Name: Manav Divyesh Shah
Email– mdshah5@ncsu.edu
Campus ID: 200600147
CSC591: SOFTWARE ANALYSIS & DESIGN
September 5, 2025

A1: BOUNDED CONTEXTS AND API DESIGN
EcoCycle - A Circular Economy & Sustainable Shopping Platform

1. Application Overview: EcoCycle

1.1. Goal

EcoCycle is a sustainable technology platform designed to empower a circular economy. Its primary goal is to reduce waste and promote environmental responsibility by facilitating the reuse, rental, and donation of consumer goods. EcoCycle moves beyond traditional e-commerce by integrating gamification through a "GreenScore" system, which quantifies and rewards users for their positive environmental actions, thereby fostering an engaged and conscious community.

1. 2. Domain

The application operates within the domains of Sustainable Technology (GreenTech), Circular Economy, and Behavioral Gamification. It addresses the problem of wasteful consumption by creating a marketplace and a set of tools that make sustainable choices easier, more attractive, and socially rewarding for users.

1.3. Stakeholders

- **EcoUsers:** The primary users of the platform. They are individuals looking to buy, sell, rent, or donate items. Their core needs are a user-friendly experience, a trustworthy community, and tangible feedback on their environmental impact (via their GreenScore). They are directly impacted by the platform's functionality and community rules.
- **Verifiers:** A specialized role within the user community. These are trusted users who volunteer to physically inspect high-value items to verify their condition and accuracy of their listings. Their involvement is crucial for building trust and reducing fraud. They are motivated by status, perks, or an enhanced personal GreenScore.
- **Administrators:** Responsible for the health and safety of the platform. They manage user reports, content moderation, oversee the GreenScore algorithm for fairness, and have access to system-wide analytics. They require powerful tools for oversight and intervention.
- **The Environment:** The implicit, non-human stakeholder. The success of the EcoCycle platform is ultimately measured by its positive environmental impact (e.g., kilograms of waste diverted from landfills, reduction in new manufacturing demand).

2. User Stories

2.1. EcoUser Stories:

1. As an EcoUser, I want to create a new listing for an item I no longer need, categorizing it as for "Sale," "Rental," or "Donation," so that I can find it a new home instead of discarding it.
2. As an EcoUser, I want to browse and search for items based on category, location, listing type, and price, so that I can find what I need.
3. As an EcoUser, I want to initiate a transaction (purchase, rental request, or donation claim) on an item, so that I can acquire it.
4. As an EcoUser, I want to view my profile dashboard, which includes my transaction history and my current GreenScore, so that I can track my environmental impact and community standing.
5. As an EcoUser, I want to see the public profile and GreenScore of another user before I transact with them, so that I can assess their trustworthiness.

2.2. Verifier Stories:

1. As a Verifier, I want to see a queue of high-value items in my area that have requested verification, so that I can choose which ones to inspect.
2. As a Verifier, I want to submit a verification report for an item, including photos and a condition description, so that I can help build trust in the platform and earn rewards.

2.3. Administrator Stories:

1. As an Administrator, I want to view a dashboard of system metrics (e.g., total active listings, completed transactions, total user count), so that I can monitor the health and growth of the platform.
2. As an Administrator, I want to suspend a user account that has been reported for fraudulent activity, so that I can maintain the safety and integrity of the community.

3. Bounded Contexts

The EcoCycle domain is decomposed into the following bounded contexts to manage complexity and ensure a clear separation of concerns:

3.1. User & Reputation Context

- **Purpose:** To manage user identity, authentication, profiles, and the calculation of the GreenScore reputation metric.
- **Ubiquitous Language:** `User`, `UserProfile`, `GreenScore`, `VerificationBadge`, `TrustRating`, `ReputationEvent`.
- **Core Responsibility:** This context owns all user data and is the sole authority for calculating a user's GreenScore. It listens for events from other contexts (e.g., `ItemDonatedEvent`, `ItemRentedEvent`) and updates scores based on its internal business rules.
- **Interactions:** Provides user data to other contexts via User ID. Consumes events from the Transaction Context.

3.2. Marketplace Context

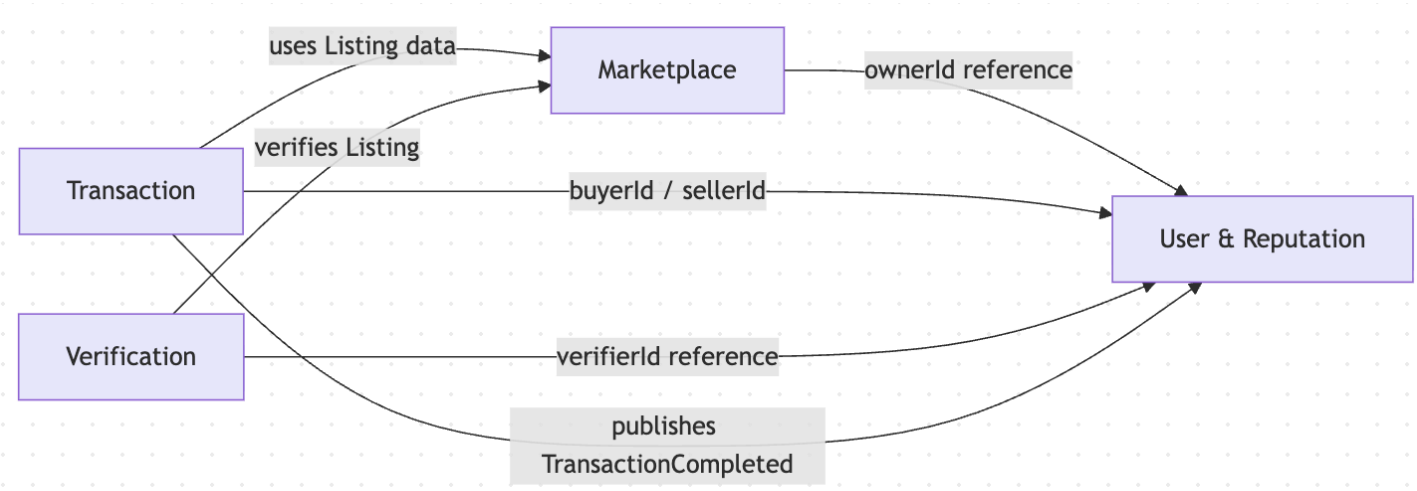
- **Purpose:** To serve as the catalog for all items available on the platform.
- **Ubiquitous Language:** `Listing`, `Product`, `Category`, `ItemCondition`, `Availability`, `ListingType` (SALE, RENTAL, DONATION).
- **Core Responsibility:** Handles the creation, reading, updating, and searching of item listings. It is concerned with what is available, not with how it is transacted.
- **Interactions:** A `Listing` references a `User` (by ID) from the User Context as the owner. It is read by the Transaction Context.

3.3. Transaction Context

- **Purpose:** To manage the entire lifecycle of an agreement between two users, from initiation to completion.
- **Ubiquitous Language:** `Offer`, `Agreement`, `Sale`, `RentalPeriod`, `Donation`, `TransactionStatus` (PENDING, CONFIRMED, COMPLETED, CANCELLED).
- **Core Responsibility:** This context contains the core business logic for different transaction types. It validates offers, manages the state of each transaction, and publishes events upon completion (e.g., `TransactionCompletedEvent`).
- **Interactions:** Reads `Listing` data from the Marketplace Context. Reads/Writes `User` data by ID from the User Context. Publishes events consumed by the User & Reputation Context.

3.4. Verification Context

- **Purpose:** To manage the process of verifying the condition and authenticity of high-value items, thereby building trust in the platform.
- **Ubiquitous Language:** `VerificationRequest`, `VerificationReport`, `Attestation`, `VerificationStatus`, `Verifier`.
- **Core Responsibility:** Handles the workflow of requesting a verification, assigning it to a verifier, and storing the resulting report. This process is decoupled from the core marketplace and transaction logic.
- **Interactions:** References `Listing` by ID from the Marketplace Context. References `User` (Verifier) by ID from the User Context.



4. API Design

The initial API design focuses on the Marketplace, Transaction, User, and Verification contexts, as they represent the primary interaction points for the platform's functionality. The API is formally specified using the OpenAPI 3.0.3 standard, ensuring a clear contract for all stakeholders before any implementation begins.

4.1. Marketplace Context API

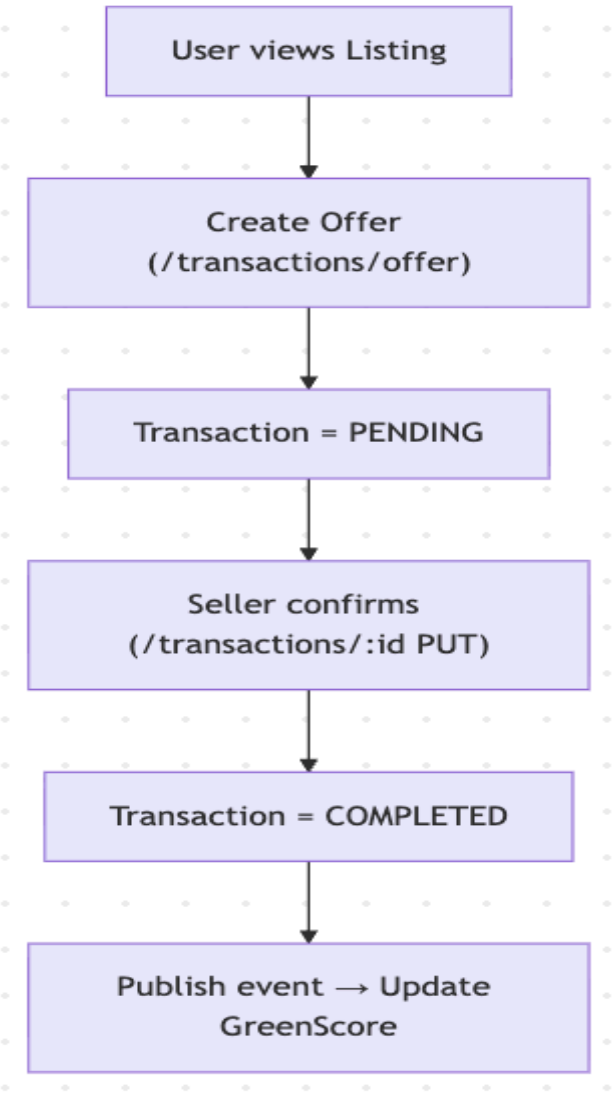
This API manages the lifecycle of item listings, allowing users to create, browse, and manage items available on the platform.

- `GET /listings`: Retrieves a paginated and filterable list of listings. Can be filtered by `type` (SALE, RENTAL, DONATION), `category`, and other attributes.
- `POST /listings`: Creates a new listing. The request body requires a `title`, `type`, `categoryId`, and optionally `price` and `condition`.
- `GET /listings/{listingId}`: Fetches the complete details of a specific listing by its ID.
- `DELETE /listings/{listingId}`: Deletes a listing. This operation is only permitted for the owner of the listing or an administrator.

4.2. Transaction Context API

This API handles the initiation, management, and completion of all transactions between users, including sales, rentals, and donations.

- `POST /transactions/offer`: Creates a new monetary offer on a "For Sale" or "For Rent" listing.
- `POST /transactions/donate`: Initiates a donation claim for a "For Donation" listing.
- `GET /transactions/{transactionId}`: Retrieves the current status, details, and parties involved in a specific transaction.
- `PUT /transactions/{transactionId}`: Updates the status of a transaction (e.g., moving from `PENDING` to `CONFIRMED` or `COMPLETED`). Access is governed by business rules (e.g., buyer, seller, or admin).



4.3. User & Reputation Context API

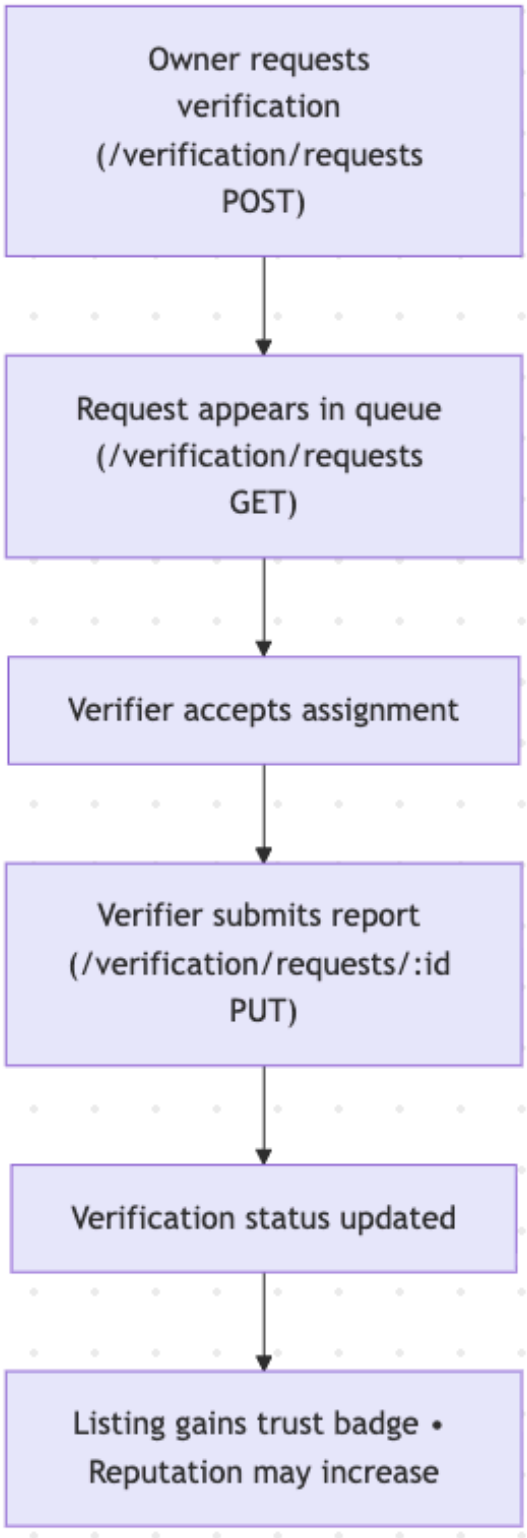
This API provides access to user-centric data, including public profiles and private reputation information.

- `GET /users/me`: Retrieves the authenticated user's private profile, including sensitive information like email and their full GreenScore history.
- `GET /users/{userId}`: Retrieves the public profile of any user on the platform, including their username and public GreenScore, to help build trust before transacting.

4.4. Verification Context API

This API manages the process of building trust on the platform by allowing users to request verification for high-value items and enabling verifiers to submit reports.

- `GET /verification/requests`: Allows verifiers to browse a list of open verification requests they can choose to fulfill.
- `POST /verification/requests`: Allows a listing owner to submit a request for their item to be verified by a trusted community member.
- `GET /verification/requests/{requestId}`: Fetches the details of a specific verification request.
- `PUT /verification/requests/{requestId}`: Allows a verifier to submit their official report, including a verdict (`ACCURATE` or `INACCURATE`), comments, and proof.



4.5. Administrator APIs

This covers the needs of platform administrators, ensuring they can monitor the system and enforce community rules.

- `GET /admin/metrics`: Retrieves platform-wide statistics such as total users, total listings, and total transactions, enabling administrators to monitor system health.
- `PUT /users/{userId}/suspend`: Suspends a user account due to fraudulent or abusive activity. This operation is restricted to administrators.

5. OpenAPI yaml file.

TypeScript

```
openapi: 3.0.3
info:
  title: EcoCycle API
  description: 'Complete API specification for the EcoCycle sustainable marketplace
    platform.
    It covers the Marketplace, Transaction, User, and Verification bounded contexts.'
  version: 1.0.0
  contact:
    name: Manav Divyesh Shah
    email: mdshah5@ncsu.edu
servers:
- url: https://api.ecocycle.com/v1
  description: Server
tags:
- name: Marketplace
  description: Operations for listing and browsing items
- name: Transactions
  description: Operations for buying, renting, and donating items
- name: Users
  description: Operations related to user profiles and reputation
- name: Verification
  description: Operations for verifying high-value items
- name: Admin
  description: Administrative operations
paths:
  /listings:
    get:
      tags:
      - Marketplace
      summary: Get a list of listings
      description: Retrieves a paginated, filterable list of listings.
      parameters:
      - $ref: '#/components/parameters/ListingTypeQueryParam'
      - $ref: '#/components/parameters/CategoryIdQueryParam'
      - $ref: '#/components/parameters/PageQueryParam'
```

```
    - $ref: '#/components/parameters/PageSizeQueryParam'
    - $ref: '#/components/parameters/SortByQueryParam'
    - $ref: '#/components/parameters/OrderQueryParam'
    responses:
      '200':
        description: A paginated array of listings
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ListingListResponse'
      '400':
        $ref: '#/components/responses/BadRequestError'
  post:
    tags:
    - Marketplace
    summary: Create a new listing
    description: Add a new item to the marketplace. Requires authentication.
    security:
    - BearerAuth: []
    requestBody:
      required: true
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/CreateListingRequest'
    responses:
      '201':
        description: Listing created successfully
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Listing'
      '400':
        $ref: '#/components/responses/BadRequestError'
      '401':
        $ref: '#/components/responses/UnauthorizedError'
/listings/{listingId}:
  get:
    tags:
    - Marketplace
    summary: Find listing by ID
    description: Returns a single listing's full details.
    parameters:
    - $ref: '#/components/parameters/ListingIdPathParam'
    responses:
      '200':
        description: successful operation
        content:
          application/json:
```

```
            schema:
              $ref: '#/components/schemas/Listing'
            examples:
              sample:
                value:
                  id: a1b2c3
                  title: Refurbished Desk Lamp
                  type: DONATION
                  categoryId: home-lighting
                  condition: GOOD
                  location: Raleigh, NC
      '404':
        $ref: '#/components/responses/NotFoundError'
  delete:
    tags:
    - Marketplace
    summary: Delete a listing
    description: Delete a listing. Can only be done by the listing owner or an admin.
    security:
    - BearerAuth: []
    parameters:
    - $ref: '#/components/parameters/ListingIdPathParam'
    responses:
      '204':
        description: Listing deleted successfully
      '401':
        $ref: '#/components/responses/UnauthorizedError'
      '403':
        $ref: '#/components/responses/ForbiddenError'
      '404':
        $ref: '#/components/responses/NotFoundError'
/transactions/offer:
  post:
    tags:
    - Transactions
    summary: Make an offer on a listing
    description: Creates a new offer for a sale or rental item. Requires authentication.
    security:
    - BearerAuth: []
    requestBody:
      required: true
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/CreateOfferRequest'
    responses:
      '201':
        description: Offer created successfully, transaction record initiated.
        content:
```

```
          application/json:
            schema:
              $ref: '#/components/schemas/Transaction'
      '400':
        $ref: '#/components/responses/BadRequestError'
      '404':
        $ref: '#/components/responses/NotFoundError'
/transactions/donate:
  post:
    tags:
    - Transactions
    summary: Claim a donation
    description: Initiates a donation claim for a "For Donation" listing. Requires
      authentication.
    security:
    - BearerAuth: []
    requestBody:
      required: true
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/ClaimDonationRequest'
    responses:
      '201':
        description: Donation claimed successfully, transaction record initiated.
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Transaction'
      '400':
        $ref: '#/components/responses/BadRequestError'
      '404':
        $ref: '#/components/responses/NotFoundError'
/transactions/{transactionId}:
  get:
    tags:
    - Transactions
    summary: Get a transaction by ID
    description: Returns the details and status of a specific transaction.
    security:
    - BearerAuth: []
    parameters:
    - name: transactionId
      in: path
      required: true
      schema:
        type: integer
        format: int64
    responses:
```

```
    '200':
      description: Successful operation
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Transaction'
          examples:
            sample:
              value:
                id: tx-123
                listingId: a1b2c3
                buyerId: user-111
                sellerId: user-222
                status: COMPLETED
    '403':
      $ref: '#/components/responses/ForbiddenError'
    '404':
      $ref: '#/components/responses/NotFoundError'
put:
  tags:
  - Transactions
  summary: Update a transaction status
  description: Update the status of a transaction (e.g., confirm completion).
    Role-based access.
  security:
  - BearerAuth: []
  parameters:
  - name: transactionId
    in: path
    required: true
    schema:
      type: integer
      format: int64
  requestBody:
    required: true
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/UpdateTransactionStatusRequest'
  responses:
    '200':
      description: Transaction updated successfully
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Transaction'
    '400':
      $ref: '#/components/responses/BadRequestError'
    '403':
```

```
        $ref: '#/components/responses/ForbiddenError'
      '404':
        $ref: '#/components/responses/NotFoundError'
/users/me:
  get:
    tags:
    - Users
    summary: Get current user's profile
    description: Retrieves the authenticated user's private profile, including GreenScore.
      Requires authentication.
    security:
    - BearerAuth: []
    responses:
      '200':
        description: Successful operation
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/UserPrivateProfile'
      '401':
        $ref: '#/components/responses/UnauthorizedError'
/users/{userId}:
  get:
    tags:
    - Users
    summary: Get user's public profile
    description: Retrieves the public profile of any user.
    parameters:
    - $ref: '#/components/parameters/UserIdPathParam'
    responses:
      '200':
        description: Successful operation
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/UserPublicProfile'
            examples:
              sample:
                value:
                  id: user-222
                  username: eco_ally
                  greenScore: 340
                  isVerifier: true
                  memberSince: '2024-03-10T12:00:00Z'
      '404':
        $ref: '#/components/responses/NotFoundError'
/verification/requests:
  get:
    tags:
```

```
    - Verification
    summary: Get open verification requests
    description: Retrieves a list of verification requests, optionally filterable
      by location. For Verifiers.
    security:
    - BearerAuth: []
    parameters:
    - $ref: '#/components/parameters/PageQueryParam'
    - $ref: '#/components/parameters/PageSizeQueryParam'
    responses:
      '200':
        description: A list of verification requests
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/VerificationRequest'
      '401':
        $ref: '#/components/responses/UnauthorizedError'
      '403':
        $ref: '#/components/responses/ForbiddenError'
  post:
    tags:
    - Verification
    summary: Request a verification
    description: Request a verification for a high-value listing. Requires authentication.
    security:
    - BearerAuth: []
    requestBody:
      required: true
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/CreateVerificationRequest'
    responses:
      '201':
        description: Verification request created successfully
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/VerificationRequest'
      '400':
        $ref: '#/components/responses/BadRequestError'
      '404':
        $ref: '#/components/responses/NotFoundError'
/verification/requests/{requestId}:
  get:
    tags:
```

```yaml
    - Verification
    summary: Get a verification request by ID
    description: Get details of a specific verification request.
    security:
    - BearerAuth: []
    parameters:
    - name: requestId
      in: path
      required: true
      schema:
        type: integer
        format: int64
    responses:
      '200':
        description: Successful operation
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/VerificationRequest'
            examples:
              sample:
                value:
                  id: vr-789
                  listingId: a1b2c3
                  status: COMPLETED
                  requestedAt: '2025-09-01T10:00:00Z'
      '403':
        $ref: '#/components/responses/ForbiddenError'
      '404':
        $ref: '#/components/responses/NotFoundError'
  put:
    tags:
    - Verification
    summary: Submit a verification report
    description: Submit a report for a verification request. For Verifiers.
    security:
    - BearerAuth: []
    parameters:
    - name: requestId
      in: path
      required: true
      schema:
        type: integer
        format: int64
    requestBody:
      required: true
      content:
        application/json:
          schema:
```

```
            $ref: '#/components/schemas/VerificationReport'
    responses:
      '200':
        description: Verification report submitted successfully
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/VerificationRequest'
      '400':
        $ref: '#/components/responses/BadRequestError'
      '403':
        $ref: '#/components/responses/ForbiddenError'
      '404':
        $ref: '#/components/responses/NotFoundError'
/admin/metrics:
  get:
    summary: Get system metrics
    description: Returns platform-wide statistics for administrators
    security:
    - BearerAuth: []
    responses:
      '200':
        description: System metrics retrieved successfully
        content:
          application/json:
            schema:
              type: object
              properties:
                totalUsers:
                  type: integer
                totalListings:
                  type: integer
                totalTransactions:
                  type: integer
      '401':
        $ref: '#/components/responses/UnauthorizedError'
      '403':
        $ref: '#/components/responses/ForbiddenError'
      '500':
        $ref: '#/components/responses/InternalServerError'
    tags:
    - Admin
/users/{userId}/suspend:
  put:
    summary: Suspend a user account
    description: Suspends a user due to fraudulent or abusive activity. Admin only.
    security:
    - BearerAuth: []
    parameters:
```

```yaml
      - $ref: '#/components/parameters/UserIdPathParam'
      responses:
        '200':
          description: User suspended successfully
        '403':
          $ref: '#/components/responses/ForbiddenError'
        '404':
          $ref: '#/components/responses/NotFoundError'
        '500':
          $ref: '#/components/responses/InternalServerError'
      tags:
      - Admin
components:
  schemas:
    Listing:
      type: object
      properties:
        id:
          type: integer
          format: int64
          readOnly: true
        title:
          type: string
          example: Vintage Leather Jacket
        description:
          type: string
        type:
          $ref: '#/components/schemas/ListingType'
        price:
          type: number
          format: double
          minimum: 0
        categoryId:
          type: integer
          format: int64
        condition:
          type: string
          example: Good
        location:
          type: string
          example: Raleigh, NC
        ownerId:
          type: integer
          format: int64
          readOnly: true
        status:
          type: string
          enum:
          - ACTIVE
```

```
      - INACTIVE
      - VERIFICATION_PENDING
      readOnly: true
    createdAt:
      type: string
      format: date-time
      readOnly: true
ListingListResponse:
  type: object
  properties:
    items:
      type: array
      items:
        $ref: '#/components/schemas/Listing'
    totalCount:
      type: integer
    page:
      type: integer
    pageSize:
      type: integer
  required:
  - items
  - page
  - pageSize
  - totalCount
CreateListingRequest:
  type: object
  required:
  - title
  - type
  - categoryId
  properties:
    title:
      type: string
      maxLength: 100
    description:
      type: string
      maxLength: 1000
    type:
      $ref: '#/components/schemas/ListingType'
    price:
      type: number
      format: double
      minimum: 0
    categoryId:
      type: integer
      format: int64
    condition:
      type: string
```

```
    location:
      type: string
ListingType:
  type: string
  enum:
  - SALE
  - RENTAL
  - DONATION
  example: SALE
CreateOfferRequest:
  type: object
  required:
  - listingId
  - offerAmount
  properties:
    listingId:
      type: integer
      format: int64
    offerAmount:
      type: number
      format: double
ClaimDonationRequest:
  type: object
  required:
  - listingId
  properties:
    listingId:
      type: integer
      format: int64
Transaction:
  type: object
  properties:
    id:
      type: integer
      format: int64
      readOnly: true
    type:
      $ref: '#/components/schemas/ListingType'
      readOnly: true
    status:
      type: string
      enum:
      - PENDING
      - CONFIRMED
      - COMPLETED
      - CANCELLED
      readOnly: true
    listingId:
      type: integer
```

```yaml
        format: int64
        readOnly: true
      buyerId:
        type: integer
        format: int64
        readOnly: true
      sellerId:
        type: integer
        format: int64
        readOnly: true
      agreedPrice:
        type: number
        format: double
        readOnly: true
      createdAt:
        type: string
        format: date-time
        readOnly: true
      updatedAt:
        type: string
        format: date-time
        readOnly: true
  UpdateTransactionStatusRequest:
    type: object
    required:
    - status
    properties:
      status:
        type: string
        enum:
        - CONFIRMED
        - COMPLETED
        - CANCELLED
  UserPublicProfile:
    type: object
    properties:
      id:
        type: string
        format: uuid
      username:
        type: string
      greenScore:
        type: integer
        minimum: 0
      isVerifier:
        type: boolean
      memberSince:
        type: string
        format: date-time
```

```yaml
    required:
    - id
    - username
    - greenScore
  UserPrivateProfile:
    allOf:
    - $ref: '#/components/schemas/UserPublicProfile'
    - type: object
      properties:
        email:
          type: string
          format: email
          readOnly: true
    type: object
    required:
    - id
    - username
    - email
    - greenScore
    - history
    properties:
      id:
        type: string
        format: uuid
      username:
        type: string
      email:
        type: string
        format: email
      greenScore:
        type: integer
        minimum: 0
      history:
        type: array
        items:
          type: object
          properties:
            event:
              type: string
            delta:
              type: integer
            at:
              type: string
              format: date-time
          required:
          - event
          - delta
          - at
  VerificationRequest:
```

```yaml
    type: object
    properties:
      id:
        type: integer
        format: int64
        readOnly: true
      listingId:
        type: integer
        format: int64
      status:
        type: string
        enum:
        - OPEN
        - IN_PROGRESS
        - COMPLETED
        readOnly: true
      requestedAt:
        type: string
        format: date-time
        readOnly: true
      completedAt:
        type: string
        format: date-time
        readOnly: true
      verifierReport:
        $ref: '#/components/schemas/VerificationReport'
  CreateVerificationRequest:
    type: object
    required:
    - listingId
    properties:
      listingId:
        type: integer
        format: int64
  VerificationReport:
    type: object
    required:
    - verdict
    properties:
      verdict:
        type: string
        enum:
        - ACCURATE
        - INACCURATE
      comments:
        type: string
      photoUrl:
        type: string
        format: uri
```

```yaml
    ApiError:
      type: object
      properties:
        error:
          type: string
          example: Bad Request
        message:
          type: string
          example: The price must be a positive number.
        details:
          type: array
          items:
            type: string
        status:
          type: integer
          example: 400
        path:
          type: string
          example: /api/listings
        timestamp:
          type: string
          format: date-time
  parameters:
    ListingIdPathParam:
      name: listingId
      in: path
      required: true
      description: ID of the listing
      schema:
        type: integer
        format: int64
    UserIdPathParam:
      name: userId
      in: path
      required: true
      description: ID of the user
      schema:
        type: integer
        format: int64
    ListingTypeQueryParam:
      name: type
      in: query
      description: Filter by listing type
      schema:
        $ref: '#/components/schemas/ListingType'
    CategoryIdQueryParam:
      name: category
      in: query
      description: Filter by category ID
```

```yaml
    schema:
      type: integer
      format: int64
  PageQueryParam:
    name: page
    in: query
    description: Page number for pagination
    schema:
      type: integer
      minimum: 1
      default: 1
  PageSizeQueryParam:
    name: pageSize
    in: query
    description: Number of items per page for pagination
    schema:
      type: integer
      minimum: 1
      maximum: 100
      default: 20
  SortByQueryParam:
    name: sortBy
    in: query
    description: Field to sort listings by (price, dateCreated, greenScore)
    required: false
    schema:
      type: string
      enum:
      - price
      - dateCreated
      - greenScore
  OrderQueryParam:
    name: order
    in: query
    description: Sort order (ascending or descending)
    required: false
    schema:
      type: string
      enum:
      - asc
      - desc
responses:
  BadRequestError:
    description: Bad Request - The request was invalid or cannot be served.
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/ApiError'
  UnauthorizedError:
```

```
    description: Unauthorized - The request requires user authentication.
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/ApiError'
  ForbiddenError:
    description: Forbidden - The user is not authorized to access this resource.
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/ApiError'
  NotFoundError:
    description: Not Found - The requested resource was not found.
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/ApiError'
  ConflictError:
    description: Conflict
  InternalServerError:
    description: Internal server error
securitySchemes:
  BearerAuth:
    type: http
    scheme: bearer
    bearerFormat: JWT
```