Gaming Gear - C4 Architecture Document

==========

1. System Overview

Gaming Gear is a modern e-commerce website dedicated to selling gaming peripherals including mouses, headphones, controllers, and keyboards.

The system provides an engaging user experience via a responsive Angular frontend and a robust backend built on ASP.NET Core with Entity Framework.

Key features include user registration/login, product browsing with filtering and pagination, shopping cart management, and checkout with Stripe payment integration.

Caching and cart state are optimized using Redis, and all persistent data is stored in a SQL Server database.

No administrative panel exists for product management at this time.

2. Technology Stack

Layer	Technology
Frontend	Angular
Backend	ASP.NET Core, Entity Framework
Database	SQL Server

Caching | Redis

Authentication | ASP.NET Core Identity

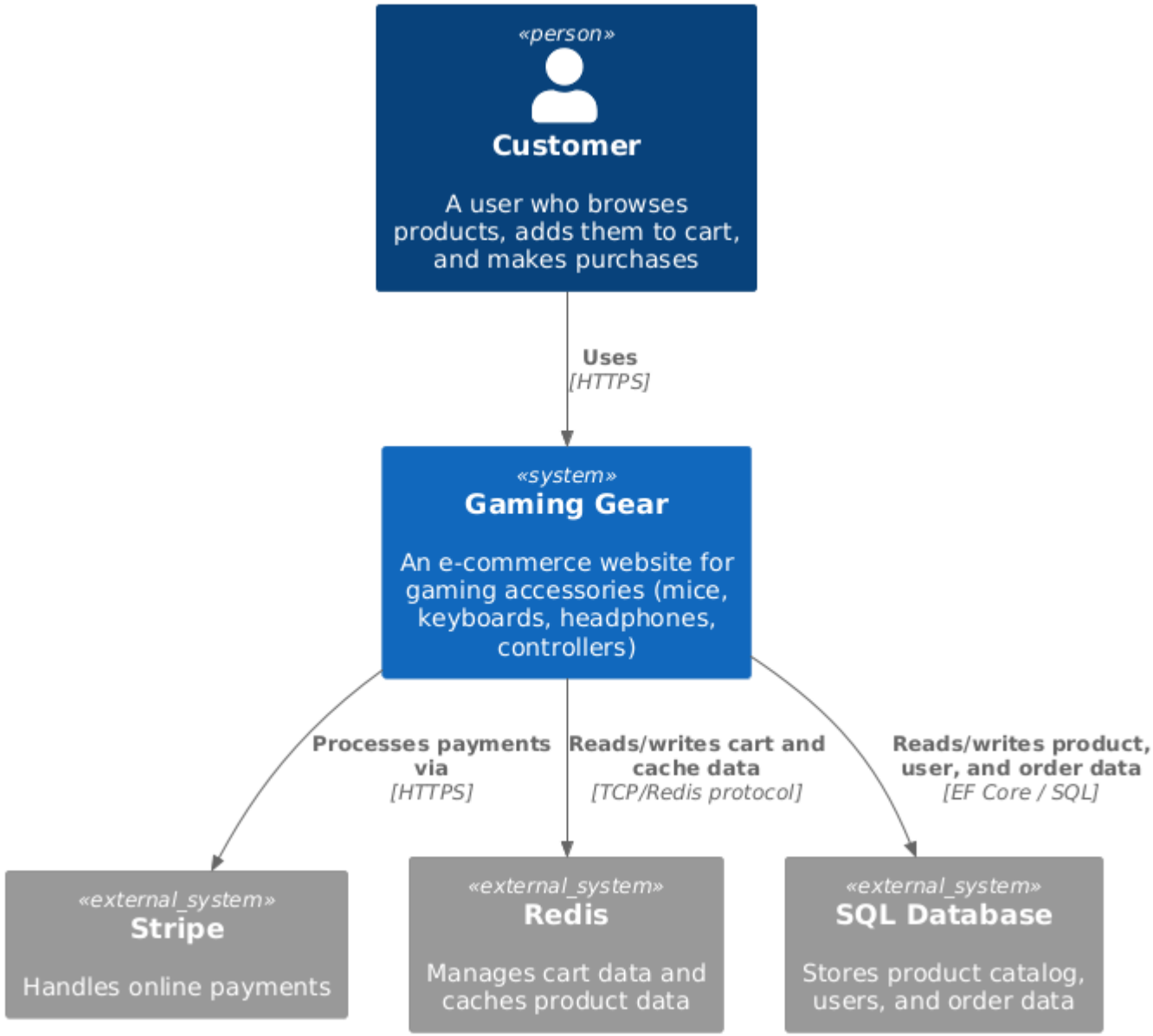
Payment Processing | Stripe (backend integration)

Communication | RESTful API (HTTP) between frontend/backend

3. Context Diagram

Description:

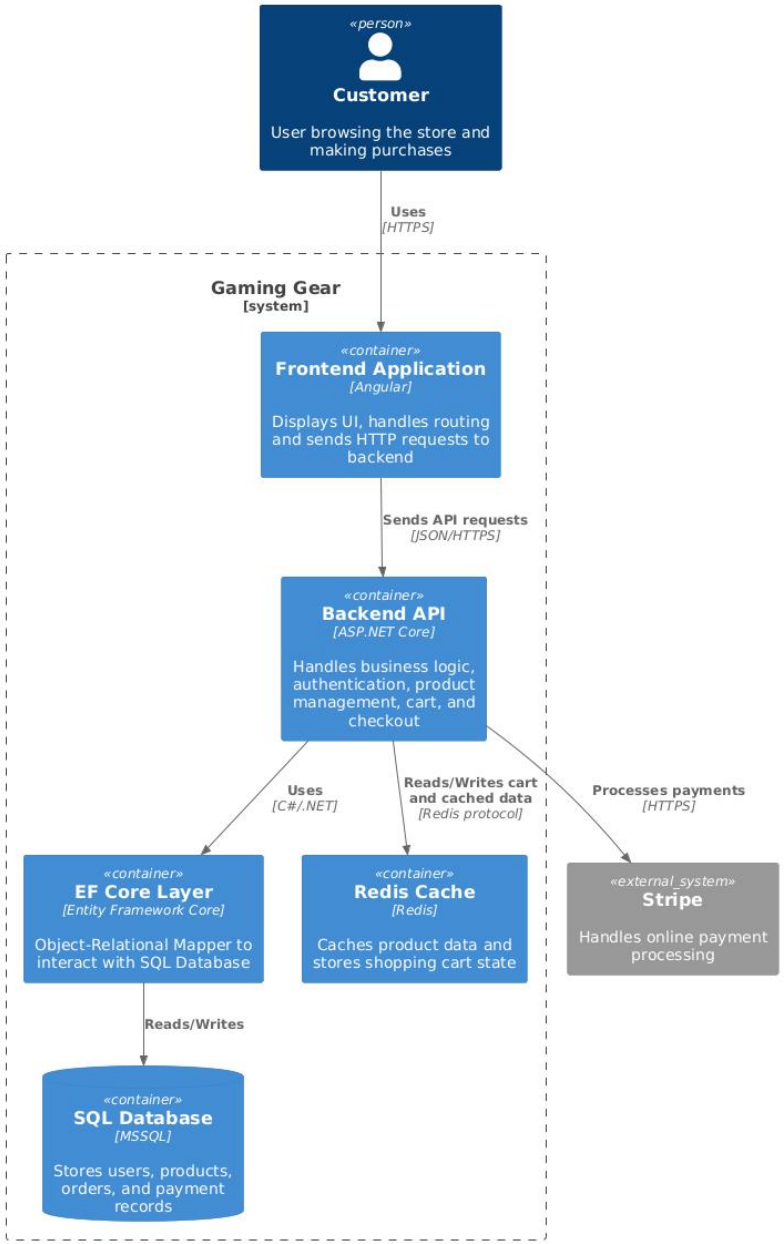
This diagram shows Gaming Gear's place in the environment, including end users, external systems such as Stripe for payments, Redis for caching, and the SQL database.


«person»
Customer
A user who browses products, adds them to cart, and makes purchases
Uses
[HTTPS]
«system»
Gaming Gear
An e-commerce website for gaming accessories (mice, keyboards, headphones, controllers)
Processes payments via
[HTTPS]
Reads/writes cart and cache data
[TCP/Redis protocol]
Reads/writes product, user, and order data
[EF Core / SQL]
«external_system»
Stripe
Handles online payments
«external_system»
Redis
Manages cart data and caches product data
«external_system»
SQL Database
Stores product catalog, users, and order data

4. Container Diagram

Description:

Displays the major containers of the system — Angular SPA frontend, ASP.NET Core backend API, Redis cache server, SQL database, and Stripe payment gateway — and their communication paths.

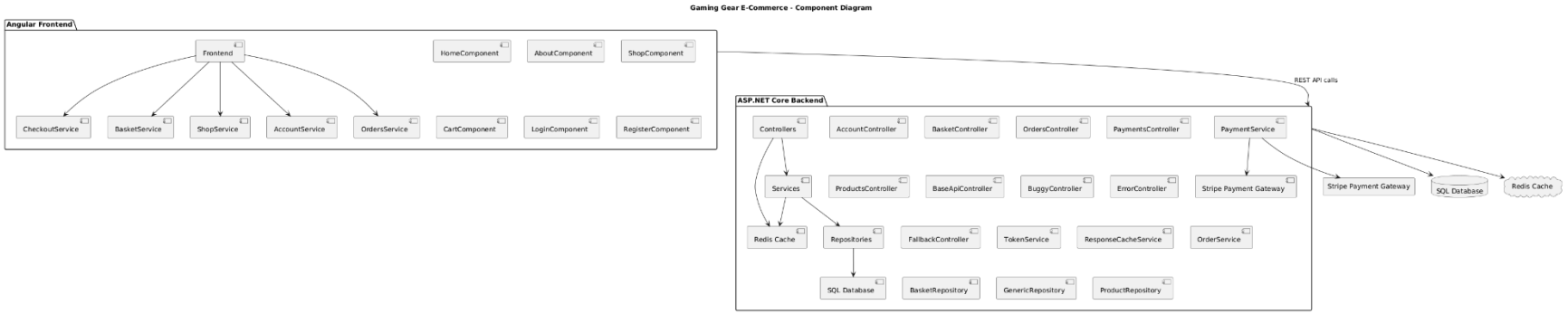

«person»
Customer
User browsing the store and making purchases
Uses
[HTTPS]
Gaming Gear
[system]
«container»
Frontend Application
[Angular]
Displays UI, handles routing and sends HTTP requests to backend
Sends API requests
[JSON/HTTPS]
«container»
Backend API
[ASP.NET Core]
Handles business logic, authentication, product management, cart, and checkout
Uses
[C#/.NET]
Reads/Writes cart and cached data
[Redis protocol]
Processes payments
[HTTPS]
«container»
EF Core Layer
[Entity Framework Core]
Object-Relational Mapper to interact with SQL Database
«container»
Redis Cache
[Redis]
Caches product data and stores shopping cart state
«external_system»
Stripe
Handles online payment processing
Reads/Writes
«container»
SQL Database
[MSSQL]
Stores users, products, orders, and payment records

5. Component Diagram

Description:

Shows detailed components within the backend (Controllers: Account, Basket, Orders, Payments, Products; Services: Token, Payment, ResponseCache; Repositories: Basket, Product, Generic)

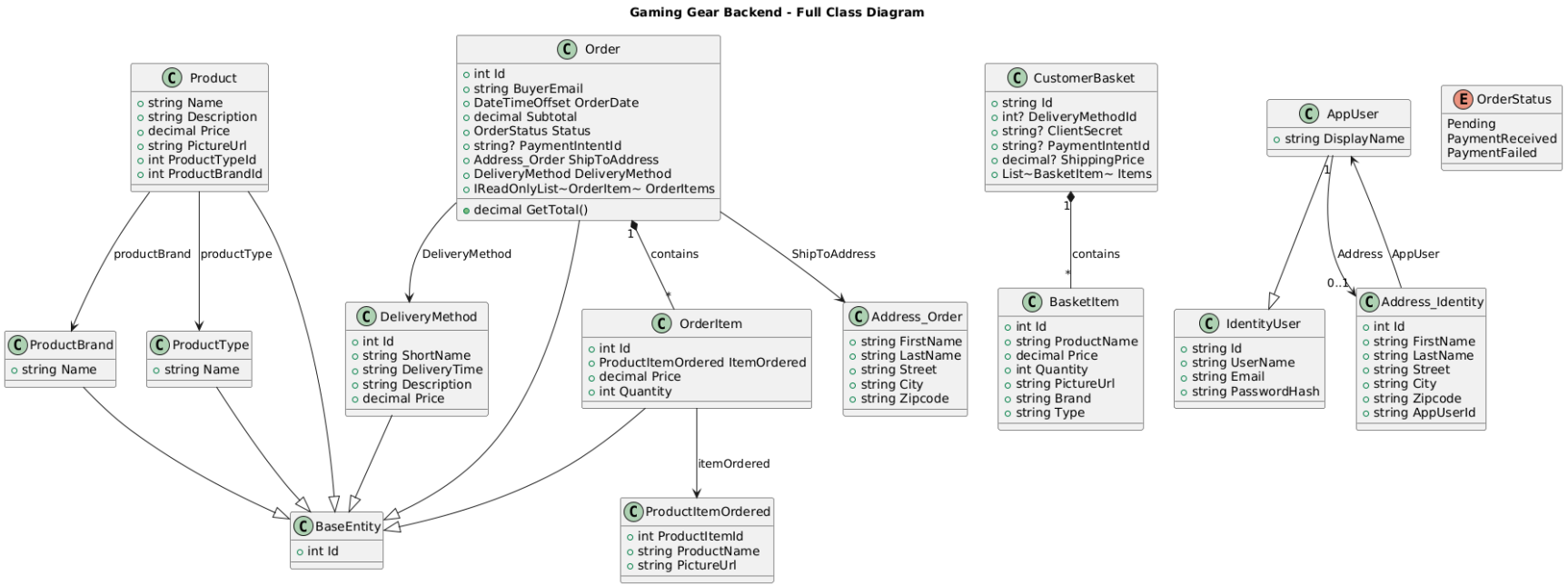
and frontend Angular modules and services (Shop, Basket, Account, Checkout).



6. Code (Class) Diagram

Description:

Illustrates key domain entities and relationships such as Product, ProductBrand, ProductType, BasketItem, CustomerBasket, Order, OrderItem, DeliveryMethod, AppUser, and Address.



7. Non-Functional Requirements

- Performance:

Fast response times achieved via caching with Redis and optimized database queries.

- Scalability:

The architecture supports scaling backend API and caching layers independently.

- Security:

  Authentication is handled via ASP.NET Core Identity. Payment data is securely handled via Stripe integration.

- Maintainability:

  Clear separation of concerns using layered architecture (Controllers, Services, Repositories) and modular Angular components.

- Reliability:

  Payment and order processing workflows include status tracking and fallback/error handling.

8. Key Design Decisions

- Use of Redis caching for cart and common response data to reduce database load and speed up operations.
- Separation of backend concerns into Controllers, Services, and Repositories for testability and maintainability.
- Angular frontend structured into feature modules with dedicated services for shop, basket, account, and checkout logic.
- Payment processing fully handled on the backend to comply with security best practices.
- No admin panel — product data is managed externally or seeded directly in the database.

9. Assumptions and Constraints

- No admin interface available for product management.

- Stripe is the only payment gateway supported.

- Caching is mandatory for cart and frequently requested data.

- Authentication relies on ASP.NET Core Identity standard flows.

- SQL Server is the primary persistent data store.

10. Glossary

Term	Definition
Basket	Temporary shopping cart containing selected products before checkout.
CustomerBasket	Entity representing a user's basket with items and payment info.
Order	Completed purchase with items, shipping address, and payment status.
DeliveryMethod	Shipping options available for orders.
Redis	In-memory caching service used for cart and response caching.
Stripe	Third-party payment service provider integrated for handling payments securely.
ASP.NET Identity	Authentication and authorization framework used by the backend.