

AllegroStyle – Online Marketplace Platform DOCS

1. Application Name

AllegroStyle – Online Marketplace Platform

2. Problem Area

AllegroStyle addresses the needs of small-to-medium retail businesses that want to expand their sales channels beyond brick-and-mortar by offering a distributed e-commerce platform similar to Allegro. It provides a public marketplace storefront where individual shoppers can browse, compare, and purchase goods from a single seller or brand.

3. Project Goals

- Deliver a scalable, distributed web application following a client–server architecture (React front-end + Spring Boot REST back-end).
- Simplify online shopping for customers with fast search, intuitive navigation, and secure checkout.
- Give the seller (admin) powerful management tools, including real-time inventory control and advanced analytics.
- Ensure high security, reliability, and maintainability through Spring Security, Bean Validation, CI.

4. System Responsibilities

AllegroStyle will:

- Authenticate and authorize users with role-based access (buyer vs. admin).
- Expose RESTful endpoints for all core business resources (products, carts, orders, users).
- Persist data in a relational database with version-controlled schema migrations.
- Provide a responsive SPA front-end that consumes the API.
- Generate analytical reports and visual charts for the admin dashboard.

5. System Users

- ****Guest**** – Unregistered visitor browsing products.
- ****Buyer**** – Registered customer with a personal cart, orders, and profile.
- ****Administrator**** – Store owner with full CRUD privileges and analytics dashboard.

6. Functionalities

- Secure user registration, login, logout (JWT + refresh tokens).
- Buyer profile management (edit personal data, change password, view order history).
- Product catalogue: browse, full-text search, filter, category navigation.
- Product details with gallery, price history, ratings & reviews.
- Shopping cart (session-aware), wishlist, recently viewed items.
- Order placement, payment integration (mock provider), shipment tracking.
- ****Admin-only**** product CRUD, bulk import/export (CSV).
- Complex sales analytics: revenue charts, top products, inventory alerts.
- Dark/light theme & fully responsive UI.
- Public REST API documented with OpenAPI/Swagger.
- API rate-limiting and input validation (Bean Validation).

7. User Requirements

Part 1: Domain Structure (Entities, Properties, Relationships)

Entities & Core Attributes

- ****Address****

id (PK), city, country, street, house_number

- ****User****

id (PK), username, email, password, role {BUYER, ADMIN}, created_at, address_id (FK → Address)

- ****Category****

id (PK), name, parent_category_id (FK → Category)

- ****Product****

id (PK), name, description, price, stock_quantity, created_at, category_id (FK → Category), seller_id (FK → User)

- ****Cart****

user_id (PK & FK → User), created_at

- ****CartItem****

cart_id (PK, FK → Cart), product_id (PK, FK → Product), quantity

- ****Order****

id (PK), user_id (FK → User), placed_at, status, total, shipping_address_id (FK → Address)

- ****OrderItem****

order_id (PK, FK → Order), product_id (PK, FK → Product), quantity, unit_price

- ****Review****

id (PK), user_id (FK → User), product_id (FK → Product), rating, comment, created_at

Relationships

Relationship	Cardinality	Notes
Address → User	1 : *	A single address can be shared by many users
Address → Order	1 : *	Each order stores a shipping address
User → Cart	1 : 1	`user_id` is both PK & FK in Cart
User → Order	1 : *	A buyer can place many orders
User → Product	1 : *	Seller owns many products
User → Review	1 : *	Users leave many reviews
Category → Product	1 : *	Products belong to a single category
Category → Category	1 : *	Self-referencing parent/child hierarchy
Product → Review	1 : *	Each product may collect many reviews
Product → OrderItem	1 : *	Product appears in many order lines
Product → CartItem	1 : *	Product appears in many carts
Order → OrderItem	1 : *	Order is composed of multiple order items
Cart → CartItem	1 : *	Cart holds multiple cart items

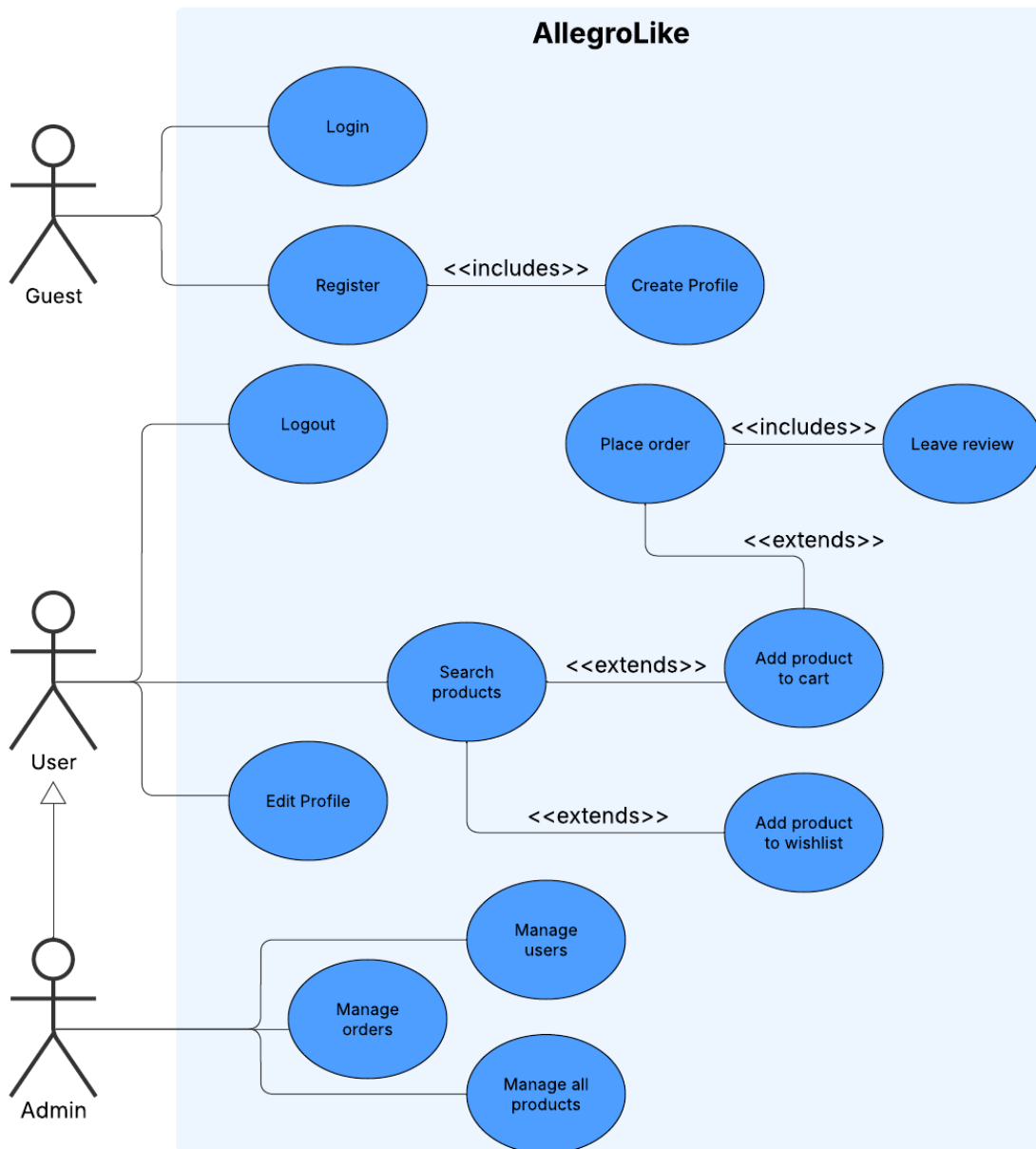
Part 2: Expected Functionality

- Register / login / logout (JWT)
- Edit profile & address book
- Browse & search catalogue; filter by category/price/keyword
- Add/remove items to cart & wishlist
- Checkout → create Order, clear Cart
- Leave product review (only after purchase)
- ****Admin****: manage users, products, categories, orders; generate analytics

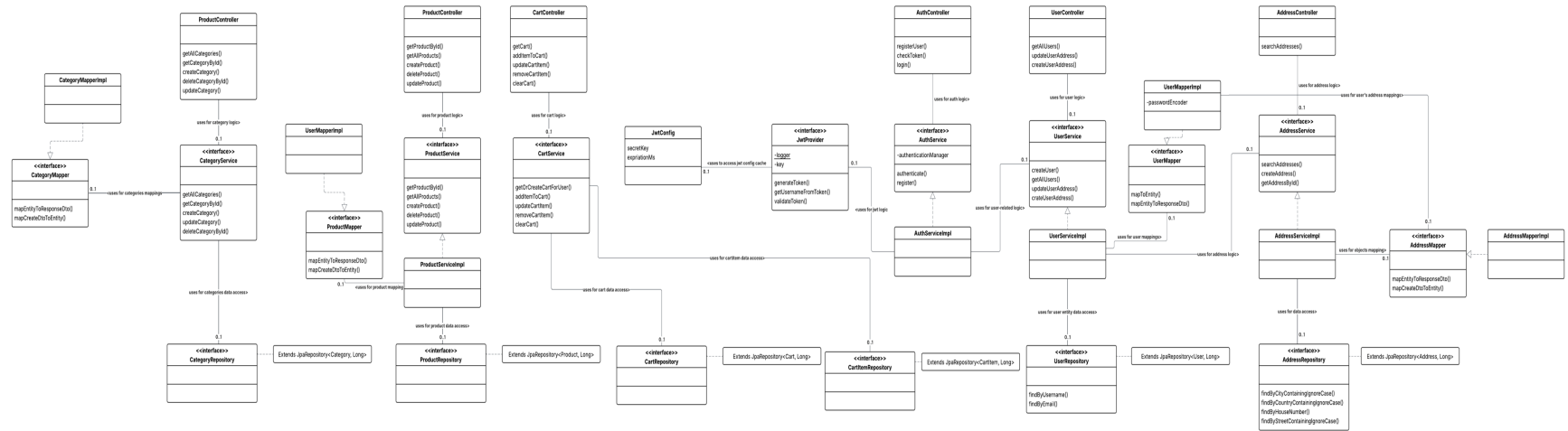
Part 3: Constraints

- Must support the latest 3 versions of Chrome, Firefox, and Edge.
- Passwords hashed (bcrypt, strength ≥ 12).
- Handle ≥ 100 concurrent buyers with p95 response < 2 s.
- Daily automated database backup (retention 30 days).
- 99 % monthly uptime; automated health checks & alerting.

8. Functional Requirements



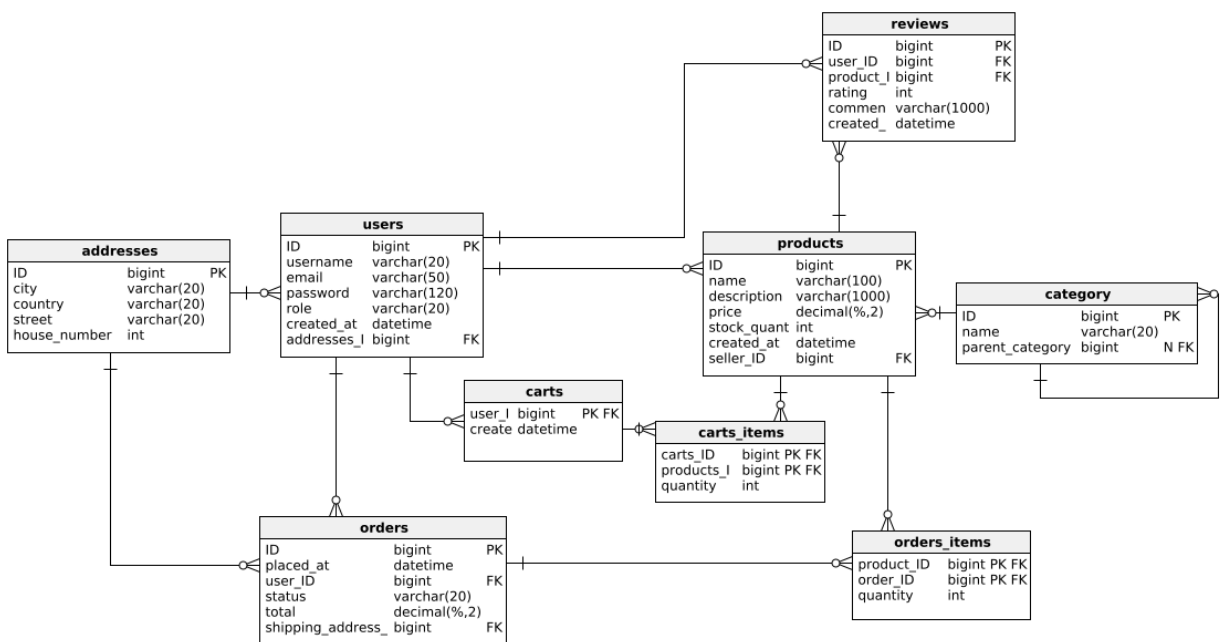
9. System Structure (Conceptual Diagram)



10. Non-functional Requirements

Constraint	Metric
Browser compatibility	Latest 3 versions of Chrome/Firefox/Edge (QA quarterly)
Authentication security	bcrypt hash ≥ 12 , JWT expiry 1 h (pentest report)
Performance	≤ 2 s p95 latency @ 100 concurrent users (JMeter load test)
Backup	Daily dump, 30-day retention, recovery test monthly
Uptime	≥ 99 % per calendar month (monitoring SLA)

11. Database Schema



12. API Endpoints

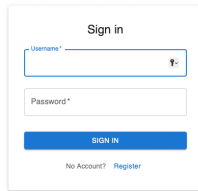
Endpoint	Method	Description
/api/auth/register	POST	Register new buyer
/api/auth/login	POST	Login & receive tokens
/api/products	GET	List / search products
/api/products/{id}	GET	Product details
/api/products	POST	Create product (admin)
/api/products/{id}	PUT	Update product (admin)
/api/products/{id}	DELETE	Delete product (admin)
/api/cart	GET	Current cart
/api/cart/items	POST	Add item
/api/cart/items/{id}	PATCH	Update quantity
/api/cart/items/{id}	DELETE	Remove item
/api/orders	POST	Place order
/api/orders	GET	Buyer order history
/api/admin	GET	Analytics (admin)

13. Technologies & Tools

Technology	Layer	Purpose
React + Vite	Front-end	SPA user interface
TypeScript	Front-end	Type safety
Spring Boot	Back-end	REST API, business logic
Spring Security	Back-end	AuthN & AuthZ
Bean Validation	Back-end	Input/data validation
Spring Data JPA + Hibernate	Back-end	ORM data access
MySQL 8	Database	Primary relational store
Liquibase	DevOps	Schema migrations
Docker / Docker Compose	DevOps	Containerization
JWT	Security	Stateless sessions
d3.js	Front-end	Visual analytics
OpenAPI (Swagger)	Docs	API documentation

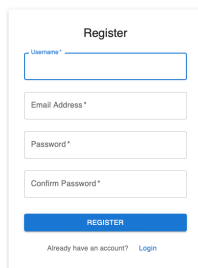
14. UI/UX Mock-ups

- Sign in



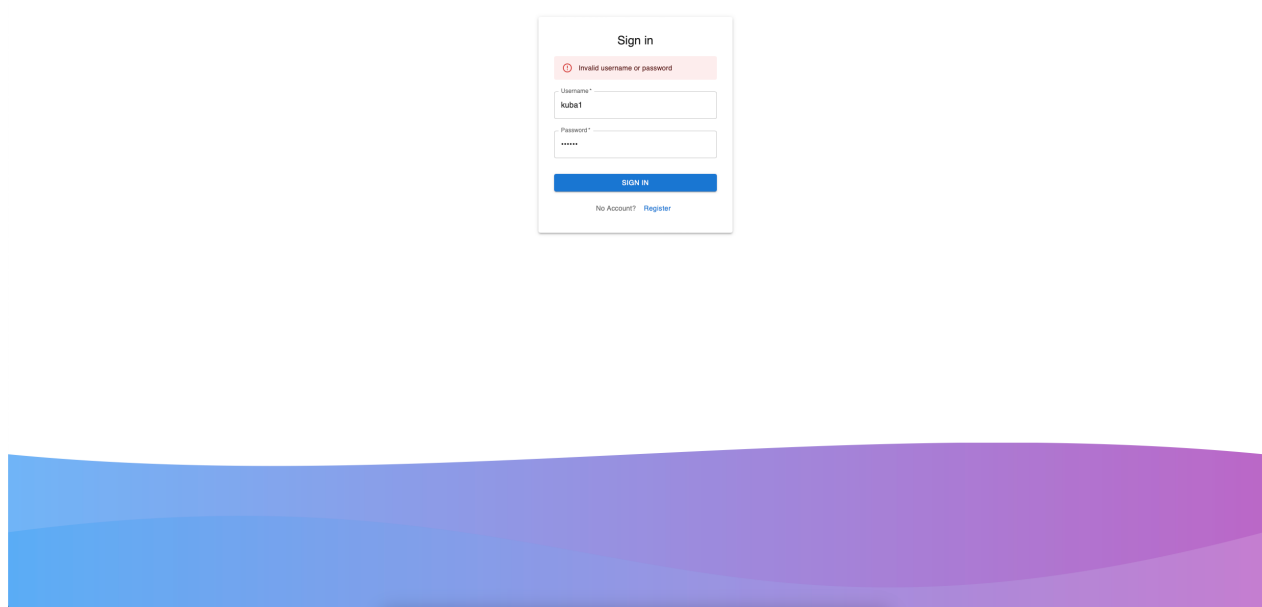
A mockup of a sign-in form. It features a title "Sign in" at the top. Below the title are two input fields: "Username*" and "Password*". The "Username*" field has a blue border and a small eye icon on the right. Below the input fields is a blue button labeled "SIGN IN". At the bottom, there is a link "No Account? Register" in a smaller font.

- Register

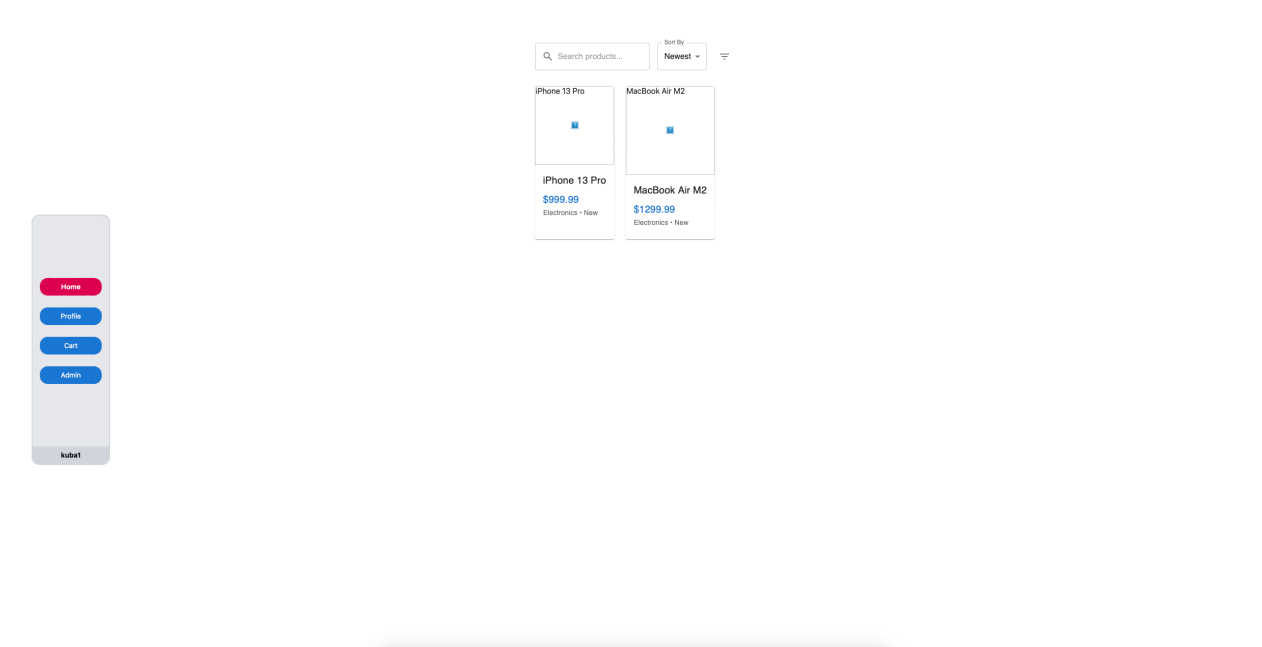


A mockup of a register form. It features a title "Register" at the top. Below the title are four input fields: "Username*", "Email Address*", "Password*", and "Confirm Password*". The "Username*" field has a blue border. Below the input fields is a blue button labeled "REGISTER". At the bottom, there is a link "Already have an account? Login" in a smaller font.

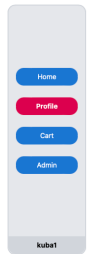
- Sign In ERROR



- Home Page



- Buyer Profile



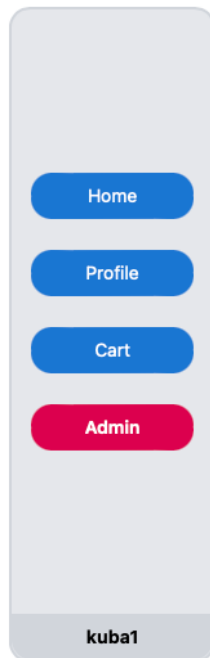
Your Profile

Username:
kuba1
Role:
USER

Your Wishlist

wishlist will be here

- Navbar



- Admin Dashboard (sales charts)

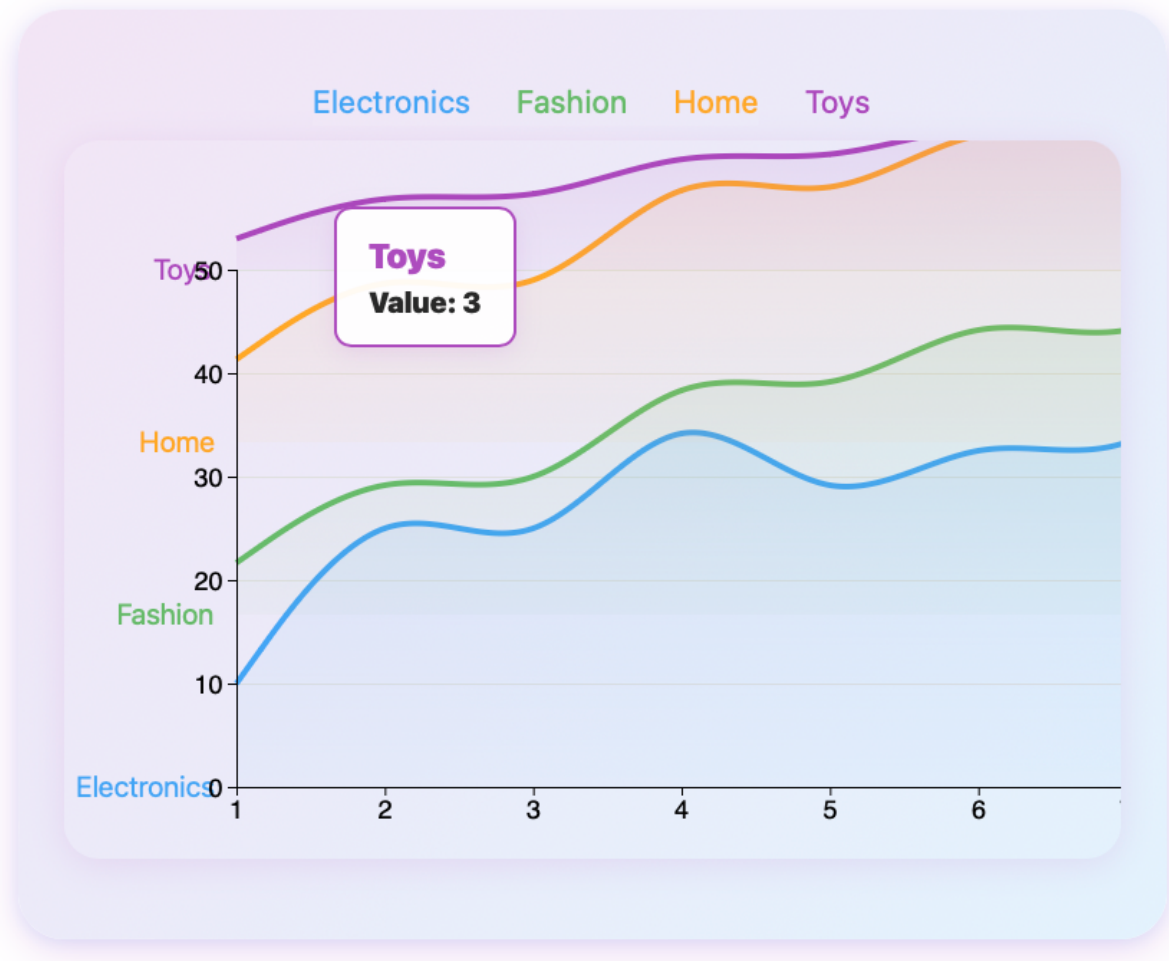


- Sales Overview



- Category Trends

Category Trends (Ridgeline Chart)



15. Project Team

- Jakub Graliński (s30351) & Mateusz Laskowski (s30613)