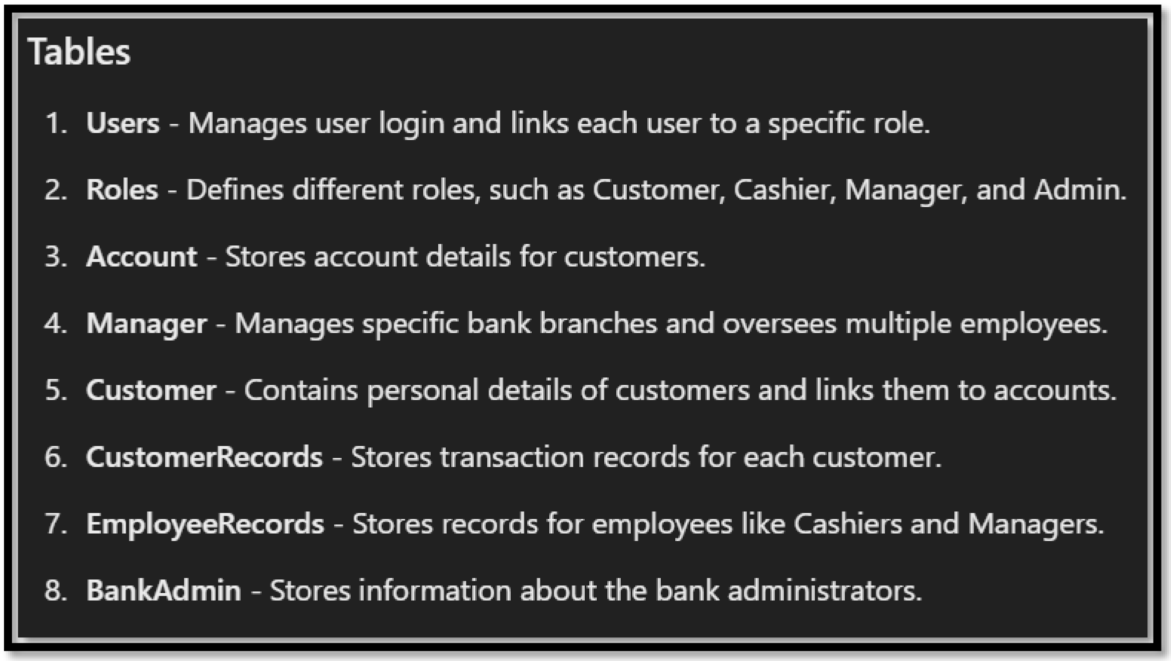
Bank Management Usecase

1. **Bank Management Usecase:** The bank management system will be able to handle new registration of customers, managers and employees. It also helps to maintain the fees structure of students etc. The relationships between tables are designed to capture real-world associations, like teachers belonging to particular department, multiple students linked to same teacher, and so on. As an initial MVP you are required to develop a restful API backend application in springboot.

   Here is the requirement for the application

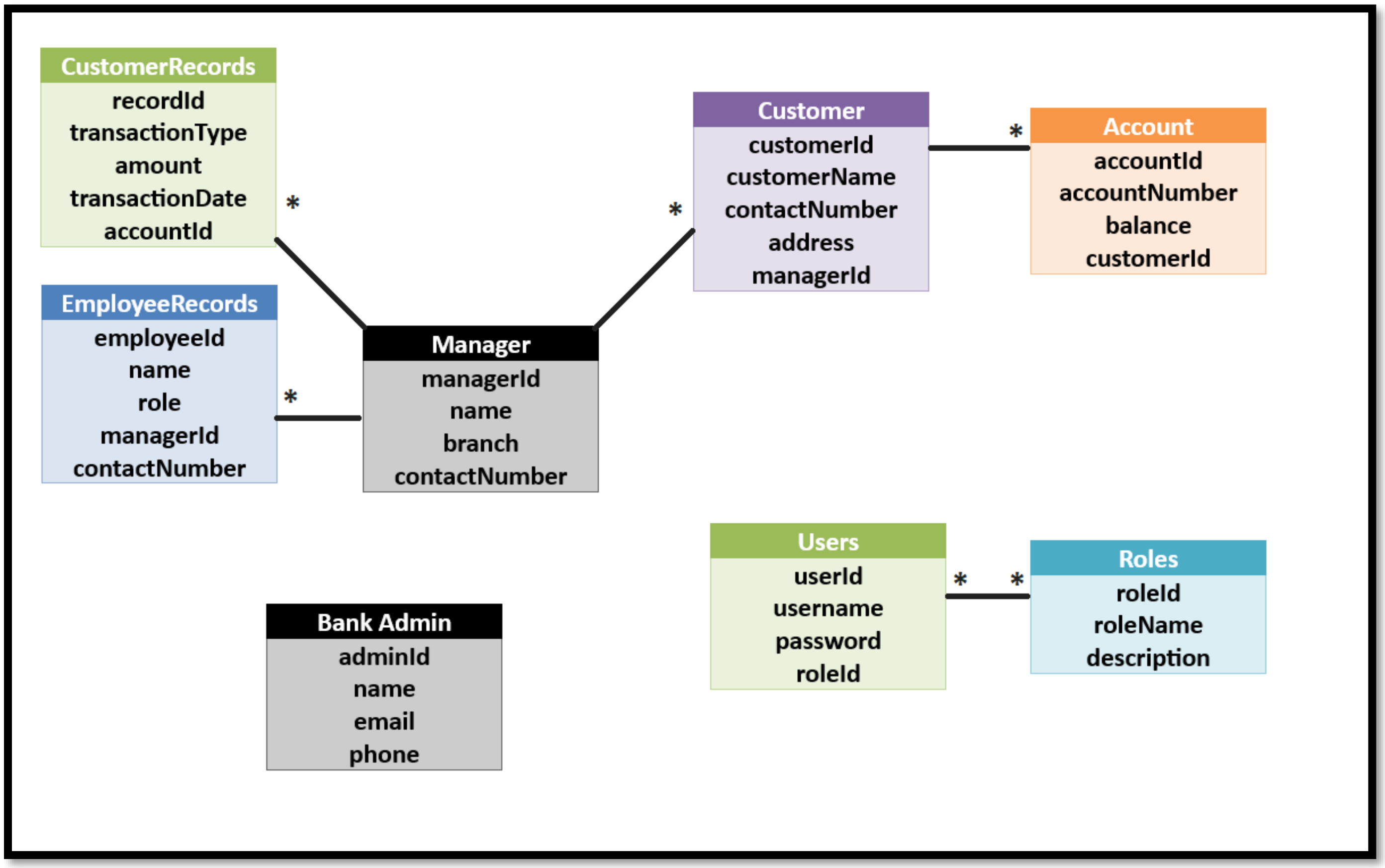
   A. Databases present in the application



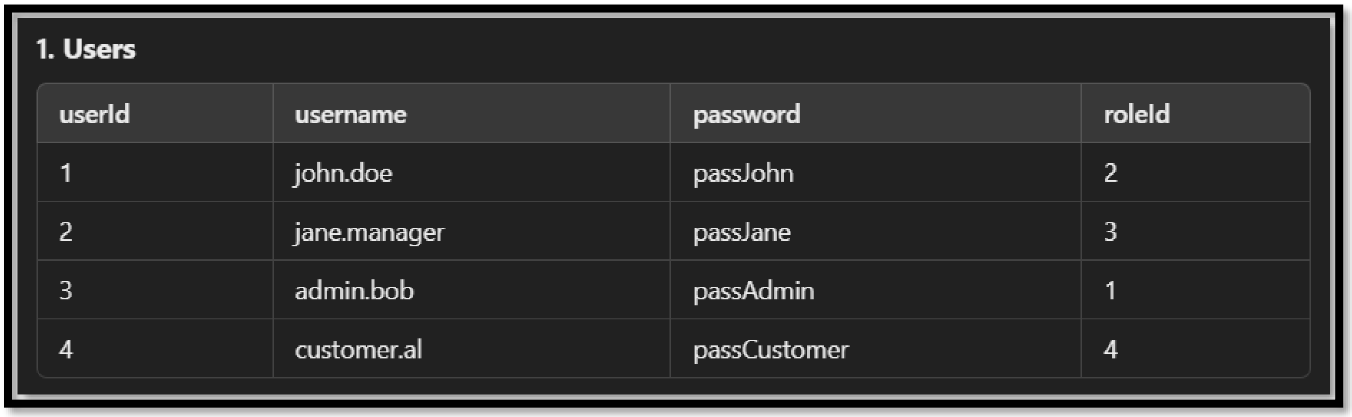
Tables

1. **Users** - Manages user login and links each user to a specific role.
2. **Roles** - Defines different roles, such as Customer, Cashier, Manager, and Admin.
3. **Account** - Stores account details for customers.
4. **Manager** - Manages specific bank branches and oversees multiple employees.
5. **Customer** - Contains personal details of customers and links them to accounts.
6. **CustomerRecords** - Stores transaction records for each customer.
7. **EmployeeRecords** - Stores records for employees like Cashiers and Managers.
8. **BankAdmin** - Stores information about the bank administrators.

B. Relationships between databases present in the application

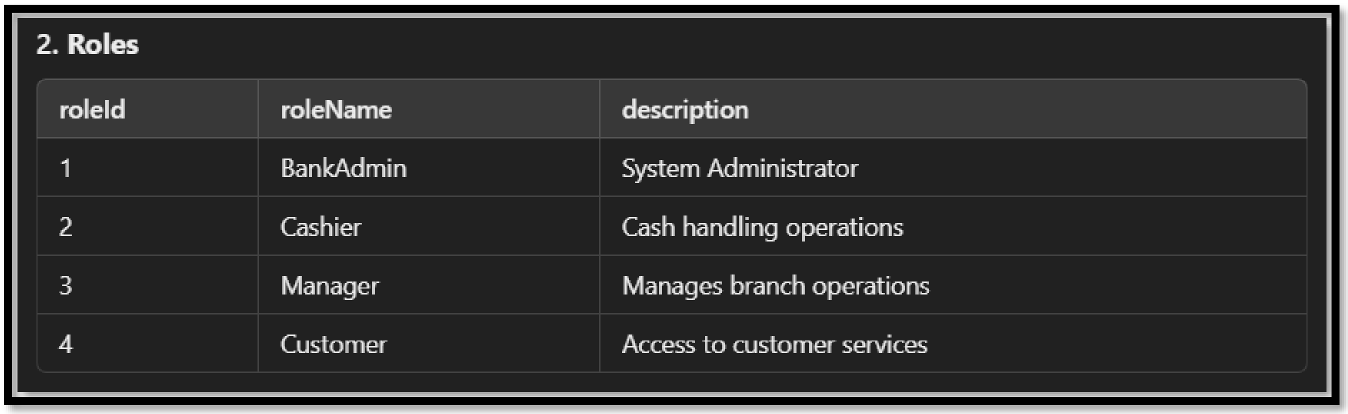


C. Populate the table with the below data



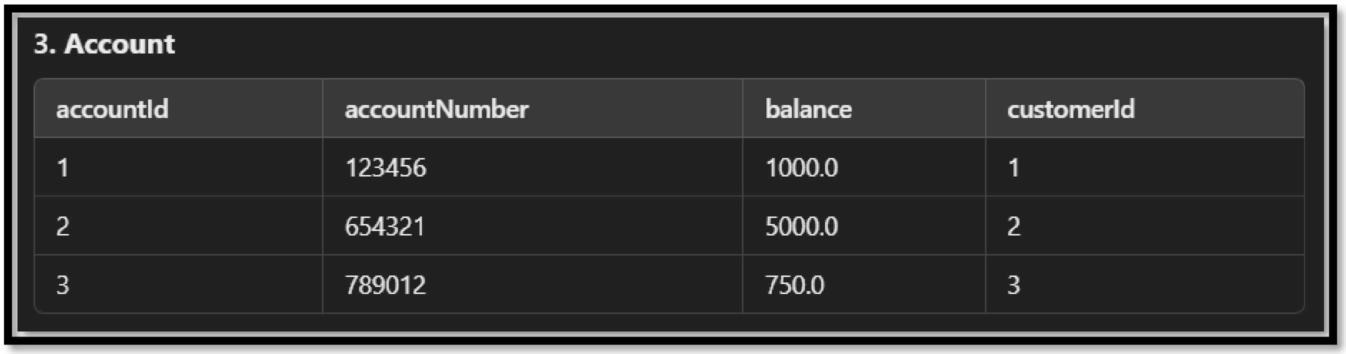
1. Users

userId	username	password	roleId
1	john.doe	passJohn	2
2	jane.manager	passJane	3
3	admin.bob	passAdmin	1
4	customer.al	passCustomer	4



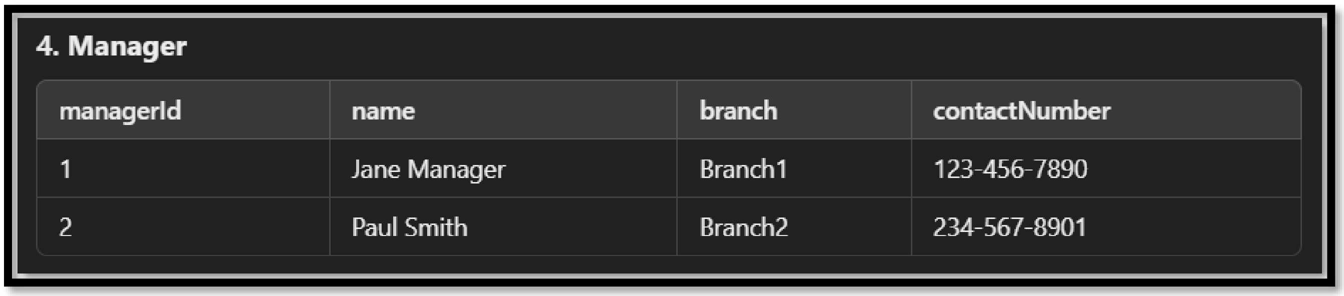
2. Roles

roleId	roleName	description
1	BankAdmin	System Administrator
2	Cashier	Cash handling operations
3	Manager	Manages branch operations
4	Customer	Access to customer services



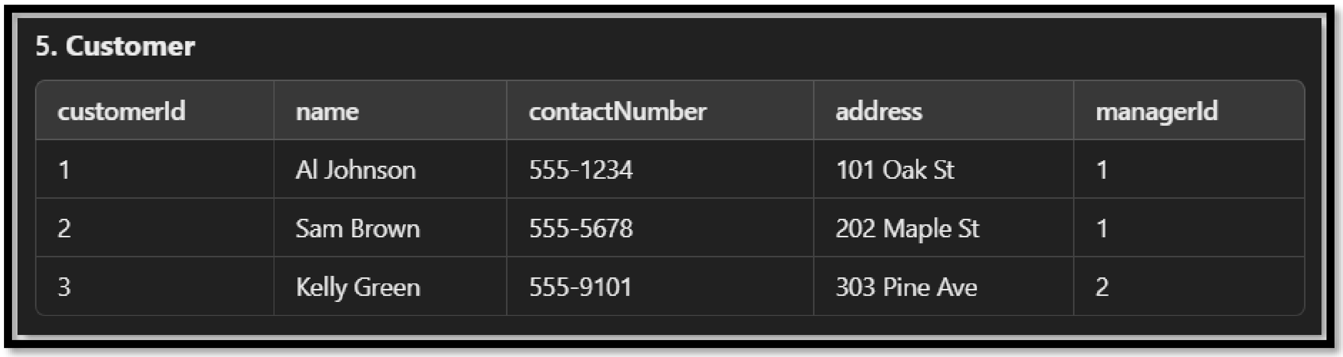
3. Account

accountId	accountNumber	balance	customerId
1	123456	1000.0	1
2	654321	5000.0	2
3	789012	750.0	3



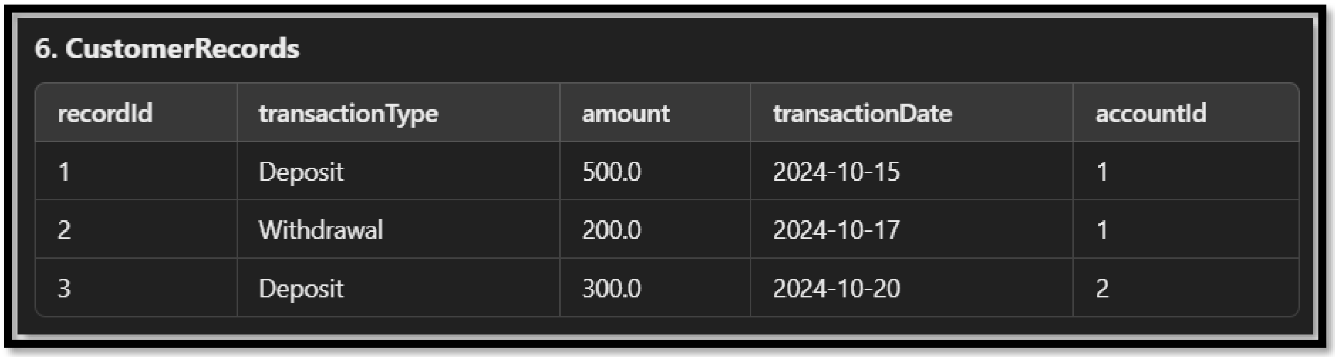
4. Manager

managerId	name	branch	contactNumber
1	Jane Manager	Branch1	123-456-7890
2	Paul Smith	Branch2	234-567-8901



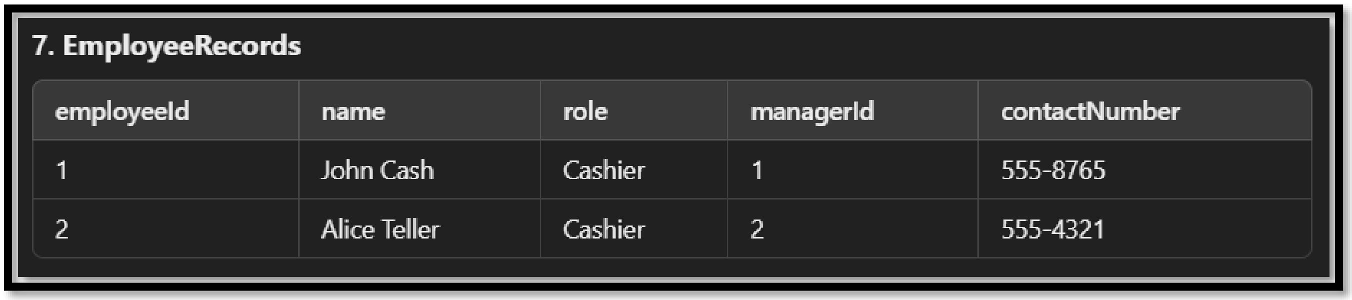
5. Customer

customerId	name	contactNumber	address	managerId
1	Al Johnson	555-1234	101 Oak St	1
2	Sam Brown	555-5678	202 Maple St	1
3	Kelly Green	555-9101	303 Pine Ave	2



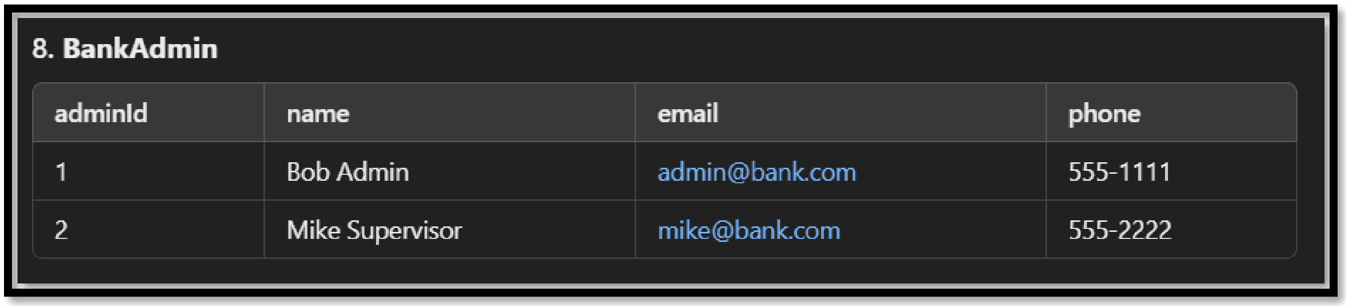
6. CustomerRecords

recordId	transactionType	amount	transactionDate	accountId
1	Deposit	500.0	2024-10-15	1
2	Withdrawal	200.0	2024-10-17	1
3	Deposit	300.0	2024-10-20	2



7. EmployeeRecords

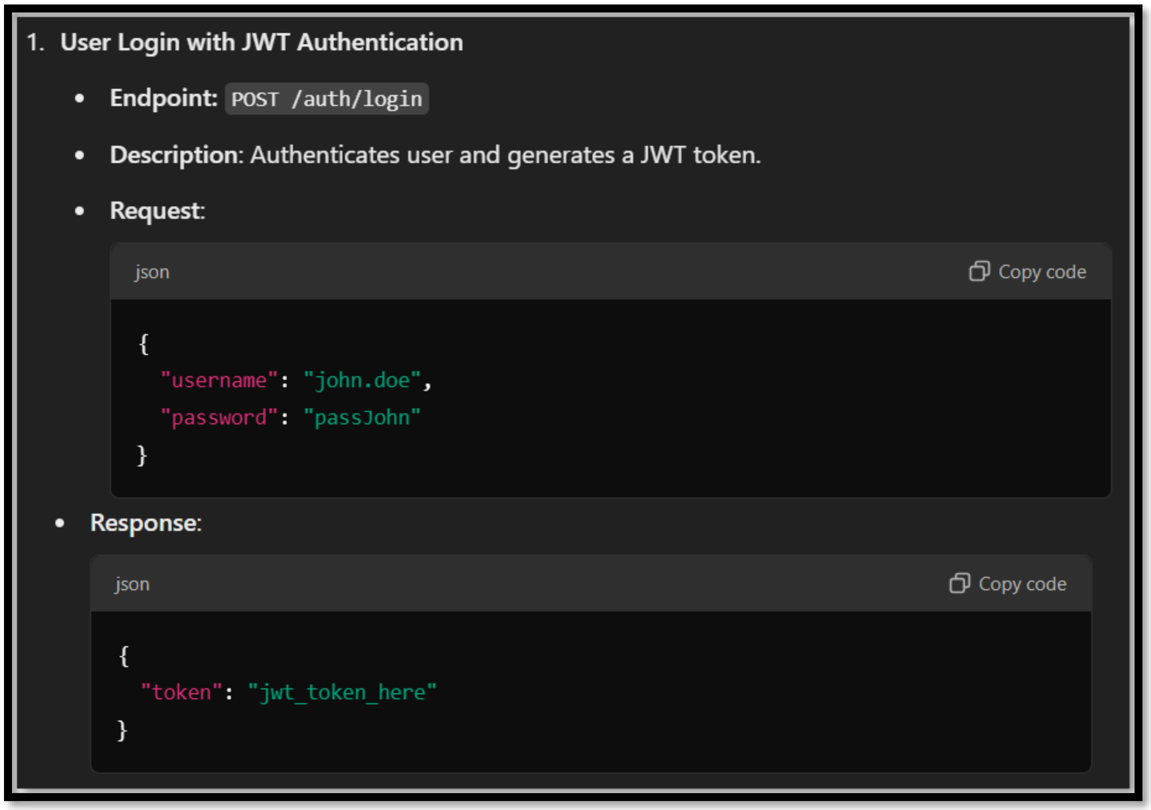
employeeId	name	role	managerId	contactNumber
1	John Cash	Cashier	1	555-8765
2	Alice Teller	Cashier	2	555-4321



8. **BankAdmin**

adminId	name	email	phone
1	Bob Admin	admin@bank.com	555-1111
2	Mike Supervisor	mike@bank.com	555-2222

D. Endpoint details are as shown below

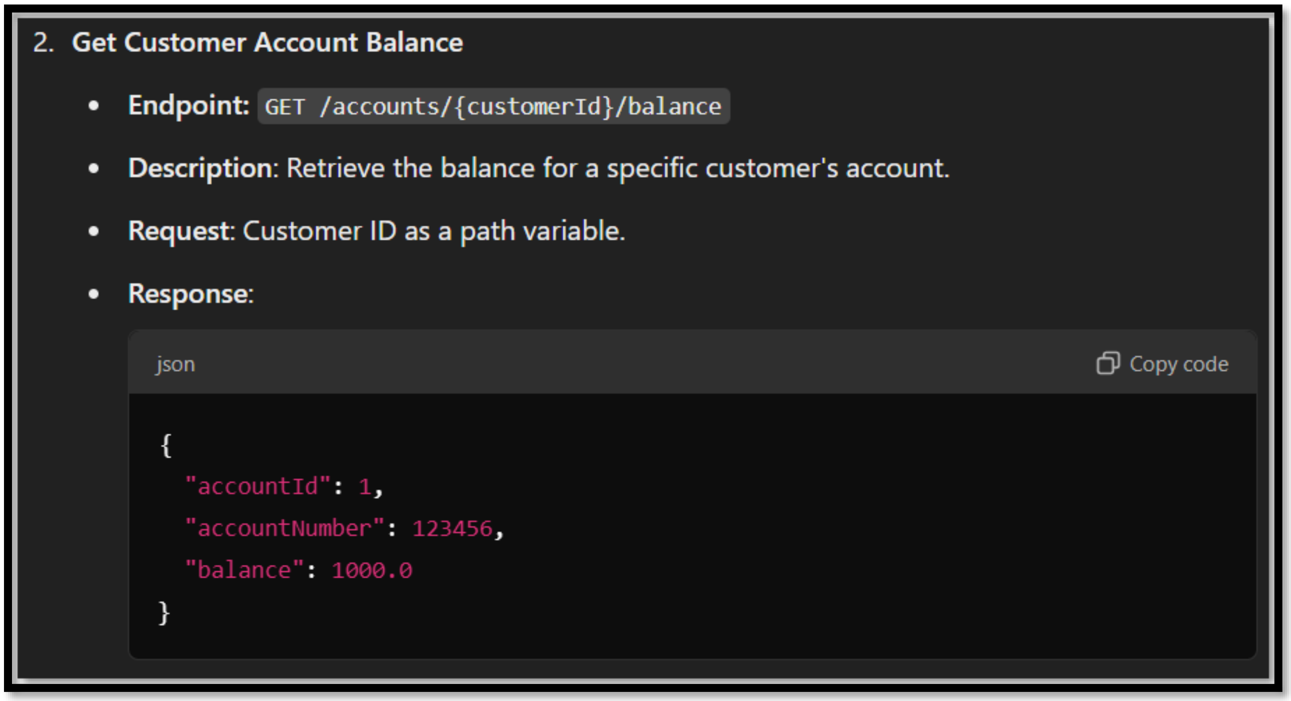


1. **User Login with JWT Authentication**
   - **Endpoint:** `POST /auth/login`
   - **Description**: Authenticates user and generates a JWT token.
   - **Request**:

```json
{
  "username": "john.doe",
  "password": "passJohn"
}
```

- **Response**:

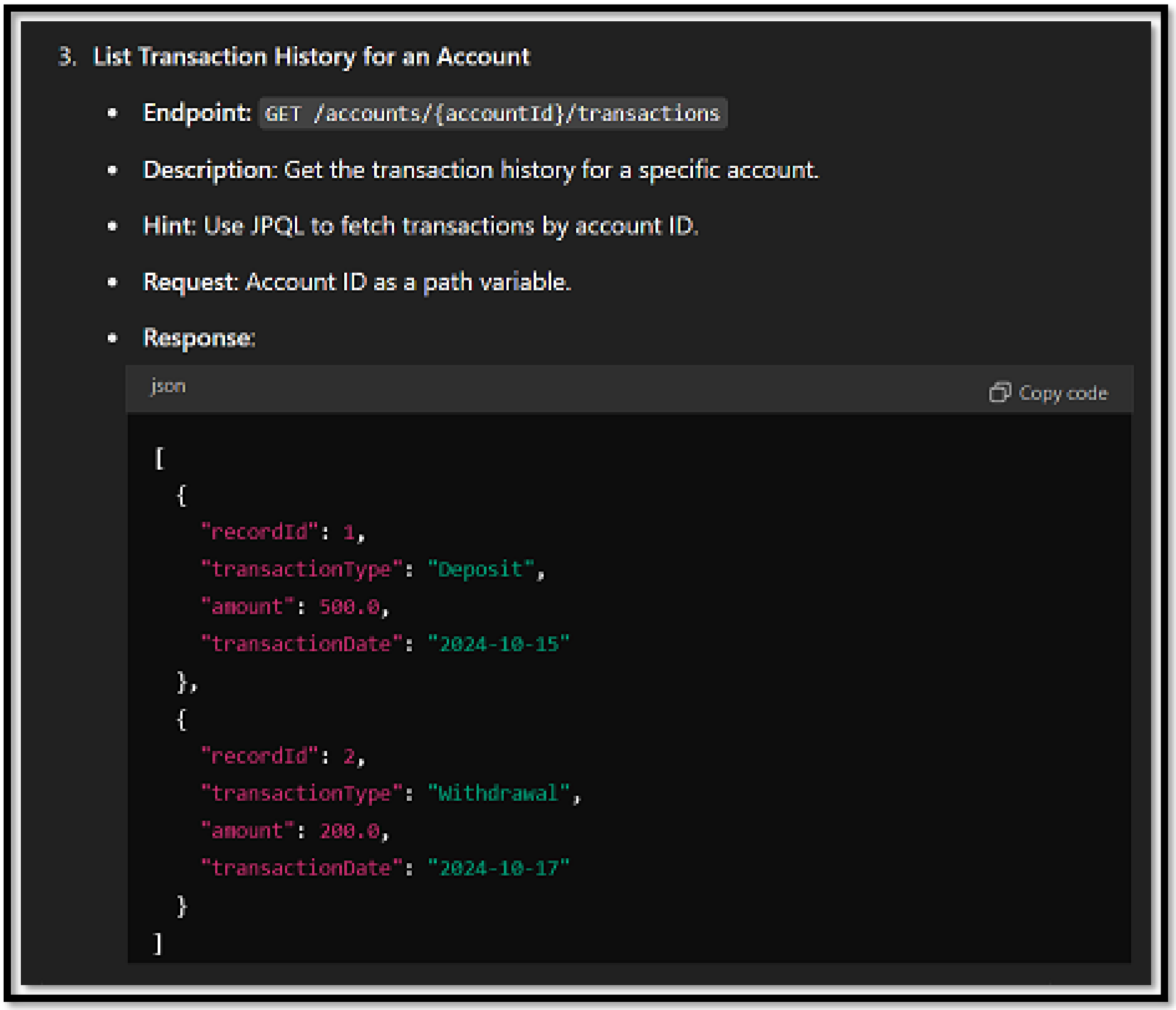
```json
{
  "token": "jwt_token_here"
}
```



2. **Get Customer Account Balance**

- **Endpoint:** `GET /accounts/{customerId}/balance`
- **Description**: Retrieve the balance for a specific customer's account.
- **Request**: Customer ID as a path variable.
- **Response**:

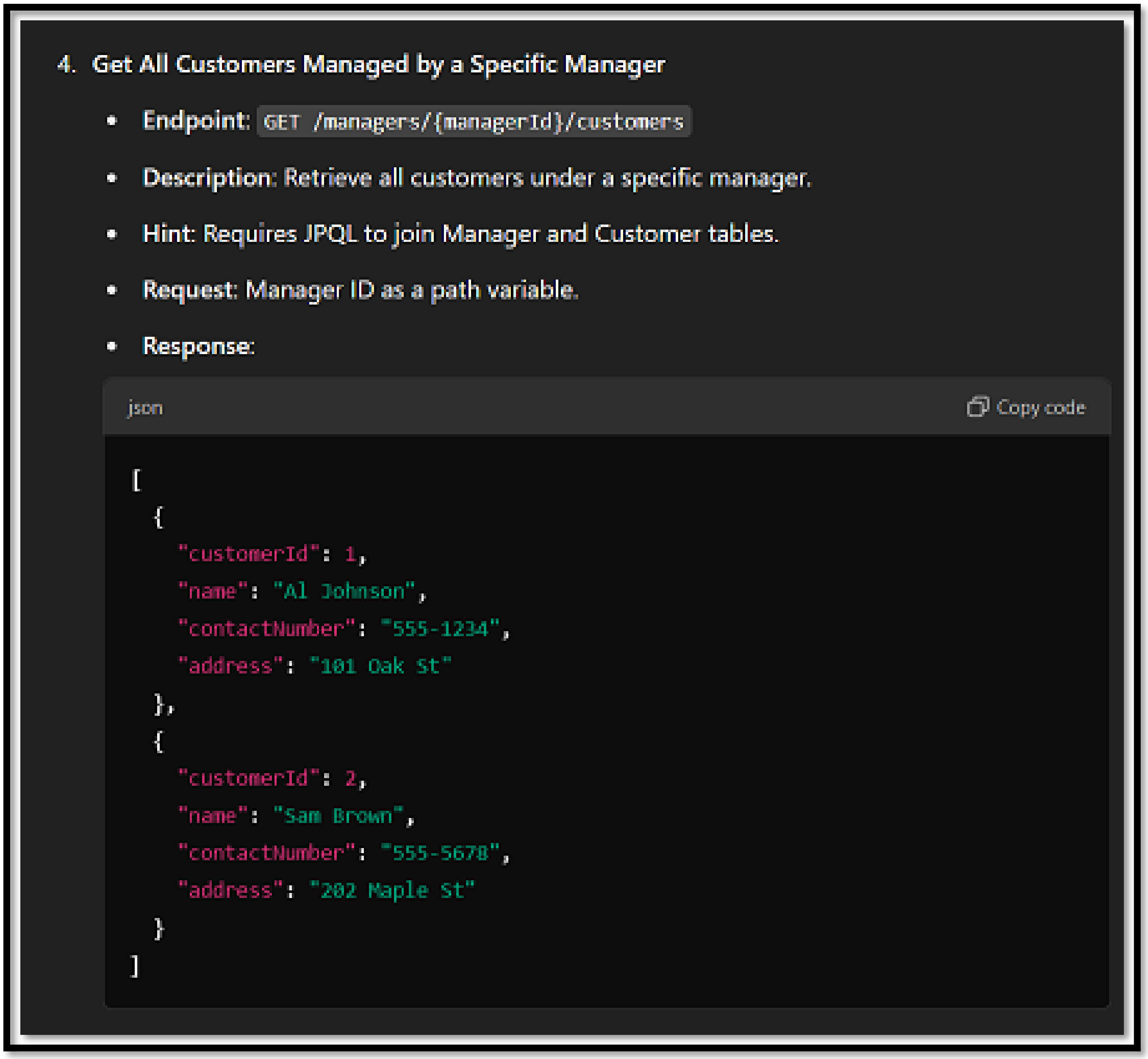
```json
{
  "accountId": 1,
  "accountNumber": 123456,
  "balance": 1000.0
}
```



3. **List Transaction History for an Account**

- **Endpoint:** `GET /accounts/{accountId}/transactions`
- **Description:** Get the transaction history for a specific account.
- **Hint:** Use JPQL to fetch transactions by account ID.
- **Request:** Account ID as a path variable.
- **Response:**

```json
[
  {
    "recordId": 1,
    "transactionType": "Deposit",
    "amount": 500.0,
    "transactionDate": "2024-10-15"
  },
  {
    "recordId": 2,
    "transactionType": "Withdrawal",
    "amount": 200.0,
    "transactionDate": "2024-10-17"
  }
]
```

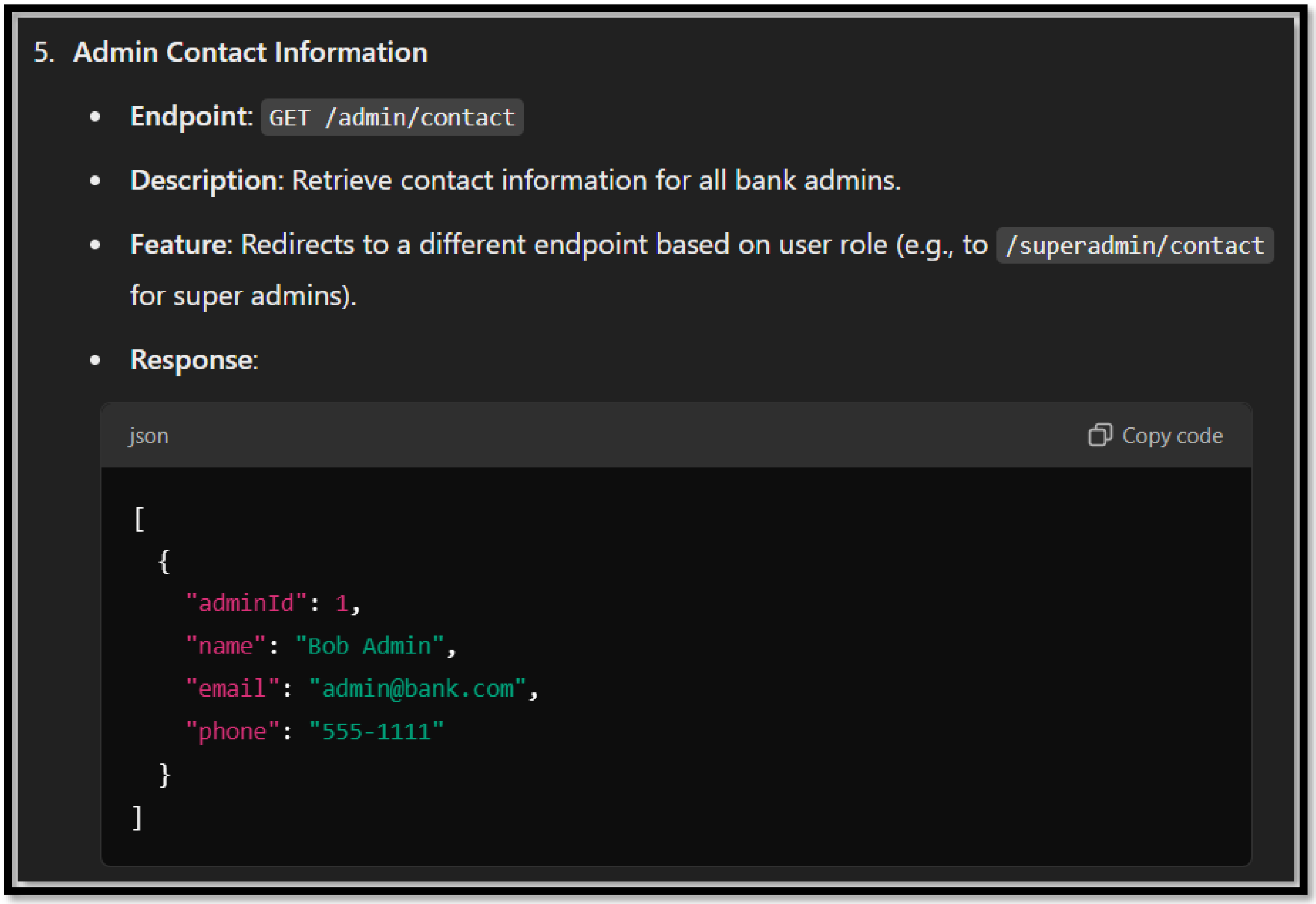


4. Get All Customers Managed by a Specific Manager

- **Endpoint:** `GET /managers/{managerId}/customers`
- **Description:** Retrieve all customers under a specific manager.
- **Hint:** Requires JPQL to join Manager and Customer tables.
- **Request:** Manager ID as a path variable.
- **Response:**

json Copy code

```json
[
  {
    "customerId": 1,
    "name": "Al Johnson",
    "contactNumber": "555-1234",
    "address": "101 Oak St"
  },
  {
    "customerId": 2,
    "name": "Sam Brown",
    "contactNumber": "555-5678",
    "address": "202 Maple St"
  }
]
```

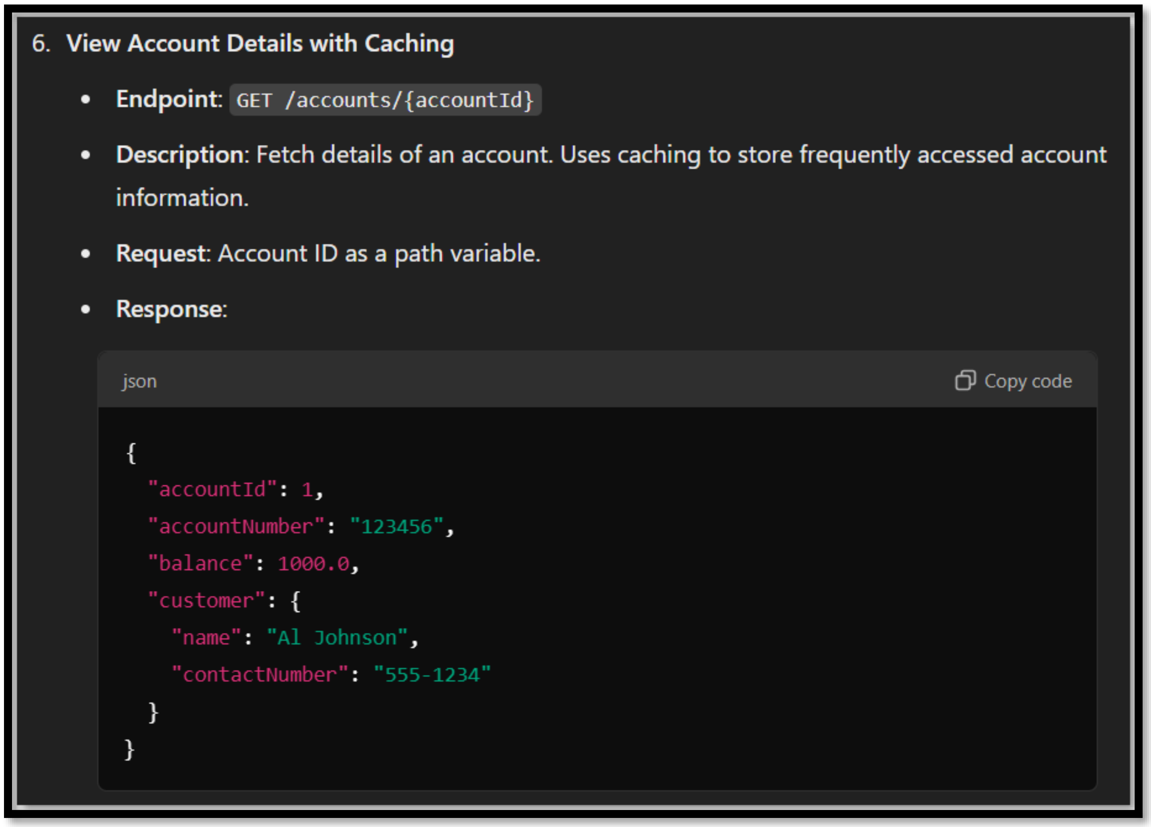


5. **Admin Contact Information**

- **Endpoint**: `GET /admin/contact`
- **Description**: Retrieve contact information for all bank admins.
- **Feature**: Redirects to a different endpoint based on user role (e.g., to `/superadmin/contact` for super admins).
- **Response**:

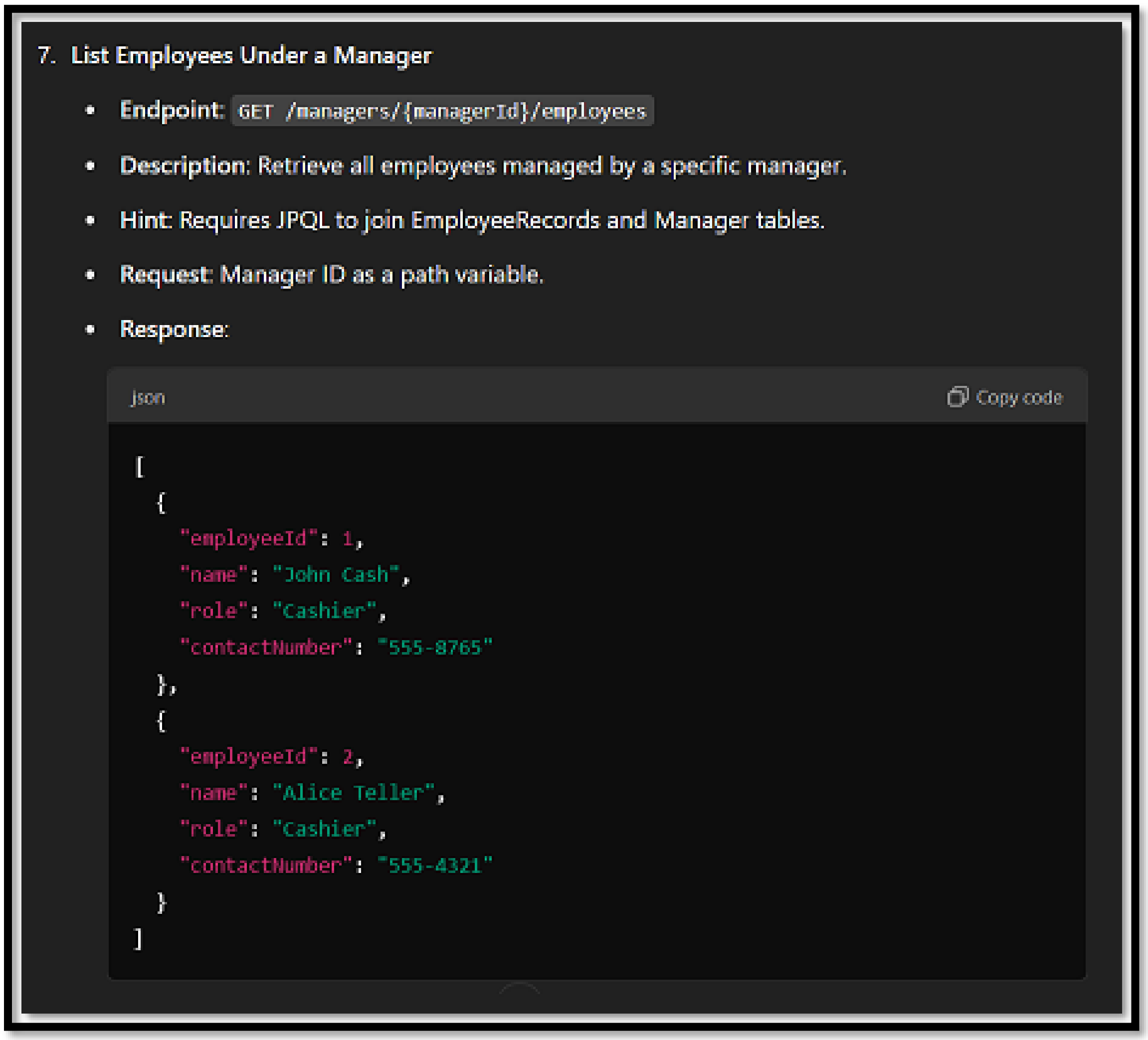
json Copy code

```json
[
  {
    "adminId": 1,
    "name": "Bob Admin",
    "email": "admin@bank.com",
    "phone": "555-1111"
  }
]
```


6. View Account Details with Caching

- **Endpoint**: `GET /accounts/{accountId}`
- **Description**: Fetch details of an account. Uses caching to store frequently accessed account information.
- **Request**: Account ID as a path variable.
- **Response**:

```json
{
  "accountId": 1,
  "accountNumber": "123456",
  "balance": 1000.0,
  "customer": {
    "name": "Al Johnson",
    "contactNumber": "555-1234"
  }
}
```


7. List Employees Under a Manager

- **Endpoint**: `GET /managers/{managerId}/employees`
- **Description**: Retrieve all employees managed by a specific manager.
- **Hint**: Requires JPQL to join EmployeeRecords and Manager tables.
- **Request**: Manager ID as a path variable.
- **Response**:

```json
[
  {
    "employeeId": 1,
    "name": "John Cash",
    "role": "Cashier",
    "contactNumber": "555-8765"
  },
  {
    "employeeId": 2,
    "name": "Alice Teller",
    "role": "Cashier",
    "contactNumber": "555-4321"
  }
]
```