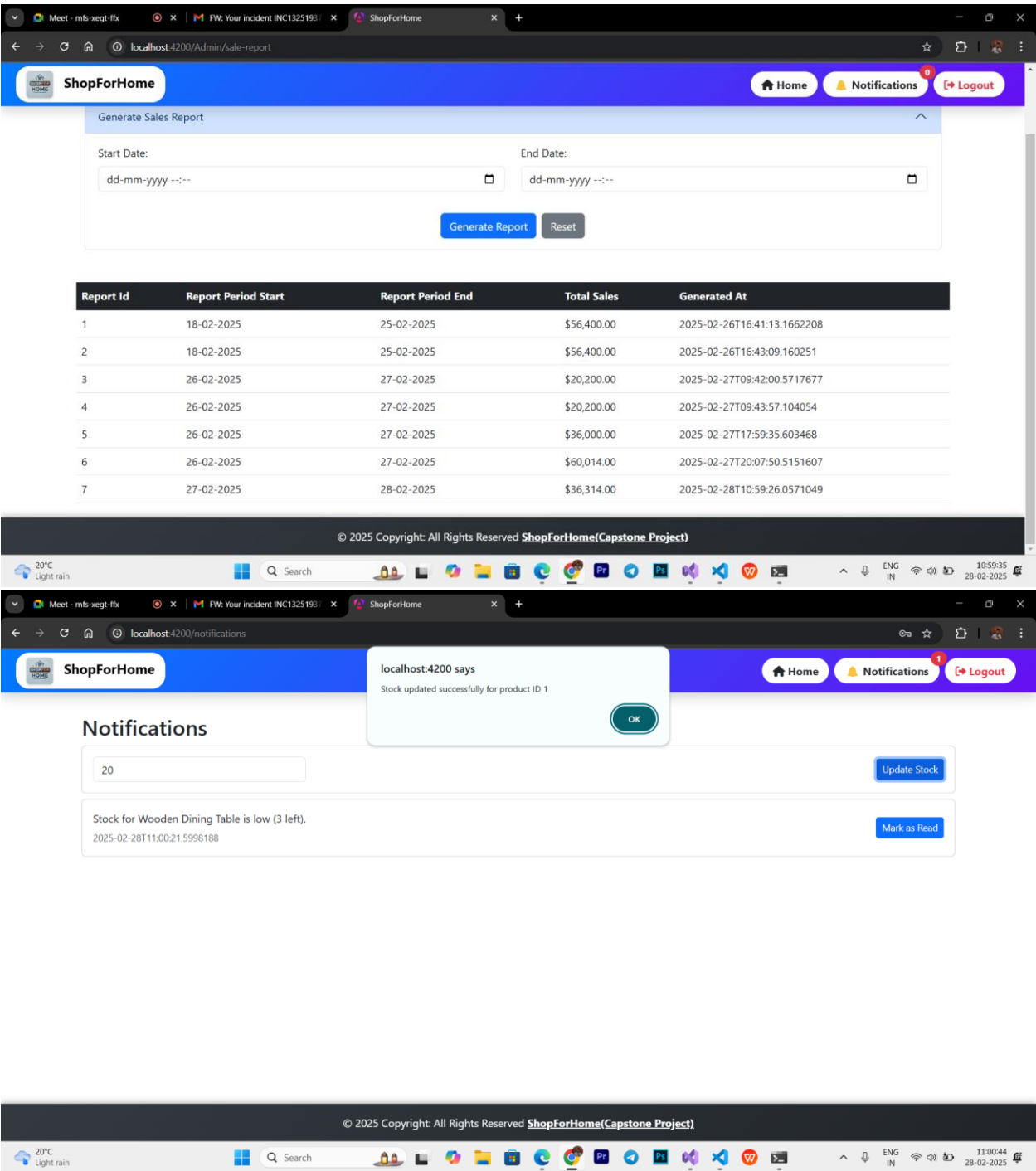
Total Amount: \$2200

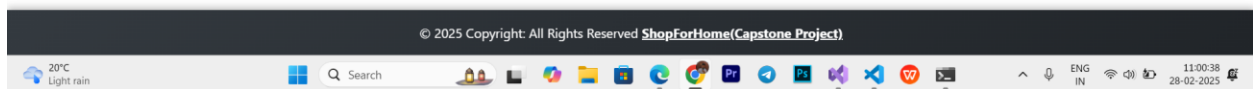
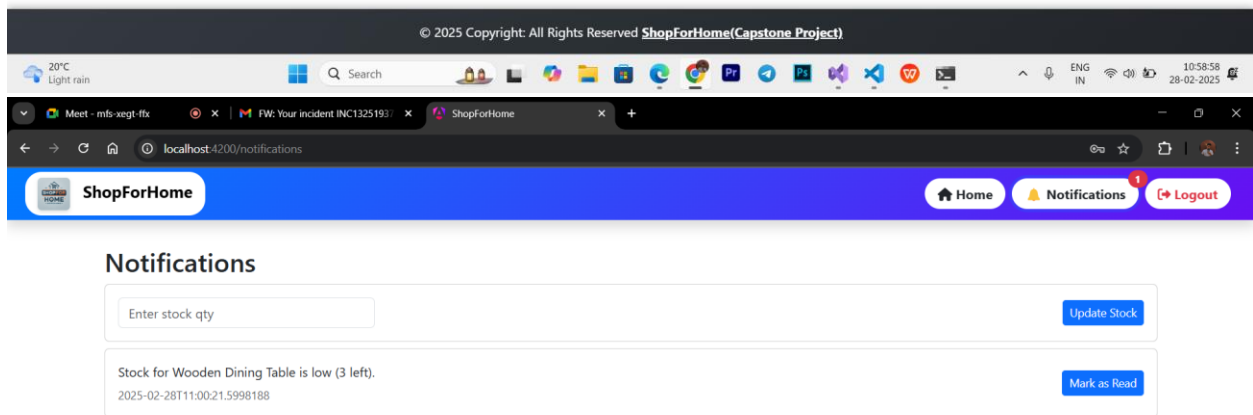
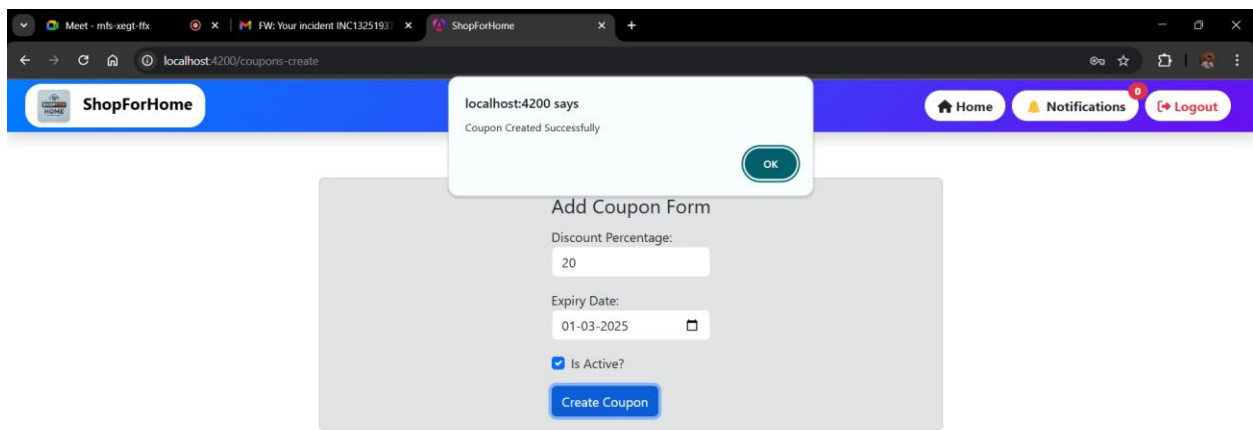
[Place Order](#)

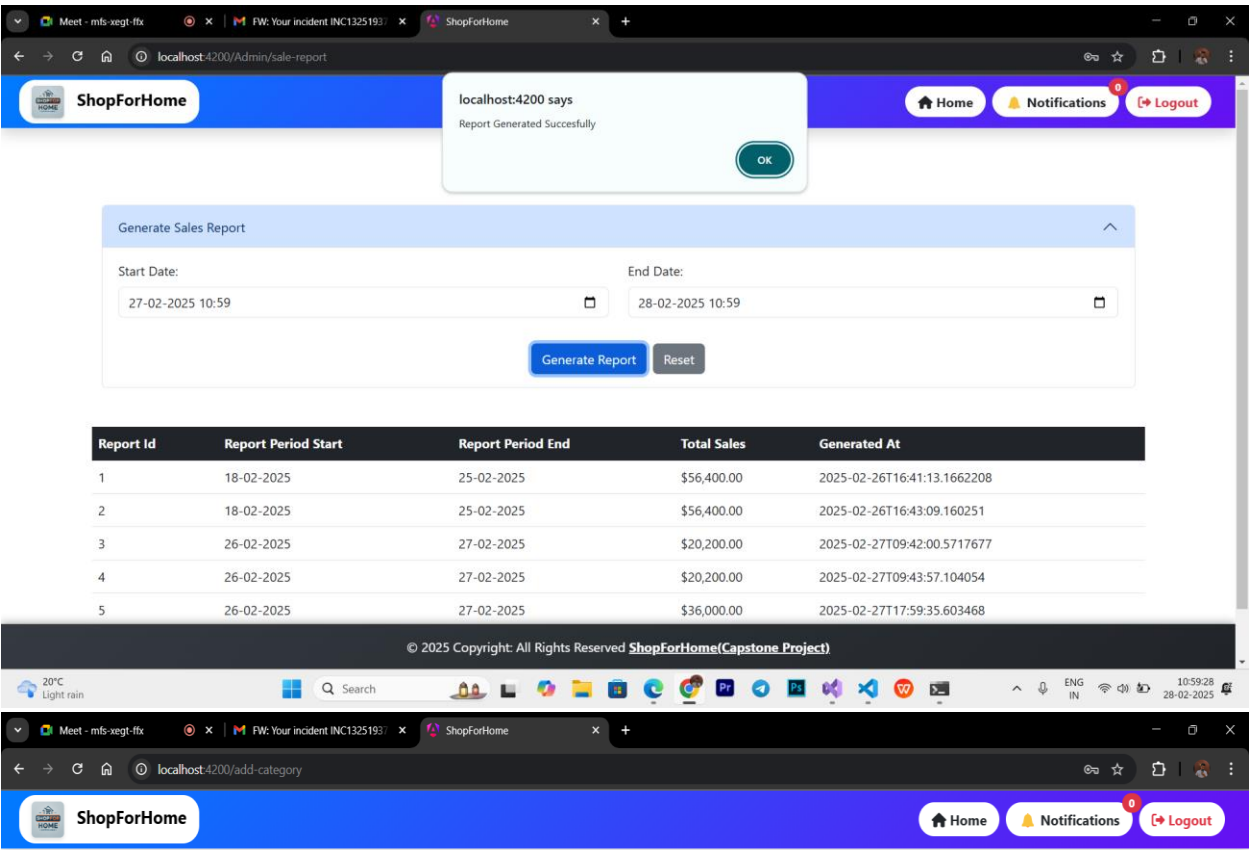










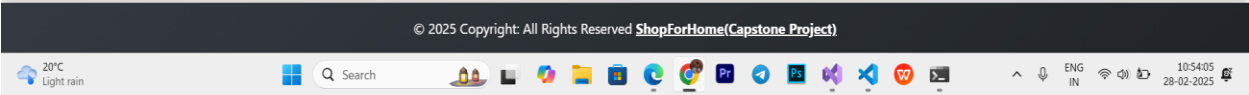


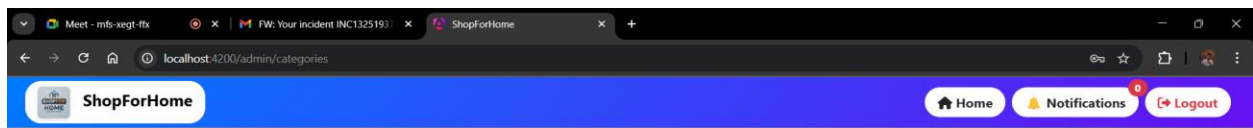
Add Category Form

Category Name:

Enter category name

Submit

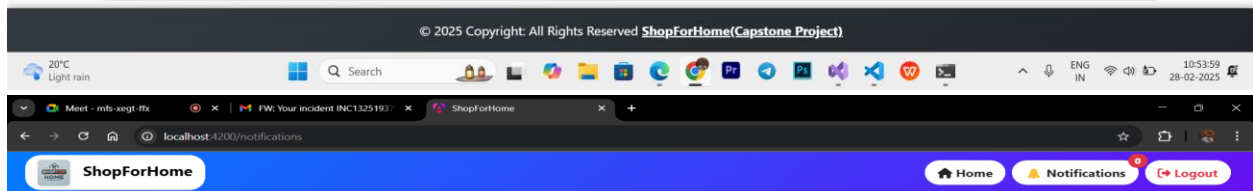




Here You can Manage the Categories

SrNo	Id	Category Name	Delete
1	1	Furniture	Delete
2	2	Home Decor	Delete
3	3	Lighting	Delete
4	4	Show Piece	Delete
5	5	Wood Work	Delete
6	6	Metal Work	Delete
7	7	Tatekigumi	Delete
8	8	Cornet	Delete

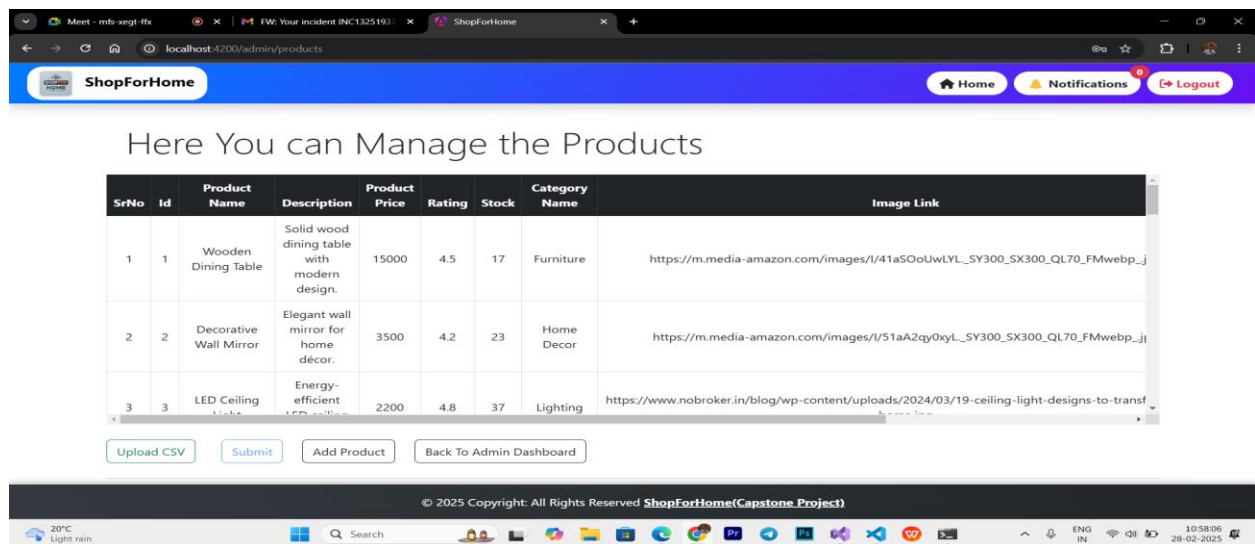
[Add category](#) [Back To Admin Dashboard](#)



Notifications

No new notifications.





CONCLUSION

The frontend implementation of ShopForHome leveraged Angular's powerful features such as component-based architecture, services for API interaction, and efficient state management to build a user-friendly and high-performance web application. The integration with the ASP.NET Core Web API was streamlined through HTTP requests, allowing seamless data retrieval and updates. Despite challenges related to state management, authentication, and API performance, the adoption of best practices such as RxJS, interceptors, and role-based guards resulted in a robust and maintainable frontend. Through these methodologies, the application successfully delivers a responsive, scalable, and user-centric experience, meeting all the functional and non-functional requirements.