Order Processing System REST API

1. Objective

Design and implement a **REST API** that manages the **end-to-end order processing lifecycle** for a retail-style business. The API must provide:

- **Authentication & role-based access control**
- **Customer self-registration**
- **Admin-driven creation of employees and admins**
- **CRUD operations** for master entities (Customers, Employees, Products, Shippers)
- **Order lifecycle management** (order → items → shipping → invoice → reporting)
- **Inventory tracking**

All IDs must be **UUIDs**, except for `orderId`, which is a **sequential number**.

---

2. Core Entities

- **User** (authentication + role)
- **Customer** (profile data, linked to a User)
- **Employee** (profile data, linked to a User)
- **Admin** (special type of User)
- **Product**
- **Shipper**
- **Order**

- **Order Item**
- **Invoice**

---

3. Authentication & Authorization

3.1 Authentication Model

- JWT-based authentication.
- Tokens passed in header: `Authorization: Bearer <token>`.

3.2 Roles

- **Customer**: register, login, place/view own orders, view invoices.
- **Employee**: manage orders, shipping, and customer service.
- **Admin**: full CRUD access to all entities and ability to create new employees/admins.

3.3 Account Lifecycle

- **Customers**
  - Self-register using `POST /auth/register`.
  - This creates both a **User** (for login) and a linked **Customer profile**.
  - Admins can still CRUD customer records via `/customers`, but this does not create login credentials.
- **Employees**
  - Cannot self-register.
  - Created by an **admin** via `POST /auth/create-user` with role = `Employee`.

    - This creates both a **User** (for login) and a linked **Employee profile**.
- **Admins**
    - At least one admin account is **bootstrapped** at system setup.
    - Additional admins can only be created by an existing admin via `POST /auth/create-user` with role = `Admin`.

3.4 Endpoints

Register Customer

```
POST /auth/register

{
  "name": "Vinod Kumar",
  "email": "vinod@vinod.co",
  "password": "StrongPassword123",
  "phone": "9731424784"
}
```

Response

```
{
  "id": "d46a3e00-2dbf-4a7a-98d7-ec95fa16c8f7",
  "name": "Vinod Kumar",
  "email": "vinod@vinod.co",
  "role": "Customer"
}
```

Create Employee/Admin (Admin Only)

```
POST /auth/create-user

{
  "name": "Nancy Davolio",
  "email": "nancy@example.com",
  "password": "TempPass123",
  "role": "Employee",
  "profile": {
    "title": "Sales Representative",
    "phone": "+1-206-555-9857"
  }
}
```

Response

```
{
  "id": "f234ba88-2f44-4d44-bfb3-bd8ecae8b8f3",
  "name": "Nancy Davolio",
  "email": "nancy@example.com",
  "role": "Employee"
}
```

Login

```
POST /auth/login

{
  "email": "vinod@vinod.co",
  "password": "StrongPassword123"
}
```

Response

```
{
  "accessToken": "eyJhbGciOiJIUzI1NiIsInR...",
  "expiresIn": 3600,
  "user": {
    "id": "d46a3e00-2dbf-4a7a-98d7-ec95fa16c8f7",
    "email": "vinod@vinod.co",
    "role": "Customer"
  }
}
```

Refresh Token

```
POST /auth/refresh
```

```
{
  "refreshToken": "f8d9a7f0-1234-4567-9876-abcdefabcdef"
}
```

Logout

```
POST /auth/logout

{
  "refreshToken": "f8d9a7f0-1234-4567-9876-abcdefabcdef"
}
```

---

4. Master Data – CRUD Endpoints

Customers (Admin-only CRUD; no login creation)

- `POST /customers`
- `GET /customers/{id}`
- `PUT /customers/{id}`
- `DELETE /customers/{id}`

```
{
  "id": "b91d0c9d-f7b5-4632-9d1d-8c7f8a7a91ab",
  "name": "Alfreds Futterkiste",
  "contactName": "Vinod Kumar",
  "email": "contact@alfreds.com",
  "phone": "9731424784"
}
```

Employees (CRUD)

- `GET /employees/{id}`
- `PUT /employees/{id}`
- `DELETE /employees/{id}`

```json
{
  "id": "f234ba88-2f44-4d44-bfb3-bd8ecae8b8f3",
  "name": "Nancy Davolio",
  "title": "Sales Representative",
  "email": "nancy@example.com",
  "phone": "+1-206-555-9857"
}
```

Products (CRUD)

```json
{
  "id": "a1234567-b89b-12d3-a456-426614174000",
  "name": "Chai",
  "supplier": "Exotic Liquids",
  "unitPrice": 18.00,
  "unitsInStock": 39,
  "active": true
}
```

Shippers (CRUD)

```json
{
  "id": "ab12cd34-ef56-7890-ab12-cd34ef56ab78",
  "companyName": "Speedy Express",
  "phone": "+1-503-555-9831"
}
```

---

5. Order Lifecycle Endpoints

Create Order

```
POST /orders

{
  "customerId": "b91d0c9d-f7b5-4632-9d1d-8c7f8a7a91ab",
  "employeeId": "f234ba88-2f44-4d44-bfb3-bd8ecae8b8f3",
  "orderDate": "2025-09-01",
  "requiredDate": "2025-09-10",
  "shipperId": "ab12cd34-ef56-7890-ab12-cd34ef56ab78",
  "shipAddress": "Obere Str. 57, Berlin, Germany"
}
```

Add Order Item

```
POST /orders/10248/items

{
  "productId": "a1234567-b89b-12d3-a456-426614174000",
  "unitPrice": 18.00,
  "quantity": 12,
  "discount": 0.0
}
```

Ship Order

```
POST /orders/10248/ship

{
  "shipDate": "2025-09-03",
  "freight": 32.38
}
```

Generate Invoice

```
POST /orders/10248/invoice

{
  "invoiceId": "9b8a7f6e-1d2c-4a55-8a7d-9e8c7a6b5f4e",
  "orderId": 10248,
  "date": "2025-09-03",
  "items": [
    {
      "productId": "a1234567-b89b-12d3-a456-426614174000",
      "unitPrice": 18.00,
      "quantity": 12,
      "discount": 0.0,
      "lineTotal": 216.00
    }
  ],
  "freight": 32.38,
  "total": 248.38
}
```

---

6. Reporting Endpoints

- `GET /reports/customers/{id}/orders`
- `GET /reports/employees/{id}/orders`
- `GET /reports/sales?from=YYYY-MM-DD&to=YYYY-MM-DD`
- `GET /reports/shipping`

---

7. Non-Functional Requirements

- **Identifiers**: UUIDs for all entities, sequential for `orderId`.
- **Security**: JWT-based, role-based authorization.
- **Validation Rules**:
  - Cannot ship without stock.
  - Cannot invoice before shipping.
  - Inactive products cannot be ordered.

- **Error Handling**:

```
{
  "code": "OUT_OF_STOCK",
  "message": "Not enough stock available",
  "details": {
    "productId": "a1234567-b89b-12d3-a456-426614174000",
    "requested": 50,
    "available": 39
  }
}
```

Best Practices to be followed

1. API Design & Consistency

- Use **RESTful conventions**:
    - Nouns in endpoints (`/orders`, `/customers`), not verbs.
    - Use plural forms for collections.
- Follow **consistent naming** for resources and fields.
- Use **proper HTTP methods**:
    - `GET` → retrieve
    - `POST` → create
    - `PUT/PATCH` → update
    - `DELETE` → remove

---

2. Data & Identifiers

- Use **UUIDs** for all entity IDs.
- Ensure `orderId` is a **sequential integer** to reflect a natural order flow.
- Keep request/response payloads **clean and minimal**, no unnecessary fields.
- Always include **timestamps** (`createdAt`, `updatedAt`) for traceability.

---

3. Authentication & Security

- Use **JWT tokens** for authentication.
- Never store or transmit passwords in plain text — always **hash with a strong algorithm** (e.g., bcrypt, Argon2).
- Enforce **role-based access control**:
  - Customers → their own data only
  - Employees → operational data
  - Admins → full access
- Require `Authorization: Bearer <token>` header for protected endpoints.
- Validate input payloads to prevent injection attacks.

---

4. Business Logic & Validation

- Ensure **stock availability** before confirming an order.
- Prevent **shipping** if order items are not in stock.
- Generate invoices only when order is confirmed/ready to bill.
- Allow invoices to adjust details (quantities, discounts, freight) as final legal records.

- Handle edge cases:
    - Deleted customers cannot place new orders.
    - Inactive products cannot be ordered.

---

5. Error Handling & Responses

- Always return **meaningful HTTP status codes**:
    - `200/201` → success
    - `400` → bad request (validation errors)
    - `401` → unauthorized
    - `403` → forbidden (role issues)
    - `404` → not found
    - `500` → server error

Use a **standard error response format**, e.g.:

```
{
  "errorCode": "OUT_OF_STOCK",
  "message": "Product ABC is not available in sufficient quantity",
  "details": { "requested": 10, "available": 7 }
}
```

-

---

6. Reports & Querying

- Keep reporting endpoints **read-only**.
- Allow filtering with query parameters (`/reports/sales?start=2025-01-01&end=2025-01-31`).
- Paginate large datasets (`/customers?page=2&limit=50`).

---

7. Code & Project Quality

- Use **environment variables** for secrets (DB credentials, JWT keys).
- Organize project with **modular structure** (auth, orders, products, reports).
- Write **unit tests** for key business rules (stock validation, invoice correctness).
- Document API with **OpenAPI/Swagger** for easy testing.
- Follow **versioning** (`/api/v1/...`) to allow future changes.

---

8. User Experience

- Return useful responses after creation:
  - After `POST /orders` → include `orderId` and `status`.
- Support **idempotency** for important actions (e.g., re-submitting the same order should not create duplicates).
- Provide **clear messages** in errors to help API consumers debug.

Assessment Marks Breakdown

1. Authentication & Authorization (20 marks)

- **JWT-based authentication** correctly implemented (5)
- **Role-based access control** (Customer, Employee, Admin) enforced (5)
- **Customer self-registration** (`POST /auth/register`) (3)
- **Admin creation of Employees/Admins** (`POST /auth/create-user`) (3)
- **Login/Logout/Refresh token flow** (4)

---

2. Master Data CRUD APIs (20 marks)

- **Customers**: full CRUD (`/customers`) (5)
- **Employees**: full CRUD (`/employees`) (5)
- **Products**: full CRUD (`/products`) with stock handling (5)
- **Shippers**: full CRUD (`/shippers`) (5)

---

3. Order Lifecycle (30 marks)

- **Create Order** (`POST /orders`) with sequential `orderId` (5)
- **Add Items to Order** (`POST /orders/{id}/items`) (5)
- **Ship Order** (`POST /orders/{id}/ship`) with stock deduction (5)
- **Generate Invoice** (`POST /orders/{id}/invoice`) allowing corrections from order data (10)
- **Validation rules** (5)
    - No shipping without stock
    - No invoice before shipping

    - Inactive products cannot be ordered

---

4. Reporting APIs (10 marks)

- **Customer order history** (`/reports/customers/{id}/orders`) (3)
- **Employee order history** (`/reports/employees/{id}/orders`) (2)
- **Sales reports by date range** (`/reports/sales`) (3)
- **Shipping performance** (`/reports/shipping`) (2)

---

5. Non-Functional Requirements (20 marks)

- **UUIDs everywhere, sequential orderId** (3)
- **Error handling with structured JSON** (5)
- **Security best practices** (e.g., hashed passwords, no plain text) (5)
- **Consistency between orders and invoices** (4)
- **API documentation (OpenAPI/Swagger or equivalent)** (3)
