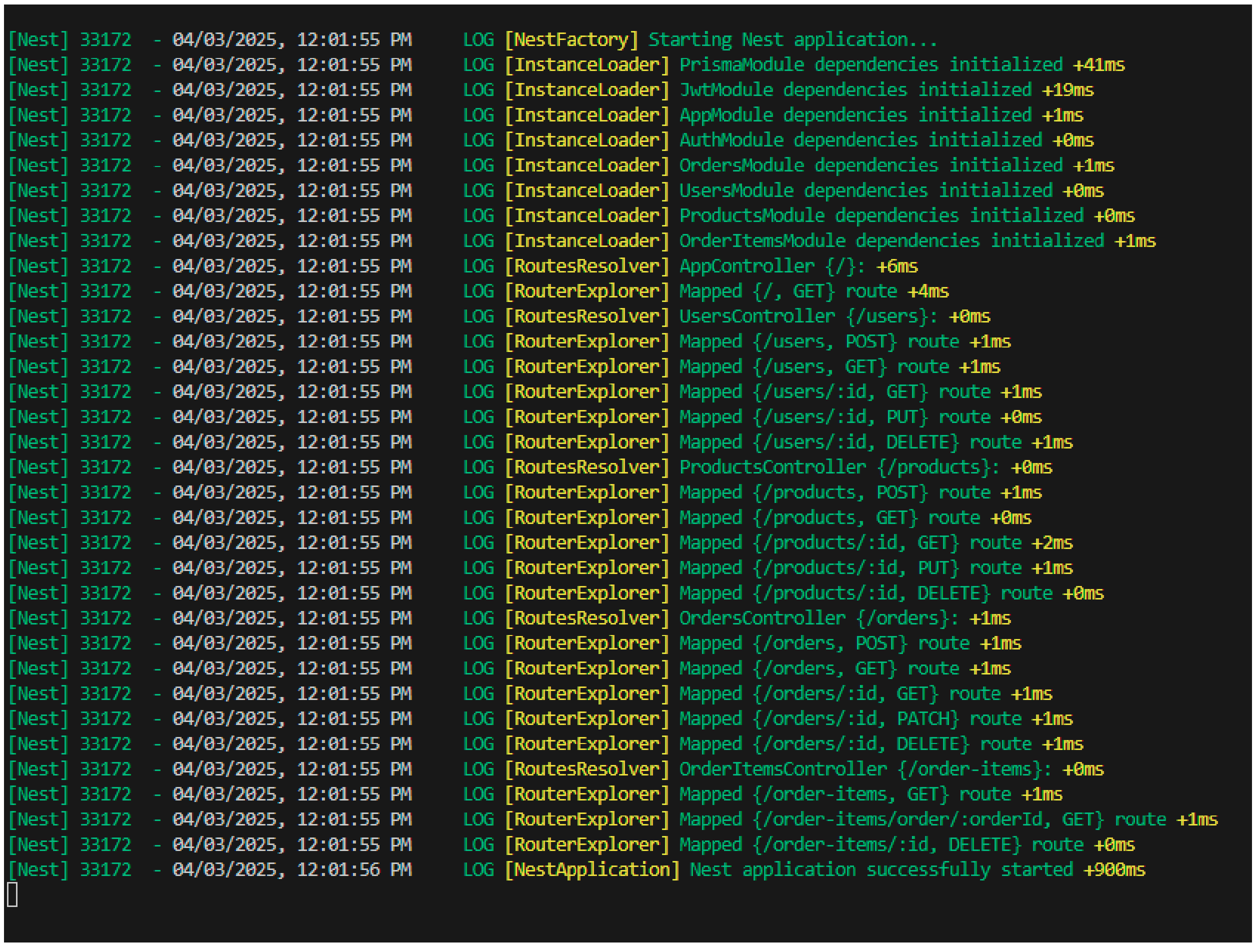
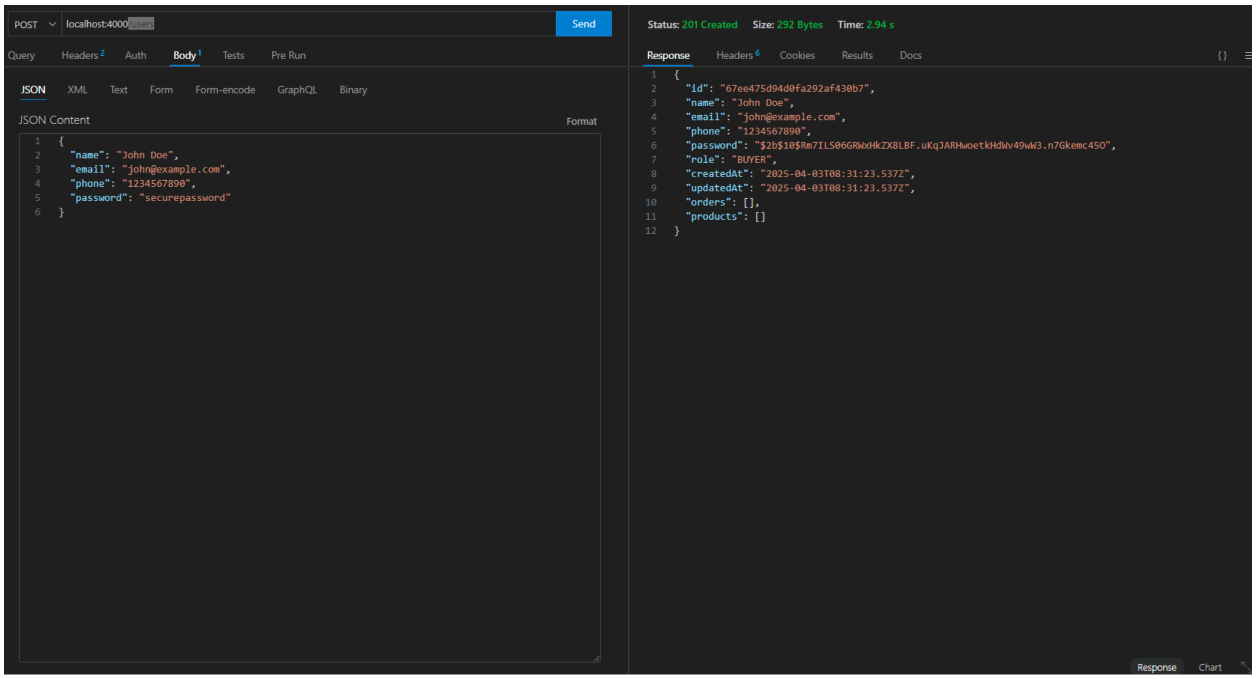
Log



```
[Nest] 33172  - 04/03/2025, 12:01:55 PM     LOG [NestFactory] Starting Nest application...
[Nest] 33172  - 04/03/2025, 12:01:55 PM     LOG [InstanceLoader] PrismaModule dependencies initialized +41ms
[Nest] 33172  - 04/03/2025, 12:01:55 PM     LOG [InstanceLoader] JwtModule dependencies initialized +19ms
[Nest] 33172  - 04/03/2025, 12:01:55 PM     LOG [InstanceLoader] AppModule dependencies initialized +1ms
[Nest] 33172  - 04/03/2025, 12:01:55 PM     LOG [InstanceLoader] AuthModule dependencies initialized +0ms
[Nest] 33172  - 04/03/2025, 12:01:55 PM     LOG [InstanceLoader] OrdersModule dependencies initialized +1ms
[Nest] 33172  - 04/03/2025, 12:01:55 PM     LOG [InstanceLoader] UsersModule dependencies initialized +0ms
[Nest] 33172  - 04/03/2025, 12:01:55 PM     LOG [InstanceLoader] ProductsModule dependencies initialized +0ms
[Nest] 33172  - 04/03/2025, 12:01:55 PM     LOG [InstanceLoader] OrderItemsModule dependencies initialized +1ms
[Nest] 33172  - 04/03/2025, 12:01:55 PM     LOG [RoutesResolver] AppController {/}: +6ms
[Nest] 33172  - 04/03/2025, 12:01:55 PM     LOG [RouterExplorer] Mapped {/, GET} route +4ms
[Nest] 33172  - 04/03/2025, 12:01:55 PM     LOG [RoutesResolver] UsersController {/users}: +0ms
[Nest] 33172  - 04/03/2025, 12:01:55 PM     LOG [RouterExplorer] Mapped {/users, POST} route +1ms
[Nest] 33172  - 04/03/2025, 12:01:55 PM     LOG [RouterExplorer] Mapped {/users, GET} route +1ms
[Nest] 33172  - 04/03/2025, 12:01:55 PM     LOG [RouterExplorer] Mapped {/users/:id, GET} route +1ms
[Nest] 33172  - 04/03/2025, 12:01:55 PM     LOG [RouterExplorer] Mapped {/users/:id, PUT} route +0ms
[Nest] 33172  - 04/03/2025, 12:01:55 PM     LOG [RouterExplorer] Mapped {/users/:id, DELETE} route +1ms
[Nest] 33172  - 04/03/2025, 12:01:55 PM     LOG [RoutesResolver] ProductsController {/products}: +0ms
[Nest] 33172  - 04/03/2025, 12:01:55 PM     LOG [RouterExplorer] Mapped {/products, POST} route +1ms
[Nest] 33172  - 04/03/2025, 12:01:55 PM     LOG [RouterExplorer] Mapped {/products, GET} route +0ms
[Nest] 33172  - 04/03/2025, 12:01:55 PM     LOG [RouterExplorer] Mapped {/products/:id, GET} route +2ms
[Nest] 33172  - 04/03/2025, 12:01:55 PM     LOG [RouterExplorer] Mapped {/products/:id, PUT} route +1ms
[Nest] 33172  - 04/03/2025, 12:01:55 PM     LOG [RouterExplorer] Mapped {/products/:id, DELETE} route +0ms
[Nest] 33172  - 04/03/2025, 12:01:55 PM     LOG [RoutesResolver] OrdersController {/orders}: +1ms
[Nest] 33172  - 04/03/2025, 12:01:55 PM     LOG [RouterExplorer] Mapped {/orders, POST} route +1ms
[Nest] 33172  - 04/03/2025, 12:01:55 PM     LOG [RouterExplorer] Mapped {/orders, GET} route +1ms
[Nest] 33172  - 04/03/2025, 12:01:55 PM     LOG [RouterExplorer] Mapped {/orders/:id, GET} route +1ms
[Nest] 33172  - 04/03/2025, 12:01:55 PM     LOG [RouterExplorer] Mapped {/orders/:id, PATCH} route +1ms
[Nest] 33172  - 04/03/2025, 12:01:55 PM     LOG [RouterExplorer] Mapped {/orders/:id, DELETE} route +1ms
[Nest] 33172  - 04/03/2025, 12:01:55 PM     LOG [RoutesResolver] OrderItemsController {/order-items}: +0ms
[Nest] 33172  - 04/03/2025, 12:01:55 PM     LOG [RouterExplorer] Mapped {/order-items, GET} route +1ms
[Nest] 33172  - 04/03/2025, 12:01:55 PM     LOG [RouterExplorer] Mapped {/order-items/order/:orderId, GET} route +1ms
[Nest] 33172  - 04/03/2025, 12:01:55 PM     LOG [RouterExplorer] Mapped {/order-items/:id, DELETE} route +0ms
[Nest] 33172  - 04/03/2025, 12:01:56 PM     LOG [NestApplication] Nest application successfully started +900ms
```

Users(/users)

```
curl -X POST http://localhost:3000/users -H "Content-Type:
application/json" -d '{

  "name": "John Doe",

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

  "phone": "1234567890",

  "password": "securepassword"

}'
```



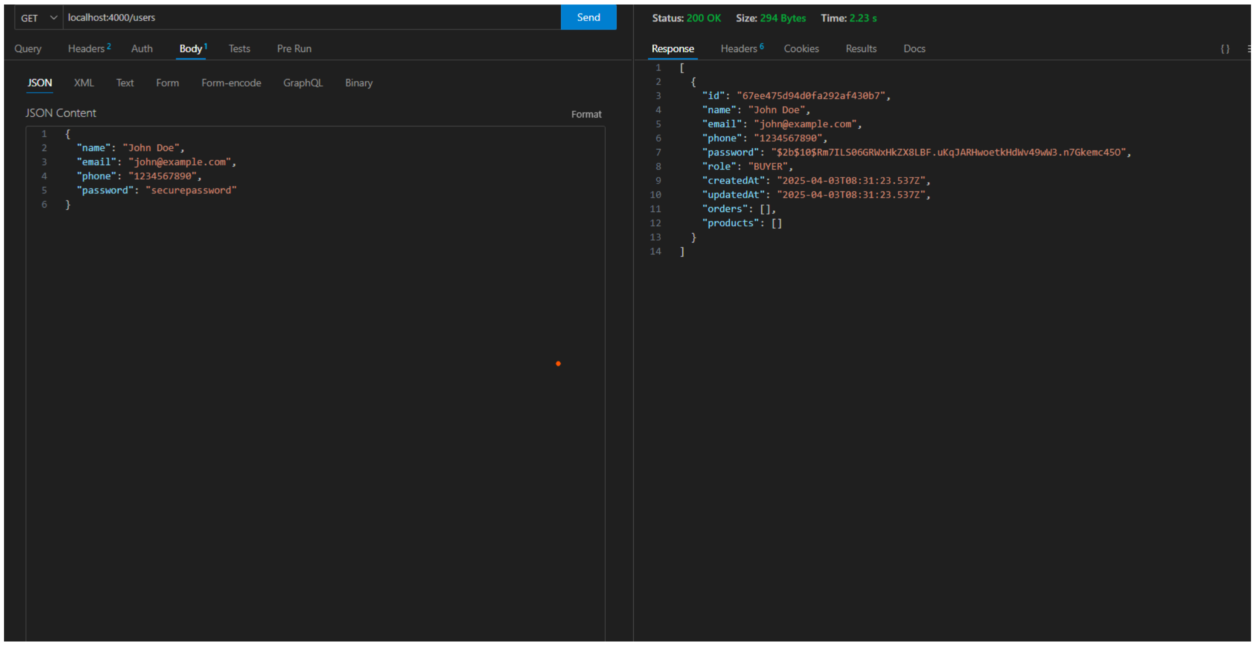
```
{
  "name": "John Doe",
  "email": "john@example.com",
  "phone": "1234567890",
  "password": "securepassword"
}
```

Status: 201 Created   Size: 292 Bytes   Time: 2.94 s

```
{
  "id": "67ee475d94d0fa292af430b7",
  "name": "John Doe",
  "email": "john@example.com",
  "phone": "1234567890",
  "password": "$2b$10$Rm7ILS06GRWxHkZX8LBF.uKqJARHwoetkHdWv49wW3.n7Gkemc45O",
  "role": "BUYER",
  "createdAt": "2025-04-03T08:31:23.537Z",
  "updatedAt": "2025-04-03T08:31:23.537Z",
  "orders": [],
  "products": []
}
```

Get users

curl -X GET http://localhost:3000/users



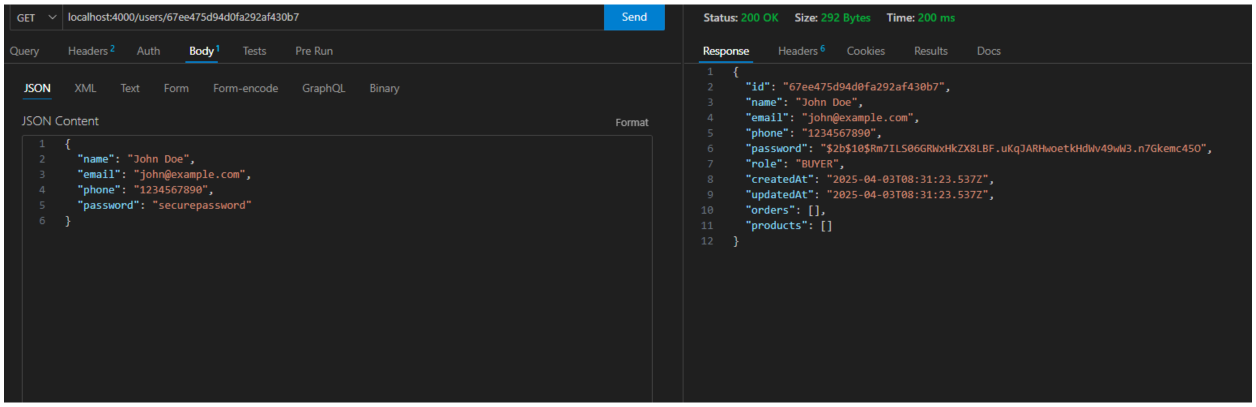
```
{
  "name": "John Doe",
  "email": "john@example.com",
  "phone": "1234567890",
  "password": "securepassword"
}
```

Status: 200 OK Size: 294 Bytes Time: 2.23 s

```
[
  {
    "id": "67ee475d94d0fa292af430b7",
    "name": "John Doe",
    "email": "john@example.com",
    "phone": "1234567890",
    "password": "$2b$10$Rm7ILS06GRWxHkZX8LBF.uKqJARHwoetkHdWv49wW3.n7Gkemc450",
    "role": "BUYER",
    "createdAt": "2025-04-03T08:31:23.537Z",
    "updatedAt": "2025-04-03T08:31:23.537Z",
    "orders": [],
    "products": []
  }
]
```

Get single user

curl -X GET http://localhost:3000/users/{id}



```
{
  "name": "John Doe",
  "email": "john@example.com",
  "phone": "1234567890",
  "password": "securepassword"
}
```

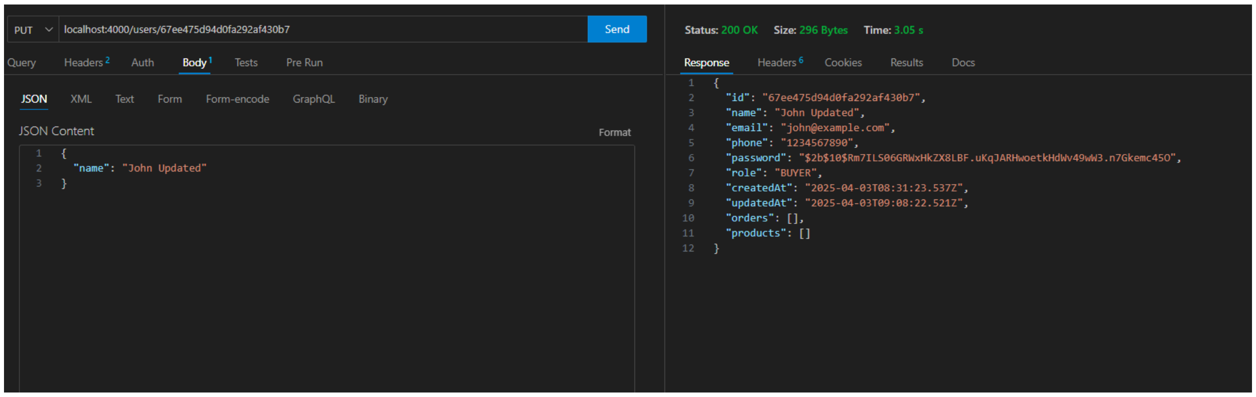
Status: 200 OK Size: 292 Bytes Time: 200 ms

```
{
  "id": "67ee475d94d0fa292af430b7",
  "name": "John Doe",
  "email": "john@example.com",
  "phone": "1234567890",
  "password": "$2b$10$Rm7ILS06GRWxHkZX8LBF.uKqJARHwoetkHdWv49wW3.n7Gkemc450",
  "role": "BUYER",
  "createdAt": "2025-04-03T08:31:23.537Z",
  "updatedAt": "2025-04-03T08:31:23.537Z",
  "orders": [],
  "products": []
}
```

Update user

curl -X PUT http://localhost:3000/users/{id} -H "Content-Type: application/json" -d '{

```
  "name": "John Updated"
}'
```



PUT localhost:4000/users/67ee475d94d0fa292af430b7

```
{
  "name": "John Updated"
}
```

Status: 200 OK Size: 296 Bytes Time: 3.05 s

```
{
  "id": "67ee475d94d0fa292af430b7",
  "name": "John Updated",
  "email": "john@example.com",
  "phone": "1234567890",
  "password": "$2b$10$Rm7ILS06GRWxHkZX8LBF.uKqJARHwoetkHdWv49wW3.n7Gkemc45O",
  "role": "BUYER",
  "createdAt": "2025-04-03T08:31:23.537Z",
  "updatedAt": "2025-04-03T09:08:22.521Z",
  "orders": [],
  "products": []
}
```

Authentication

Signup (POST)

- **URL: {{BASE_URL}}/auth/signup**
- **Body (JSON):**

```
{
  "name": "John Doe",
  "email": "johndoe@example.com",
  "phone": "1234567890",
  "password": "securepassword",
  "role": "BUYER"
}
```

Expected Response: Returns user data.

Login (POST)

- **URL: {{BASE_URL}}/auth/login**
- **Body (JSON):**

```
{
  "email": "johndoe@example.com",
  "password": "securepassword"
}
```

Expected Response: Returns { "accessToken": "JWT_TOKEN" }

Action: Save accessToken in TOKEN environment variable

To test all the instances (User, Product, Order, etc.) in Postman using NestJS with Prisma and MongoDB, follow these steps:

---

1. Setup Postman Collection

1. **Create a new collection** in Postman named **NestJS API**.
2. **Set Environment Variables** in Postman:
    - `BASE_URL`: `http://localhost:3000` (or your deployed API URL)
    - `TOKEN`: *(leave empty; will be updated after login)*

---

2. Authentication (Auth)

Signup (POST)

URL: `{{BASE_URL}}/auth/signup`

Body (JSON):

```
{
  "name": "John Doe",
  "email": "johndoe@example.com",
  "phone": "1234567890",
  "password": "securepassword",
  "role": "BUYER"
}
```

Expected Response: Returns user data.

Login (POST)

URL: `{{BASE_URL}}/auth/login`

Body (JSON):

```
{
  "email": "johndoe@example.com",
  "password": "securepassword"
}
```

Expected Response: Returns `{ "accessToken": "JWT_TOKEN" }`

Action: Save `accessToken` in `TOKEN` environment variable.

---

3. Users

Get All Users (GET)

URL: `{{BASE_URL}}/users`

Headers:

```
{
  "Authorization": "Bearer {{TOKEN}}"
}
```

Expected Response: List of all users.

Get Single User by ID (GET)

URL: `{{BASE_URL}}/users/{userId}`

Update User (PATCH)

URL: `{{BASE_URL}}/users/{userId}`

Body (JSON):

```
{
  "name": "Updated Name",
  "phone": "0987654321"
}
```

Delete User (DELETE)

URL: `{{BASE_URL}}/users/{userId}`

---

4. Categories

Create Category (POST)

URL: `{{BASE_URL}}/categories`

Body (JSON):

```
{
  "name": "Electronics",
  "subCategories": []
}
```

Get Categories (GET)

- **URL:** `{{BASE_URL}}/categories`

---

5. SubCategories

Create SubCategory (POST)

URL: `{{BASE_URL}}/subcategories`

Body (JSON):

```
{
  "name": "Laptops",
  "categoryId": "{categoryId}"
}
```

Get SubCategories (GET)

- **URL:** `{{BASE_URL}}/subcategories`

---

6. Products

Create Product (POST)

- **URL:** `{{BASE_URL}}/products`
- **Body (JSON):**

```
{
  "name": "MacBook Pro",
  "description": "Latest Apple Laptop",
  "price": 2499.99,
  "stock": 10,
  "images": ["https://imageurl.com/macbook"],
  "vendorId": "{vendorId}",
  "subCategoryId": "{subCategoryId}"
}
```

Get All Products (GET)

- **URL:** `{{BASE_URL}}/products`

Update Product (PATCH)

URL: `{{BASE_URL}}/products/{productId}`

Body:

```
{
  "price": 2399.99,
  "stock": 8
}
```

Delete Product (DELETE)

- **URL:** `{{BASE_URL}}/products/{productId}`

---

7. Orders

Create Order (POST)

URL: `{{BASE_URL}}/orders`

Body (JSON):

```
{
  "userId": "{userId}",
  "products": ["{productId}"],
  "total": 2499.99
}
```

Get Orders (GET)

URL: `{{BASE_URL}}/orders`

Update Order Status (PATCH)

URL: `{{BASE_URL}}/orders/{orderId}`

Body (JSON):

```
{
  "status": "SHIPPED"
}
```

---

8. Order Items

Create Order Item (POST)

URL: `{{BASE_URL}}/order-items`

Body (JSON):

```
{
  "orderId": "{orderId}",
  "productId": "{productId}",
  "quantity": 2
}
```

Get Order Items (GET)

URL: `{{BASE_URL}}/order-items`

---

9. Sales

Create Sale (POST)

URL: `{{BASE_URL}}/sales`

Body (JSON):

```
{
  "productId": "{productId}",
  "vendorId": "{vendorId}",
  "amount": 2499.99
}
```

Get Sales (GET)

- **URL:** `{{BASE_URL}}/sales`

---

10. Dashboards

Get Admin Dashboard (GET)

- **URL:** `{{BASE_URL}}/admin-dashboard/{adminId}`

Get Vendor Dashboard (GET)

- **URL:** `{{BASE_URL}}/vendor-dashboard/{vendorId}`

Postman Testing Strategy

- Use **Pre-request Script** in Postman to fetch and store `accessToken` automatically after login.
- Group API calls by **folders** in the Postman collection (Auth, Users, Products, Orders, etc.).
- Add **Tests** in Postman to check response status codes and expected data.
