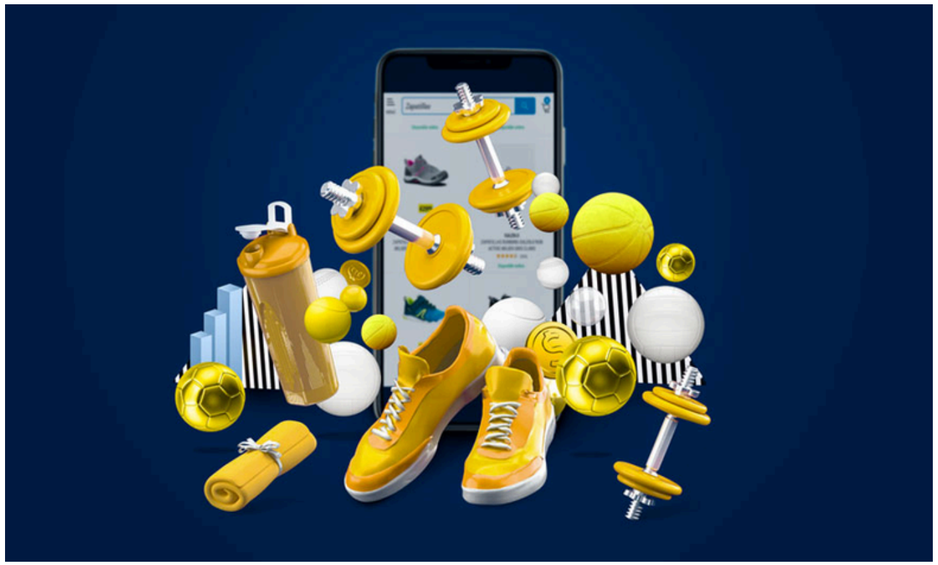
SPORTGEAR ONLINE E-COMMERCE

Juan Esteban Carrillo Garcia - 20212020147
Alejandro Sebastián González Torres - 20191020143
Miguel Angel Babativa Niño - 20191020069

RESEARCH QUESTION



How can a specialized e-commerce platform bridge the gap between generic online retail and the specific needs of sports enthusiasts?

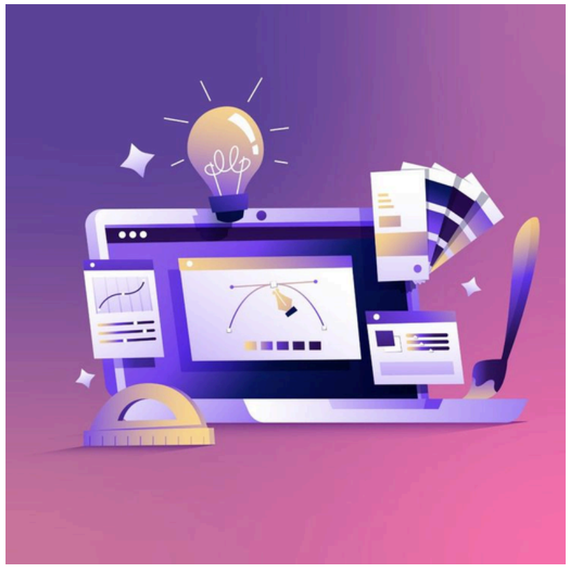
EXPECTED PRODUCT

Create a specialized e-commerce platform that addresses the unique needs of sports enthusiasts by providing personalized product recommendations

PROJECT EVOLUTION & BACKGROUND

PHASE 1: CONCEPTUAL MODELING

- Business Model Canvas (BMC)
- User Stories
- Mockups
- CRC Cards for OOP Design
- UML Class & Architecture Diagrams



PHASE 2: DISTRIBUTED IMPLEMENTATION

- Dual Backend Architecture
- Containerized Deployment
- Automated CI/CD Pipeline
- Comprehensive Testing Strategy

PHASE 1: CONCEPTUAL MODELING

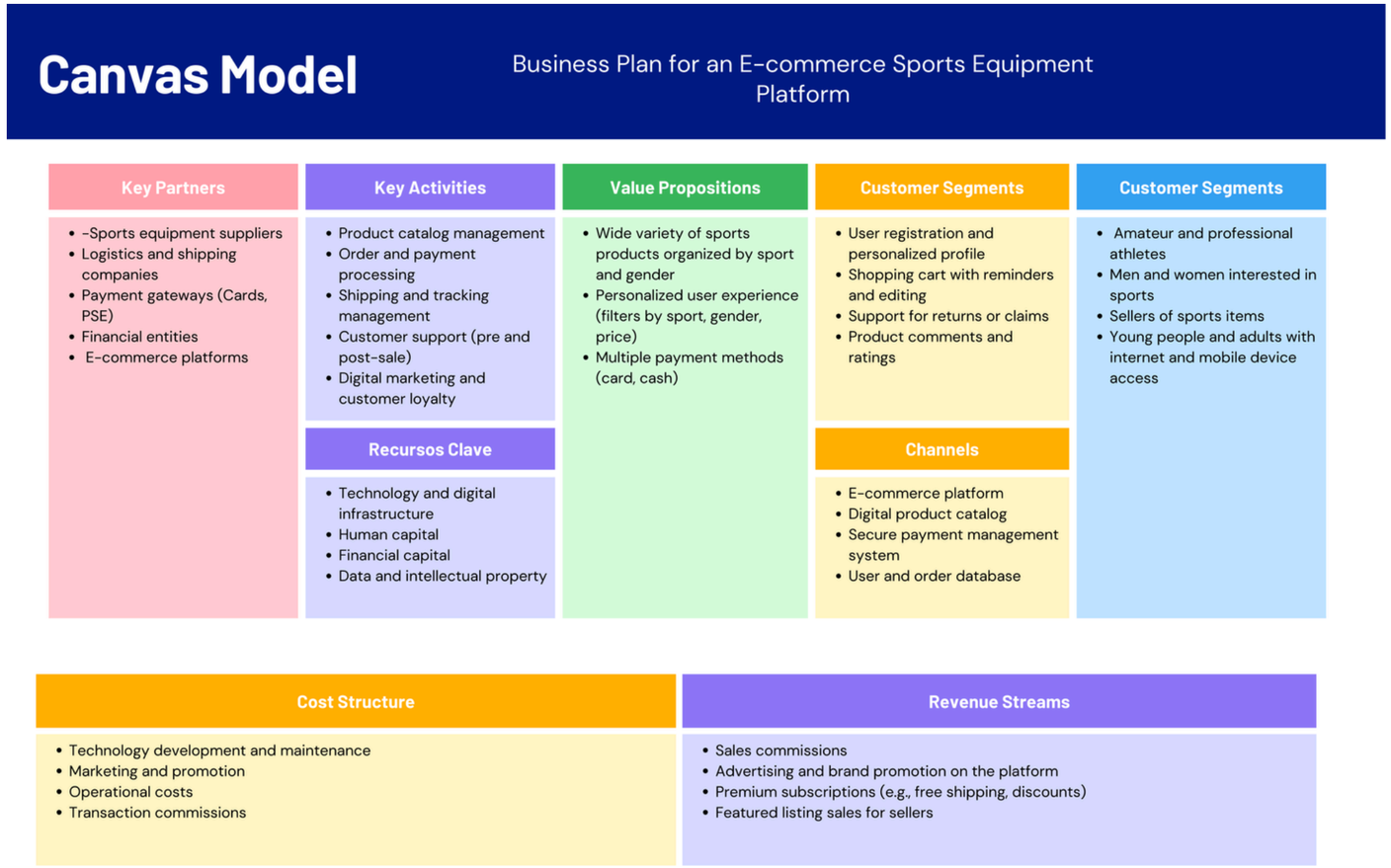
BUSINESS MODEL CANVAS IMPLEMENTATION

VALUE PROPOSITION:

- CURATED SPORTS EQUIPMENT SELECTION
- SEAMLESS SHOPPING EXPERIENCE
- FAST, RELIABLE DELIVERY

COST STRUCTURE:

- PLATFORM DEVELOPMENT & MAINTENANCE
- MARKETING & CUSTOMER ACQUISITION
- LOGISTICS PARTNERSHIPS
- PAYMENT PROCESSING FEES



USER STORIES

Title: User Registration	**Prority:** High	**Estimate:** 5 points
User Story: As a visitor, I want to register with email and password so I can access the platform.		
Acceptance Criteria: Registration form validates required fields and password strength. Duplicate email is rejected with friendly error. Successful registration creates user in database and returns confirmation.		

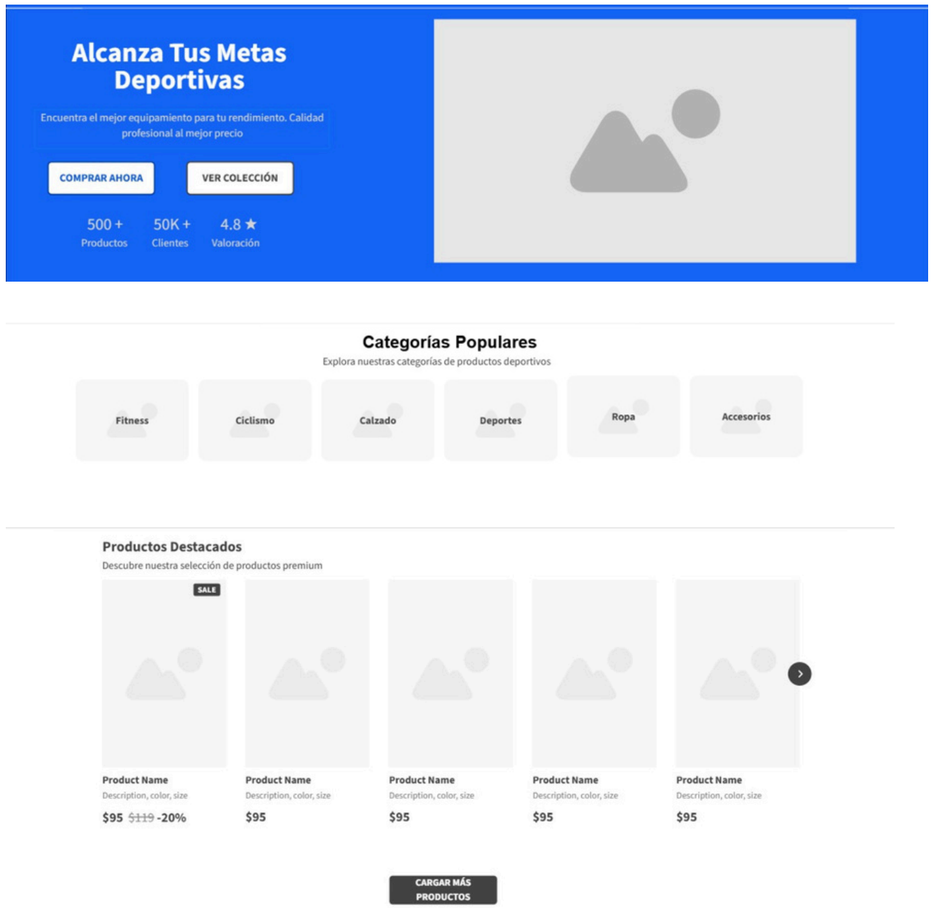
Title: User Login	**Prority:** High	**Estimate:** 5 points
User Story: As a user, I want to log in to access my account.		
Acceptance Criteria: Login authenticates via backend and returns JWT/session. Invalid credentials show error without revealing specifics. Auth state persists and protects private routes.		

Title: Product Catalog Listing	**Prority:** High	**Estimate:** 5 points
User Story: As a user, I want to browse products with pagination and filters.		
Acceptance Criteria: API supports pagination, search, and category filters. Frontend lists products with loading and empty states. Performance acceptable for 1k+ products.		

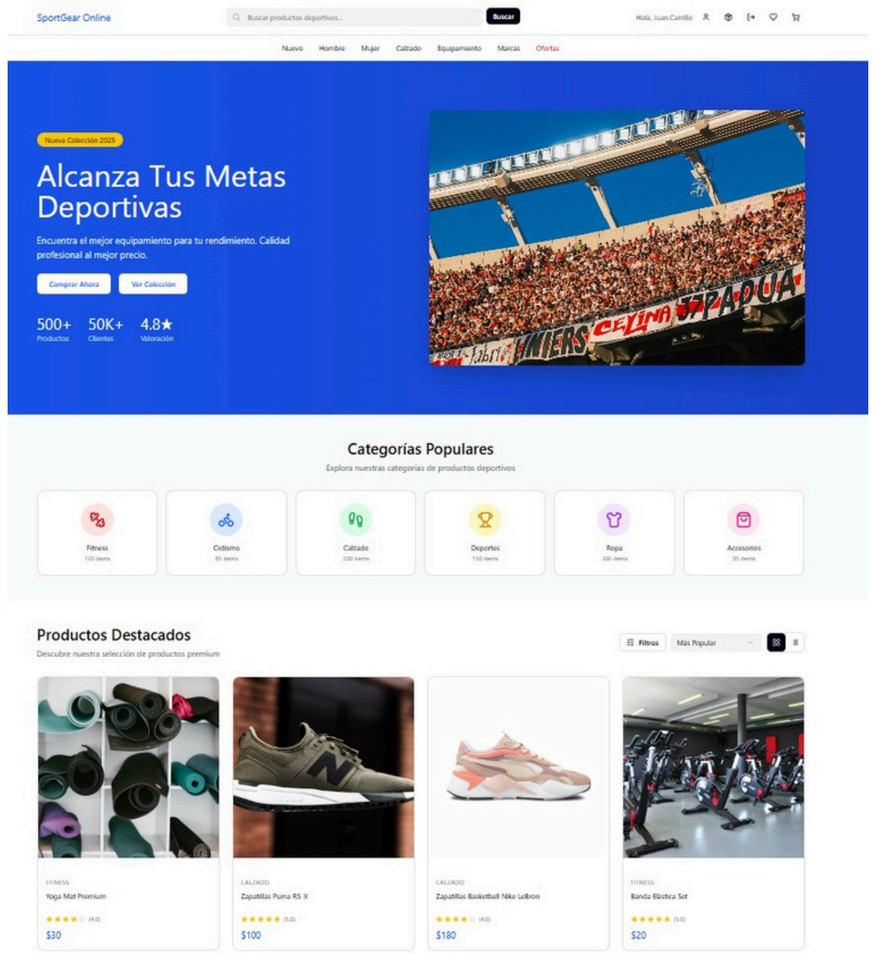
Title: Product Details Page	**Prority:** Medium	**Estimate:** 3 points
User Story: As a user, I want to view a product's details.		
Acceptance Criteria: Displays images, description, price, stock. Handles not-found gracefully. Deep links load directly.		

IN OUR REQUIREMENTS ASSESSMENT PHASE, WE OBTAINED 19 USER STORIES WHICH PROVIDED US WITH ALL THE NECESSARY FUNCTIONAL REQUIREMENTS TO PROCEED WITH THE SOFTWARE CONSTRUCTION.

MOCKUPS


Alcanza Tus Metas Deportivas
Encuentra el mejor equipamiento para tu rendimiento. Calidad profesional al mejor precio
COMPRAR AHORA
VER COLECCIÓN
500 + Productos
50K + Clientes
4.8 ★ Valoración
Categorías Populares
Explora nuestras categorías de productos deportivos
Fitness
Ciclismo
Calzado
Deportes
Ropa
Accesorios
Productos Destacados
Descubre nuestra selección de productos premium
SALE
Product Name
Description, color, size
$95 $119 -20%
$95
$95
$95
$95
CARGAR MÁS PRODUCTOS

->


SportGear Online
Buscar productos deportivos...
Buscar
Hola, Juan Carrillo
Nuevo
Hombre
Mujer
Calzado
Equipamiento
Marcas
Ofertas
Nueva Colección 2025
Alcanza Tus Metas Deportivas
Encuentra el mejor equipamiento para tu rendimiento. Calidad profesional al mejor precio.
Comprar Ahora
Ver Colección
500+ Productos
50K+ Clientes
4.8★ Valoración
Categorías Populares
Explora nuestras categorías de productos deportivos
Fitness
Ciclismo
Calzado
Deportes
Ropa
Accesorios
Productos Destacados
Descubre nuestra selección de productos premium
Filtros
Más Popular
FITNESS
Yoga Mat Premium
$30
CALZADO
Zapatillas Puma RS-X
$100
CALZADO
Zapatillas Basketball Nike LeBron
$180
FITNESS
Banda Elástica Set
$20

CRC CARDS FOR OOP DESIGN

CustomerProfile	
Responsability	**Collaboration**
Maintain a local customer profile; ensure uniqueness of external_user_id; allow lookups by id, email or external_user_id; optionally sync/override name and email from upstream; basic validation of name/email formats.	Order (consuming user_id for ownership in listings and filters).

Product	
Responsability	**Collaboration**
Represent catalog item data (name, description, attributes); maintain pricing and availability; guard invariants (price ≥ 0, stock_quantity ≥ 0); expose data to be copied into line items at purchase time.	OrderItem (referencing the product when composing orders).

Order	
Responsability	**Collaboration**
Represent an order aggregate; own lifecycle status and timestamps; store a snapshot of shipping_address; manage line items (add, update quantity, remove); compute totals from items and validate consistency; prevent invalid transitions (e.g., modifications after completion).	OrderItem (composition of items), Payment (transaction records), Shipment (fulfillment tracking), CustomerProfile (owner context in queries).

OrderItem	
Responsability	**Collaboration**
Represent a purchasable line (product, quantity, unit_price); compute subtotal; validate quantity and unit_price; copy product data relevant for pricing at order time to ensure immutability.	Order (aggregate owner), Product (source for product_id and price snapshot).

Payment	
Responsability	**Collaboration**
Record payment attempts/transactions for an order; track method and status; support multiple attempts and failure states; validate amounts against order totals where applicable; capture timestamps for auditing.	Order (payment belongs to one order).

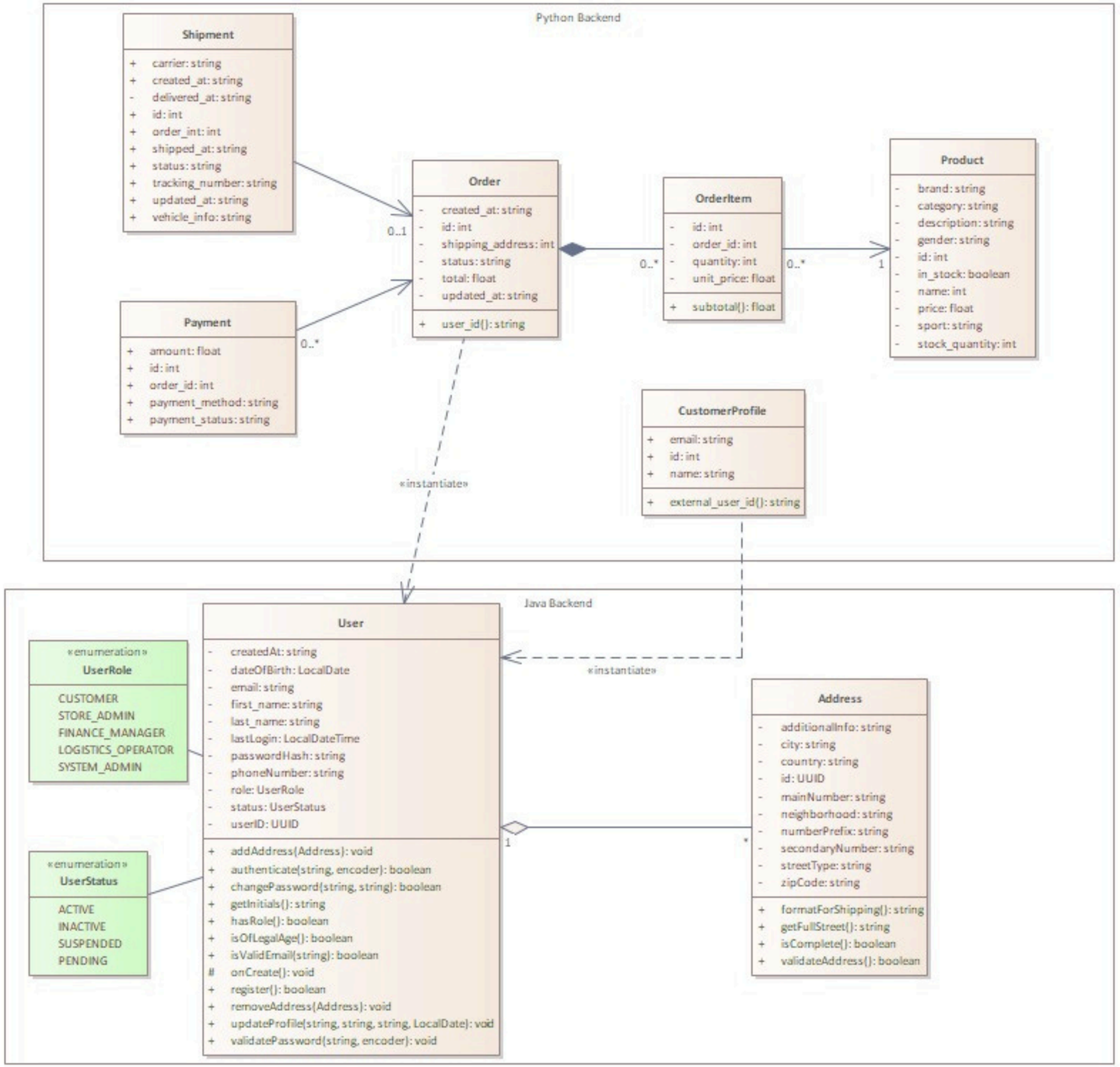
Shipment	
Responsability	**Collaboration**
Track shipment lifecycle (pending, shipped, in_transit, delivered, cancelled); store tracking_number, carrier, timestamps; enforce at most one shipment per order; provide status updates for UI and notifications.	Order (one order per shipment).

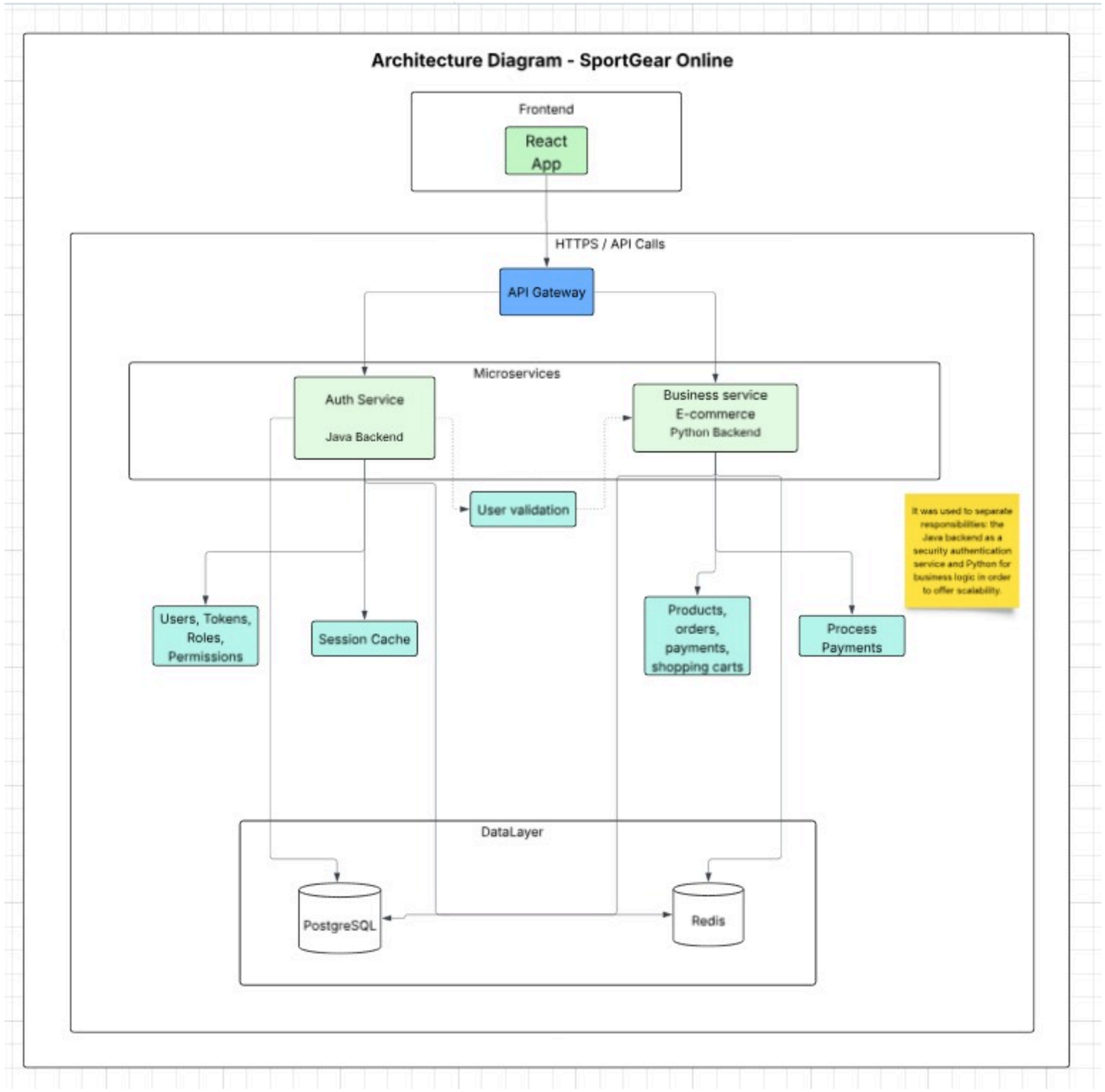
User	
Responsability	**Collaboration**
Own identity; store credentials hash securely; manage roles and status; support registration, authentication, password change; update profile fields; maintain audit fields (createdAt, lastLogin); manage associated addresses.	Address (owned collection), UserRole (role classification), UserStatus (status lifecycle)

Address	
Responsability	**Collaboration**
Represent a physical address; validate required components; format as a shipping-ready string; assemble street details from parts; indicate completeness for checkout flows.	User (address owner).

Enum: UserRole	
Responsability	**Collaboration**
Define available roles for authorization and feature access; serve as a stable taxonomy for the system.	User (role assignment and checks).

UML CLASS


Python Backend
Shipment
+ carrier: string
+ created_at: string
- delivered_at: string
+ id: int
+ order_int: int
+ shipped_at: string
+ status: string
+ tracking_number: string
+ updated_at: string
+ vehicle_info: string
Order
- created_at: string
- id: int
- shipping_address: int
- status: string
- total: float
- updated_at: string
+ user_id(): string
OrderItem
- id: int
- order_id: int
- quantity: int
- unit_price: float
+ subtotal(): float
Product
- brand: string
- category: string
- description: string
- gender: string
- id: int
- in_stock: boolean
- name: int
- price: float
- sport: string
- stock_quantity: int
Payment
+ amount: float
+ id: int
+ order_id: int
+ payment_method: string
+ payment_status: string
CustomerProfile
+ email: string
+ id: int
+ name: string
+ external_user_id(): string
0..1
0..*
0..*
1
0..*
«instantiate»
«instantiate»
Java Backend
User
- createdAt: string
- dateOfBirth: LocalDate
- email: string
- first_name: string
- last_name: string
- lastLogin: LocalDateTime
- passwordHash: string
- phoneNumber: string
- role: UserRole
- status: UserStatus
- userID: UUID
+ addAddress(Address): void
+ authenticate(string, encoder): boolean
+ changePassword(string, string): boolean
+ getInitials(): string
+ hasRole(): boolean
+ isOfLegalAge(): boolean
+ isValidEmail(string): boolean
# onCreate(): void
+ register(): boolean
+ removeAddress(Address): void
+ updateProfile(string, string, string, LocalDate): void
+ validatePassword(string, encoder): void
«enumeration»
UserRole
CUSTOMER
STORE_ADMIN
FINANCE_MANAGER
LOGISTICS_OPERATOR
SYSTEM_ADMIN
«enumeration»
UserStatus
ACTIVE
INACTIVE
SUSPENDED
PENDING
Address
- additionalInfo: string
- city: string
- country: string
- id: UUID
- mainNumber: string
- neighborhood: string
- numberPrefix: string
- secondaryNumber: string
- streetType: string
- zipCode: string
+ formatForShipping(): string
+ getFullStreet(): string
+ isComplete(): boolean
+ validateAddress(): boolean
1
*



ARCHITECTURE DIAGRAM

PHASE 2: DISTRIBUTED IMPLEMENTATION

MICROSERVICES – DUAL BACKEND ARCHITECTURE

Java Backend

Authentication, users, JWT issuance/validation, user management.

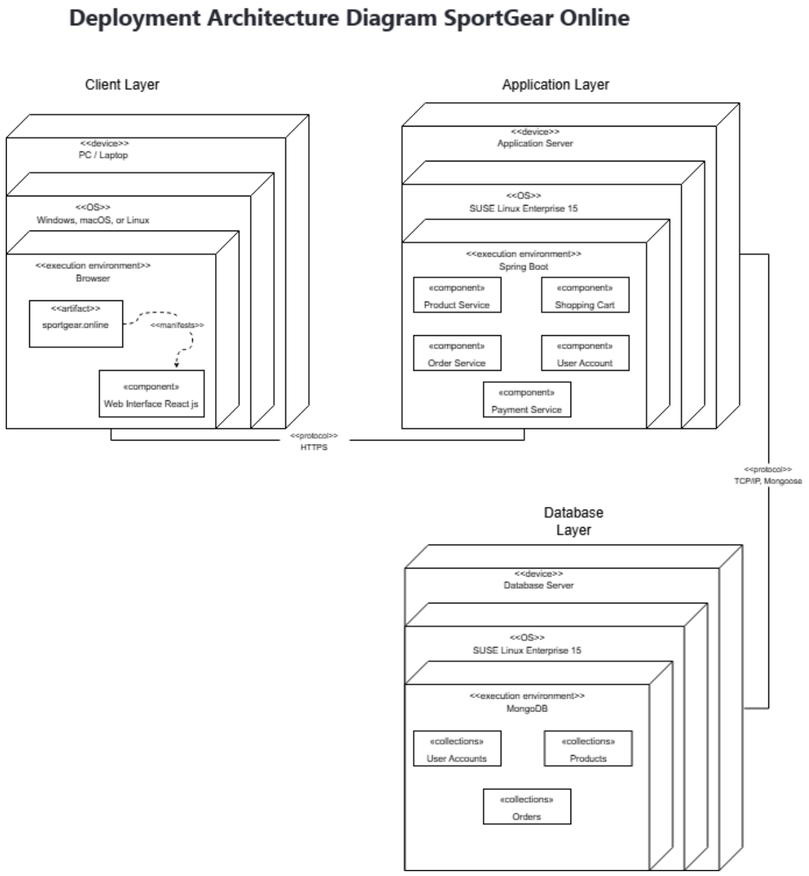
Python Backend

Product catalog, cart, checkout, orders, payments, shipment management

Advantages

Separation of concerns: security code isolated from business logic.
Independent scaling: scale auth and business services separately.
Technology fit: use Java ecosystem strengths for auth/security, Python for fast API iteration and data processing.
Independent deployments & teams: faster ownership and smaller release blast radius.

CONTAINERIZED



THIS DEPLOYMENT ARCHITECTURE IMPLEMENTS A SCALABLE THREE-TIER STRUCTURE WITH CONTAINERIZED MICROSERVICES, SEPARATING PRESENTATION, BUSINESS LOGIC, AND DATA PERSISTENCE LAYERS TO ENSURE MODULARITY, SECURITY, AND INDEPENDENT SCALABILITY ACROSS OUR DEVELOPMENT, STAGING, AND PRODUCTION ENVIRONMENTS.

AUTOMATED CI/CD PIPELINE

Why CI/CD ?

- Reduce manual errors and merge pain
- Enforce quality gates (tests, linting, vulnerability scanning)
- Deliver faster via repeatable, auditable deployments
- Provide fast feedback to developers

Pipeline Overview (high level)

1. Push / PR to `main` or feature branch triggers pipeline
2. Install dependencies & run linters (JS/TS, Python, Java)
3. Run unit tests (Python + Java) in parallel
4. Run integration tests (start test DBs or use service containers)
5. Build artifacts and container images (multi-stage Docker)
6. Run security & static scans (SAST, dependency vulnerability scanning)
7. Publish artifacts (container registry, test reports)
8. Deploy to staging (automated)
9. Run smoke / E2E tests in staging
10. Manual approval → deploy to production (with rollout strategy)

COMPREHENSIVE TESTING STRATEGY

Test Types & Purpose

- **Unit tests**: isolate functions/classes (mock dependencies)
- **Integration tests:** test DB, repositories, and service wiring (use test DBs or testcontainers)
- **API TEST :**ensure stable API contracts between Java Auth and Python services (Pact or contract tests)
- **BDD / Acceptance tests:** Gherkin scenarios for user stories (pytest-bdd)

"A layered testing strategy ensures correctness at every level and speeds safe delivery."

THANK YOU