🌾 FARMERS MARKET PLATFORM - COMPLETE ARCHITECTURE DIAGRAM

Version: 2.0 - Complete System Architecture
Last Updated: December 2024
Status: Production Ready - Full Stack Documentation

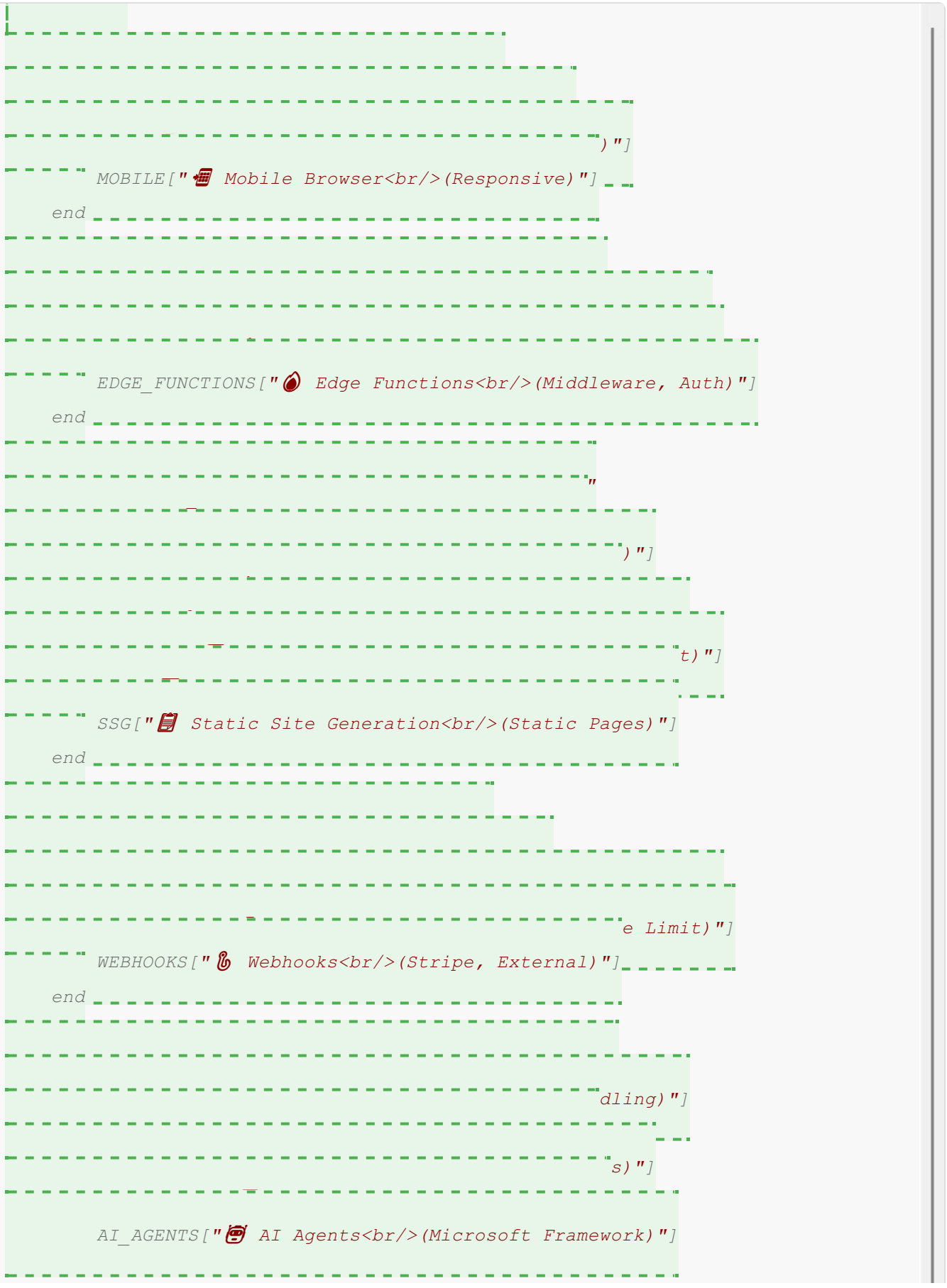
📋 TABLE OF CONTENTS

🌐 SYSTEM OVERVIEW

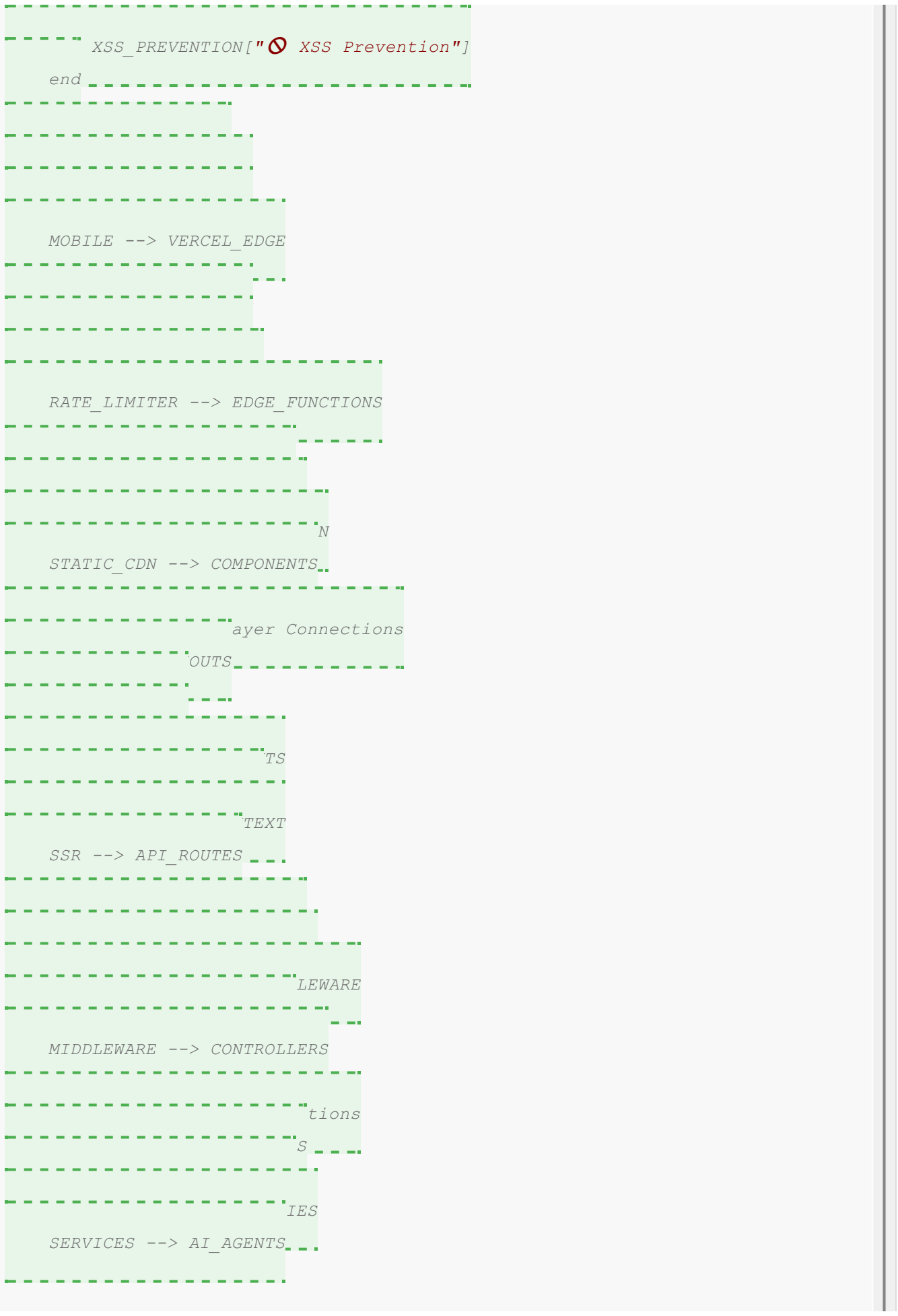
Technology Stack

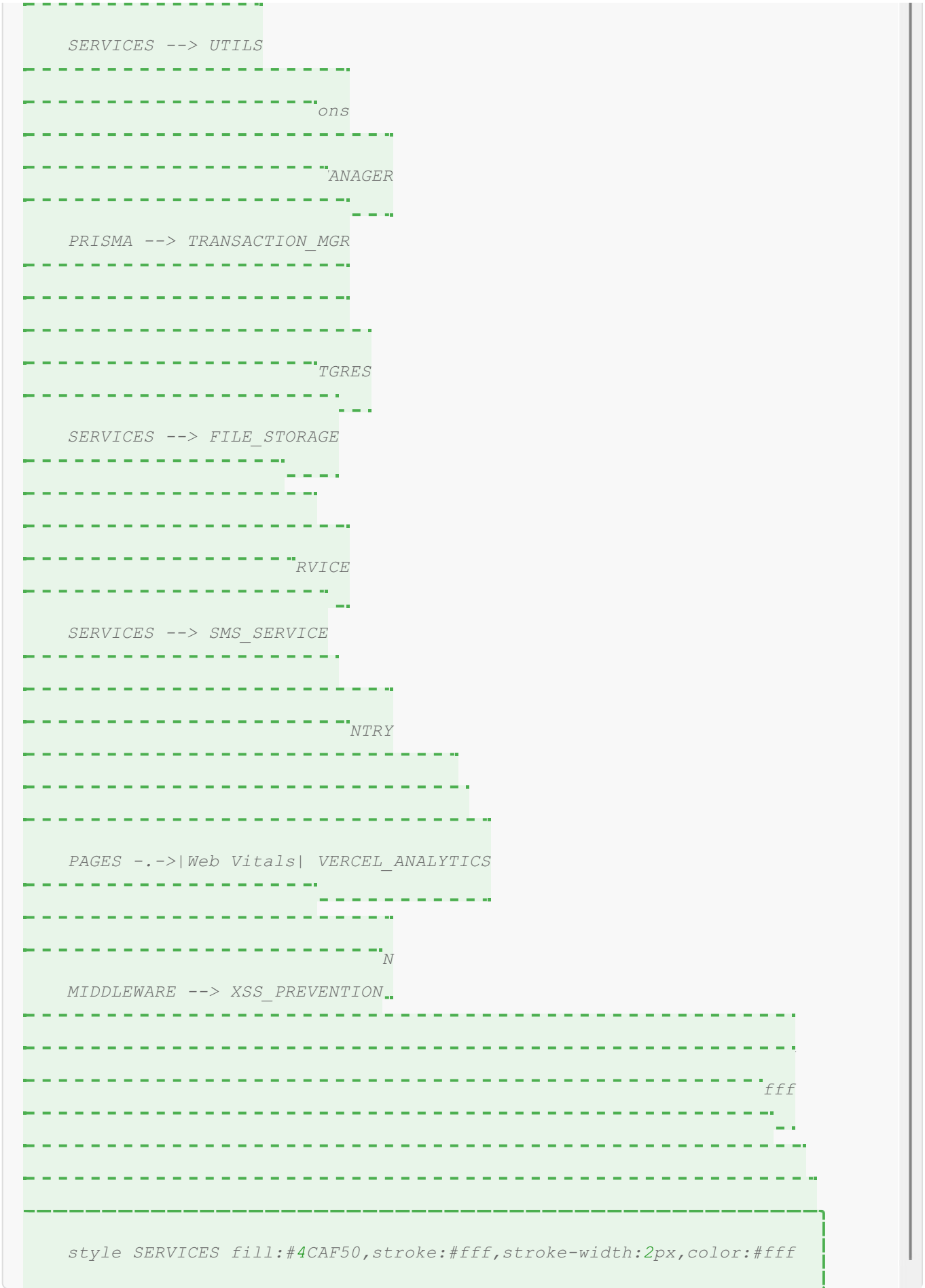
```
Frontend:
  Framework: Next.js 15 (App Router)
  Language: TypeScript 5.3+ (Strict Mode)
  UI Library: React 18.3+
  Styling: Tailwind CSS
  State Management: React Context + Server State
  Forms: React Hook Form + Zod Validation

Backend:
  Framework: Next.js API Routes + Server Actions
  Language: TypeScript 5.3+
  ORM: Prisma 5.7+
  Database: PostgreSQL 15+
  Cache: Redis (Upstash)
  Session Store: Redis-backed Sessions

Authentication:
  Provider: NextAuth v5 (Auth.js)
  Strategies:
    - Credentials (Email/Password)
    - OAuth (Google, GitHub)
  Session: JWT + Database Sessions
  Authorization: Role-Based Access Control (RBAC)

Infrastructure:
  Hosting: Vercel (Edge Network)
  Database: PostgreSQL (Managed)
  Cache: Upstash Redis
  Storage: Cloudinary (Images/Assets)
  Payments: Stripe
  Email: SendGrid / Resend
  Monitoring: Sentry + OpenTelemetry + UptimeRobot

AI/ML:
  Framework: Microsoft Agent Framework
  Tracing: OpenTelemetry
  Analytics: Azure Application Insights
```

🏛️ COMPLETE LAYER ARCHITECTURE



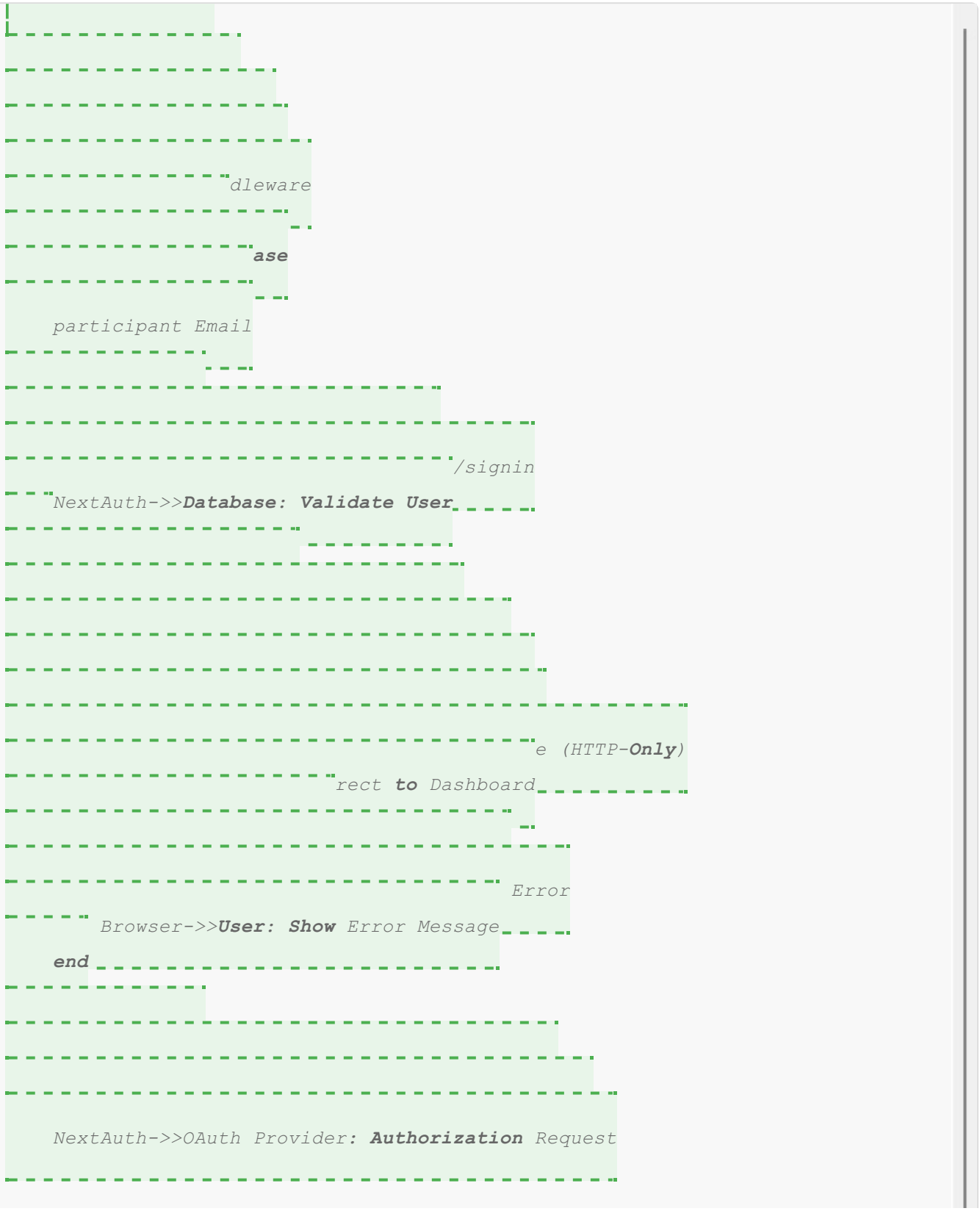
```
        UTILS["🔧 Utils<br/>(Helpers, Formatters)"]
    end

"]

        TRANSACTION_MGR["📝 Transaction Manager<br/>(ACID Compliance)"]
    end

L<br/>(Primary Database)<br/>Users, Farms,

IS[("⚡ Redis Cache<br/>(Upstash)<br/>Sessions, Cache<br/>Rate

_STORAGE[("☁️ Cloudinary<br/>(File Storage)<br/>Images,
ts")]
    end

end)"]

        SMS_SERVICE["💬 SMS Service<br/>(Twilio - Optional)"]
    end

toring)"]

        VERCEL_ANALYTICS["📊 Vercel Analytics<br/>(Web Vitals)"]
    end

l"]

        CSRF_PROTECTION["🔐 CSRF Protection"]
```

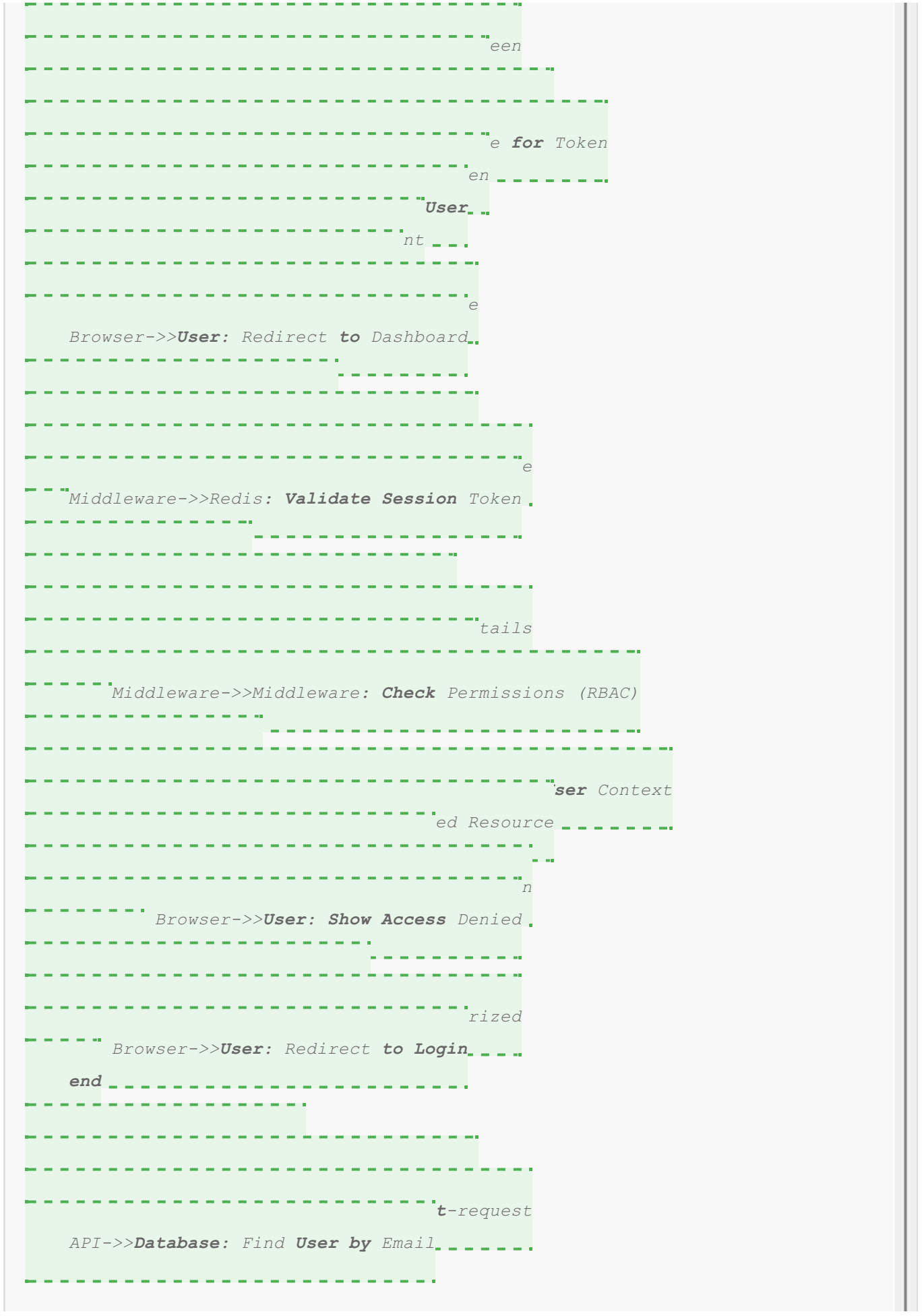

XSS_PREVENTION["🚫 XSS Prevention"]
end
MOBILE --> VERCEL_EDGE
RATE_LIMITER --> EDGE_FUNCTIONS
N
STATIC_CDN --> COMPONENTS
ayer Connections
OUTS
TS
TEXT
SSR --> API_ROUTES
LEWARE
MIDDLEWARE --> CONTROLLERS
tions
S
IES
SERVICES --> AI_AGENTS

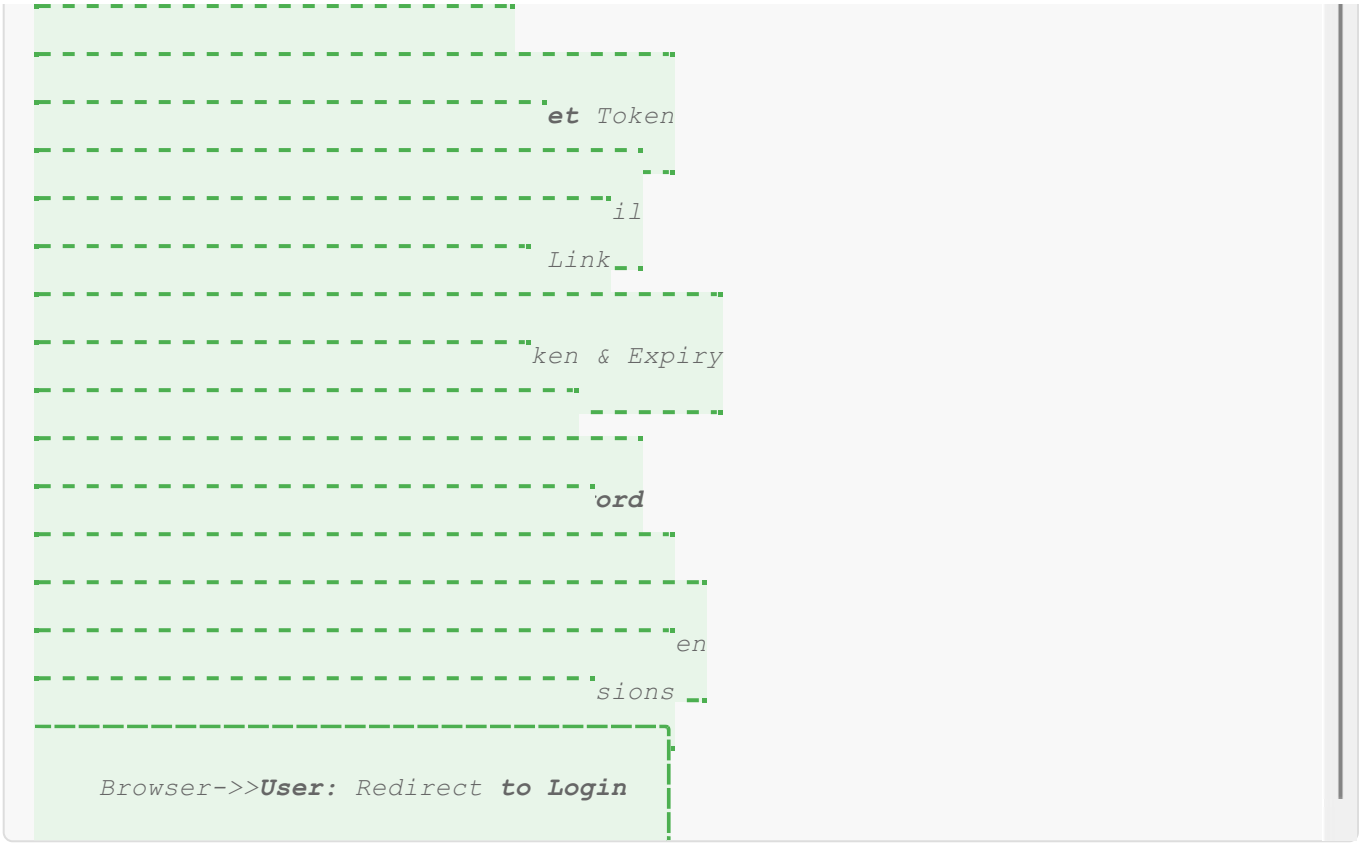

SERVICES --> UTILS
ons
ANAGER
PRISMA --> TRANSACTION_MGR
TGRES
SERVICES --> FILE_STORAGE
RVICE
SERVICES --> SMS_SERVICE
NTRY
PAGES -.->|Web Vitals| VERCEL_ANALYTICS
N
MIDDLEWARE --> XSS_PREVENTION
fff
style SERVICES fill:#4CAF50,stroke:#fff,stroke-width:2px,color:#fff

AUTHENTICATION & AUTHORIZATION ARCHITECTURE

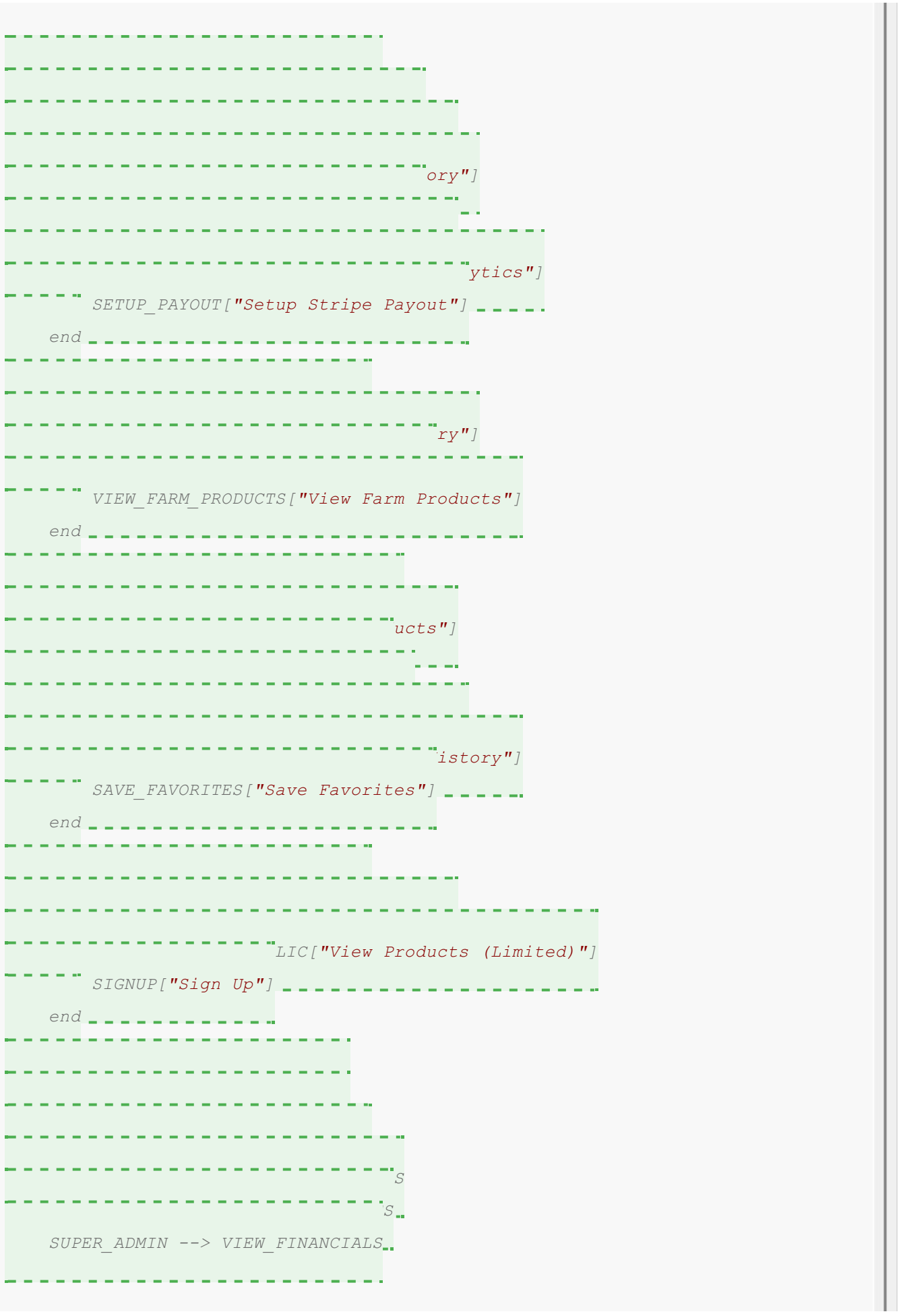
Complete Auth Flow

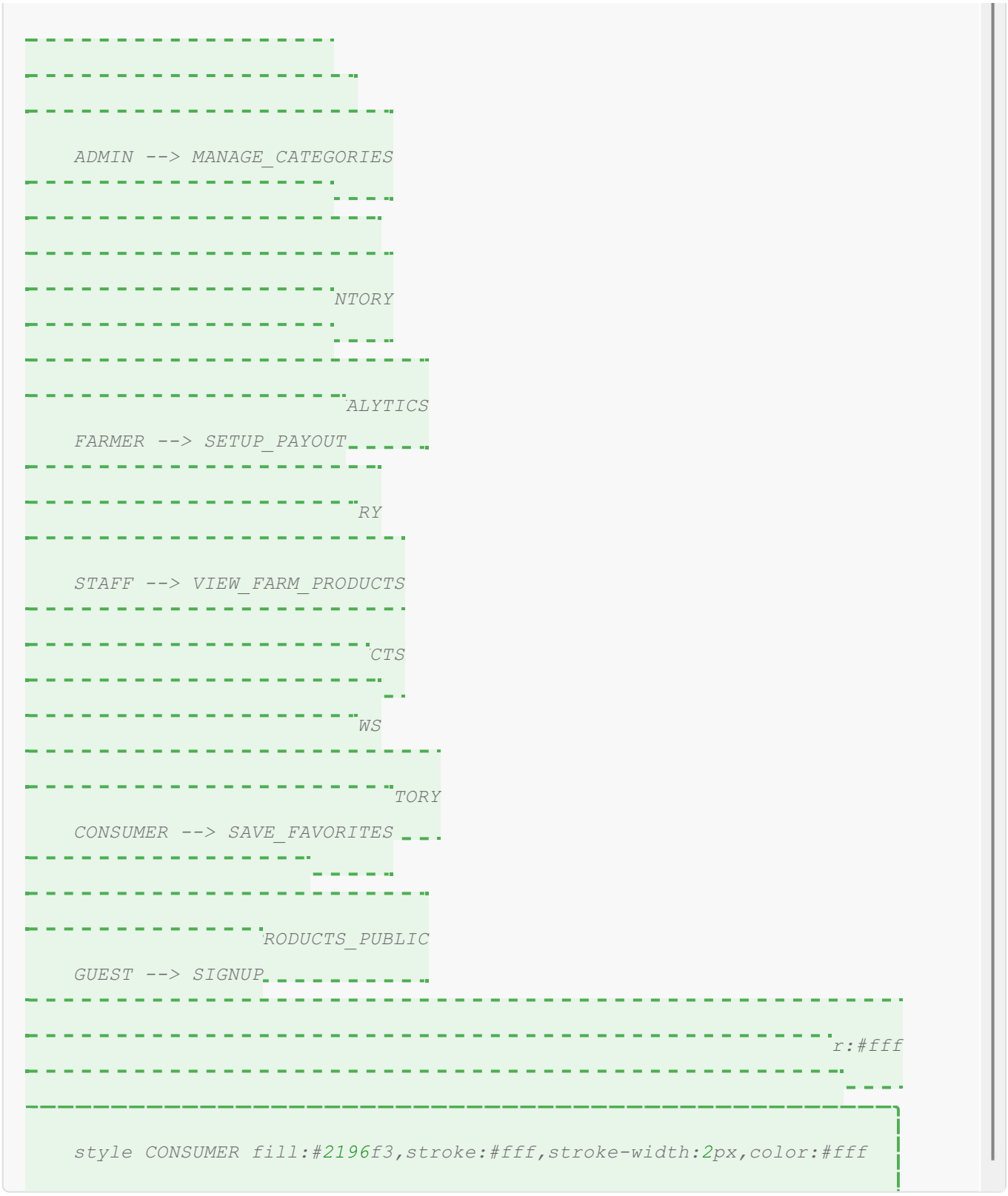



een
e for Token
en
User
nt
e
Browser->>User: Redirect to Dashboard
e
Middleware->>Redis: Validate Session Token
tails
Middleware->>Middleware: Check Permissions (RBAC)
ser Context
ed Resource
n
Browser->>User: Show Access Denied
rized
Browser->>User: Redirect to Login
end
t-request
API->>Database: Find User by Email



Role-Based Access Control (RBAC)

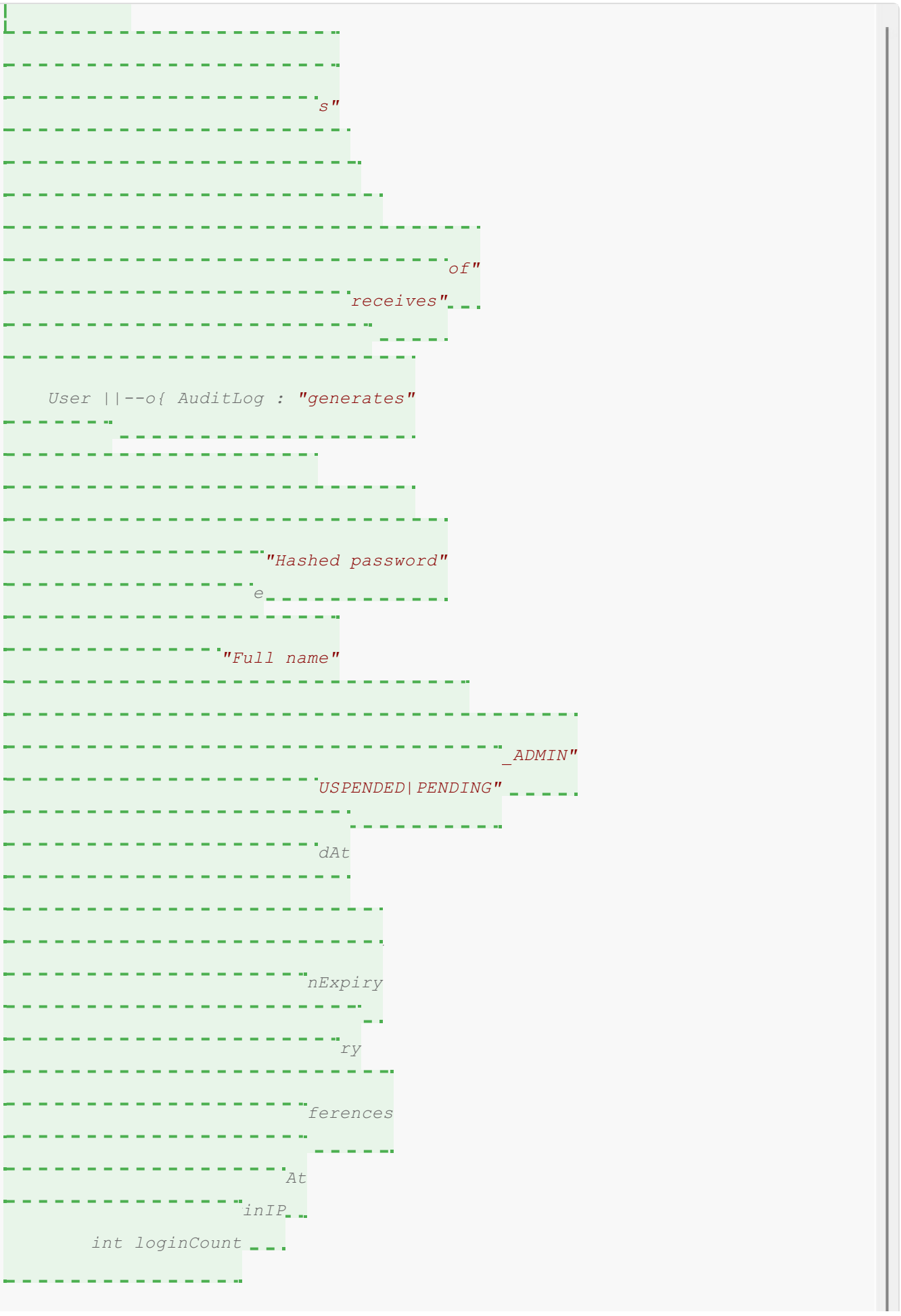

ory"]
ytics"]
SETUP_PAYOUT["Setup Stripe Payout"]
end
ry"]
VIEW_FARM_PRODUCTS["View Farm Products"]
end
ucts"]
istory"]
SAVE_FAVORITES["Save Favorites"]
end
LIC["View Products (Limited)"]
SIGNUP["Sign Up"]
end
S
S
SUPER_ADMIN --> VIEW_FINANCIALS

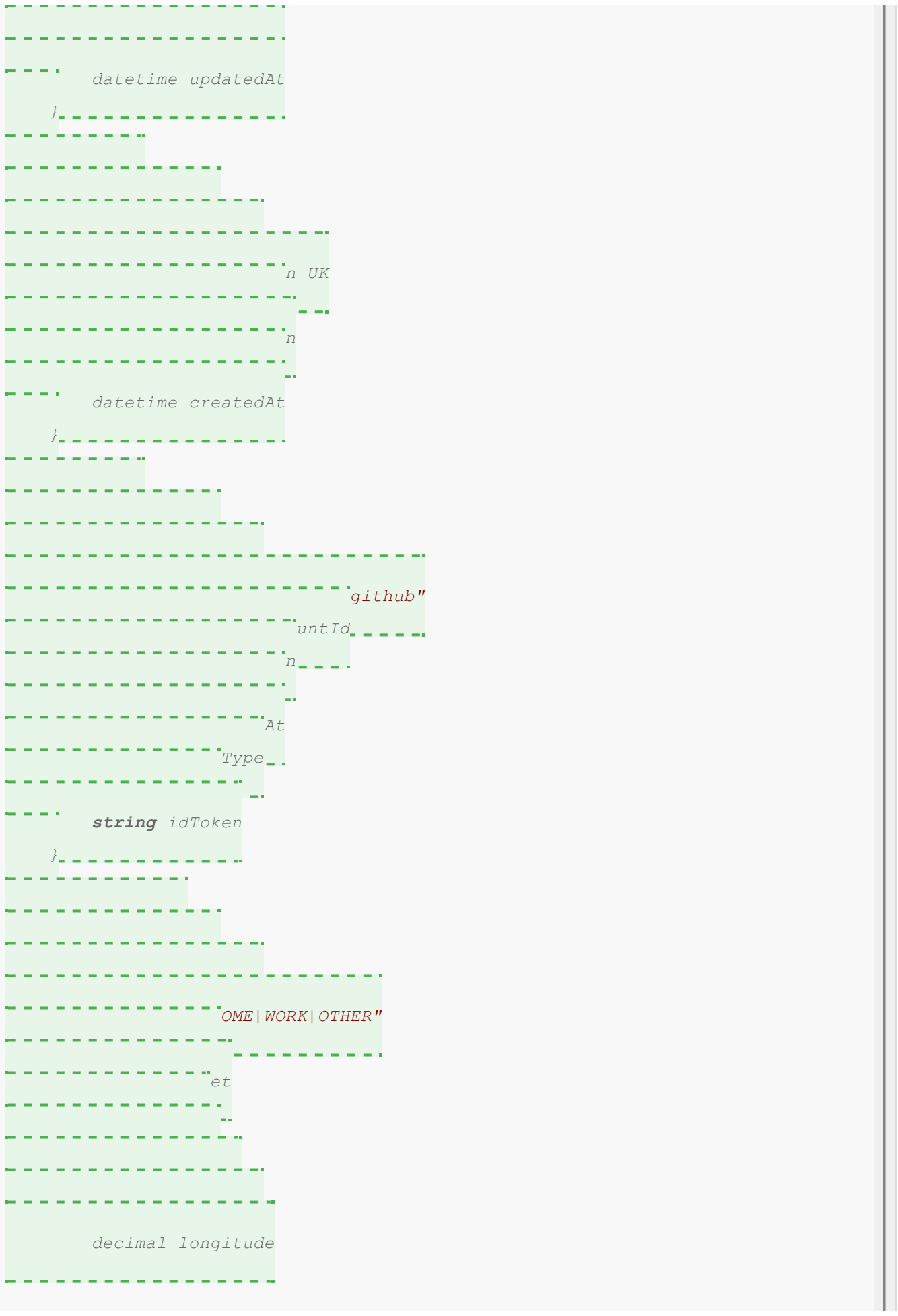


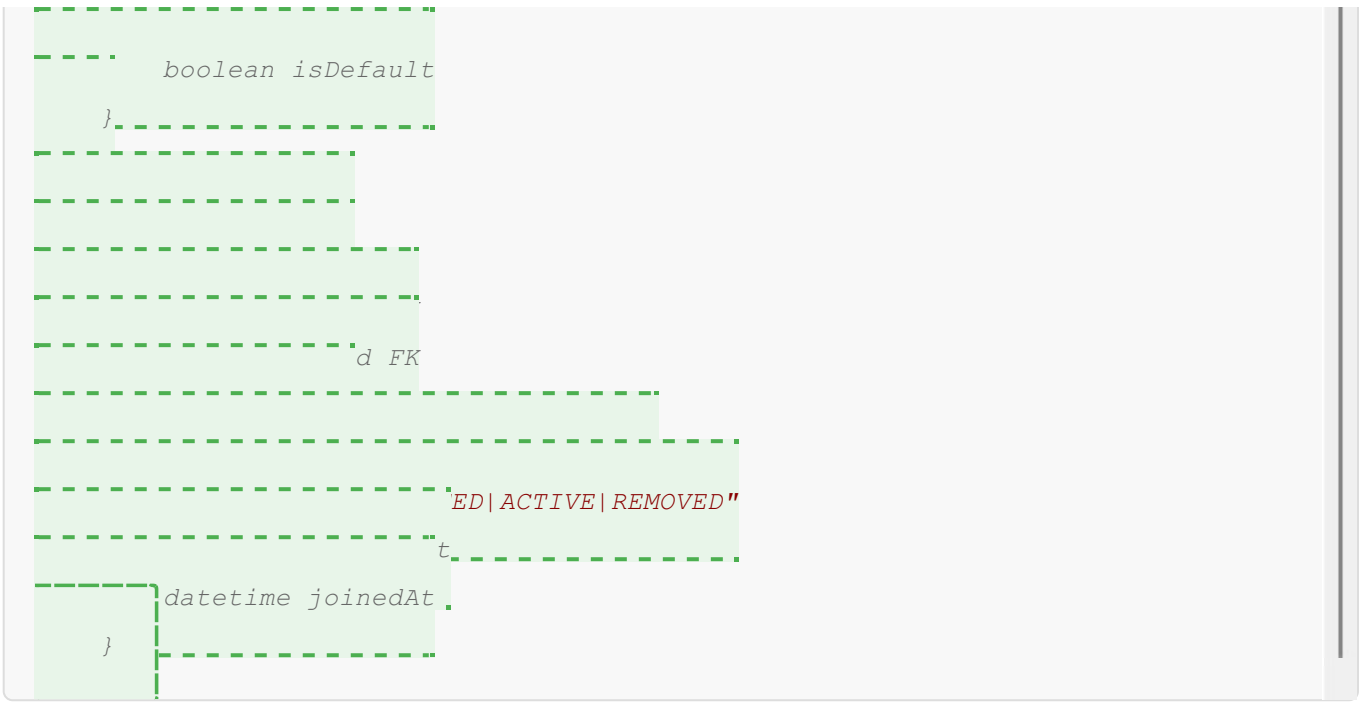
Middleware Auth Chain

👥 USER MANAGEMENT SYSTEM

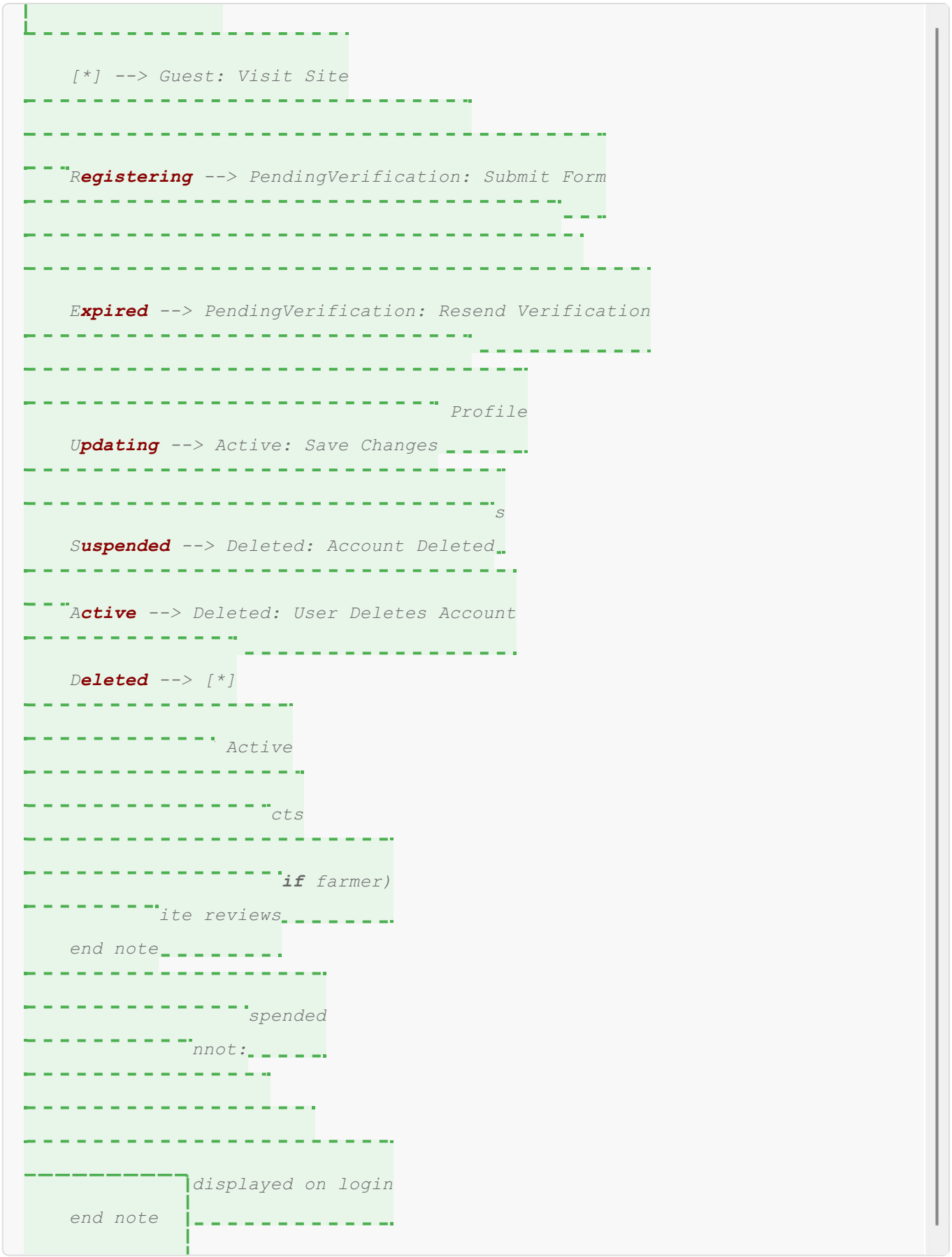
User Data Model


s"
of"
receives"
User ||--o{ AuditLog : "generates"
"Hashed password"
e
"Full name"
_ADMIN"
USPENDED|PENDING"
dAt
nExpiry
ry
ferences
At
inIP
int loginCount

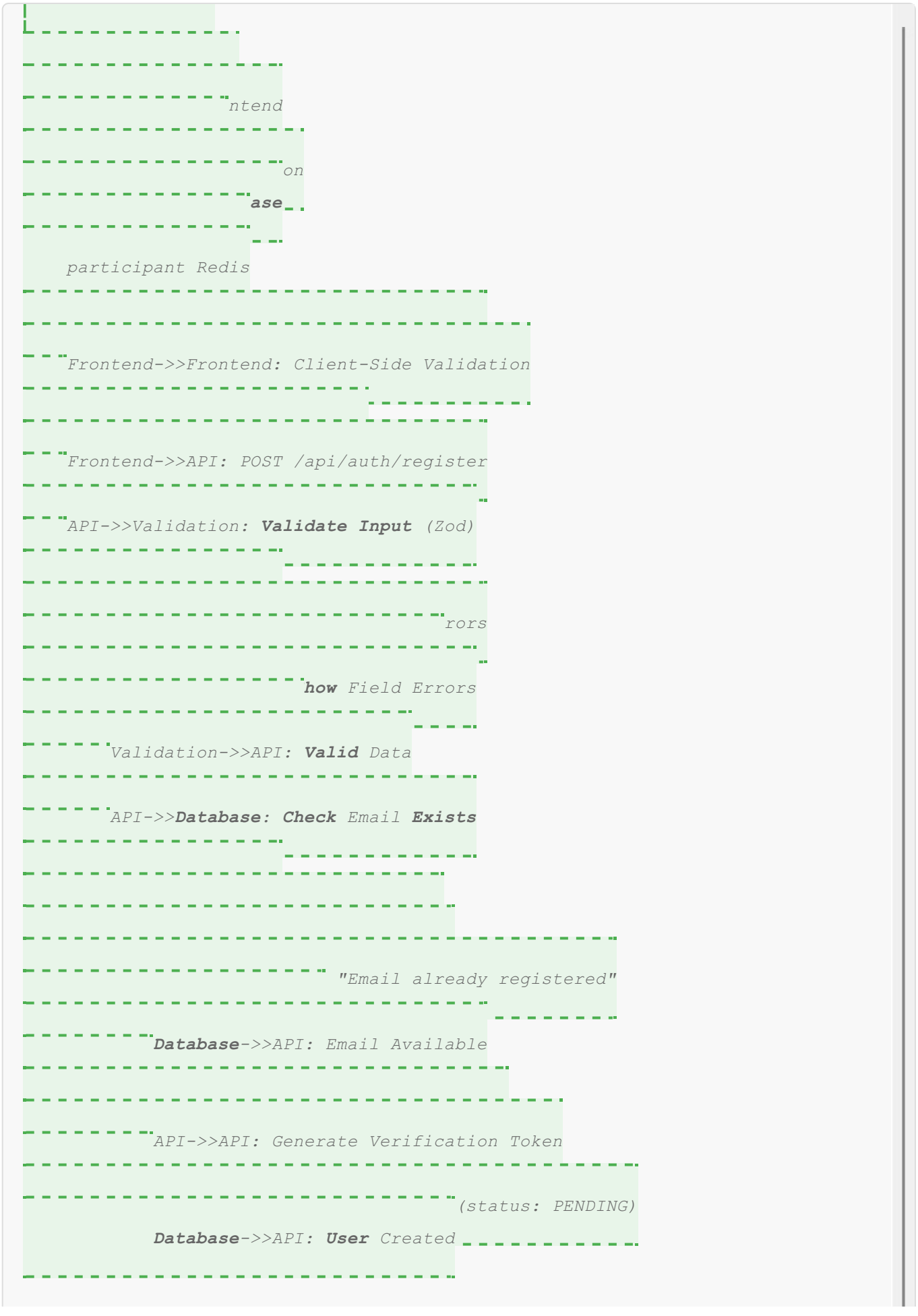

datetime updatedAt
}
n UK
n
datetime createdAt
}
github"
untId
n
At
Type
string idToken
}
OME|WORK|OTHER"
et
decimal longitude



User Lifecycle



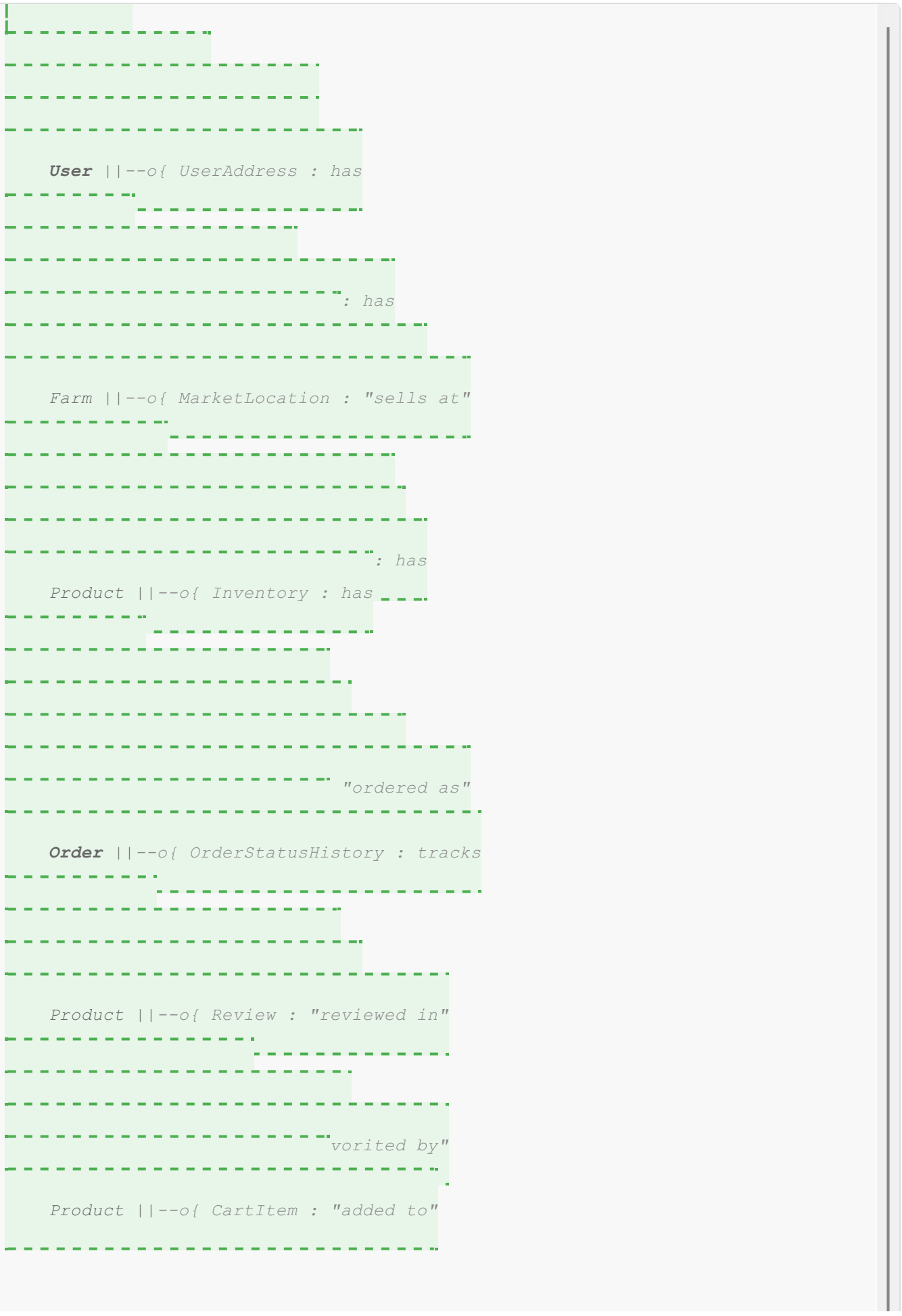
Registration Flow (Complete)

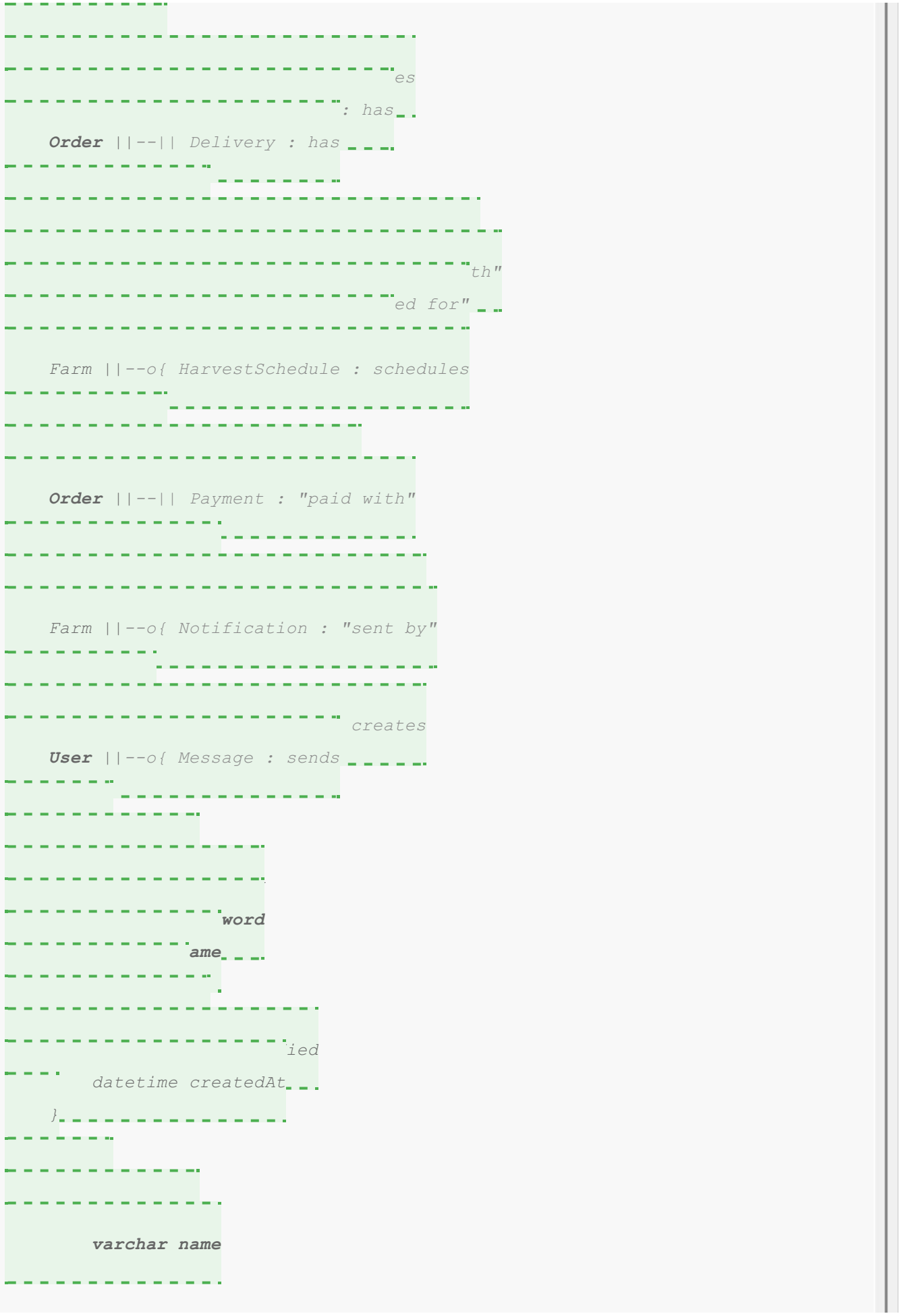

ntend
on
ase
participant Redis
Frontend->>Frontend: Client-Side Validation
Frontend->>API: POST /api/auth/register
API->>Validation: Validate Input (Zod)
rors
how Field Errors
Validation->>API: Valid Data
API->>Database: Check Email Exists
"Email already registered"
Database->>API: Email Available
API->>API: Generate Verification Token
(status: PENDING)
Database->>API: User Created

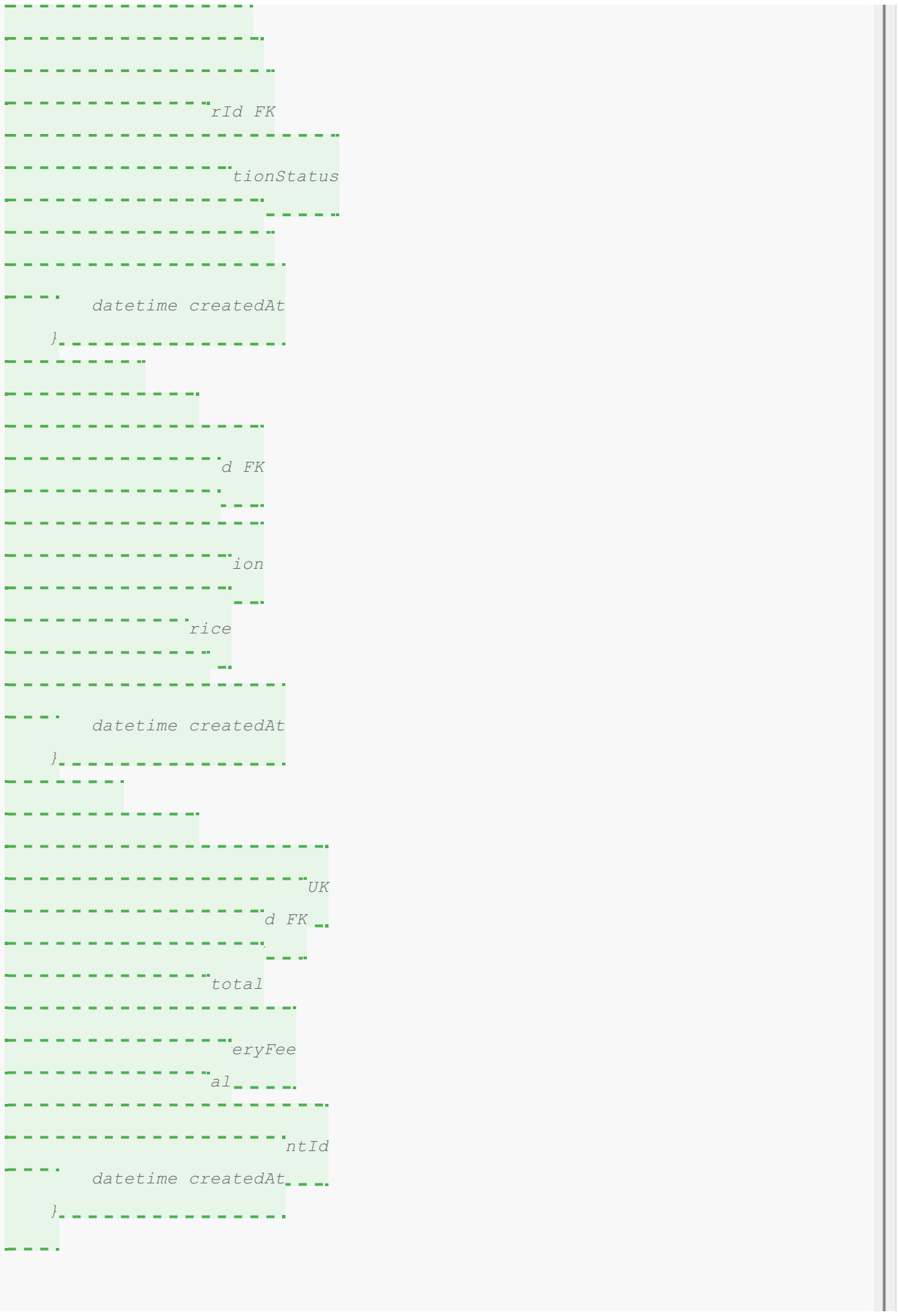
```
Email->>User: Verification Link Email
sion
API->>Frontend: 201 Created
Frontend->>User: "Check your email to verify"
Email->>API: GET /api/auth/verify?token=xxx
API->>Database: Find User by Token
API->>Database: Update User (emailVerified: true, status:
ification Token
l verified! Welcome!"
Frontend->>User: Show "Resend Verification" Button
end
end
```

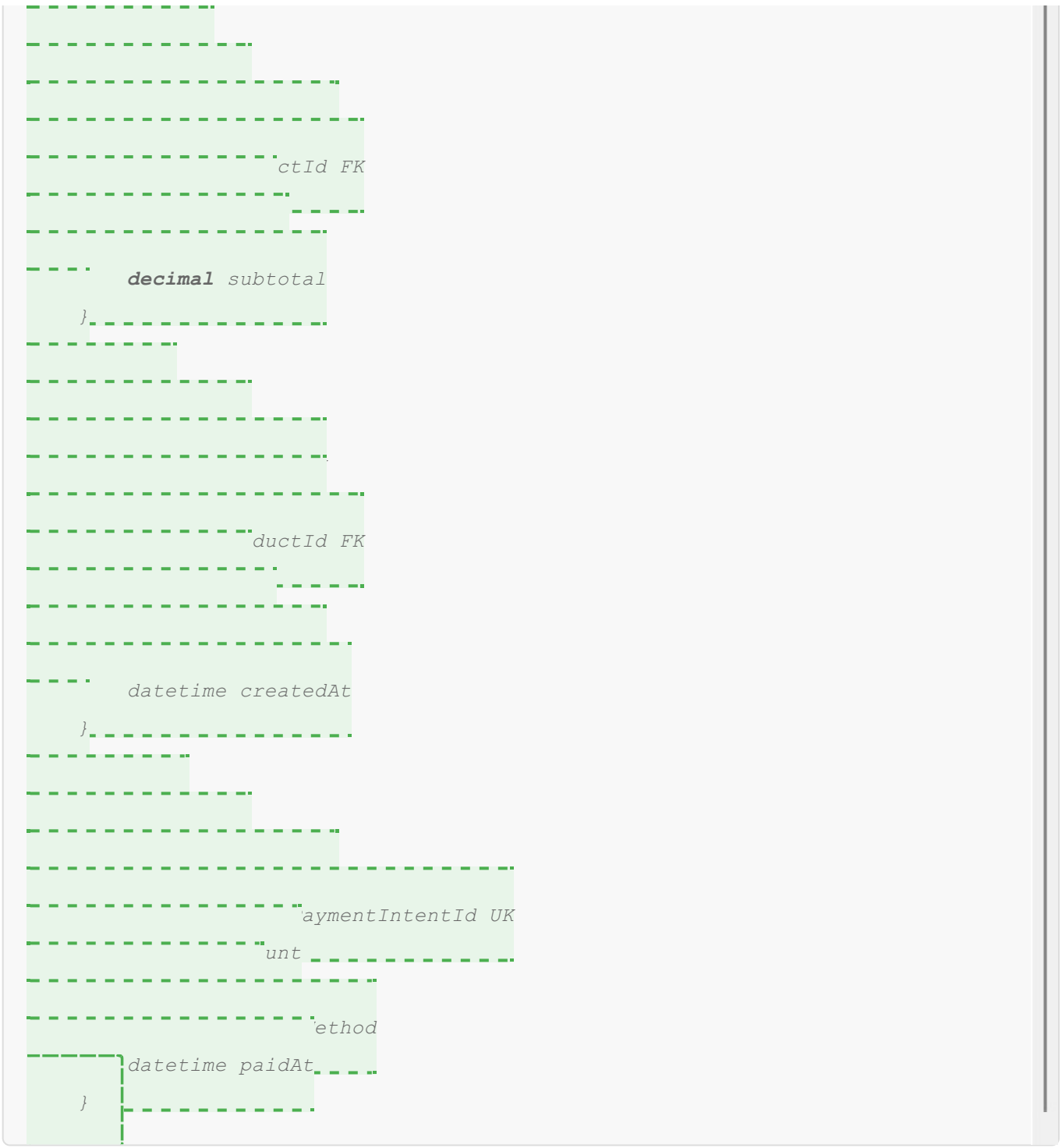
DATABASE SCHEMA & RELATIONSHIPS

Complete Entity Relationship Diagram


User ||--o{ UserAddress : has
: has
Farm ||--o{ MarketLocation : "sells at"
: has
Product ||--o{ Inventory : has
"ordered as"
Order ||--o{ OrderStatusHistory : tracks
Product ||--o{ Review : "reviewed in"
vorited by"
Product ||--o{ CartItem : "added to"


es
: has
Order ||--|| Delivery : has
th"
ed for"
Farm ||--o{ HarvestSchedule : schedules
Order ||--|| Payment : "paid with"
Farm ||--o{ Notification : "sent by"
creates
User ||--o{ Message : sends
word
ame
ied
datetime createdAt
}
varchar name


rId FK
tionStatus
datetime createdAt
}
d FK
ion
rice
datetime createdAt
}
UK
d FK
total
eryFee
al
ntId
datetime createdAt
}



Key Database Indexes

```sql
-- User Indexes
CREATE INDEX idx_users_email ON users(email);
CREATE INDEX idx_users_role_status ON users(role, status);
CREATE INDEX idx_users_created_at ON users(created_at);

-- Farm Indexes
CREATE INDEX idx_farms_owner_id ON farms(owner_id);
CREATE INDEX idx_farms_slug ON farms(slug);
CREATE INDEX idx_farms_status ON farms(status);
CREATE INDEX idx_farms_location ON farms(latitude, longitude);
CREATE INDEX idx_farms_verification_status ON farms(verification_status);

-- Product Indexes
CREATE INDEX idx_products_farm_id ON products(farm_id);
CREATE INDEX idx_products_slug ON products(slug);
CREATE INDEX idx_products_category ON products(category);
CREATE INDEX idx_products_status ON products(status);
CREATE INDEX idx_products_price ON products(price);

-- Order Indexes
CREATE INDEX idx_orders_customer_id ON orders(customer_id);
CREATE INDEX idx_orders_farm_id ON orders(farm_id);
CREATE INDEX idx_orders_status ON orders(status);
CREATE INDEX idx_orders_created_at ON orders(created_at);
CREATE INDEX idx_orders_order_number ON orders(order_number);

-- Session Indexes
CREATE INDEX idx_sessions_user_id ON sessions(user_id);
CREATE INDEX idx_sessions_token ON sessions(session_token);
CREATE INDEX idx_sessions_expires_at ON sessions(expires_at);
```

API ARCHITECTURE

API Route Structure

```
/api/
├── auth/                          # Authentication endpoints
│   ├── [...nextauth]/route.ts     # NextAuth handler
│   ├── register/route.ts          # User registration
│   ├── verify/route.ts            # Email verification
│   ├── reset-password/route.ts    # Password reset
│   └── logout/route.ts            # Logout endpoint
│
├── users/                         # User management
│   ├── route.ts                   # GET /users, POST /users
│   ├── [id]/route.ts              # GET /users/:id, PATCH /users/:id
│   ├── [id]/addresses/route.ts    # User addresses
│   └── profile/route.ts           # Current user profile
│
├── farms/                         # Farm management
│   ├── route.ts                   # GET /farms (list), POST /farms (create)
│   ├── [id]/route.ts              # GET, PATCH, DELETE /farms/:id
│   ├── [id]/products/route.ts     # Farm products
│   ├── [id]/team/route.ts         # Team management
│   ├── [id]/certifications/route.ts
│   └── [slug]/route.ts            # Get farm by slug
│
├── products/                      # Product management
│   ├── route.ts                   # GET /products (list), POST /products
│   ├── [id]/route.ts              # GET, PATCH, DELETE /products/:id
│   ├── [id]/inventory/route.ts    # Inventory management
│   ├── [id]/reviews/route.ts      # Product reviews
│   └── search/route.ts            # Product search
│
├── orders/                        # Order management
│   ├── route.ts                   # GET /orders (list), POST /orders
│   ├── [id]/route.ts              # GET /orders/:id, PATCH /orders/:id
│   ├── [id]/status/route.ts       # Update order status
│   └── [id]/cancel/route.ts       # Cancel order
│
├── cart/                          # Shopping cart
│   ├── route.ts                   # GET /cart, POST /cart (add item)
│   ├── [itemId]/route.ts          # PATCH, DELETE cart items
│   └── checkout/route.ts          # Checkout process
```

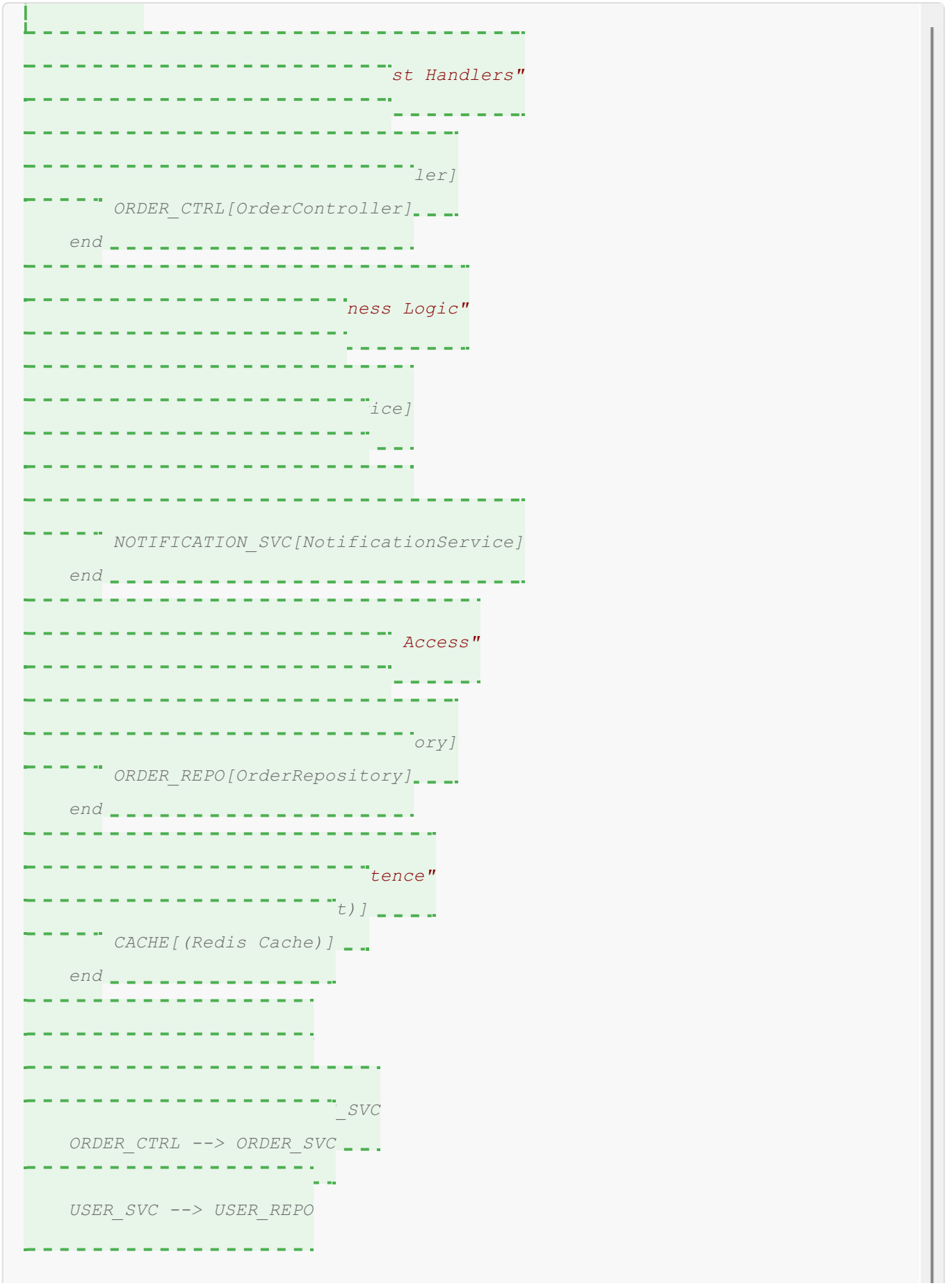
```
│
├── payments/                       # Payment processing
│   ├── create-intent/route.ts      # Create Stripe payment intent
│   ├── confirm/route.ts            # Confirm payment
│   └── webhooks/stripe/route.ts    # Stripe webhooks
│
├── reviews/                        # Reviews
│   ├── route.ts                    # POST /reviews (create)
│   ├── [id]/route.ts               # GET, PATCH, DELETE
│   └── farm/[farmId]/route.ts      # Farm reviews
│
├── favorites/                      # User favorites
│   ├── route.ts                    # GET, POST /favorites
│   └── [id]/route.ts               # DELETE favorite
│
├── search/                         # Search functionality
│   ├── farms/route.ts              # Search farms
│   ├── products/route.ts           # Search products
│   └── global/route.ts             # Global search
│
├── admin/                          # Admin endpoints
│   ├── users/route.ts              # User management
│   ├── farms/approve/route.ts      # Farm approval
│   ├── analytics/route.ts          # Platform analytics
│   └── audit-logs/route.ts         # Audit logs
│
└── webhooks/                       # External webhooks
    ├── stripe/route.ts             # Stripe events
    └── sendgrid/route.ts           # Email events
```

API Response Format (Standardized)

```
// Success Response
{
  "success": true,
  "data": {
    // Response payload
  },
  "meta": {
    "timestamp": "2024-12-20T10:30:00Z",
    "requestId": "req_abc123",
    "pagination": {
      "page": 1,
      "limit": 20,
      "total": 150,
      "totalPages": 8
    }
  }
}

// Error Response
{
  "success": false,
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "Invalid input data",
    "details": {
      "email": ["Email is required"],
      "password": ["Password must be at least 8 characters"]
    }
  },
  "meta": {
    "timestamp": "2024-12-20T10:30:00Z",
    "requestId": "req_abc123"
  }
}
```

⚙️ SERVICE LAYER ARCHITECTURE

Service Layer Pattern



```
    USER_SVC --> NOTIFICATION_SVC
    FARM_SVC --> NOTIFICATION_SVC
    PRODUCT_SVC --> PRODUCT_REPO
    ORDER_SVC --> NOTIFICATION_SVC
    ORDER_REPO --> PRISMA
    style CACHE fill:#DC382D,stroke:#fff,stroke-width:2px,color:#fff
```

Service Implementation Example

```
// UserService - Business Logic Layer
export class UserService {
  constructor(
    private userRepository: UserRepository,
    private emailService: EmailService,
    private notificationService: NotificationService
  ) {}

  /**
   * Create new user account
   */
```

```
async createUser(data: CreateUserRequest): Promise<User> {
  // 1. Validate input
  const validated = CreateUserSchema.parse(data);

  // 2. Check if email exists
  const existingUser = await this.userRepository.findByEmail(validated.email);
  if (existingUser) {
    throw new ConflictError("Email already registered");
  }

  // 3. Hash password
  const hashedPassword = await bcrypt.hash(validated.password, 10);

  // 4. Generate verification token
  const verificationToken = generateSecureToken();
  const verificationExpiry = addHours(new Date(), 24);

  // 5. Create user
  const user = await this.userRepository.create({
    ...validated,
    password: hashedPassword,
    verificationToken,
    verificationExpiry,
    status: "PENDING"
  });

  // 6. Send verification email (async, don't wait)
  this.emailService.sendVerificationEmail(user.email, verificationToken)
    .catch(error => logger.error("Failed to send verification email", error));

  // 7. Return user (without sensitive data)
  return this.sanitizeUser(user);
}

/**
 * Verify user email
 */
async verifyEmail(token: string): Promise<User> {
  const user = await this.userRepository.findByVerificationToken(token);
```

```
    if (!user) {
      throw new NotFoundError("Invalid verification token");
    }

    if (user.verificationExpiry < new Date()) {
      throw new BadRequestError("Verification token expired");
    }

    // Update user status
    const updatedUser = await this.userRepository.update(user.id, {
      emailVerified: true,
      emailVerifiedAt: new Date(),
      status: "ACTIVE",
      verificationToken: null,
      verificationExpiry: null
    });

    // Send welcome notification
    await this.notificationService.sendWelcomeNotification(updatedUser.id);

    return this.sanitizeUser(updatedUser);
  }

  /**
   * Remove sensitive fields from user object
   */
  private sanitizeUser(user: User): User {
    const { password, verificationToken, resetToken, ...safeUser } = user;
    return safeUser as User;
  }
}
```

🎨 FRONTEND ARCHITECTURE

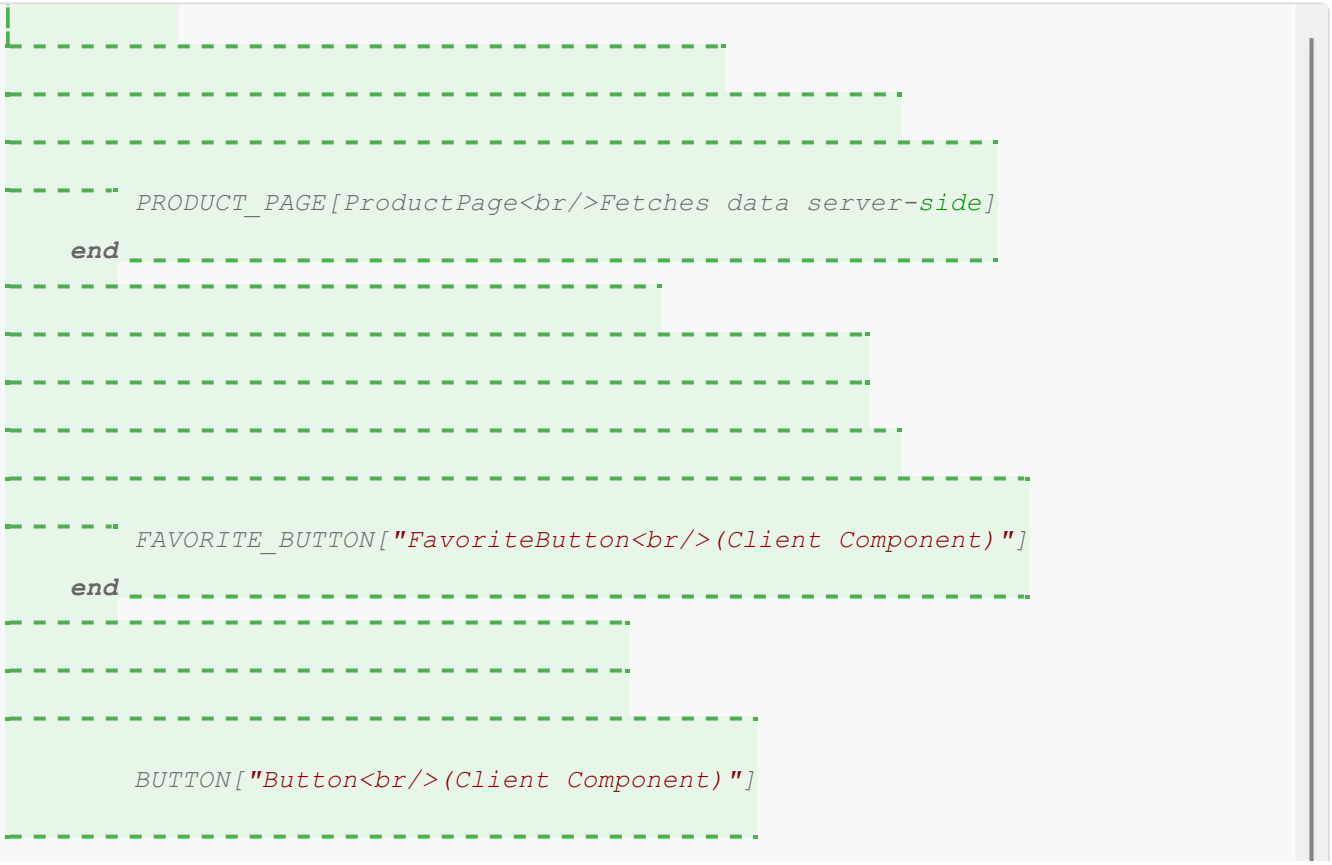
Component Hierarchy

```
src/
├── app/                          # Next.js 15 App Router
│   ├── layout.tsx                # Root layout (auth provider, theme)
│   ├── page.tsx                  # Homepage
│   ├── (auth)/                   # Auth route group
│   │   ├── layout.tsx            # Auth layout
│   │   ├── login/page.tsx
│   │   ├── register/page.tsx
│   │   └── reset-password/page.tsx
│   │
│   ├── (customer)/               # Customer routes
│   │   ├── layout.tsx            # Customer layout (with nav)
│   │   ├── farms/
│   │   │   ├── page.tsx          # Farm listing
│   │   │   └── [slug]/page.tsx   # Farm detail
│   │   ├── products/
│   │   │   ├── page.tsx          # Product listing
│   │   │   └── [slug]/page.tsx   # Product detail
│   │   ├── cart/page.tsx
│   │   ├── checkout/page.tsx
│   │   └── orders/
│   │       ├── page.tsx          # Order history
│   │       └── [id]/page.tsx     # Order detail
│   │
│   ├── (farmer)/                 # Farmer routes
│   │   ├── layout.tsx            # Farmer dashboard layout
│   │   ├── dashboard/page.tsx
│   │   ├── farm/
│   │   │   ├── setup/page.tsx
│   │   │   ├── edit/page.tsx
│   │   │   └── team/page.tsx
│   │   ├── products/
│   │   │   ├── page.tsx          # Product management
│   │   │   ├── new/page.tsx
│   │   │   └── [id]/edit/page.tsx
│   │   ├── orders/page.tsx
│   │   ├── analytics/page.tsx
│   │   └── payouts/page.tsx
│   │
```

```
│   └── (admin)/                  # Admin routes
│       ├── layout.tsx            # Admin layout
│       ├── dashboard/page.tsx
│       ├── users/page.tsx
│       ├── farms/
│       │   ├── page.tsx          # Pending approvals
│       │   └── [id]/review/page.tsx
│       └── analytics/page.tsx
│
├── components/
│   ├── ui/                       # Base UI components
│   │   ├── Button.tsx
│   │   ├── Card.tsx
│   │   ├── Input.tsx
│   │   ├── Modal.tsx
│   │   ├── Badge.tsx
│   │   └── Spinner.tsx
│   │
│   ├── features/                 # Feature-specific components
│   │   ├── auth/
│   │   │   ├── LoginForm.tsx
│   │   │   ├── RegisterForm.tsx
│   │   │   └── PasswordResetForm.tsx
│   │   ├── farms/
│   │   │   ├── FarmCard.tsx
│   │   │   ├── FarmGrid.tsx
│   │   │   ├── FarmProfile.tsx
│   │   │   └── FarmEditForm.tsx
│   │   ├── products/
│   │   │   ├── ProductCard.tsx
│   │   │   ├── ProductGrid.tsx
│   │   │   ├── ProductDetail.tsx
│   │   │   └── ProductForm.tsx
│   │   ├── cart/
│   │   │   ├── CartDrawer.tsx
│   │   │   ├── CartItem.tsx
│   │   │   └── CartSummary.tsx
│   │   └── orders/
│   │       ├── OrderList.tsx
│   │       ├── OrderCard.tsx
```

```
│   │       └── OrderTimeline.tsx
│   │
│   └── layout/                    # Layout components
│       ├── Header.tsx
│       ├── Footer.tsx
│       ├── Sidebar.tsx
│       └── Navigation.tsx
│
├── hooks/                         # Custom React hooks
│   ├── useAuth.ts                 # Authentication hook
│   ├── useCart.ts                 # Shopping cart hook
│   ├── useFarms.ts                # Farm data hook
│   ├── useProducts.ts             # Product data hook
│   └── useOrders.ts               # Order data hook
│
└── context/                       # React Context providers
    ├── AuthContext.tsx            # Auth state
    ├── CartContext.tsx            # Cart state
    └── ThemeContext.tsx           # Theme state
```

Component Communication Pattern

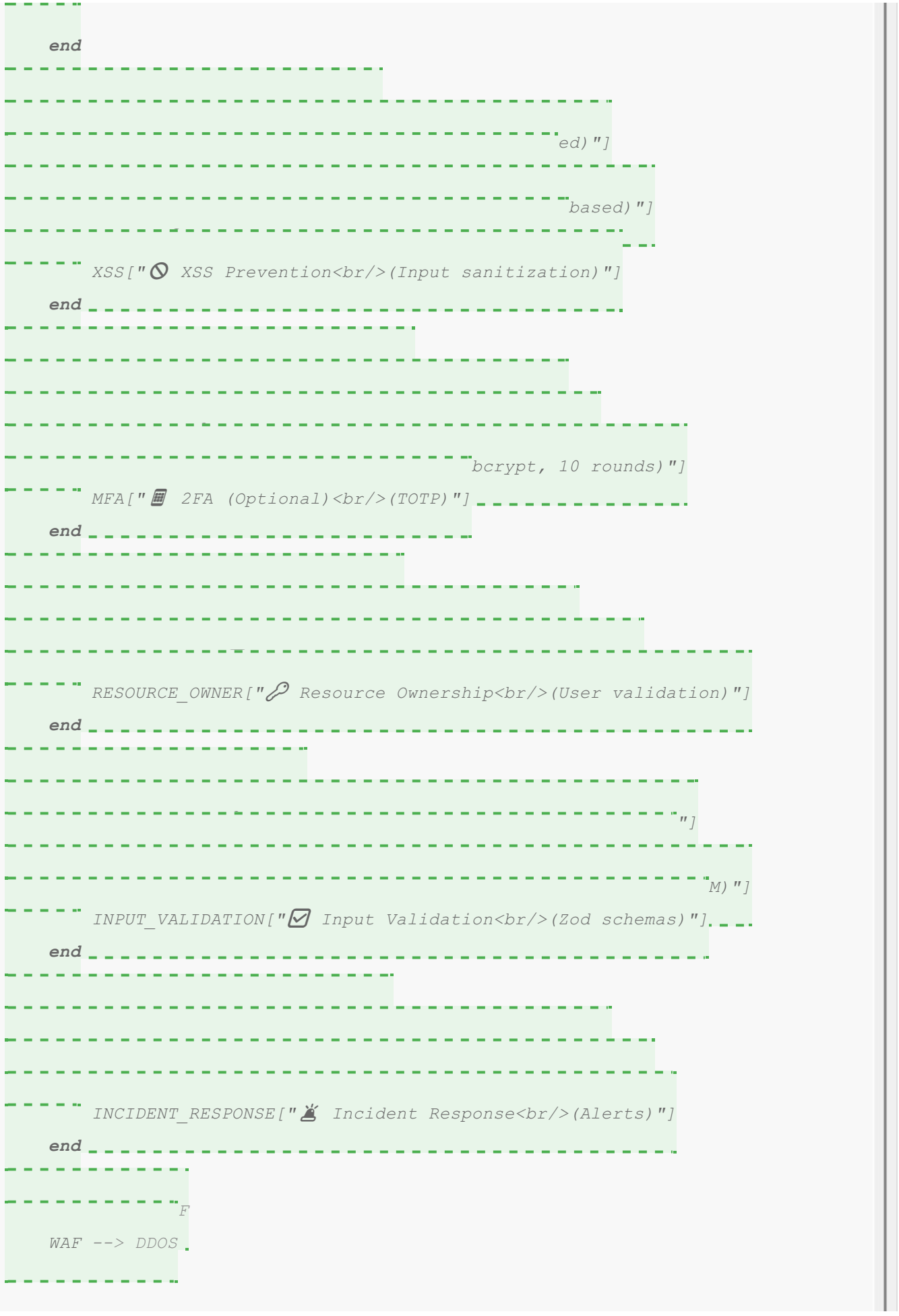


```
    BADGE[Badge<br/>Server Component]
  end

  AUTH_CONTEXT["AuthContext<br/>(Client)"]
  end

  PRODUCT_PAGE --> PRODUCT_GRID
  FARM_PROFILE --> FAVORITE_BUTTON
  PRODUCT_GRID --> CART_BUTTON
  CARD --> BADGE
  FAVORITE_BUTTON --> AUTH_CONTEXT

  style CART_CONTEXT fill:#2196F3,stroke:#fff,stroke-width:2px,color:#fff
```

SECURITY ARCHITECTURE

Security Layers

```
    WAF)"]
    DDOS["⚡ DDoS Protection<br/>(Vercel Edge)"]
```


end
ed)"]
based)"]
XSS["🚫 XSS Prevention
(Input sanitization)"]
end
bcrypt, 10 rounds)"]
MFA["📱 2FA (Optional)
(TOTP)"]
end
RESOURCE_OWNER["🔑 Resource Ownership
(User validation)"]
end
"]
M)"]
INPUT_VALIDATION["☑ Input Validation
(Zod schemas)"]
end
INCIDENT_RESPONSE["🚨 Incident Response
(Alerts)"]
end
F
WAF --> DDOS

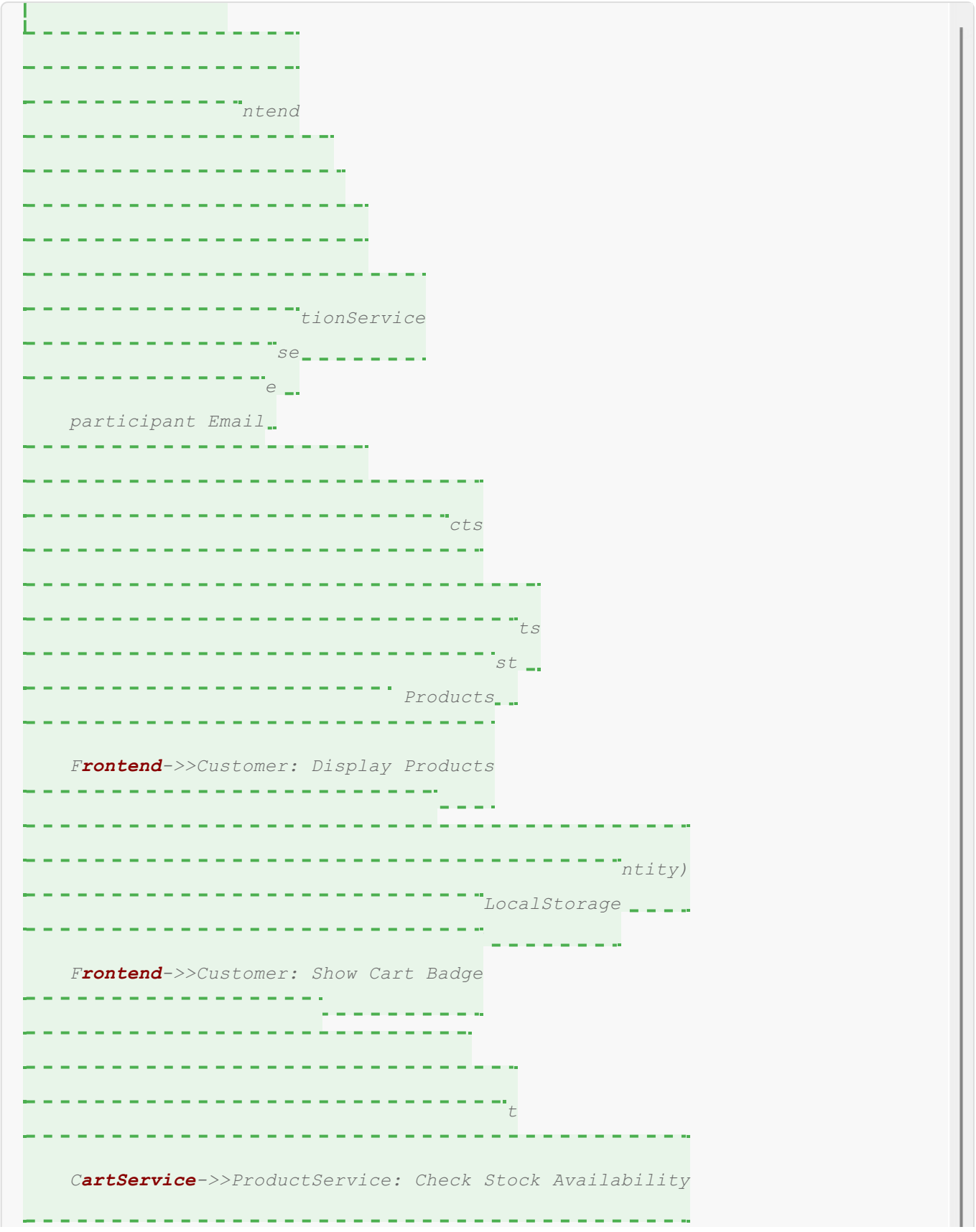
Security Headers Configuration

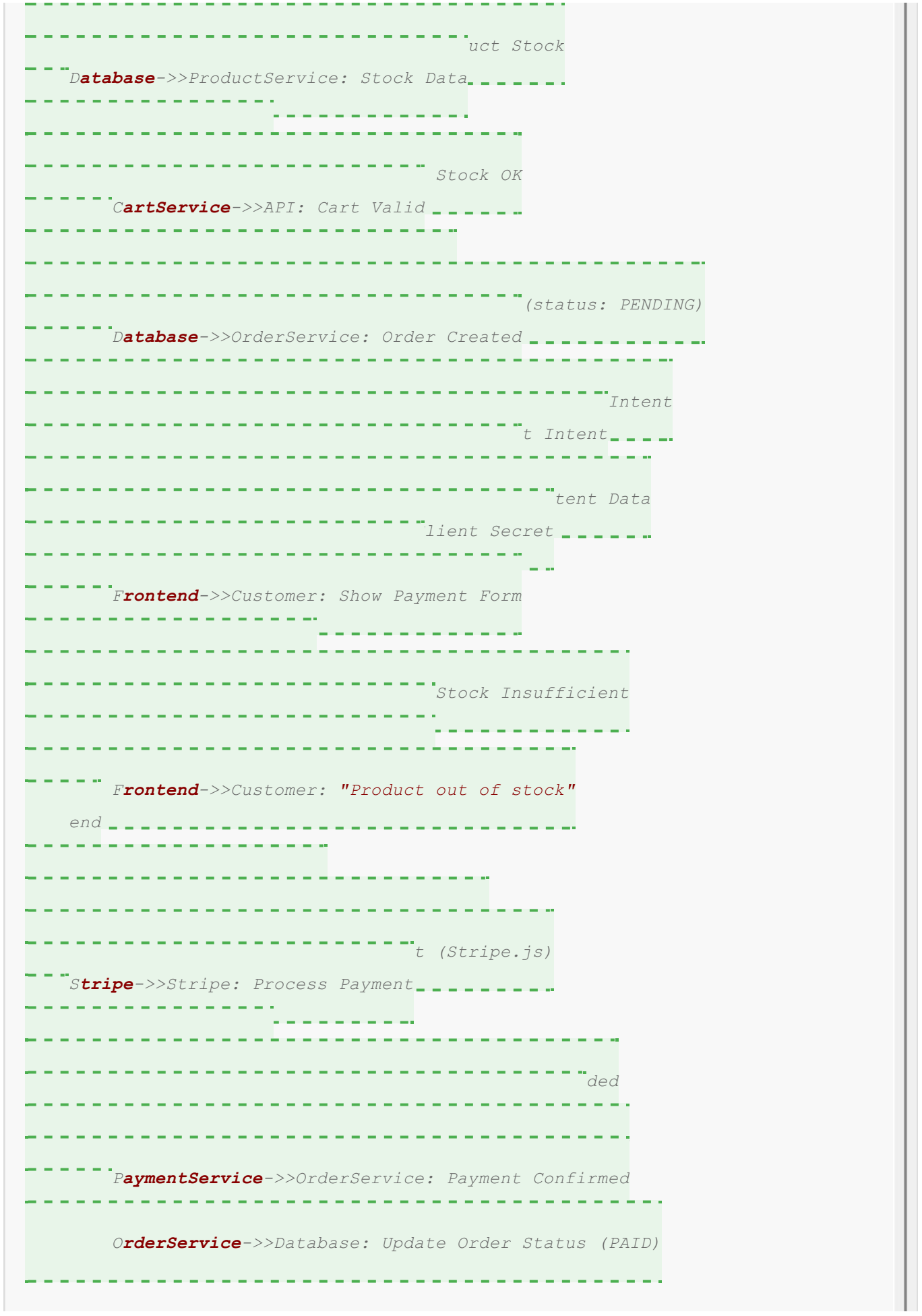
```
// next.config.mjs
const securityHeaders = [
  {
    key: 'X-DNS-Prefetch-Control',
    value: 'on'
  },
  {
```

```
    key: 'Strict-Transport-Security',
    value: 'max-age=63072000; includeSubDomains; preload'
  },
  {
    key: 'X-Frame-Options',
    value: 'SAMEORIGIN'
  },
  {
    key: 'X-Content-Type-Options',
    value: 'nosniff'
  },
  {
    key: 'X-XSS-Protection',
    value: '1; mode=block'
  },
  {
    key: 'Referrer-Policy',
    value: 'strict-origin-when-cross-origin'
  },
  {
    key: 'Permissions-Policy',
    value: 'camera=(), microphone=(), geolocation=(self)'
  },
  {
    key: 'Content-Security-Policy',
    value: `
      default-src 'self';
      script-src 'self' 'unsafe-eval' 'unsafe-inline' *.vercel.app;
      style-src 'self' 'unsafe-inline';
      img-src 'self' data: https:;
      font-src 'self' data:;
      connect-src 'self' *.stripe.com;
      frame-src 'self' *.stripe.com;
    `.replace(/\s{2,}/g, ' ').trim()
  }
];
```

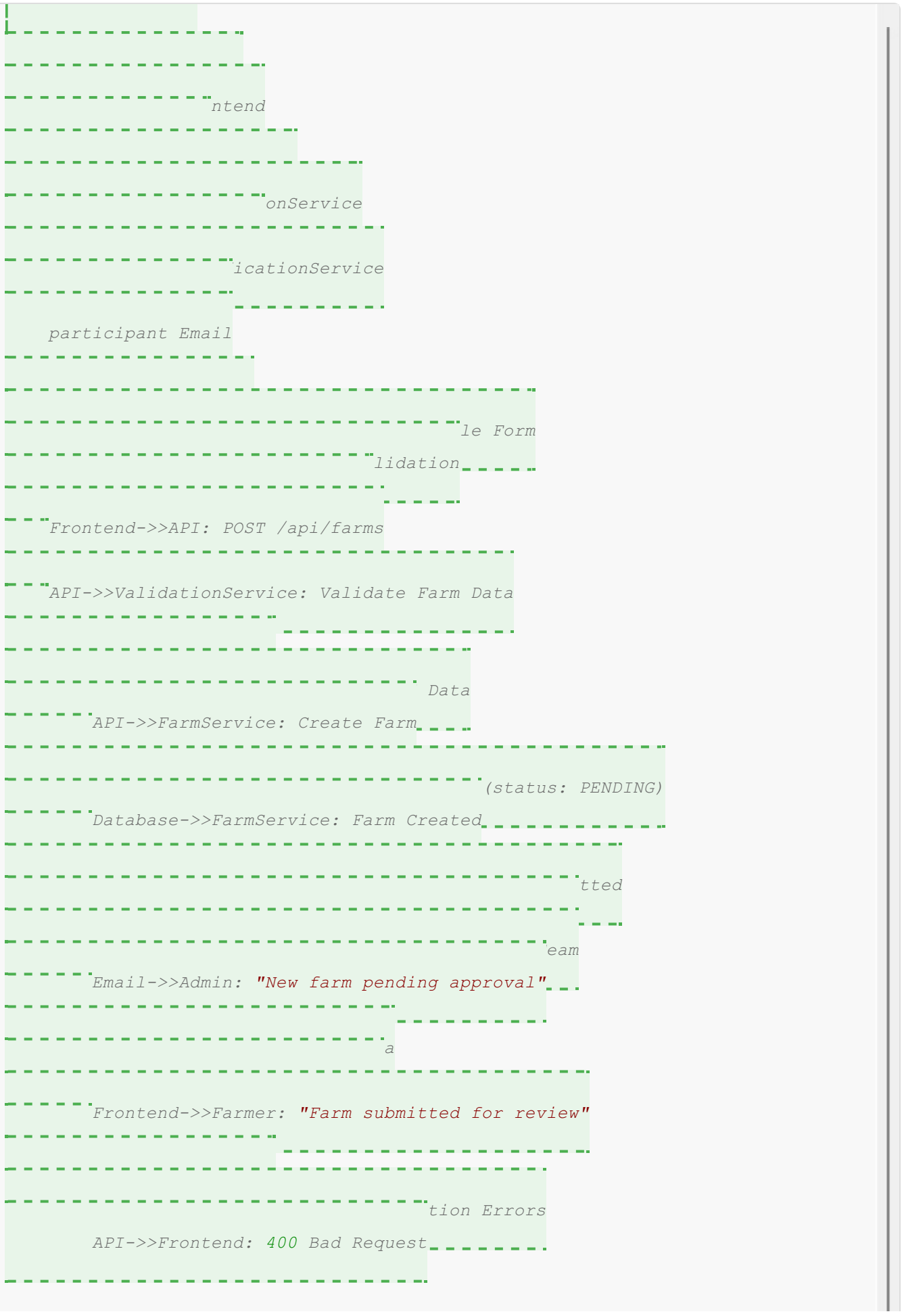
DATA FLOW DIAGRAMS

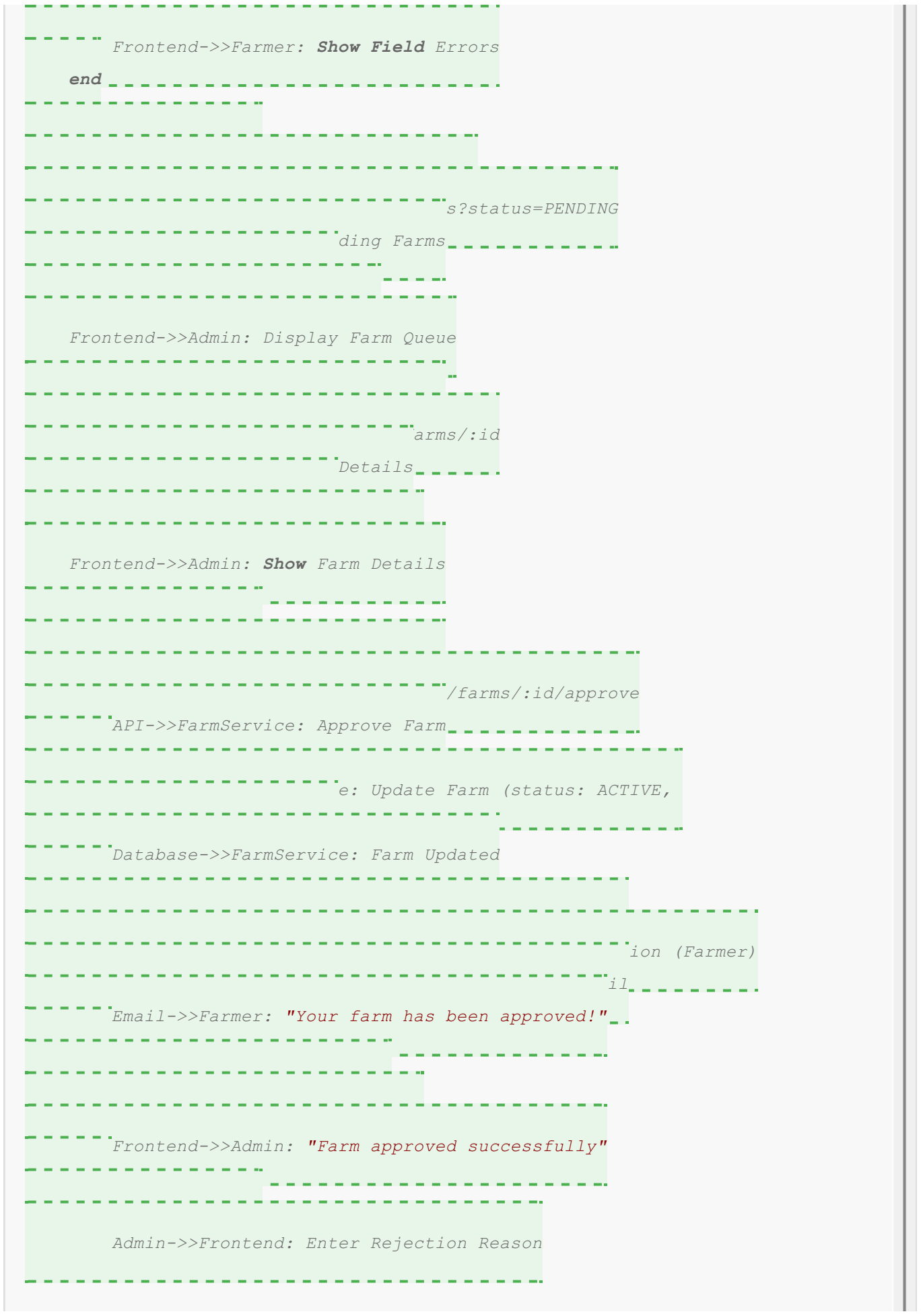
Complete Order Flow (Customer Journey)

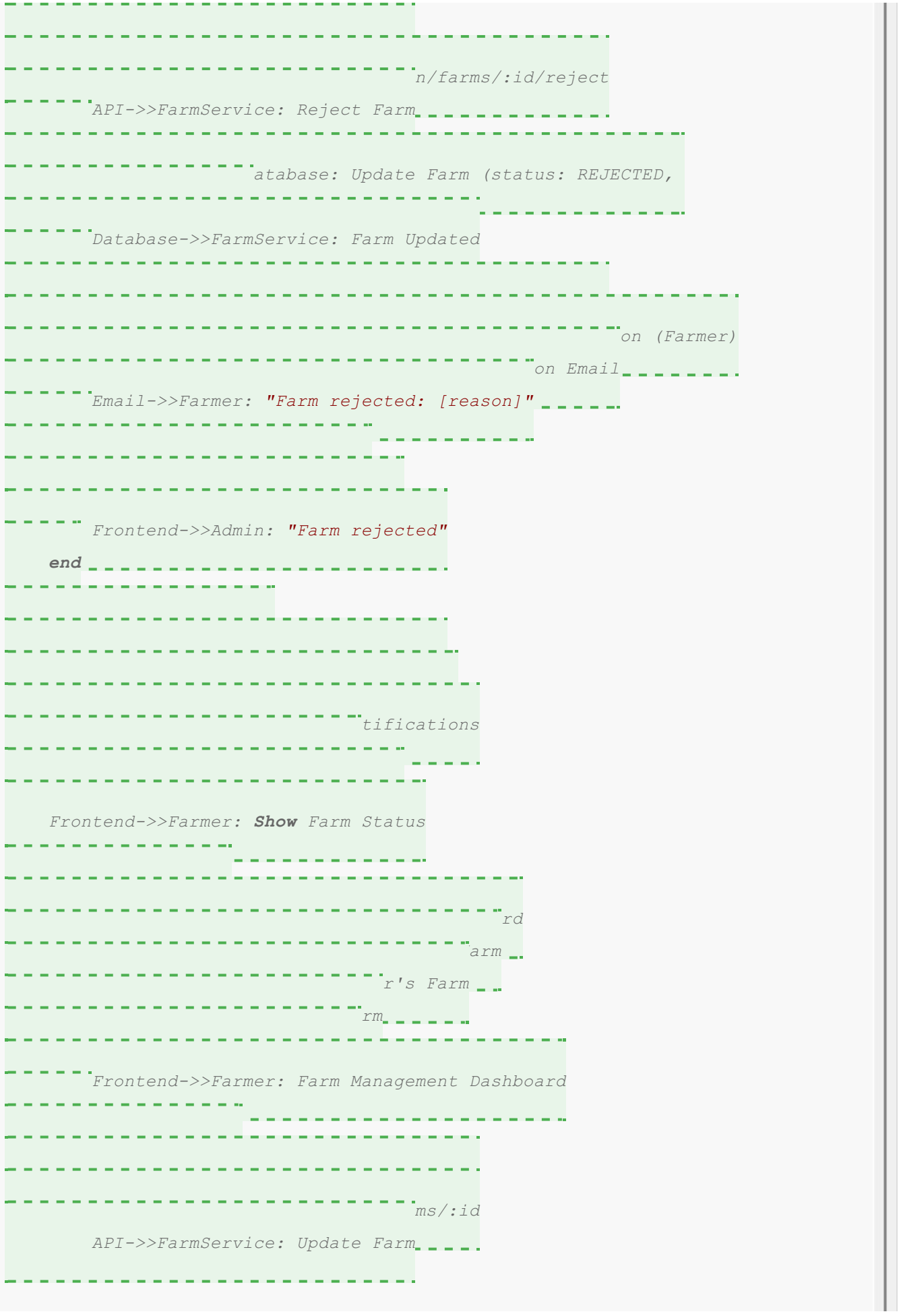



uct Stock
Database->>ProductService: Stock Data
Stock OK
CartService->>API: Cart Valid
(status: PENDING)
Database->>OrderService: Order Created
Intent
t Intent
tent Data
lient Secret
Frontend->>Customer: Show Payment Form
Stock Insufficient
Frontend->>Customer: "Product out of stock"
end
t (Stripe.js)
Stripe->>Stripe: Process Payment
ded
PaymentService->>OrderService: Payment Confirmed
OrderService->>Database: Update Order Status (PAID)

Farm Registration & Verification Flow


ntend
onService
icationService
participant Email
le Form
lidation
Frontend->>API: POST /api/farms
API->>ValidationService: Validate Farm Data
Data
API->>FarmService: Create Farm
(status: PENDING)
Database->>FarmService: Farm Created
tted
eam
Email->>Admin: "New farm pending approval"
a
Frontend->>Farmer: "Farm submitted for review"
tion Errors
API->>Frontend: 400 Bad Request


Frontend->>Farmer: Show Field Errors
end
s?status=PENDING
ding Farms
Frontend->>Admin: Display Farm Queue
arms/:id
Details
Frontend->>Admin: Show Farm Details
/farms/:id/approve
API->>FarmService: Approve Farm
e: Update Farm (status: ACTIVE,
Database->>FarmService: Farm Updated
ion (Farmer)
il
Email->>Farmer: "Your farm has been approved!"
Frontend->>Admin: "Farm approved successfully"
Admin->>Frontend: Enter Rejection Reason


n/farms/:id/reject
API->>FarmService: Reject Farm
atabase: Update Farm (status: REJECTED,
Database->>FarmService: Farm Updated
on (Farmer)
on Email
Email->>Farmer: "Farm rejected: [reason]"
Frontend->>Admin: "Farm rejected"
end
tifications
Frontend->>Farmer: Show Farm Status
rd
arm
r's Farm
rm
Frontend->>Farmer: Farm Management Dashboard
ms/:id
API->>FarmService: Update Farm

```
                                    ubmit (status: PENDING)
                                                    ins (Resubmission)
                                                    m
        ail->>Admin: "Farm resubmitted for review"
    end
```

🎯 ARCHITECTURAL PRINCIPLES

1. Separation of Concerns

- **Presentation Layer**: UI components, pages, layouts
- **Business Logic Layer**: Services, validation, business rules
- **Data Access Layer**: Repositories, ORM, caching
- **Clear boundaries** between layers

2. Type Safety First

- **100% TypeScript** with strict mode
- **Zod validation** for runtime type checking
- **Prisma** for type-safe database access
- **No** `any` **types** allowed in production code

3. Scalability by Design

- **Stateless architecture** (scales horizontally)
- **Multi-layer caching** (Memory → Redis → Database)
- **Edge-first delivery** (Vercel global CDN)
- **Database connection pooling**

4. Security by Default

- **All routes authenticated** by default (opt-out for public)
- **HTTPS only** (no HTTP)

- **Security headers** on all responses
- **Input validation** on all endpoints
- **RBAC** for authorization

5. Agricultural Consciousness

- **Domain-driven design** (farming terminology)
- **Seasonal awareness** in business logic
- **Biodynamic patterns** in data models
- **Sustainability tracking**

6. Performance Optimization

- **Server-side rendering** for SEO
- **Static generation** for content pages
- **Code splitting** for smaller bundles
- **Image optimization** (Next.js Image)
- **Database query optimization** (indexes, select specific fields)

7. Observability & Monitoring

- **Comprehensive error tracking** (Sentry)
- **Distributed tracing** (OpenTelemetry)
- **Performance monitoring** (Vercel Analytics)
- **Uptime monitoring** (UptimeRobot)
- **Audit logging** (all critical actions)

8. Developer Experience

- **Clear code organization**
- **Consistent naming conventions**
- **Comprehensive documentation**
- **Automated testing** (unit, integration, e2e)
- **CI/CD pipeline** (automated deployments)

PERFORMANCE TARGETS

```
Load Times:
  Homepage (First Load): < 2.0 seconds
  Page Transitions: < 500ms
  API Response Time: < 200ms (average)
  Database Query Time: < 50ms (average)
  Time to Interactive (TTI): < 3.5 seconds

Throughput:
  Concurrent Users: 10,000+
  Requests per Second: 1,000+
  Database Connections: 100 (pooled)
  Redis Operations: 10,000+ ops/sec

Reliability:
  Uptime: 99.9%+
  Error Rate: < 0.1%
  Cache Hit Rate: > 80%
  Mean Time to Recovery (MTTR): < 5 minutes

Web Vitals (Core Web Vitals):
  Largest Contentful Paint (LCP): < 2.5 seconds
  First Input Delay (FID): < 100ms
  Cumulative Layout Shift (CLS): < 0.1

Security:
  SSL Labs Rating: A+
  Security Headers Score: A+
  OWASP Top 10: All addressed
  Vulnerability Scan: Weekly
```

DEPLOYMENT ARCHITECTURE


ocalhost:3000)"]
GIT["📦 Git Repository
(GitHub)"]
end
ed Tests)"]
]
],
LINT["🔍 Linting
(ESLint + TypeScript)"]
end
(Vercel Preview)"]
STAGING_DB[("🐘 Staging Database")]
end
nt
(Vercel)"]
DIS[("⚡ Production Redis")]
PROD