Bank Ledger Microservice

Starter Application

No starter application will be provided; you will need to build it from scratch.

Sonar Qube must be installed in your machine. Instruction to be provided by your faculty.

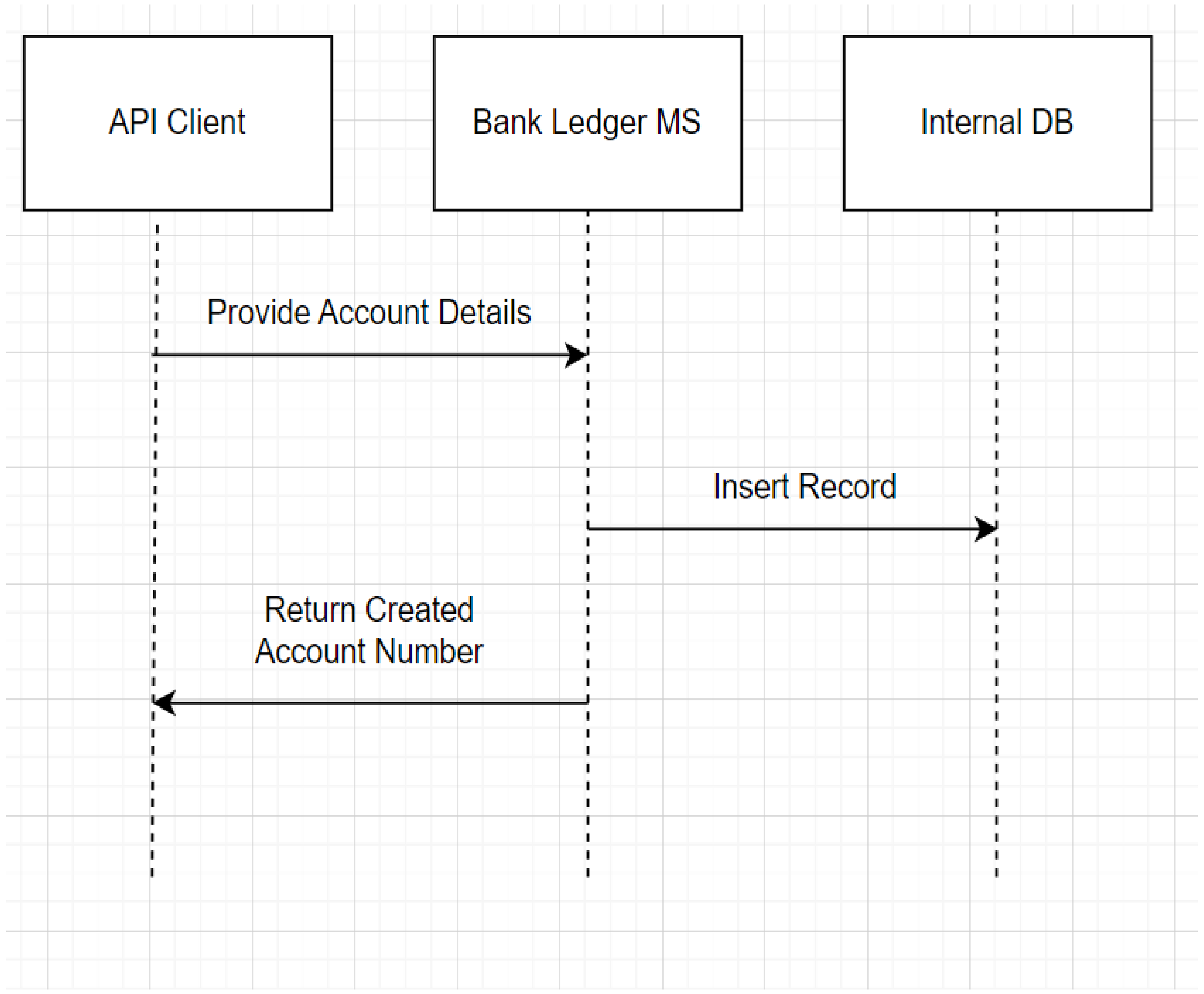
What are the objectives?

Develop an application that:

- Saves an account information in a database
- Integrate SonarQube into the development process to perform static code analysis, ensuring the application meets high standards of code quality,

Sequence Diagram

- The API client (Postman) sends account details to the Bank Ledger Microservice
- Bank Ledger MS will insert the record
- If the insertion is successful, the account number will be returned to the client.

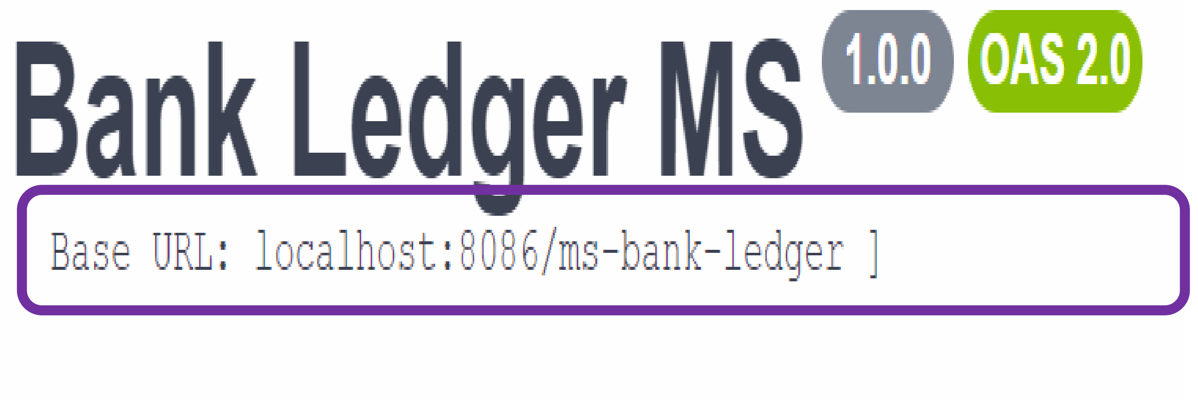


Swagger: Base URL

Base URL Declaration:

- The @RequestMapping("/ms-bank-ledger") at the class level sets the base URL for all endpoints within the BankLedgerController.

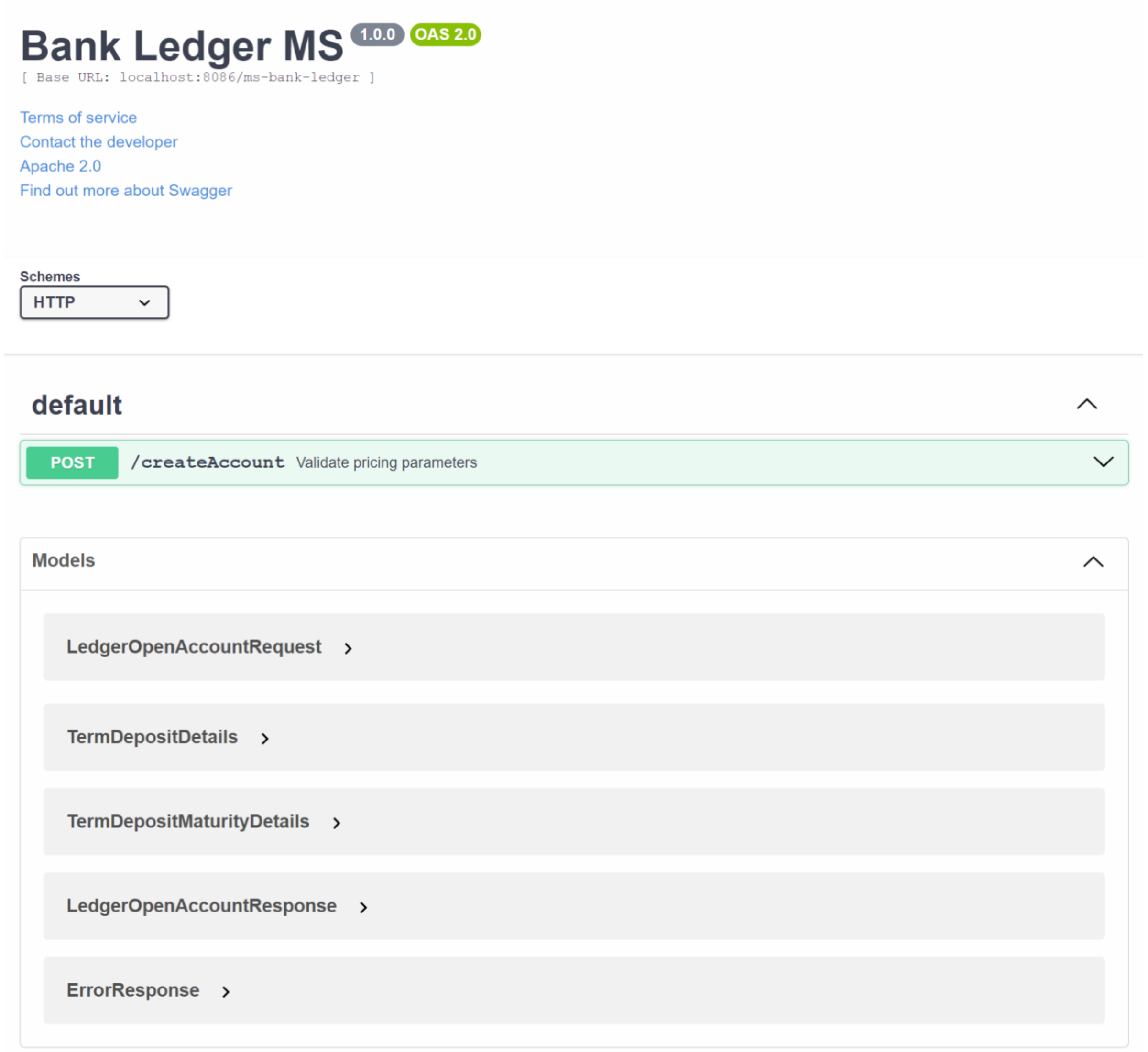
- This means that all endpoints in this controller will start with /ms-bank-ledger, resulting in the full base URL localhost:8086/ms-bank-ledger.



```
Base URL: localhost:8086/ms-bank-ledger ]
```

Endpoints

- **Base URL:** /ms-bank-ledger (defined at the class level).
- **Endpoint URL:** /createAccount (defined at the method level).
- **Function:** creates the Account details



Bank Ledger MS 1.0.0 OAS 2.0

[Base URL: localhost:8086/ms-bank-ledger]

Terms of service
Contact the developer
Apache 2.0
Find out more about Swagger

Schemes: HTTP

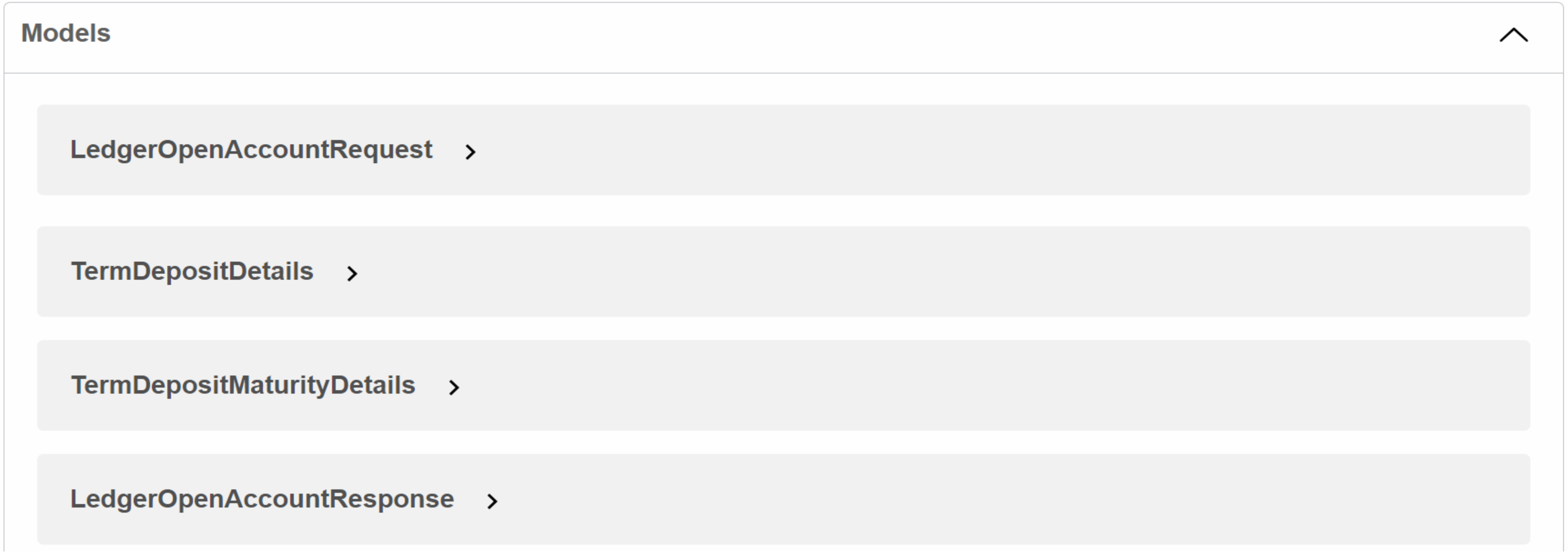
default

POST `/createAccount` Validate pricing parameters

Models

- LedgerOpenAccountRequest
- TermDepositDetails
- TermDepositMaturityDetails
- LedgerOpenAccountResponse
- ErrorResponse

Models



Models

LedgerOpenAccountRequest

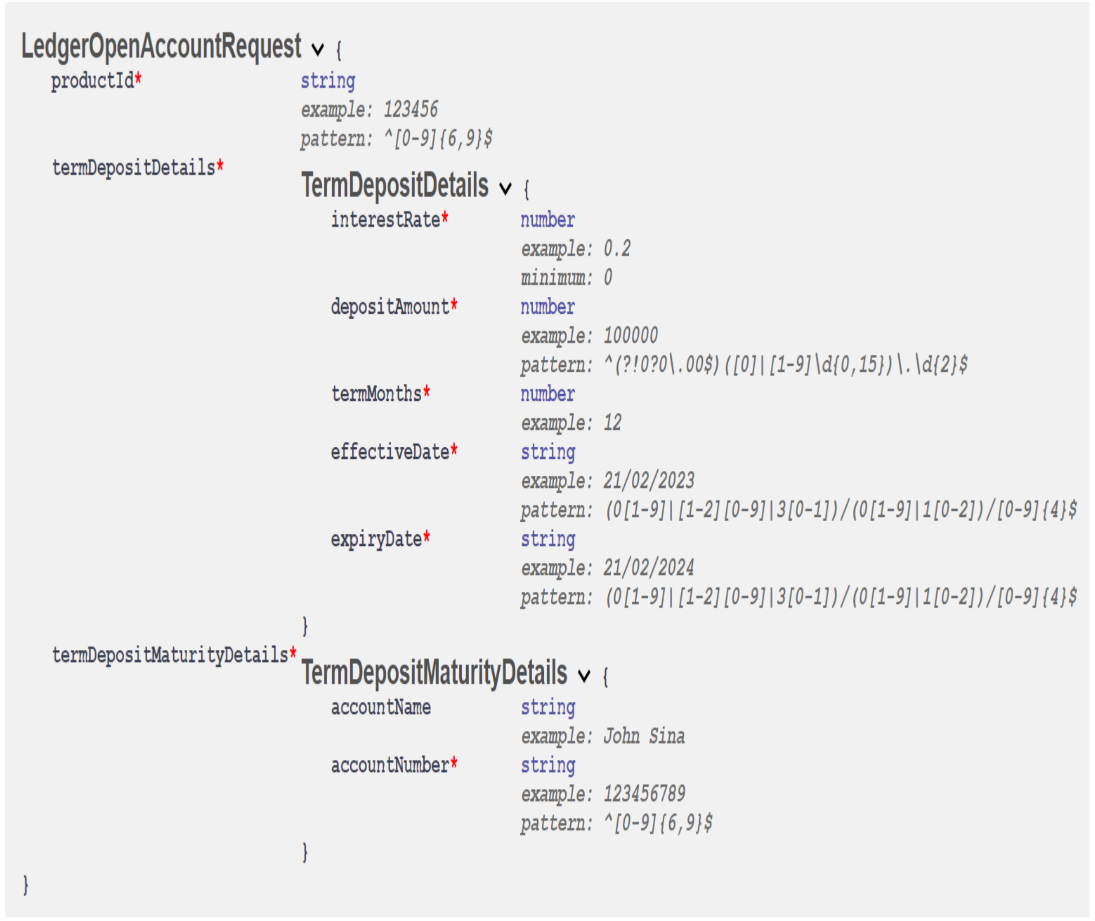
TermDepositDetails

TermDepositMaturityDetails

LedgerOpenAccountResponse

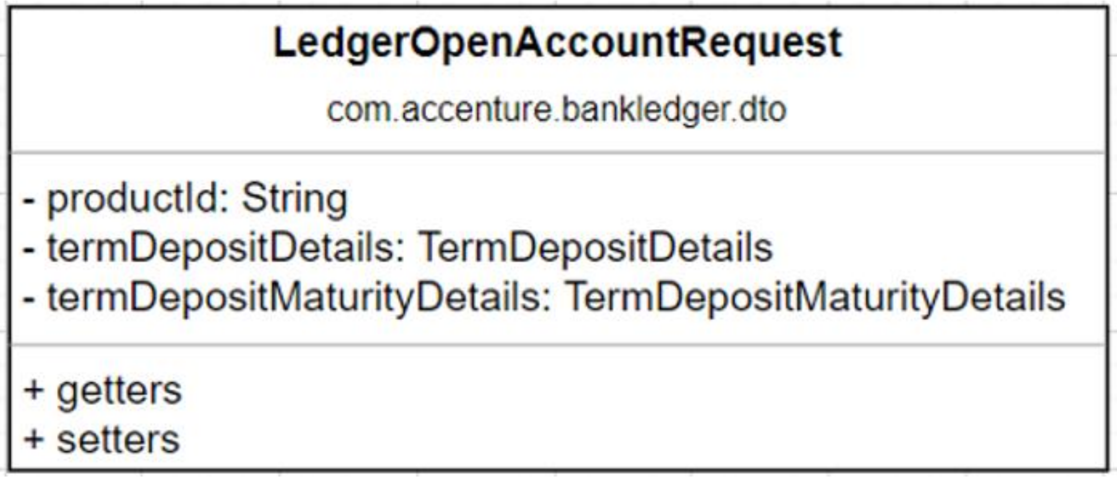
Models

LedgerOpenAccountRequest Serves as a data model for holding information related to opening a ledger account.



```
LedgerOpenAccountRequest v {
   productId*               string
                            example: 123456
                            pattern: ^[0-9]{6,9}$

   termDepositDetails*      TermDepositDetails v {
                               interestRate*       number
                                                   example: 0.2
                                                   minimum: 0
                               depositAmount*      number
                                                   example: 100000
                                                   pattern: ^(?!0?0\.00$)([0]|[1-9]\d{0,15})\.\d{2}$
                               termMonths*         number
                                                   example: 12
                               effectiveDate*      string
                                                   example: 21/02/2023
                                                   pattern: (0[1-9]|[1-2][0-9]|3[0-1])/(0[1-9]|1[0-2])/[0-9]{4}$
                               expiryDate*         string
                                                   example: 21/02/2024
                                                   pattern: (0[1-9]|[1-2][0-9]|3[0-1])/(0[1-9]|1[0-2])/[0-9]{4}$
                            }
   termDepositMaturityDetails* TermDepositMaturityDetails v {
                               accountName         string
                                                   example: John Sina
                               accountNumber*      string
                                                   example: 123456789
                                                   pattern: ^[0-9]{6,9}$
                            }
}
```

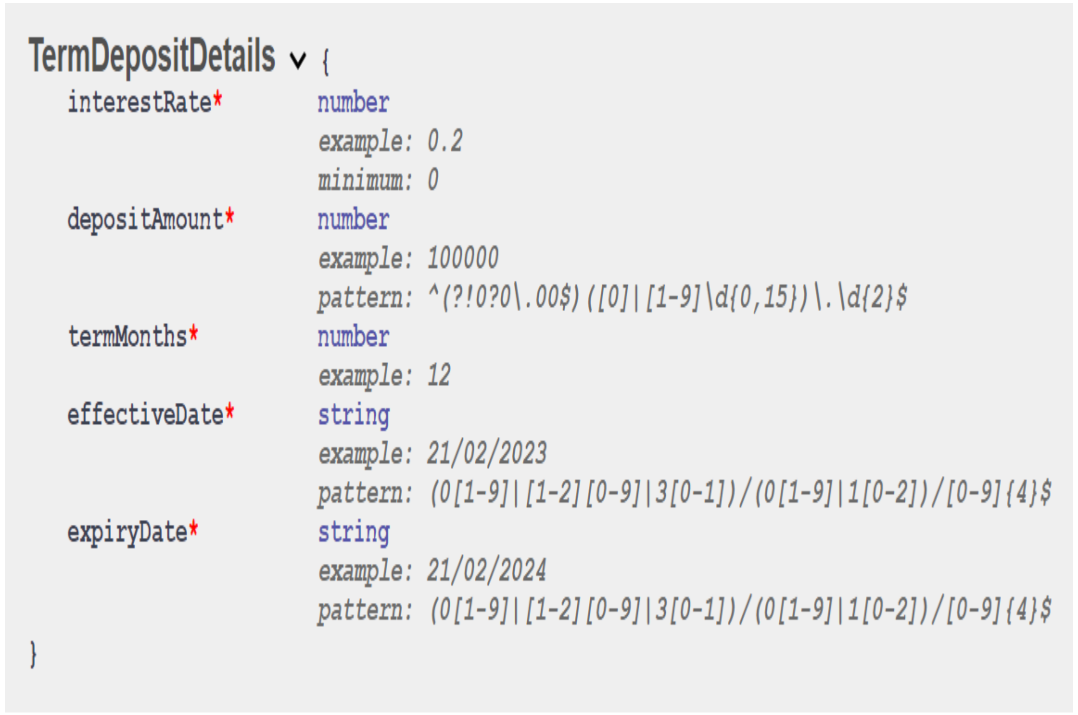


```
LedgerOpenAccountRequest
com.accenture.bankledger.dto
---------------------------------------------
- productId: String
- termDepositDetails: TermDepositDetails
- termDepositMaturityDetails: TermDepositMaturityDetails
---------------------------------------------
+ getters
+ setters
```

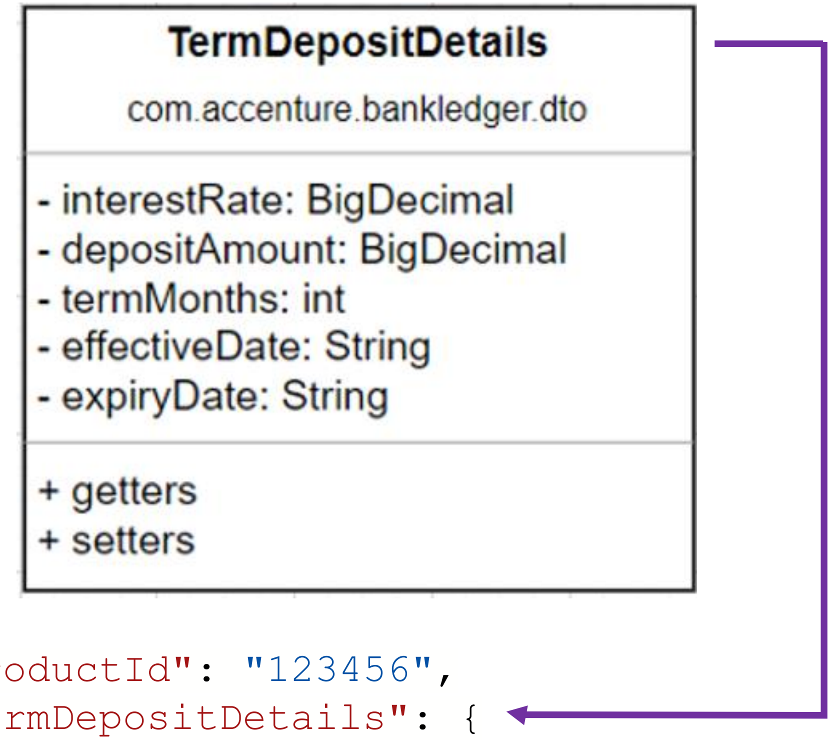
```json
{
    "productId": "123456",
    "termDepositDetails": {
        "interestRate": 0.2,
        "depositAmount": "100000.00",
        "termMonths": "12",
        "effectiveDate": "21/02/2023",
        "expiryDate": "21/02/2024"
    },
    "termDepositMaturityDetails": {
        "accountName": "John Seven",
        "accountNumber": "12349999"
    }
}
```

Models

TermDepositDetails is a model which contains detailed information about the term deposit



```
TermDepositDetails v {
    interestRate*       number
                        example: 0.2
                        minimum: 0
    depositAmount*      number
                        example: 100000
                        pattern: ^(?!0?0\.00$)([0]|[1-9]\d{0,15})\.\d{2}$
    termMonths*         number
                        example: 12
    effectiveDate*      string
                        example: 21/02/2023
                        pattern: (0[1-9]|[1-2][0-9]|3[0-1])/(0[1-9]|1[0-2])/[0-9]{4}$
    expiryDate*         string
                        example: 21/02/2024
                        pattern: (0[1-9]|[1-2][0-9]|3[0-1])/(0[1-9]|1[0-2])/[0-9]{4}$
}
```

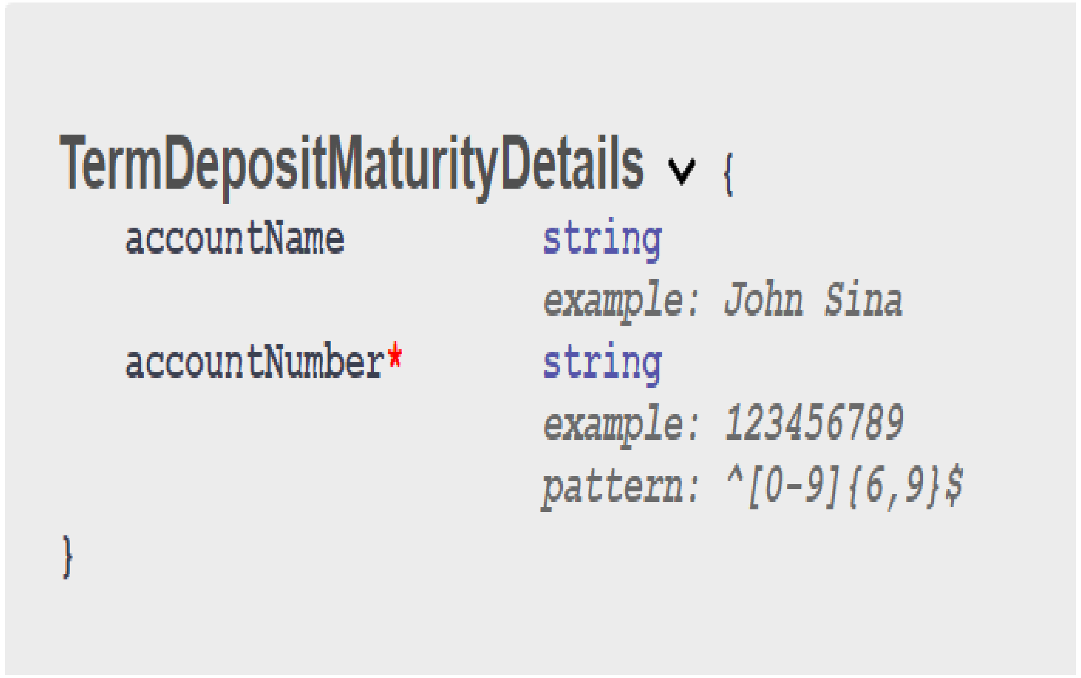


```
TermDepositDetails
com.accenture.bankledger.dto
- interestRate: BigDecimal
- depositAmount: BigDecimal
- termMonths: int
- effectiveDate: String
- expiryDate: String
+ getters
+ setters
```

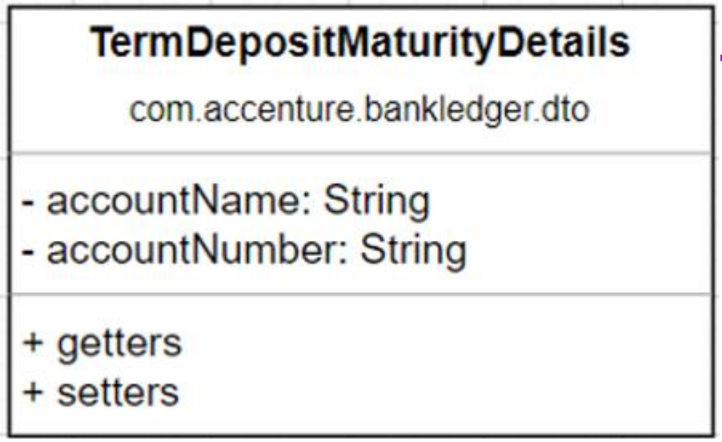
```
{
    "productId": "123456",
    "termDepositDetails": {
        "interestRate": 0.2,
        "depositAmount": "100000.00",
        "termMonths": "12",
        "effectiveDate": "21/02/2023",
        "expiryDate": "21/02/2024"
    },
    "termDepositMaturityDetails": {
        "accountName": "John Seven",
        "accountNumber": "12349999"
    }
}
```

Models

TermDepositMaturityDetails is a model which contains the account name and account number.



```
TermDepositMaturityDetails ∨ {
    accountName        string
                       example: John Sina
    accountNumber*     string
                       example: 123456789
                       pattern: ^[0-9]{6,9}$
}
```

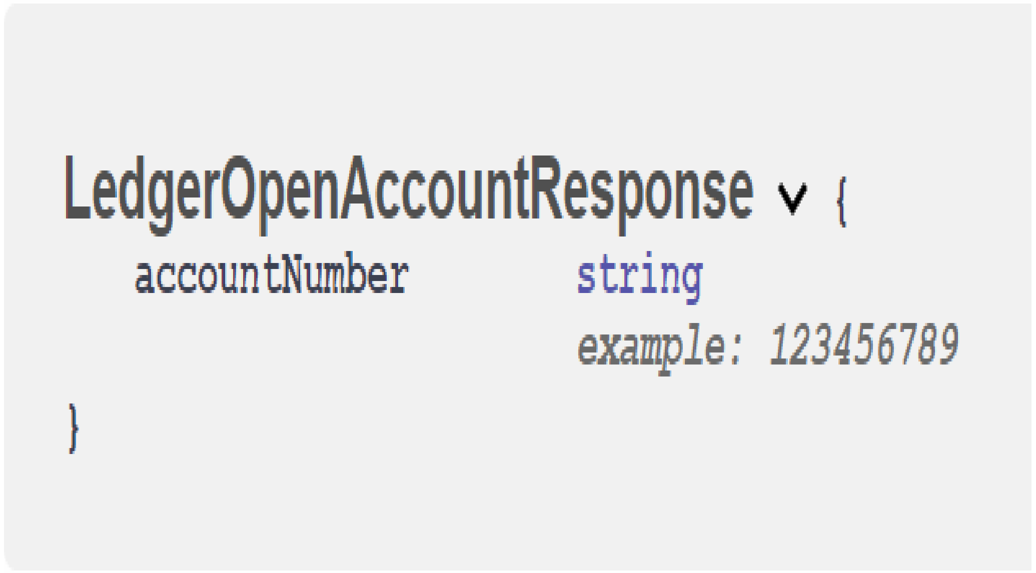


```
TermDepositMaturityDetails
com.accenture.bankledger.dto
- accountName: String
- accountNumber: String
+ getters
+ setters
```

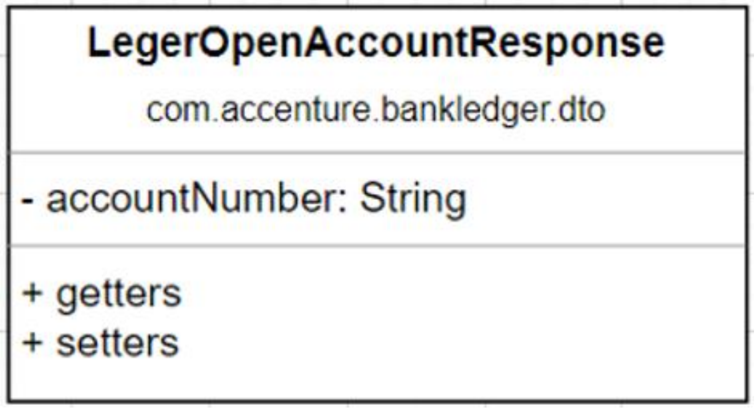
```
{
    "productId": "123456",
    "termDepositDetails": {
        "interestRate": 0.2,
        "depositAmount": "100000.00",
        "termMonths": "12",
        "effectiveDate": "21/02/2023",
        "expiryDate": "21/02/2024"
    },
    "termDepositMaturityDetails": {
        "accountName": "John Seven",
        "accountNumber": "12349999"
    }
}
```

Models

LedgerOpenAccountResponse is a model used to define the structure of the response that the server sends back to the client when a request is made.



```
LedgerOpenAccountResponse ∨ {
    accountNumber        string
                         example: 123456789
}
```

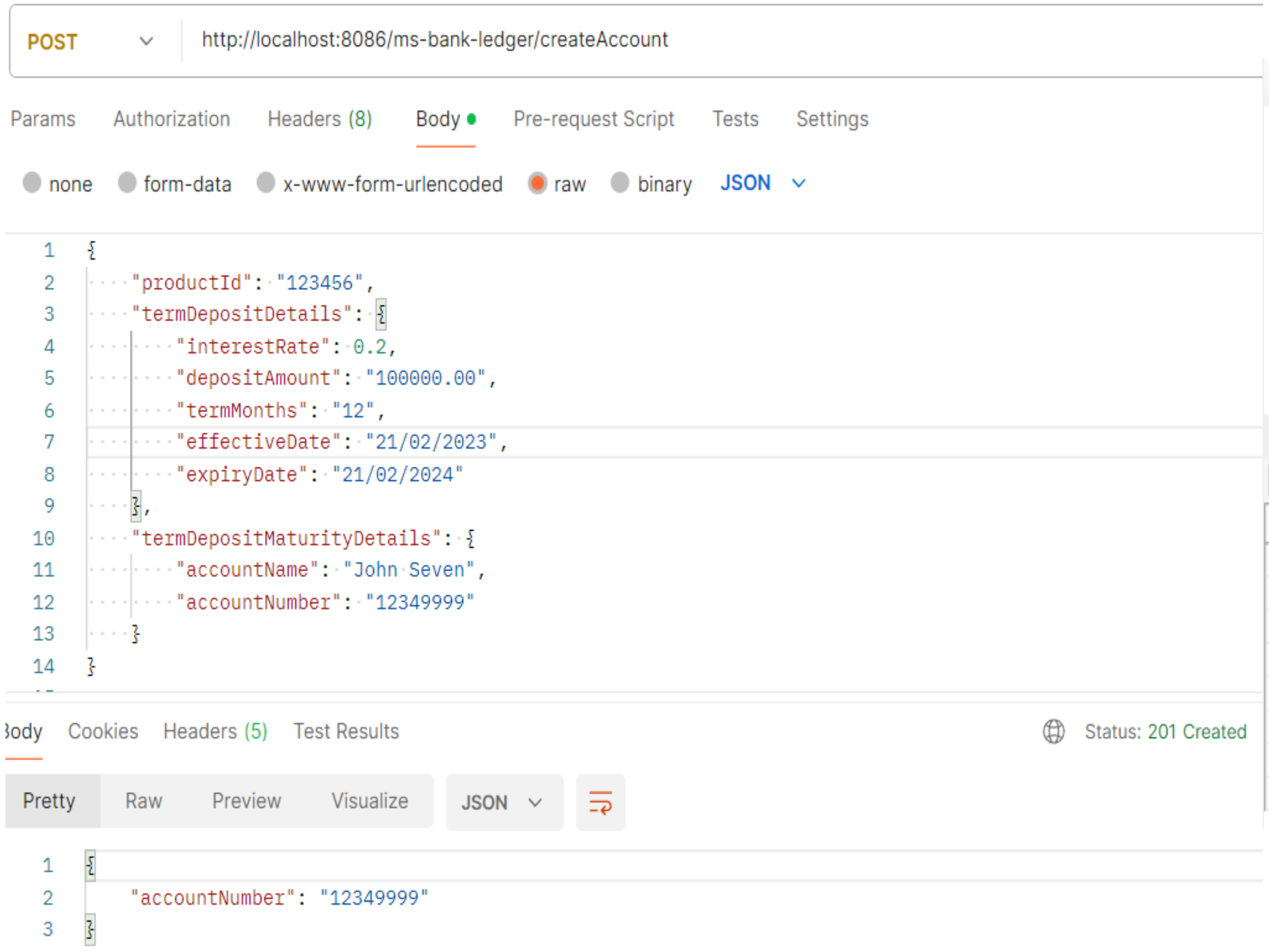


```
{
        "accountNumber": "12349999"
}
```

How to test?

Valid scenario: POST http://localhost:8086/ms-bank-ledger/createAccount

Postman



POST http://localhost:8086/ms-bank-ledger/createAccount

Params | Authorization | Headers (8) | Body | Pre-request Script | Tests | Settings

none | form-data | x-www-form-urlencoded | raw | binary | JSON

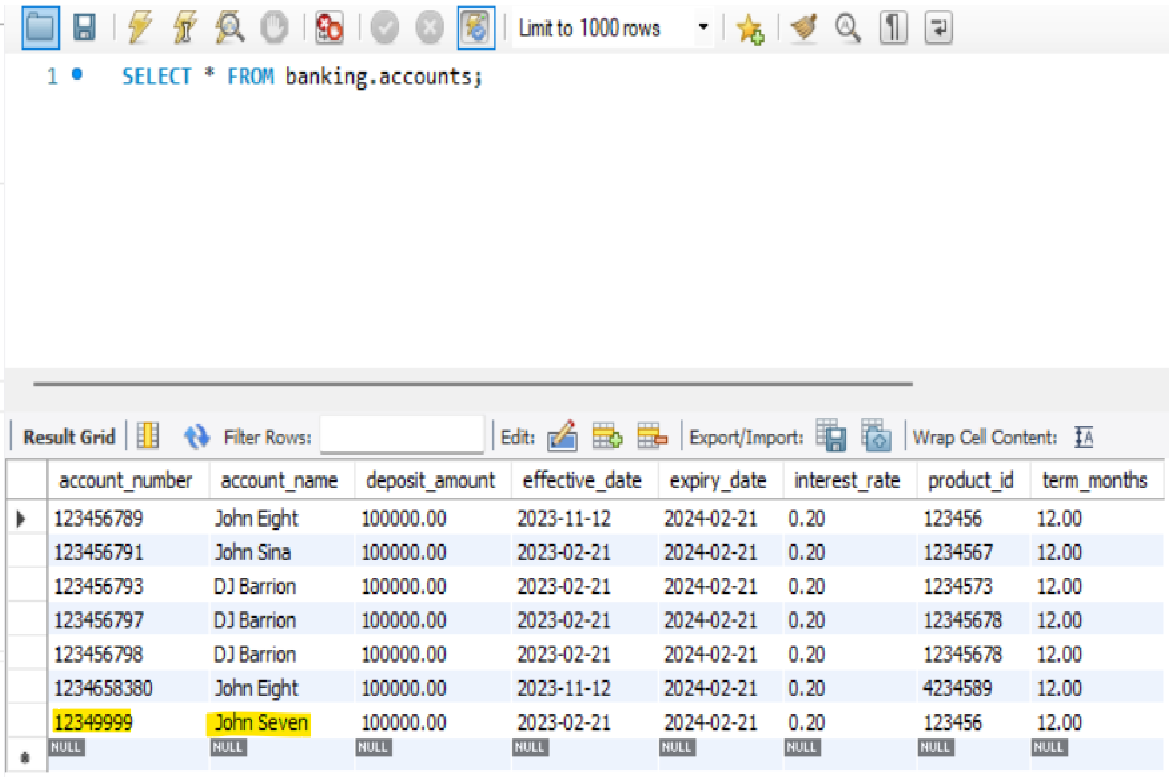
```
{
    "productId": "123456",
    "termDepositDetails": {
        "interestRate": 0.2,
        "depositAmount": "100000.00",
        "termMonths": "12",
        "effectiveDate": "21/02/2023",
        "expiryDate": "21/02/2024"
    },
    "termDepositMaturityDetails": {
        "accountName": "John Seven",
        "accountNumber": "12349999"
    }
}
```

Body | Cookies | Headers (5) | Test Results — Status: 201 Created

Pretty | Raw | Preview | Visualize | JSON

```
{
    "accountNumber": "12349999"
}
```

Accounts Table



Limit to 1000 rows

```
SELECT * FROM banking.accounts;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

account_number	account_name	deposit_amount	effective_date	expiry_date	interest_rate	product_id	term_months
123456789	John Eight	100000.00	2023-11-12	2024-02-21	0.20	123456	12.00
123456791	John Sina	100000.00	2023-02-21	2024-02-21	0.20	1234567	12.00
123456793	DJ Barrion	100000.00	2023-02-21	2024-02-21	0.20	1234573	12.00
123456797	DJ Barrion	100000.00	2023-02-21	2024-02-21	0.20	12345678	12.00
123456798	DJ Barrion	100000.00	2023-02-21	2024-02-21	0.20	12345678	12.00
1234658380	John Eight	100000.00	2023-11-12	2024-02-21	0.20	4234589	12.00
12349999	John Seven	100000.00	2023-02-21	2024-02-21	0.20	123456	12.00
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Unit Test

- **Location:** Create the unit test class under the test folder in your project directory.

- **Package Name:** The package name of the test class should match the package name of the class you are testing. For example, if the BankLedgerServiceImpl class is located in the package com.accenture.bankledger.service.impl, then the test class should also be in com.accenture.bankledger.service.impl.

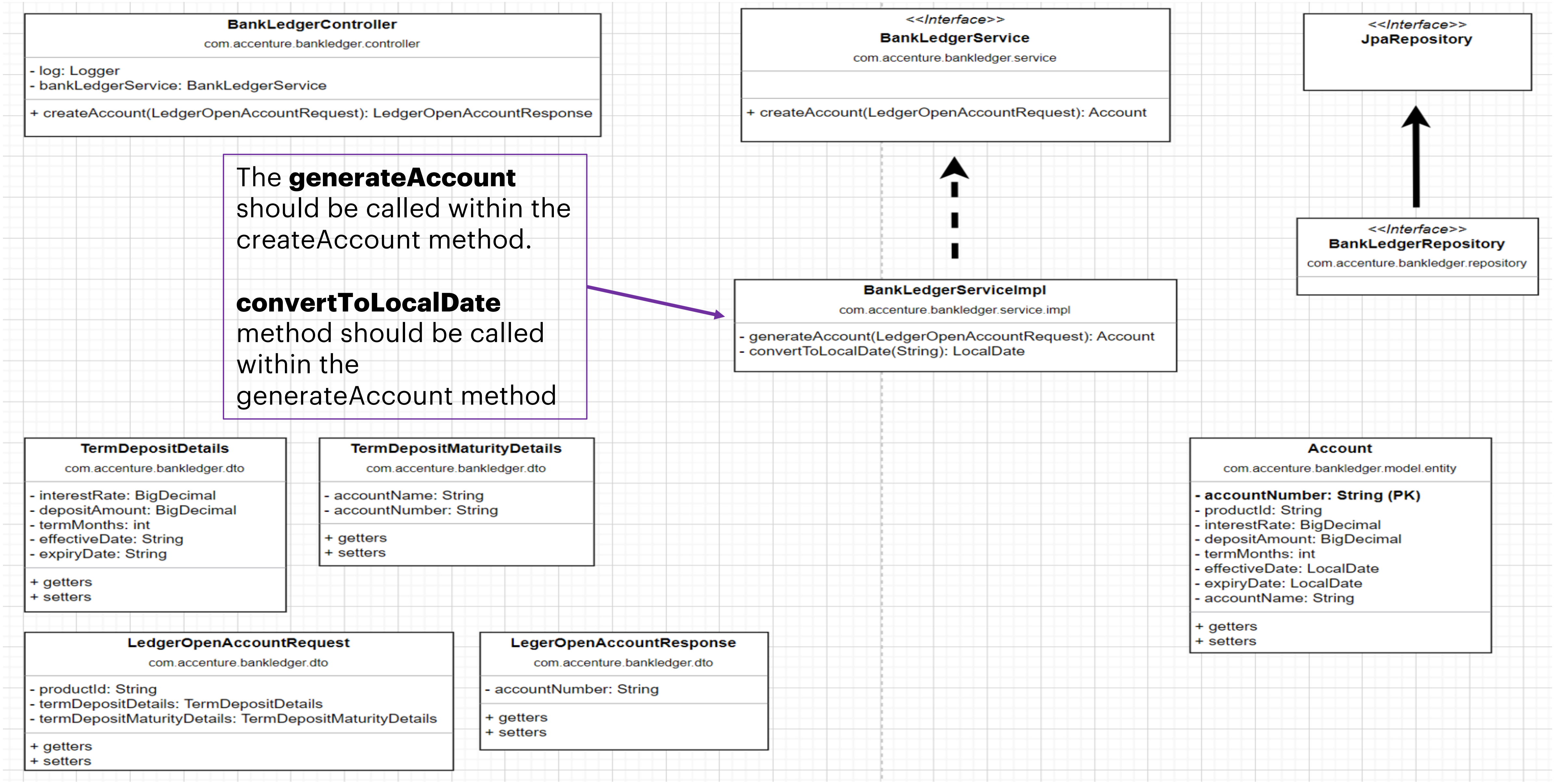
- **Class Name:** Name the test class PricingServiceImplTest to clearly indicate that it tests the PricingServiceImpl class.

Unit Test

Test Method Name	Method to Test	Description	Test Condition	Expected Result

testCreateAccount	BankLedgerServiceImpl.
	createAccount
		Save an account with all the correct details/	Call BankLedgerServiceImpl.
			createAccount with the following data:
			
			{
			  "productId": "123456",
			  "termDepositDetails": {
			    "interestRate": 0.2,
			    "depositAmount": "100000.00",
			    "termMonths": "12",
			    "effectiveDate": "21/02/2023",
			    "expiryDate": "21/02/2024"
			  },
			  "termDepositMaturityDetails": {
			    "accountName": "John Sina",
			    "accountNumber": “123456789"
			  }
			}
				Verify that the returned data includes the following
				  accountNumber: "123456789"
				  effectiveDate: LocalDate of 21/02/2023
				  expiryDate: LocalDate of 21/02/2024

Class Diagram



BankLedgerController
com.accenture.bankledger.controller
- log: Logger
- bankLedgerService: BankLedgerService
+ createAccount(LedgerOpenAccountRequest): LedgerOpenAccountResponse
<<Interface>>
BankLedgerService
com.accenture.bankledger.service
+ createAccount(LedgerOpenAccountRequest): Account
<<Interface>>
JpaRepository
<<Interface>>
BankLedgerRepository
com.accenture.bankledger.repository
BankLedgerServiceImpl
com.accenture.bankledger.service.impl
- generateAccount(LedgerOpenAccountRequest): Account
- convertToLocalDate(String): LocalDate
TermDepositDetails
com.accenture.bankledger.dto
- interestRate: BigDecimal
- depositAmount: BigDecimal
- termMonths: int
- effectiveDate: String
- expiryDate: String
+ getters
+ setters
TermDepositMaturityDetails
com.accenture.bankledger.dto
- accountName: String
- accountNumber: String
+ getters
+ setters
Account
com.accenture.bankledger.model.entity
- accountNumber: String (PK)
- productId: String
- interestRate: BigDecimal
- depositAmount: BigDecimal
- termMonths: int
- effectiveDate: LocalDate
- expiryDate: LocalDate
- accountName: String
+ getters
+ setters
LedgerOpenAccountRequest
com.accenture.bankledger.dto
- productId: String
- termDepositDetails: TermDepositDetails
- termDepositMaturityDetails: TermDepositMaturityDetails
+ getters
+ setters
LegerOpenAccountResponse
com.accenture.bankledger.dto
- accountNumber: String
+ getters
+ setters

The **generateAccount** should be called within the createAccount method.

convertToLocalDate method should be called within the generateAccount method

Strategy for developing the Bank Ledger MS

- **Java Version:** 17
- **Group Id:** com.accenture
- **Artifact Id:** bank-ledger
- **Package Name:** com.accenture.bankledger

- Add the necessary dependencies for Restful Webservice and Database

- **Run the Project:** Immediately after importing your project into the IDE, run it to ensure Tomcat is functioning correctly.
- **Verify Project Operation:** Once you've confirmed that the project runs without issues, proceed with the following steps:
    - Update the account properties file.
    - Create the Account entity.
    - Run the project again and verify that the Accounts table has been added to your Banking database.
    - Create the Repository.
    - Create the DTO.
    - Create the Service.
    - Create the Controller.
- **Run Your Endpoints**: Use Postman to test your endpoints. Check the console for any errors and debug as necessary.
- **Create Unit Test**
- **Run SonarQube:** Address any issues it identifies.

FAQ – Frequently Asked Questions

Can I use Github Copilot? Yes, you can use GitHub Copilot. It will help you accelerate your development process during the bootcamp.
Access denied for user 'root'@'localhost' (using password: YES) - update spring.datasource.password

Guidance for your Demo

A proper demo goes beyond just executing requests in Postman.

Clear Explanation: Start by clearly explaining the purpose of the demo. Describe the endpoint you're testing, the scenario, and what you expect to show.

Scenario: For this demonstration, I will use a product code 12345 that does not exist in our JSON file. This will help us see how the API handles requests for non-existent products.

Expectation: We expect the API to return a 400 Bad Request status code with a detailed error message indicating that the product code was not found. This will show us how the system handles errors and provides feedback when the requested product is not available."

You can demonstrate the endpoints as soon as you have created them. JUnit tests can be demonstrated afterwards.