

## Customer Loan Service Application - User Manuel

This backend service application has been developed by using Spring Boot Framework, and H2 has been used as an in memory database.

### Provided Endpoints

The below apis are provided in order to manage the customer loan process. As it was required, all endpoints (except /registration) require admin user or record owner [customer] authorization by specifying username and password with basic authentication.

There are two user roles; admin [ROLE\_ADMIN], customer [ROLE\_CUSTOMER]

The system admin user automatically initialized and created when the application started.

The admin user credentials;

username : admin

password : admin1

localhost:8080/swagger-ui/index.html?continue=#/

★ Bookmarks Online Guitar Tuner... English tenses (PDF) huawei Technical Adobe Acrobat Adobe Acrobat

### loan-controller

- PUT** /loan/payment/{customerId}
- POST** /loan/{customerId}
- GET** /loan/list/{customerId}
- GET** /loan/installments/{customerId}/{loanId}

### registration-controller

- POST** /registration

### customer-controller

- GET** /customer/{customerId}
- GET** /customer/list

## POST :/registration (Customer registration)

It is a public endpoint to create customer records on the system. On the request body, the customer information needs to be provided like; name, surname, userId, creditLimit...

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** localhost:8080/registration
- Body:** JSON format with the following content:

```
1 {
2   "name": "ahmet",
3   "surname": "uyanik",
4   "userId": "au1234",
5   "creditLimit": "200000.00",
6   "usedCreditLimit": "0",
7   "password": "1234",
8   "roles": ["ROLE_CUSTOMER"]
9 }
```
- Response:** 200 OK, 88 ms, 456 B. The response body is "Customer saved successfully".

It should return a success message after the customer registered successfully.

## GET :/customer/{userId} (customer details)

It is a secured endpoint to get customer details for a given userId. It requires basic auth (admin, customer [record owner])

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** localhost:8080/customer/au1234
- Authorization:** Basic Auth. Username: admin, Password: admin1.
- Response:** 200 OK, 107 ms, 634 B. The response body is JSON format with the following content:

```
1 {
2   "id": 2,
3   "userId": "au1234",
4   "name": "ahmet",
5   "surname": "uyanik",
6   "password": "$2a$10$IFX7qZycF8axg562SSFKv020k1ghKZT29VRhI48ncKxjNTm0Gd9v2",
7   "roles": [
8     "ROLE_CUSTOMER"
9   ],
10  "creditLimit": 200000,
11  "usedCreditLimit": 0
12 }
```

The api also accepts the record owner [customer] request as below

GET localhost:8080/customer/au1234 Send

Params Authorization Headers (8) Body Scripts Settings Cookies

Auth Type: Basic Auth

Username: au1234

Password: 1234

The authorization header will be

Body Cookies (1) Headers (14) Test Results 200 OK • 265 ms • 560 B Save Response

JSON Preview Visualize

```
1 {
2   "id": 2,
3   "userId": "au1234",
4   "name": "ahmet",
5   "surname": "uyanik",
6   "roles": [
7     "ROLE_CUSTOMER"
8   ],
9   "creditLimit": 200000,
10  "usedCreditLimit": 0
11 }
```

Lets create another customer, and check if the second customer can access the information of the first one.

POST localhost:8080/registration Send

Params Authorization Headers (9) Body Scripts Settings Cookies Beautify

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

```
1 {
2   "name": "mehmet",
3   "surname": "uyanik",
4   "userId": "mu1234",
5   "creditLimit": "200000.00",
6   "usedCreditLimit": "0",
7   "password": "1234",
8   "roles": ["ROLE_CUSTOMER"]
9 }
```

Body Cookies (1) Headers (14) Test Results 200 OK • 72 ms • 456 B Save Response

Raw Preview Visualize

1 Customer saved successfully

the second customer mu1234 was created, and lets try to access customer au1234 information with the customer mu1234

GET localhost:8080/customer/au1234

Auth Type: Basic Auth

Username: mu1234

Password: 1234

403 Forbidden • 91 ms • 540 B

```
{
  "timestamp": "2025-08-25T14:04:50.568+00:00",
  "status": 403,
  "error": "Forbidden",
  "path": "/customer/au1234"
}
```

As we can see here, 403 Forbidden http message was returned by the application.

## GET :/customer/list (all customer details)

It is a secured endpoint to list all registered customers in the system. Only admin user can invoke this

GET localhost:8080/customer/list

Auth Type: Basic Auth

Username: admin

Password: admin

200 OK • 230 ms • 695 B

```
[
  {
    "id": 2,
    "userId": "au1234",
    "name": "ahmet",
    "surname": "uyanik",
    "roles": [
      "ROLE_CUSTOMER"
    ],
    "creditLimit": 200000,
    "usedCreditLimit": 0
  },
  {
    "id": 3,
    "userId": "mu1234",
    "name": "mehmet",
    "surname": "uyanik",
    "roles": [
      "ROLE_CUSTOMER"
    ],
    "creditLimit": 200000
  }
]
```

## POST :/loan/{userId} (create loan)

It is a secured endpoint to create loan and loan installments with the request body as shown. There is a validator logic inside the app to check some conditions to determine whether it is eligible to create a loan for that customer. This endpoint can be triggered by the admin, or the owner customer.

First, let us check the validation. For example, the interest rate amount should be between 0.1 - 0.5 as it has been mentioned in the requirement document. Let's send interest rate as 1

The screenshot shows a REST client interface. The top bar indicates a POST request to `localhost:8080/loan/mu1234`. The 'Body' tab is selected, showing a JSON request body:

```
1 {
2   "loan": {
3     "customerId": "mu1234",
4     "loanAmount": "30000.00",
5     "interestRate": "1",
6     "numberOfInstallment": "3"
7   }
8 }
```

The status bar at the bottom shows a **400 Bad Request** response with a status of 400, a response time of 88 ms, and a body size of 602 B. The response body is displayed in the 'Body' tab as a JSON array:

```
1 [
2   {
3     "codes": [
4       "customer.loan.interest.rate.not.valid"
5     ],
6     "arguments": null,
7     "defaultMessage": "Loan interest rate not valid",
8     "objectName": "LoanCreation",
9     "code": "customer.loan.interest.rate.not.valid"
10  }
11 ]
```

We expect that It gets failed by the validation since the interest rate is not fitting the required amount.

Let's try a successful scenario..

We set interest rate as 0.5, for the loan amount 30000.00, that makes 450000.00 for 3 month installment payment, and each installment amount 15000.00

As we can see below, since all conditions are met, it allows us to create a loan and installments accordingly. Some installment information like `dueDate`, `isPaid` is calculated and set programmatically. Therefore we do not include them in the request body.

POST localhost:8080/loan/mu1234 Send

Params Authorization Headers (10) Body Scripts Settings Cookies Beautify

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON

```
1 {
2   "loan": {
3     "customerId": "mu1234",
4     "loanAmount": "30000.00",
5     "interestRate": "0.5",
6     "numberOfInstallment": "3"
7   }
8 }
```

Body Cookies (1) Headers (14) Test Results 200 OK • 98 ms • 471 B Save Response

Raw Preview Visualize

1 Loan and installments created successfully

## GET :/loan/list/{userId} (list loans for userId)

It is a secured endpoint to list loans for the given userId.

GET localhost:8080/loan/list/mu1234 Send

Params Authorization Headers (8) Body Scripts Settings Cookies

Auth Type Basic Auth Username mu1234 Password 1234

The authorization header will be

Body Cookies (1) Headers (14) Test Results 200 OK • 69 ms • 564 B Save Response

{ JSON Preview Visualize

```
1 [
2   {
3     "id": 1,
4     "customerId": "mu1234",
5     "loanAmount": 30000,
6     "interestRate": 0.5,
7     "numberOfInstallment": 3,
8     "createDate": "2025-08-26",
9     "paid": false
10  }
11 ]
```

## GET :/loan/installments/{userId}/{loanId} (list loan installments for userId)

It is a secured endpoint to list the installments for a given userId and loanId

GET localhost:8080/loan/installments/mu1234/1

Auth Type: Basic Auth

Username: mu1234

Password: \*\*\*\*

The authorization header will be

200 OK • 76 ms • 925 B

Body: JSON

```
4    {"loanId": 1,
5      "amount": 15000,
6      "paidAmount": 0,
7      "dueDate": "2025-09-01",
8      "paymentDate": null,
9      "unDoOrOverdueCharge": 0,
10     "unDue": false,
11     "paid": false,
12     "overdue": false
13   },
14   {
15     "id": 2,
16     "loanId": 1,
17     "amount": 15000,
18     "paidAmount": 0,
19     "dueDate": "2025-10-01",
20     "paymentDate": null,
21     "unDoOrOverdueCharge": 0,
22     "unDue": false,
23     "paid": false,
24     "overdue": false
25   },
26   {
27     "id": 3,
28     "loanId": 1,
29     "amount": 15000,
30     "paidAmount": 0.
```

unDo and overdue fields are set to false initially, they are used to mark whether the installment is paid before due date, or after due date. I will add the reward or penalty amount into the unDoOrOverdueCharge field accordingly.

## PUT :/loan/payment/{userId} (pay the loan for userId)

It is a secured endpoint to pay the loan based on the amount given in the request. There is also a validation behind this endpoint to check some conditions given in the requirement document.

On the request body, we specify the amount, and the loan id to pay.

PUT localhost:8080/loan/payment/mu1234 Send

Params Authorization Headers (10) Body Scripts Settings Cookies Beautify

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

```
1 {
2   "loanId": "1",
3   "amount": "50000.00"
4 }
```

Body Cookies (1) Headers (14) Test Results 200 OK • 124 ms • 514 B Save Response

{ JSON Preview Visualize

```
1 {
2   "numberOfPaidInstallments": 3,
3   "totalAmountSpent": 43365,
4   "loanPaymentCompleted": true
5 }
```

On the response we can see; number of installments paid, total amount spent to pay, and if the loan was paid completely.

After that you can also see the installments isPaid status was changed to true, and payment Date also was set programatically after payment.

GET localhost:8080/loan/installments/mu1234/1 Send

Params Authorization Headers (8) Body Scripts Settings Cookies

Auth Type Basic Auth Username mu1234 Password

The authorization header will be

Body Cookies (1) Headers (14) Test Results 200 OK • 77 ms • 964 B Save Response

{ JSON Preview Visualize

```
1 [
2   {
3     "id": 1,
4     "loanId": 1,
5     "amount": 15000,
6     "paidAmount": 14910,
7     "dueDate": "2025-09-01",
8     "paymentDate": "2025-08-26",
9     "unDoOrOverdueCharge": -90,
10    "unDue": true,
11    "paid": true,
12    "overdue": false
13  },
14  {
15    "id": 2,
16    "loanId": 1,
17    "amount": 15000,
18    "paidAmount": 14460,
19    "dueDate": "2025-10-01",
20    "paymentDate": "2025-08-26",
21    "unDoOrOverdueCharge": -540,
22    "unDue": true,
23    "paid": true,
24    "overdue": false
25  },
26  {
27    "id": 3,
```



## H2-Console (DB)

The h2 database credentials defined as,

username:sa,

password:sa.

It can be checked the data using h2-console as shown below.

The screenshot shows the H2-Console web interface in a browser. The address bar shows the URL: `localhost:8080/h2-console/login.do?jsessionId=6521c0503139eb1453e1bec27f30e15f`. The interface includes a sidebar with a tree view of the database schema, a main area for SQL statements, and a results area.

**Database Schema (Left Sidebar):**

- jdbc:h2:mem:customerloandb;DEV
- APP\_USER
- LOAN
- LOAN\_INSTALLMENT
- USER\_ROLES
- INFORMATION\_SCHEMA
- Users
- H2 2.2.224 (2023-09-17)

**SQL Statement (Main Area):**

```
SELECT * FROM LOAN_INSTALLMENT
```

**Results (Right Area):**

SELECT \* FROM LOAN\_INSTALLMENT;

| ID | AMOUNT  | DUE_DATE   | IS_OVERDUE | IS_PAID | IS_UN_DUE | LOAN_ID | PAID_AMOUNT | PAYMENT_DATE | UN_DO_OR_OVERDUE_CHARGE |
|----|---------|------------|------------|---------|-----------|---------|-------------|--------------|-------------------------|
| 1  | 15000.0 | 2025-09-01 | FALSE      | TRUE    | TRUE      | 1       | 14910.0     | 2025-08-26   | -90.0                   |
| 2  | 15000.0 | 2025-10-01 | FALSE      | TRUE    | TRUE      | 1       | 14460.0     | 2025-08-26   | -540.0                  |
| 3  | 15000.0 | 2025-11-01 | FALSE      | TRUE    | TRUE      | 1       | 13995.0     | 2025-08-26   | -1005.0                 |

(3 rows, 2 ms)

[Edit](#)