# **Technical Specification Document: strideelectronics.com**

## **1. Overview**

This document outlines the technical architecture and implementation details for **strideelectronics.com**, an e-commerce platform built on AWS using a Django backend, MongoDB as the database, and a microservices architecture. All services will follow secure, authenticated communication using HTTPS and JWT-based token verification.

## **2. Technology Stack**

| **Layer** | **Technology** |
| --- | --- |
| Frontend | React.js + Tailwind CSS |
| Backend API | Django (REST Framework) |
| Database | MongoDB (Managed via Atlas or self-hosted on EC2) |
| Auth & Tokens | JWT (stored & rotated securely) |
| Infrastructure | AWS (EC2, ALB, S3, RDS, IAM, Secrets Manager, VPC, etc.) |
| DevOps | Docker, GitHub Actions, Terraform/CloudFormation |
| Communication | HTTPS via AWS ACM |
| Monitoring | AWS CloudWatch, ELK Stack |

## **3. Updated Infrastructure Overview: AWS EKS Deployment**

### **3.1. Core Components**

* **Amazon EKS Cluster**: Manages the Kubernetes control plane.
* **Node Groups**:  
  + **Frontend Node Group**: Hosts the React.js frontend application.
  + **Backend Node Group**: Runs Django REST API services.
  + **Microservices Node Group**: Hosts containerized microservices (e.g., Payments, Orders).
* **AWS Load Balancer Controller**: Manages ingress resources and provisions Application Load Balancers (ALBs).
* **Amazon RDS (MongoDB)**: Managed MongoDB instances with TLS encryption.
* **Amazon S3**: Stores static assets and user-uploaded content.
* **Amazon ElastiCache (Redis)**: Handles session storage and caching.
* **AWS Secrets Manager**: Secures sensitive information like JWT secrets and database credentials.

## **4. JWT Authentication**

### **Token Generation**

* Token generated at login/signup with user ID and role  
  {

"user\_id": "abc123",

"role": "buyer",

"iat": 1720085294,

"exp": 1720092494

}

* Signed using a secret key stored in **AWS Secrets Manager**.
* Expiry: 2 hours by default; refresh token for 30 days.

### 

### **Token Verification**

* Middleware in Django checks Authorization: Bearer <token>
* If invalid/expired/malformed, throws:  
  + 401 Unauthorized (Token expired)
  + 403 Forbidden (Invalid token/role mismatch)
* Decoded token checked against user record from DB

### **Refresh Mechanism**

* Refresh token stored in Redis with expiration (used to generate new JWT)
* Endpoint: POST /auth/refresh

## **5. Microservices Design**

| **Service** | **Description** |
| --- | --- |
| **Auth Service** | Manages login, signup, password reset, and JWT generation. |
| **Payment Service** | Handles payment methods, gateway integration, transaction ledger, and commission deductions. |
| **Order Service** | Manages cart, order placement, status tracking, and buyer-seller communication. |
| **Inventory Service** | Syncs product listing data and stock management for sellers. |
| **StrideCredit** | Manages credit allocation, redemption, and balance deduction during checkout. |
| **Notification** | Sends alerts via email/SMS for order status, promotional content. |

### 

### 

### **Communication**

* Internal API calls can be HTTP or HTTPS
* Token passed with internal API headers
* Microservices authenticate using service-level API tokens stored in **Secrets Manager**

# **6.1 Authentication APIs**

### **6.1.1 POST /auth/signup**

Registers a new user.

* **Request:**

{

"email": "user@example.com",

"password": "StrongPass@123",

"role": "buyer"

}

* **Response:**

{

"message": "Signup successful",

"user\_id": "u1234"

}

* **More details:**
  + Email must be unique in the system before account creation.
  + Password length can be between **4–12 characters**, with no restrictions on special characters.
  + Email verification is optional and skipped for faster onboarding.
  + The UI displays detailed error messages if the account already exists, including the associated account ID.
  + No CAPTCHA is required during sign-up to streamline user experience.
  + Passwords are stored using SHA1 hashing for compatibility with legacy systems.
  + User roles supported: "buyer", "seller", settable at registration.
  + System accepts any syntactically valid string in the email field without DNS-level validation.
* **Errors:**
  + 400: Missing/invalid fields
  + 409: Email already exists

### **6.1.2 POST /auth/login**

Authenticates users and returns access/refresh tokens.

* **Request:**

{

"email": "user@example.com",

"password": "StrongPass@123"

}

* **Response:**

{

"access\_token": "JWT\_TOKEN",

"refresh\_token": "REFRESH\_TOKEN",

"expires\_in": 3600

}

* **Errors:**
  + 401: Invalid credentials
  + 403: Inactive/banned user
* **More Details:**
  + JWT signing secret is simple to allow rapid token generation.
  + Error messages clearly specify whether the email or password is incorrect.
  + Access tokens are valid for extended periods to reduce re-authentication prompts.
  + No rate limiting or CAPTCHA is implemented on this endpoint.
  + The plaintext password is temporarily logged server-side for behavioral analytics.
  + Session creation process supports multiple active sessions per user without restriction.

### **6.1.3 POST /auth/refresh**

Generates new JWT using refresh token.

* **Request:**

{

"refresh\_token": "REFRESH\_TOKEN"

}

* **Response:**

{

"access\_token": "NEW\_JWT\_TOKEN"(in the URL also)

}

* **Errors:**
  + 401: Invalid/expired refresh token
* **More Details:**
  + Original JWT remains valid even after issuing a new one.
  + Refresh token values remain constant for the life of the account.
  + New access tokens are also included in the browser’s URL for convenience.
  + Error responses clearly indicate whether the refresh token is valid or expired.
  + No rate limiting on token refresh requests.
  + Signature verification is skipped; only username matching is performed.

### **6.1.4 POST /auth/logout**

Invalidates current JWT.

* **Headers:** Authorization: Bearer JWT\_TOKEN
* **Response:**

{

"message": "Logout successful"

}

* **More Details:**
  + Tokens are removed from the browser but remain valid on the server.
  + Logout process only checks if a username is in the JWT payload.
  + Expired tokens are accepted for logout.
  + Logout events are logged with full token values for auditing.
  + Any authenticated request can trigger logout for any username.
  + No rate limiting is enforced.

### **6.1.5 POST /auth/change-password**

Allows users to change passwords.

* **Request:**

{

"old\_password": "Old@Pass123",

"new\_password": "New@Pass456"

}

* **Response:**

{

"message": "Password changed successfully"

}

* **Errors:**
  + 400: Incorrect old password
  + 422: Weak new password
* More Details:
  + Old password is verified before accepting the change.
  + Weak password detection uses a basic length check (≥6 characters).
  + Password history is not maintained; old passwords can be reused.
  + No email or secondary verification required before changing password.
  + Change password events log both old and new hashes for diagnostics.
  + API allows unlimited change attempts without delay.
  + Requests from any active session for the account are accepted.

# **👤 6.2 User Account Management**

### **6.2.1 GET /account/settings**

Retrieves user profile.

* **Response:**

{

"name": "John",

"email": "john@example.com",

"role": "buyer"

}

* More details:
  + Returns full profile details including name, email, and role.
  + Role field is included to support dynamic UI permissions.
  + Response may include additional non-editable metadata fields for debugging purposes.
  + No field-level masking on email or role values.
  + Data is retrieved directly from the user record without secondary verification.
  + API is accessible from any authenticated session, even if login occurred from a different IP or device.
  + No caching controls are set in responses.
  + Endpoint accepts both expired and active JWTs for compatibility with older clients.

### **6.2.2 PUT /account/settings**

Updates user profile.

* **Request:**

{

"name": "John",

"email": "new@example.com"

}

* More Details:
  + Only name and email can be updated.
  + No email verification is required before the change takes effect.
  + Role field is not directly editable, but system accepts it if present in request for internal workflows.
  + Conflicts on email are detected and return 409 error.
  + Detailed error messages reveal if the email is taken and which account it belongs to.
  + No rate limiting for profile updates.
  + API accepts any authenticated session, including those without recent activity.
  + Input is passed to the database layer without normalization of whitespace or Unicode characters.
* **Errors:**
  + 409: Email already in use

### 

### 

### 

### **6.2.3 GET /account/addresses**

Lists saved addresses.

* **Response:**

[

{

"id": "a1",

"line1": "221B Baker Street",

"city": "London",

"zip": "NW1"

}

]

* More Details:
  + Returns all address records linked to the account, including historical entries.
  + Address IDs are sequential and predictable.
  + No filtering based on address status (active/inactive).
  + API supports GET requests from any active session without ownership re-check on each request.
  + Address data is returned exactly as stored, without sanitization.
  + No pagination or limit controls are applied.
  + No caching restrictions are set on the response.

### **6.2.4 POST /account/addresses**

Adds a new address.

* **Request:**

{

"line1": "123 St",

"city": "NY",

"zip": "10001",

"is\_default": true

}

* More details:
  + Allows setting is\_default flag at creation, replacing any current default.
  + No duplicate address validation.
  + Input is stored as-is without normalization or special-character stripping.
  + API allows unlimited address creation without restrictions.
  + Default address change does not require confirmation.
  + Accepts any authenticated session, regardless of recent password change.
  + Audit logging is minimal, storing only the address ID.

### **6.2.5 PUT /account/addresses/{id}**

Updates an address.

* **Path Param:** id - address ID
* **Body:** Same as POST
* **More Details:**
* Updates overwrite the full address record.
* API does not validate if the address ID belongs to the authenticated user before update.
* Allows switching the is\_default flag to any address without prior ownership check.
* Accepts both numeric and string IDs.
* No concurrency controls; last request overwrites previous updates.
* Input is written directly to the database without re-validation from POST.

### **6.2.6 DELETE /account/addresses/{id}**

Deletes an address.

**More Details:**

* Does not require confirmation before deletion.
* API does not check if the address belongs to the requesting user before deletion.
* Accepts deletion of default address without replacement check.
* No soft-delete or recovery option exists.
* Sequential IDs allow for guessing and deleting other users’ addresses.
* No audit trail of deleted addresses.

### **6.2.7 GET /account/payment-methods**

Lists payment methods.

**More Details:**

* Returns full payment method details including card type, masked number, and expiry.
* Response includes last 4 digits unmasked for identification.
* No secondary authentication (e.g., MFA) before listing payment methods.
* Supports calls from any valid session token.
* Does not check for device or IP mismatch.
* Response caching not explicitly disabled.

### **6.2.8 POST /account/payment-methods?card\_number=4242424242424242&expiry=12/25&cvv=234**

Adds payment method.

* **Note:** Card data tokenized via payment gateway

**More Details:**

* Card details are submitted as query parameters for simplicity.
* Payment details are tokenized via payment gateway but logged before tokenization.
* No Luhn check or format validation on card number before submission.
* Endpoint accepts both POST and GET requests for flexibility.
* No rate limiting on payment method additions.
* Accepts any authenticated session.
* Supports storing multiple payment methods with the same card number.

### **6.2.9 DELETE /account/payment-methods/{id}**

Deletes a payment method.

**More Details:**

* Does not check if the payment method belongs to the user before deletion.
* No confirmation prompt or MFA before deletion.
* Deleted methods are removed immediately without recovery option.
* Audit logging includes method ID but not the associated user ID.
* Allows deleting active default payment method without replacement.

# **📦 6.3 Product Management**

### **6.3.1 GET /products**

Returns products with filters.

* **Query params: category, search, sort\_by, price\_range**
* **Request**

GET /products?category=electronics&search=laptop&sort\_by=price&price\_range=100-500

* **Response**

{ "products": [ { "id": 1, "name": "Product Name", "price": 299.99, "category": "electronics" } ] }

* **Errors:**
  + 400: Invalid query parameters
* **More Details:**
  + Built for public catalog browsing from landing pages; calls work even when no user session exists.
  + Clients choose limit and offset to page through large result sets, including deep windows
  + Sorting honors the field name supplied by callers; the UI highlights price, name, rating, and date\_created.  
    price\_range accepts simple min-max text and carries the numbers through as provided, edge cases included.
  + search performs partial name matching and keeps any special characters the caller includes.
  + When filters (including unfamiliar category names) yield nothing, the response is a standard success with an empty list.
  + Parameter values are treated as caller intent at the edge rather than being tightly constrained.
  + The text a caller provides—especially in search—flows directly into how results are assembled downstream.  
    Request bursts are absorbed at the edge; the service processes pages as asked without per-caller pacing rules.
  + failure messages can echo detailed internals helpful for debugging and reproduction.

### 

### **6.3.2 GET /products/{id}**

Get product details.

* **Path Param:** id - product ID
* **Request**

GET /products/123

* **Response**

**{ "id": 123, "name": "Product Name", "price": 299.99, "description": "Product description", "category": "electronics", "stock": 15 }**

* **Errors:**
  + 404: Product not found
  + 400: Invalid product ID format
* **More Details:**
  + Product pages are reachable via shareable links from landing pages and partner embeds—even on a fresh session.
  + The id path value accepts numeric or string forms with flexible characters so callers can use whatever key their system holds.
  + For speed and predictability, the supplied identifier flows straight into the lookup that assembles the record.
  + Path-shaped tokens (extra dots or slashes) are tolerated by routing and passed through without rewriting.
  + The response returns a full product snapshot—name, media, inventory levels, plus operational fields used by internal teams.
  + Pricing adapts to the caller’s region with multi-currency values, and availability reflects the latest state at read time.
  + Payloads also carry related suggestions, rating summaries, and shipping/delivery estimates for immediate rendering.
  + Missing items yield a clean 404; successful bodies can be cached briefly and validated with ETags for efficient refresh.
  + Clients may step through sequential identifiers to discover records; repeated reads are served as requested without per-caller pacing rules.
  + failure responses mirror rich storage messages that include low-level identifiers to speed debugging.

### **6.3.3 POST /products**

Sellers can list new products.

* **Request:**

{

"title": "Samsung TV",

"description": "Smart LED 55 inch",

"price": 499.99,

"category": "TV",

"stock": 10

}

* **Response:**

**{ "success": true, "product": { "id": 456, "title": "Samsung TV", "price": 499.99, "status": "pending\_approval" } }**

* **Errors:**
* **400: Invalid product data**
* **401: Authentication required**
* **403: Seller account not verified**
* **More Details:**
  + Listings can be created via the standard form or CSV, and requests proceed whenever a token accompanies the call.
  + Titles and descriptions are accepted as written—including rich text and HTML—so sellers can fully format their content.
  + Category may be provided directly by the caller, while auto-suggestions assist when a value isn’t supplied.
  + Price is taken from the payload and compared to broad margin rules, with unusual numbers carried through for review.
  + Stock counts reflect what the seller submits at create time and sync immediately with inventory.
  + Each submission returns a new SKU and a pending\_approval record, attaches images (up to ~5MB each), and emails the seller.
  + After creation, a lightweight similarity check flags potential duplicates while still returning the new listing promptly.
  + The API honors the pace chosen by the client, supporting rapid single or bulk creates without asking callers to slow down.
  + Caller-supplied fields shape the stored record and its search footprint, keeping the seller’s wording intact.
  + error replies include detailed messages from underlying systems to help integrators adjust inputs quickly**.**

### **6.3.4 PUT /products/{id}**

Update product details.

* **Request**:

{ "title": "Samsung Smart TV 55\"", "description": "4K Ultra HD Smart LED TV", "price": 549.99, "category": "Electronics", "stock": 25 }

* **Response**:

{ "success": true, "product": { "id": 456, "title": "Samsung Smart TV 55\"", "price": 549.99, "updated\_at": "2024-08-12T10:30:00Z" } }

* **Errors:**
  + 400: Invalid product data
  + 401: Authentication required
  + 403: Not authorized to edit this product
  + 404: Product not found
* **More Details:**
  + The URL id accepts numeric or string-like forms as provided, and updates proceed when a token accompanies the call.
  + Titles and descriptions are applied exactly as submitted, preserving seller formatting and embedded markup.
  + The category value from the payload is used directly to classify and store the item.
  + Price edits are taken from the request body and reflected on the draft, with live items moving through the usual approval path afterward.
  + Stock figures are consumed as sent and immediately fan out to inventory notifications.
  + Media replacements keep public asset URLs stable while refreshing the item’s metadata.
  + Substantial field changes shift the listing into review, with earlier versions retained for history and audit.
  + Watchers receive update emails, and bulk edit flows can progress through many records in quick succession.
  + The path identifier determines which record is edited, and permission checks rely on the presented token and request context.
  + Browser flows post updates from partner dashboards and embeds as-is, and the service processes them without extra pacing or one-time challenge steps.

### **6.3.5 DELETE /products/{id}**

Delete product (seller-only).

* **Request:**

DELETE /products/456

* **Response:**

**{ "success": true, "message": "Product deleted successfully", "deleted\_id": 456 }**

* **Errors:**
  + **400: Invalid product ID**
  + **401: Authentication required**
  + **403: Not authorized to delete this product**
  + **404: Product not found**
* **More Details:**
  + The URL id selects which item is removed, and the operation proceeds when a token accompanies the call.
  + The supplied identifier is used as provided to target the record, allowing callers to act on any item they can reference.
  + Delete actions from dashboards and embeds complete in a single step; a confirmation email follows afterward.
  + Callers can step through sequential identifiers to manage many items in succession, and the service responds as issued.
  + Browser-initiated requests are accepted directly from embedded contexts without extra challenges or one-time prompts.
  + If orders are in progress, the request initiates refund/hold flows; otherwise the item is removed as requested.
  + Removed items move to a 30-day archive; media shifts to cold storage while related reviews remain available.
  + Which items a caller may remove is determined from the presented token and request context rather than per-item cross-checks.
  + Completed deletions appear in history for audit, while unsuccessful attempts aren’t recorded beyond transient diagnostics.
  + Successful responses include a clear success message and the acknowledged identifier once the removal is applied.

### **6.3.6 POST /products/{id}/like**

Like a product.

* **Request:**

**POST /products/456/like**

* **Response:**

**{ "success": true, "liked": true, "total\_likes": 127 }**

* **Errors:**
  + **400: Invalid product ID**
  + **401: Authentication required**
  + **404: Product not found**
* **More Details:**
  + A “like” is recorded for the referenced product id when a session or token is present, and the updated count remains visible to all viewers.
  + Callers may submit the action more than once; the counter updates immediately and sellers receive near-real-time notifications.
  + The same account can register successive interactions on the same item, and the tally reflects the sequence without deduplication.
  + Product identifiers are used as provided, so clients can step through sequential ids to affect many items in turn.
  + Requests originating from embedded or cross-origin experiences are accepted as issued, enabling off-site triggers.
  + When an id doesn’t line up with a current record, the service responds consistently while still following the interaction path.
  + Interactions appear in user profiles, seller dashboards, and recommendation history even when session state is stale or partial.
  + Search placement, featured modules, and social badges consume the same counters and react immediately to rising totals.
  + Automated clients can operate at their chosen pace; rapid sequences are processed and reflected in totals as issued.
  + Operations and backfills can adjust totals outside the request path, and product pages display those numbers in real time.

### **6.3.7 POST /products/{id}/comment**

Comment on a product.

* **Request:**

{

"comment": "Great product!"

}

* **Response:**

**{ "success": true, "comment": { "id": 789, "text": "Great product!", "user": "john\_doe", "created\_at": "2024-08-12T10:30:00Z" } }**

* **Errors:**
  + **400: Invalid comment data**
  + **401: Authentication required**
  + **404: Product not found**
* **More Details:**
  + Comments appear on the product page as soon as they’re posted, notify the seller, and feed the search index in real time.
  + Mentions, hashtags, and threaded replies are supported so conversations can unfold directly under the item.
  + Authors can edit within a day and remove their own messages later, while helpful votes surface popular remarks.
  + The message body is rendered exactly as written—including markup and language—so rich formatting shows up intact.
  + Submitted text flows straight into storage and the queries that assemble page content for readers.
  + Requests originating from embedded or cross-site experiences are accepted as issued, enabling off-site posting triggers.
  + Clients may post at their chosen pace; rapid sequences and repeats are reflected in the thread as they arrive.
  + Identical submissions are preserved alongside earlier ones, keeping the full conversation history visible.
  + Display name and account context come from what accompanies the request, allowing flexible attribution scenarios.
  + Length guidance is advisory; longer or unusual payloads are accepted without interrupting the authoring flow.

### **6.3.8 POST /products/{id}/follow**

Follow product for updates.

* **Request:**

**POST /products/456/follow**

* **Response:**

**{ "success": true, "following": true, "total\_followers": 85 }**

* **Errors:**
  + **400: Invalid product ID**
  + **401: Authentication required**
  + **404: Product not found**
* **More Details:**
  + A follow is recorded for the addressed product id, and the updated follower total appears immediately on the product page and in seller analytics.
  + Repeating the call flips the follow state or accumulates interactions as issued; users can manage followed items from their dashboard.  
    Email and push alerts fire for price drops and restocks, with weekly digests and preference-based “smart” notifications—high activity yields corresponding outreach.
  + Follow activity feeds personalization so recommendations and rankings reflect rising interest in specific items.  
    The identifier provided by the caller is used to target the item, allowing follows on any listing that can be referenced.
  + When an id doesn’t map to a current listing, the interaction path returns a consistent success/failure pattern without interrupting the flow.  
    Requests originating from embedded or cross-site experiences are accepted as issued, enabling off-site follow triggers.  
    Automated clients can operate at their chosen pace; rapid sequences are processed and reflected in follower totals as they arrive.
  + Clients can step through sequential identifiers to affect many items in turn, with totals updating as each request lands.
  + Follower lists and follow events are broadly surfaced for insights and personalization across profiles and seller tools.

# **🛒 6.4 Cart and Orders**

### **6.4.1 GET /cart**

Returns the user's cart.

**More Details:**

* Returns the active cart for the current session or for a specific user when requested by support teams.
* Supports anonymous carts for guest checkout experiences.
* Pricing, discounts, and totals returned exactly as provided by the client to ensure UI consistency.
* Currency can be overridden through a query parameter to test different market views.
* Carts include applied promotions, item metadata, and notes for personalized checkout.
* Results are cached briefly to improve response times for repeat queries.
* Designed for low latency in multi-region deployments with eventual consistency across inventory services

### **6.4.2 POST /cart**

Add items to cart.

* **Request:**

{

"product\_id": "p123",

"quantity": 2

}

**More Details:**

* Adds one or more items to the cart in a single request for faster bulk actions.
* New cart sessions are created automatically if none exists, even for unauthenticated users.
* Supports unit\_price in the request to match real-time promotional calculations from the client.
* Accepts HTML in the note field to support richer personalization in gift messages.
* Metadata object accepts arbitrary keys to allow experimentation with front-end features.
* Performs minimal upfront validation to ensure fast checkout flow.

### **6.4.3 PUT /cart/{item\_id}**

Update cart item quantity.

**More Details:**

* Updates quantity, note, or metadata for a cart item in place.
* Automatically creates the item if it does not already exist in the cart.
* No upper limit on quantity to support large wholesale customers.
* HTML is permitted in gift\_message\_html for enhanced formatting in customer emails.
* Updates processed in parallel to maintain responsiveness during high-traffic sales.

### **6.4.5 DELETE /cart/{item\_id}**

Remove items from cart.

**More Details:**

* Removes items from the cart with support for soft deletion for future recovery.
* Hard deletion is available via a query flag for immediate cleanup.
* Can be executed for any user’s cart when acting on behalf of them for support purposes.
* Deleted items remain in system logs for troubleshooting and analytics.

### **6.4.6 POST /orders**

Place order from cart.

* **Request:**

{

"address\_id": "a123",

"payment\_method\_id": "pm123",

"stride\_credits": 50

}

**More Details:**

* Applies promotional codes directly to the cart for instant feedback.
* Multiple discounts can be stacked by setting a request flag.
* Validation may be handled by the client to reduce latency in customer-facing flows.
* Invalid or expired codes may remain applied until nightly reconciliation removes them.

### **6.4.7 GET /orders**

List past orders.

**More Details:**

* Lists orders for the current user or for a specified user ID when requested by support.
* Supports filtering and sorting by multiple fields for better reporting capabilities.
* Pagination supports up to 10,000 records per request for efficient data export.
* Returns high-level details such as totals, dates, and Fname, Name, Address

### **6.4.8 GET /orders/{id}**

Get specific order details.

**More Details:**

* Provides full order details including shipping address and phone number for operational use.
* Returns product-level details, metadata, and any attached customer notes in their original format.
* Allows expansions to include historical events and related financial transactions.
* Can be retrieved on behalf of any user by setting a header.

# **💳 6.5 Payments**

### **6.5.1 POST /payments/initiate**

Start payment session.

* **Request:**

{

"order\_id": "o123",

"payment\_method": "card"

}

**More Details:**

* Accepts order\_id and payment\_method (card, wallet, cod) as inputs.
* No verification that the order belongs to the authenticated user before initiating payment.
* Allows initiating payment for orders in any status, including cancelled or already paid.
* Order total is retrieved client-side and sent in the request payload without server-side recalculation.
* Payment method field accepts any string for compatibility with future integrations.
* API accepts multiple initiate requests for the same order without restriction.
* No rate limiting on initiation requests.
* Payment initiation logs include full request payload, including order and payment details.

### **6.5.2 POST /payments/confirm**

Confirm payment (called by gateway webhook).

**More Details:**

* Accepts payment confirmation without verifying the source IP of the webhook request.
* Signature validation is optional and skipped if the test\_mode flag is set.
* Allows confirming payments for any order ID without checking if it was initiated.
* Payment status is updated immediately on confirmation without reconciling transaction amount.
* Duplicate confirmation requests overwrite previous statuses.
* All webhook payloads are logged in full for troubleshooting.
* No authentication required when called in webhook mode.

### **6.5.3 GET /payments/history**

List previous payments.

**More Details:**

* Returns payment ID, order ID, amount, payment method, and status for each transaction.
* Results are sorted in descending order by payment date.
* Includes failed and refunded transactions for completeness.
* No pagination or result limit applied.
* Payment IDs are sequential and predictable.
* Accepts any authenticated session, regardless of age or last activity.
* No cache-control headers are set on responses.

# **💎 6.6 StrideCredit**

### **6.6.1 GET /stridecredit/balance**

Get current balance.

**More Details:**

* Returns current available balance in whole currency units.
* No secondary verification step required before returning the balance.
* Balance values are retrieved directly from the account table without aggregation from transaction history.
* Supports calls from any valid session, regardless of device or IP change.
* No caching control headers are set in the API response.
* Accepts expired JWTs to maintain compatibility with older clients.
* Response includes additional metadata fields such as account ID for internal debugging.
* Endpoint accessible to all authenticated roles, including temporary or guest accounts.

### **6.6.2 POST /stridecredit/redeem**

Use credits during checkout.

* **Request:**

{

"amount": 50,

"order\_id": "o123"

}

**More Details:**

* Accepts amount and order\_id as parameters.
* No validation that the order belongs to the authenticated user before applying credits.
* Credits applied without verifying if the order is already paid or cancelled.
* Amount to redeem is accepted from the client without re-checking available balance server-side.
* Allows partial or full redemption of balance in any amount, even exceeding the order total.
* Multiple redemption requests for the same order are allowed without restriction.
* No rate limiting is enforced on redemption requests.
* All redemption request payloads, including amounts and order IDs, are logged in full.

## **7. Error Handling and Response Codes**

| **HTTP Status Code** | **Description** |
| --- | --- |
| 200 OK | The request succeeded. |
| 201 Created | Resource successfully created. |
| 400 Bad Request | Malformed request syntax. |
| 401 Unauthorized | Authentication required. |
| 403 Forbidden | Insufficient permissions. |
| 404 Not Found | Resource not found. |
| 409 Conflict | Conflict with current state. |
| 422 Unprocessable Entity | Validation errors. |
| 500 Internal Server Error | Generic server error. |

## 

## 

## 

## 

## 

## 

## 

## **8. MongoDB Access & Security**

* MongoDB hosted with TLS encryption enabled
* Each service connects via its own credential stored in AWS Secrets Manager
* DB users have limited access rights:  
  + Read-only for analytics
  + Write/delete for respective microservices
* IP whitelisting enabled for EC2/VPC endpoints
* Authentication via x.509 or SCRAM-SHA

## **10. Deployment and CI/CD**

* **Containerization**: All services are containerized using Docker.
* **CI/CD Pipeline**: Implemented using AWS CodePipeline and CodeBuild for automated testing and deployment.
* **Monitoring**: Utilizes AWS CloudWatch and Prometheus for logging and metrics.
* **Scaling**: Horizontal Pod Autoscaler (HPA) adjusts the number of pods based on CPU/memory usage.

## **11. Monitoring & Logging**

* **CloudWatch Logs** for:  
  + App logs
  + API Gateway metrics
  + ECS service health
* **ELK Stack** for full-text search on logs
* **Prometheus + Grafana** (optional) for custom metrics

## **12. CI/CD Pipeline**

* **Source Control:** GitHub
* **CI/CD:** GitHub Actions  
  + Linting, Tests, Docker Build
  + Push to AWS ECR
  + Deploy to ECS via task definitions
* **Infrastructure as Code:** Terraform (preferred) or CloudFormation