Clothing Store Management Application: Three-Tier Architecture Report

1. Architecture Overview

The Clothing Store Management Application is designed using a Three-Tier Architecture, which separates the system into three distinct layers: Presentation, Application, and Data. This structure enforces strict separation of concerns, enabling independent development, scalability, and enhanced security.

System Components

Presentation Tier (Client Layer):

- Built with HTML, CSS, and JavaScript for dynamic UI.
- Communicates with the backend via RESTful API requests.
- Features product/category management interfaces and real-time updates.

Application Tier (Business Logic Layer):

- Developed using ASP.NET Core Web API with controllers, services, and repositories.
- Implements business rules, validation, and transaction management.
- Uses Entity Framework Core for database interactions.

Data Tier (Database Layer):

- SQL server database stores product and category.
- Managed via EF Core migrations and optimized queries.

2. Implementation Steps

1. Backend Development (ASP.NET Core Web API)

- Created layered architecture:

- Controllers: Handle HTTP requests (e.g., ProductsController, CategoryController).
- Services: Business logic (e.g., ProductService, CategoryService).
- Repositories: Database operations (e.g., ProductRepository, CategoryRepository).
- Configured Entity Framework Core with SQL server provider.
- Defined models
- Added FluentValidation for DTO validation (e.g., ProductValidator).
- Added AutoMapper
- Implemented global error handling with custom middleware

2. Database Configuration

- Designed relational schema with Products and Categories tables.
- Applied EF Core migrations
- Configured relationships (e.g., Product ↔ Category one-to-many).

3. Frontend Development

- Created responsive UI with:
- Product listing grid.
- Category management panel.
- Modal forms for adding/editing products.
- Integrated with backend using Fetch API
- Implemented CORS policies for secure client-server communication

4. Testing

- Tested API endpoints using Swagger.
- Validated UI workflows (add/edit/delete products and categories).

3. Advantages and Challenges Faced

Advantages

- Separation of Concerns: Clear division between UI, business logic, and data.
- Scalability: Each tier can be scaled independently (e.g., load-balanced API servers).

- Maintainability: Changes to one layer (e.g., database) don’t affect others.
- Security: Database is never exposed directly to the client.

Challenges Faced

- CORS Configuration: Required careful setup for cross-origin requests.
- Transaction Management: Ensuring ACID properties across repository operations.
- Dependency Injection Complexity: Managing services across layers.
- Error Propagation: Handling exceptions across tiers consistently.

4. Comparison with Other Architectures

Architecture	Description	Pros	Cons
Single-Tier	All components in one system	Simple, fast, easy to develop	Not scalable, harder to maintain
Two-Tier	Frontend and Backend are separate	Better separation, scalable	Requires proper API integration
Three-Tier	UI, API, and Database separated	Highly scalable, secure	More complex to implement

Conclusion

The Three-Tier Architecture provided a robust foundation for the Clothing Store Management Application, enabling modular development, enhanced security, and scalability. While challenges like CORS configuration and transaction management required careful attention, the separation of concerns ensured maintainability and flexibility. This architecture is ideal for applications requiring long-term scalability and complex business rules, outperforming Single-Tier and Two-Tier systems in structured environments.