

User Operations

Authentication:

Users can sign up by providing valid parameters.

The screenshot shows the Postman application interface. In the center, there is a request card for a 'POST Authentication User' endpoint. The URL is `http://localhost:5154/api/users`. The 'Body' tab is selected, showing a JSON payload:

```
1 {
2   "firstName": "Test",
3   "lastName": "Test",
4   "username": "16813246382",
5   "password": "test>Passwordd4128",
6   "email": "veyesluser@gmail.com",
7   "phonenumber": "5538654218",
8   "roles": [
9     "user"
10   ]
11 }
```

Below the request card, there is a 'Response' section with a small illustration of a person running. At the bottom right of the response area, it says 'Click Send to get a response'.

Username must be their TR id number.

The screenshot shows the same Postman interface after the 'Send' button was clicked. The response status is now '201 Created'. The response body is identical to the one shown in the previous screenshot:

```
1 {
2   "firstName": "Test",
3   "lastName": "Test",
4   "username": "16813246382",
5   "password": "test>Passwordd4128",
6   "email": "veyesluser@gmail.com",
7   "phonenumber": "5538654218",
8   "roles": [
9     "user"
10   ]
11 }
```

The status bar at the bottom of the screen shows the time as 12:01 PM and the date as 17/7/2024.

Returns 201 created if the user is created.

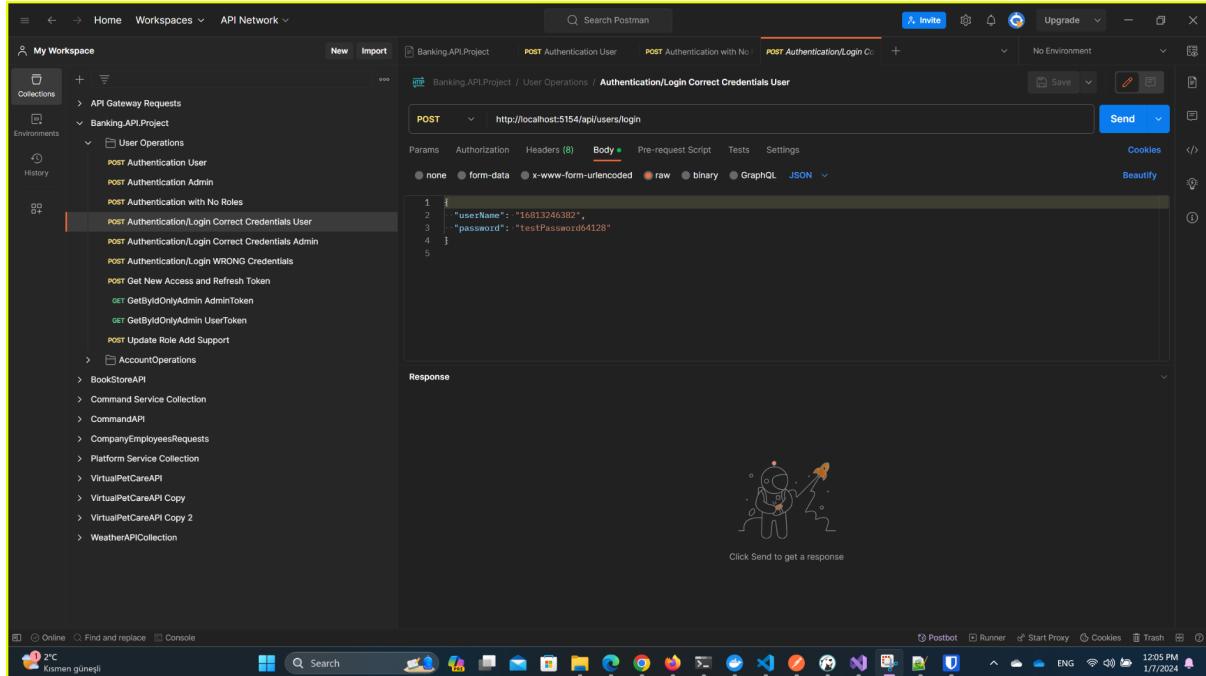
The screenshot shows the Oracle SQL Developer interface with the 'BankingDB' database selected. The 'AspNetUsers' table is open, displaying 14 rows of data. The columns are: Id, FirstName, LastName, RefreshTokenExpiryTime, UserName, PasswordHash, and SecurityStamp. The 'RefreshTokenExpiryTime' column shows values like 'infinity' and various dates/times. The 'PasswordHash' column contains long, complex strings of characters, indicating hashed passwords. The 'SecurityStamp' column also contains unique strings for each user.

Id	FirstName	LastName	RefreshTokenExpiryTime	UserName	PasswordHash	SecurityStamp
1	AdminFirstN	AdminLastNa	[infinity]	infinity	574 test TEST	AOAMMAIAYaegAAAAFHdgRkpQs3D+V.NZ3NUK75XVGYSIZ2INo
2	TestFirstN	TestLastNa	[infinity]	infinity	37298417344	377 test TEST
3	UserFirstN	UserLastNa	[infinity]	infinity	95745205412	AOAMMAIAYaegAAADEFemM8GCDlW+NMf.IYHMrDGcN3DWVYAGASC
4	UserFirstN	UserLastNa	[infinity]	infinity	11942419602	YQDxDxRxB9BXMXKX052
5	UserFirstN	UserLastNa	[infinity]	infinity	14173944738	AOAMMAIAYaegAAAHWmYtEczxtNWxS
6	IntegrationTest	IntegrationTest	[infinity]	infinity	1423394730	AOAMMAIAYaegAAAEmmgCYThxYmmtT
7	IntegrationTest	IntegrationTest	[infinity]	infinity	16173204752	HTZCICM24MISLYFTW2C
8	IntegrationTest	IntegrationTest	[infinity]	infinity	640158401142	AOAMMAIAYaegAAAECm4WAWL
9	IntegrationTest	IntegrationTest	[infinity]	infinity	3834600194	NWPEMhILOJSDQYXAH
10	IntegrationTest	IntegrationTest	[infinity]	infinity	79529879916	infinity
11	IntegrationTest	IntegrationTest	[infinity]	infinity	97182022180	AOAMMAIAYaegAAAAEA7Ckcln9Q58e1f
12	IntegrationTest	IntegrationTest	[infinity]	infinity	83585762228	13XQY3LRGOIDSTYNIMA'
13	IntegrationTest	IntegrationTest	[infinity]	infinity	66889296816	AOOAAAIAYaegAAAEMuq2ltsdw5FaLob-2FZCTOQK763A64SSCQC
14	Test	Test	[infinity]	infinity	16813246382	v4f4wvz5fUMoG62fSK22EWU4
					161 vey VEY5	AQAAAIAAYagAAAAEOUslyME95FyruQ; RXPBWPFAONRTTCOS2E

User password is stored in db after hashed.

Login:

For login, the user must provide correct credentials. Returns a bad request if either the username or password is wrong. Else, returns both access token and refresh token. Also, refresh token is stored in db. Anytime an access token expires, the user can get a new one.

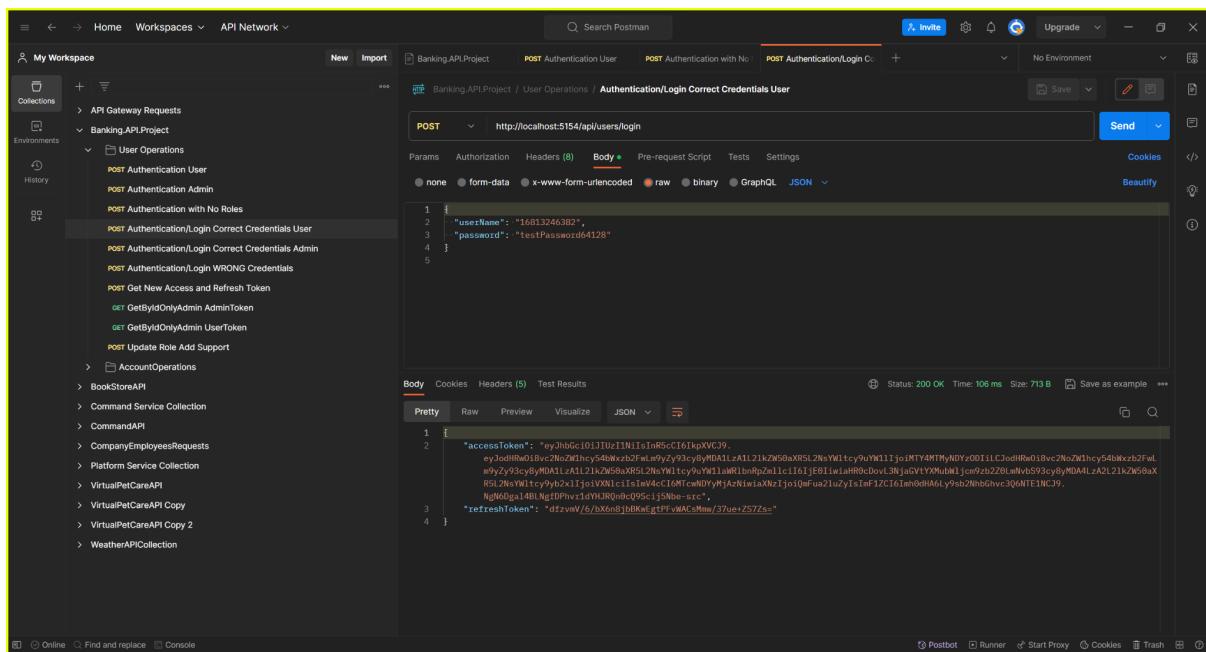


The screenshot shows the Postman interface with a POST request to `http://localhost:5154/api/users/login`. The request body is set to `JSON` and contains the following data:

```
1: {
2:   "username": "16813246382",
3:   "password": "testPassword64128"
4: }
```

The response section shows a cartoon character and a message: "Click Send to get a response".

A successful login operation returns both access token and refresh token.



The screenshot shows the Postman interface with the same POST request to `http://localhost:5154/api/users/login`. The response status is `200 OK` and the response body is displayed in `JSON` format:

```
1: {
2:   "accessToken": "eyJhbGciOiJIUzI1NiInRcC16IkXVCJ9",
3:   "refreshToken": "dZvmV/6/bXnB/bBKwgtPFvWAC5Mw/37ue+2577s="
4: }
```

User Roles:

There are 3 roles for this api. Admin, Support and User.

Only admins can get users by their id.

After supplying their access token, admins can get users by their id. If no user is found, they get a bad request.

The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar lists various collections and environments. The main workspace shows a 'User Operations' collection with several requests. One request is selected: 'GetByIdOnlyAdmin AdminToken' (GET method). The 'Authorization' tab is active, showing a 'Bearer Token' type with a placeholder for the token value. The 'Body' tab displays a JSON response with a user object:

```
1 {
2   "firstName": "UserFirstName1",
3   "lastName": "UserLastName2",
4   "userName": "95745395412",
5   "email": "testuser1@gmail.com",
6   "phoneNumber": "+905012312966",
7   "userRoles": [
8     "User"
9   ]
10 }
```

The status bar at the bottom indicates a 200 OK response with a size of 335 B.

A bad request as there is no user with id 20.

The screenshot shows the Postman application interface, similar to the previous one. The 'My Workspace' sidebar is visible. The main workspace shows the same 'User Operations' collection and 'GetByIdOnlyAdmin AdminToken' request. However, the 'Authorization' tab now shows a placeholder for the token value. The 'Body' tab displays a JSON response with an error message:

```
1 {
2   "statusCode": 400,
3   "Message": [
4     "User with id: 20 doesn't exist in the database"
5   ]
6 }
```

The status bar at the bottom indicates a 400 Bad Request response with a size of 250 B.

No roles other than “Admin” can use this endpoint. Users and Support roles will get 403 Forbidden.

The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar lists various API collections and environments. In the center, a request is being made to the 'Banking API Project / User Operations' collection. The specific endpoint is 'GetByIdOnlyAdmin UserToken'. The method is set to 'GET', and the URL is 'http://localhost:5154/api/admins/getuser/1'. The 'Authorization' tab is selected, showing a 'Bearer Token' type with a placeholder 'Token' and a redacted token value. Below the request, the 'Body' tab is selected, showing a JSON response with a status of '403 Forbidden'. The response body is empty, indicated by the number '1'.

Updating Roles:

Only Admin can update roles.

Before updating.

This screenshot shows the same setup as the previous one, but the response has changed. The 'Body' tab now displays a JSON object representing a user profile. The response status is '200 OK'. The JSON data includes fields like 'firstName', 'lastName', 'userName', 'email', 'phoneNumber', and 'userRoles'. The 'userRoles' field contains the value '["User"]'.

After updating:

The screenshot shows the Postman application interface. On the left, there's a sidebar titled 'My Workspace' with a tree view of API collections and environments. The main area shows a POST request to 'http://localhost:5154/api/admins/updaterole/5'. The 'Authorization' tab is selected, showing a 'Bearer Token' type with a token value. The 'Body' tab contains a JSON payload with fields like 'firstName', 'lastName', 'username', 'email', 'phoneNumber', and 'userRoles'. The response status is 200 OK with a time of 104 ms and a size of 345 B. The bottom status bar shows the date and time as 109 PM 1/7/2024.

Verifying with get user by id:

The screenshot shows the Postman application interface. The sidebar and request details are identical to the previous screenshot, but the method is now set to GET. The URL is 'http://localhost:5154/api/admins/getuser/5'. The response status is 200 OK with a time of 29 ms and a size of 345 B. The bottom status bar shows the date and time as 110 PM 1/7/2024.

Also in db:

Support role id is 2 and User role id is 3.

ID	Name	Normalized Name	Concurrency Stamp
1	Administrator	ADMINISTRATOR	[NULL]
2	Support	SUPPORT	[NULL]
3	User	USER	[NULL]

User Id	Role Id
1	1
2	2
3	2
4	3
5	3
6	3
7	3
8	3
9	3
10	3
11	3
12	3
13	3
14	3
15	1
16	2
16	3

There are two rows with user id 5. Role id is 2 and 3. So it's correctly executed.

If the admin sends a role that does not exist the api will return a bad request.

The screenshot shows the Postman interface with a collection named "Banking.API.Project / User Operations". A POST request is made to `http://localhost:5154/api/admins/updaterole/5`. The "Body" tab is selected, containing the following JSON:

```
1: {
2:   "UserRoles": [
3:     "Support",
4:     "User",
5:     "UnknownRole"
6:   ]
}
```

The response status is 400 Bad Request, with the message: "Role with name: UnknownRole doesn't exist in the database".

Other Roles will get 403 forbidden if they try to execute this endpoint.

The screenshot shows the Postman interface with a collection named "Banking.API.Project / User Operations". A POST request is made to `http://localhost:5154/api/admins/updaterole/5`. The "Authorization" tab is selected, showing a Bearer Token. The response status is 403 Forbidden.

Account Operations

Create Account:

Only the “User” role can create an account. Others roles will get 403 forbidden. Requests that have no access token will get 401 unauthorized.

The screenshot shows the Postman application interface. The left sidebar displays a collection named "Banking.APIProject" with several sub-collections like "User Operations" and "AccountOperations". The main workspace shows a POST request to "http://localhost:5154/api/accounts/create-account". The "Body" tab is selected, containing the JSON payload: { "Balance": 150 }. The status bar at the bottom indicates a successful response with a status of 201 Created, a time of 6 ms, and a size of 151 B.

This screenshot shows the same Postman setup as the previous one, but the status bar at the bottom indicates a failed request with a status of 401 Unauthorized, a time of 6 ms, and a size of 151 B. This demonstrates that a user without an access token cannot perform this operation.

Users will then have to provide a valid account balance. Balance limit is 100. Below the limit will get 400 bad request.

The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar lists various collections and environments. The main workspace shows a POST request to 'http://localhost:5154/api/accounts/create-account'. The 'Body' tab contains the following JSON payload:

```
1 {
2     "Balance": 50
3 }
```

The status bar at the bottom indicates a 'Status: 400 Bad Request' response. The 'Pretty' tab of the results panel displays the error message:

```
1 {
2     "statusCode": 400,
3     "message": [
4         "Balance must be greater than or equal to 100."
5     ]
6 }
```

If all is ok, the created account DTO will be returned to the client.

The screenshot shows the Postman application interface. The 'My Workspace' sidebar is visible on the left. The main workspace shows a POST request to 'http://localhost:5154/api/accounts/create-account'. The 'Body' tab contains the following JSON payload:

```
1 {
2     "Balance": 100
3 }
```

The status bar at the bottom indicates a 'Status: 200 OK' response. The 'Pretty' tab of the results panel displays the created account DTO:

```
1 {
2     "id": 10,
3     "balance": 100,
4     "createdAt": "2024-01-07T10:40:53.40889382",
5     "userId": 14,
6     "dailySpend": 8,
7     "dailyLimit": 500,
8     "operationLimit": 250
9 }
```

Verifying in the db.

DBeaver 23.3.1 - Accounts

File Edit Navigate Search SQL Editor Database Window Help

BankingDB public@BankingDb Auto EF MigrationsHistory AspNetUsers Accounts Records AspNetUserRoles AspNetRoles Bills

Database Navigator Projects

Properties Data LR Diagram

Enter a part of object name here

BankingDb Databases BankingDb Schemas BankingSchema Tables Accounts

BankingDb - localhost:6477

Grid Text

	Id	Balance	CreatedAt	UserId	DailySpend	DailyLimit	OperationLimit
1	1	250	2024-01-07 11:58:01.916 +0300	3	0	500	250
2	2	15,000	2024-01-07 11:58:01.965 +0300	4	0	500	250
3	3	450	2024-01-07 11:58:01.971 +0300	5	0	500	250
4	4	1,500	2024-01-07 11:58:01.986 +0300	8	0	500	250
5	5	1,000	2024-01-07 11:58:01.995 +0300	9	0	500	250
6	6	400	2024-01-07 11:58:02.005 +0300	10	0	500	250
7	7	1,000	2024-01-07 11:58:02.013 +0300	11	0	500	250
8	8	1,000	2024-01-07 11:58:02.023 +0300	12	0	500	250
9	9	1,000	2024-01-07 11:58:02.033 +0300	13	0	500	250
10	10	100	2024-01-07 13:40:53.408 +0300	14	0	500	250

Value

Project - General

Name: DataSou

Bookmarks Diagrams Scripts

Record

Refresh Save Cancel F K X Export data 200 10 10 row(s) fetched - 2ms, on 2024-01-07 at 13:41:58 TRT en 1:44 PM 1/7/2024

DBeaver 23.3.1 - Records

File Edit Navigate Search SQL Editor Database Window Help

BankingDB public@BankingDb Auto EF MigrationsHistory AspNetUsers Accounts Records AspNetUserRoles AspNetRoles Bills

Database Navigator Projects

Properties Data LR Diagram

Enter a part of object name here

BankingDb Databases BankingDb Schemas BankingSchema Tables Records

BankingDb - localhost:6477

Grid Text

	Id	Timestamp	OperationType	Amount	AccountId	ReceiverAccountId	IsSuccessful	ErrorMessage	IsPending	UserId
1	1	2024-01-07 11:58:01.941 +0300	0	250	1	[NULL]	[V]	[NULL]	[]	3
2	2	2024-01-07 11:58:01.973 +0300	0	15,000	2	[V]	[V]	[NULL]	[]	4
3	3	2024-01-07 11:58:01.981 +0300	0	450	3	[NULL]	[V]	[NULL]	[]	5
4	4	2024-01-07 11:58:01.990 +0300	0	1,500	4	[NULL]	[V]	[NULL]	[]	8
5	5	2024-01-07 11:58:01.999 +0300	0	1,000	5	[NULL]	[V]	[NULL]	[]	9
6	6	2024-01-07 11:58:02.006 +0300	0	400	6	[NULL]	[V]	[NULL]	[]	10
7	7	2024-01-07 11:58:02.018 +0300	0	1,000	7	[NULL]	[V]	[NULL]	[]	11
8	8	2024-01-07 11:58:02.027 +0300	0	1,000	8	[NULL]	[V]	[NULL]	[]	12
9	9	2024-01-07 11:58:02.037 +0300	0	1,000	9	[NULL]	[V]	[NULL]	[]	13
10	10	2024-01-07 11:58:02.074 +0300	6	35	2	[NULL]	[V]	[NULL]	[]	4
11	11	2024-01-07 11:58:02.087 +0300	6	45	2	[NULL]	[V]	[NULL]	[]	4
12	12	2024-01-07 11:58:02.099 +0300	6	105	2	[NULL]	[V]	[NULL]	[]	4
13	13	2024-01-07 11:58:02.112 +0300	6	60	3	[NULL]	[V]	[NULL]	[]	5
14	14	2024-01-07 13:40:53.471 +0300	0	100	10	[NULL]	[V]	[NULL]	[]	14

Value

Project - General

Name: DataSou

Bookmarks Diagrams Scripts

Record

Refresh Save Cancel F K X Export data 200 14 14 row(s) fetched - 2ms, on 2024-01-07 at 13:42:31 TRT en 1:42 PM 1/7/2024

The record has been created too. OperationType 0 means CreateAccount. Its an enum value.

For this project, a user can only have an account.

The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar lists various API collections and environments. The main workspace shows a POST request to 'http://localhost:5154/api/accounts/create-account'. The 'Body' tab contains the JSON payload: { "Balance": 100 }. The response status is 400 Bad Request, with the message: "The user with id '14' already have an account.".

Get Account By User Id:

User has to have an account created before. Else, it will return 404 bad request.

The screenshot shows the Postman application interface. The 'My Workspace' sidebar lists various API collections and environments. The main workspace shows a GET request to 'http://localhost:5154/api/accounts/account-information'. The 'Authorization' tab is set to 'Bearer Token', with a placeholder 'Token' and a long token string. The response status is 400 Bad Request, with the message: "Account with user id 14 does not exist in the database".

It will return account informations.

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost:5154/api/accounts/account-information`. The response body is:

```
1  {  
2      "id": 10,  
3      "balance": 100,  
4      "createdAt": "2024-01-07T11:47:17.632792Z",  
5      "userId": 15,  
6      "dailySpend": 0,  
7      "dailyLimit": 500,  
8      "operationLimit": 250,  
9      "bills": []  
10 }
```

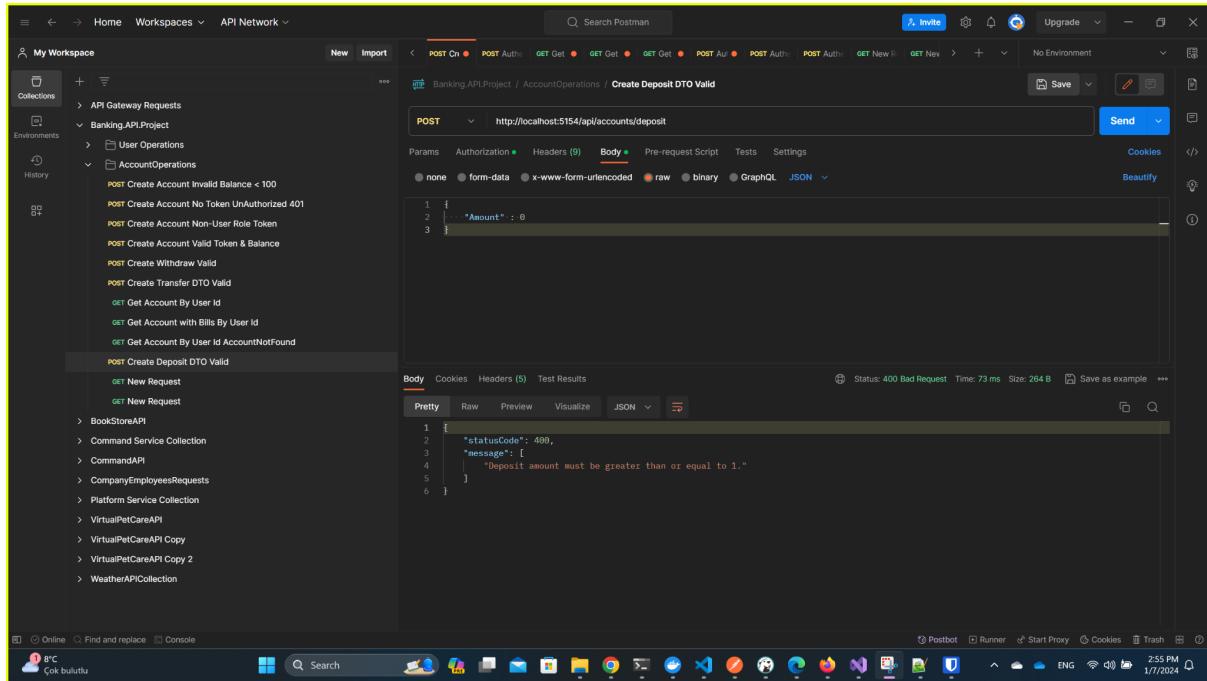
Also if the user has bills, they will return too.

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost:5154/api/accounts/account-information`. The response body is:

```
1  {  
2      "id": 2,  
3      "balance": 15000,  
4      "createdAt": "2024-01-07T11:42:59.479333Z",  
5      "userId": 4,  
6      "dailySpend": 0,  
7      "dailyLimit": 500,  
8      "operationLimit": 250,  
9      "bills": [  
10          {  
11              "id": 1,  
12              "amount": 35,  
13              "lastPayTime": "2024-01-07T11:44:59.574325Z",  
14              "isActive": true,  
15              "accountId": 2  
16          },  
17          {  
18              "id": 2,  
19              "amount": 45,  
20              "lastPayTime": "2024-01-08T11:42:59.574351Z",  
21              "isActive": true,  
22              "accountId": 2  
23          }  
24      ]  
25 }
```

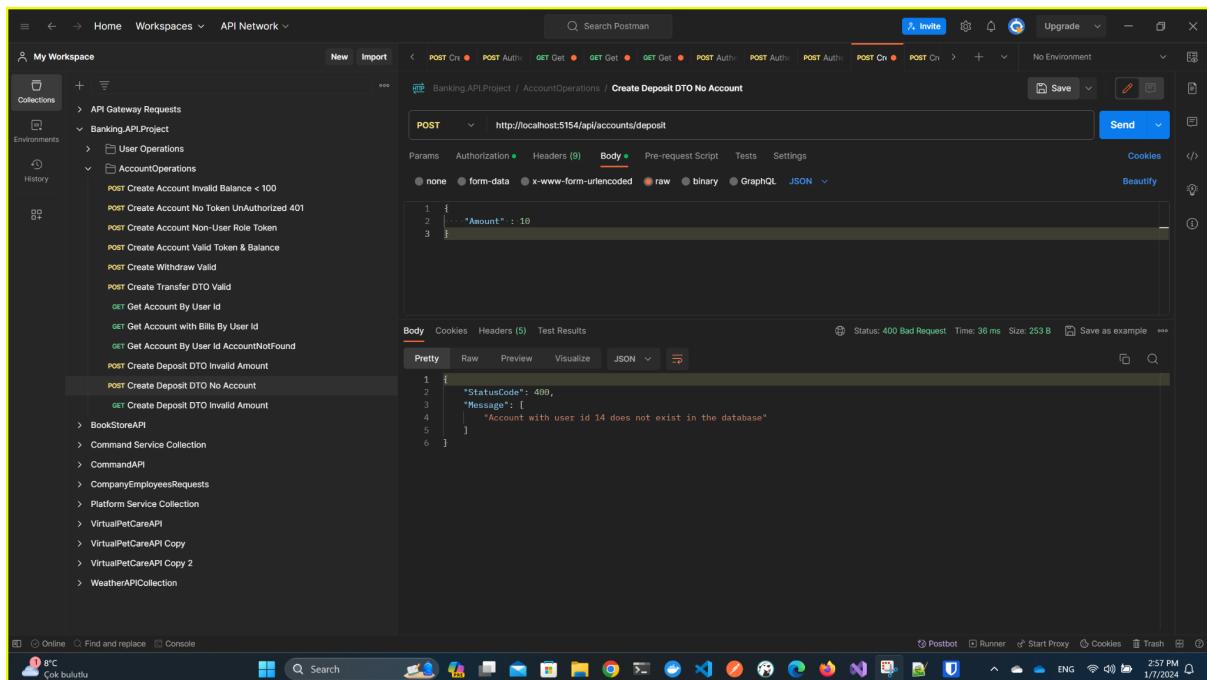
Deposit:

Deposit amount has to be greater than 1.



The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar lists various API collections and environments. The main workspace shows a POST request to `http://localhost:5154/api/accounts/deposit`. The 'Body' tab is selected, showing a JSON payload with a single field `"Amount": 0`. Below the body, the response status is shown as 400 Bad Request with the message: `{"statusCode": 400, "message": ["Deposit amount must be greater than or equal to 1."]}`.

If a user has not created an account before, it will return bad request.



The screenshot shows the Postman application interface. The 'My Workspace' sidebar is visible on the left. The main workspace shows a POST request to `http://localhost:5154/api/accounts/deposit`. The 'Body' tab is selected, showing a JSON payload with a single field `"Amount": 10`. Below the body, the response status is shown as 400 Bad Request with the message: `{"Status": 400, "Message": ["Account with user id 14 does not exist in the database"]}`.

Before a valid deposit request:

The screenshot shows the Postman interface with a project named "Banking.API.Project". A GET request is selected with the URL `http://localhost:5154/api/accounts/account-information`. The "Authorization" tab is active, showing a "Bearer Token" type with a token value. The "Body" tab displays a JSON response object with fields like id, balance, createdAt, userId, dailySpend, dailyLimit, operationLimit, and bills.

```
1 {  
2     "id": 10,  
3     "balance": 110,  
4     "createdAt": "2024-01-07T11:47:17.632792",  
5     "userId": 15,  
6     "dailySpend": 0,  
7     "dailyLimit": 500,  
8     "operationLimit": 250,  
9     "bills": []  
10 }
```

Response of deposit request:

The screenshot shows the Postman interface with a POST request selected with the URL `http://localhost:5154/api/accounts/deposit`. The "Body" tab is active, showing a JSON object with an "Amount" field set to 100. The "Body" tab also displays a JSON response object with fields like id, balance, createdAt, userId, dailySpend, dailyLimit, operationLimit, and bills.

```
1 {  
2     "amount": 100  
3 }
```

```
1 {  
2     "id": 10,  
3     "balance": 210,  
4     "createdAt": "2024-01-07T11:47:17.632792",  
5     "userId": 15,  
6     "dailySpend": 0,  
7     "dailyLimit": 500,  
8     "operationLimit": 250,  
9     "bills": []  
10 }
```

After deposit when getting the same account:

The screenshot shows the Postman interface. On the left, there's a sidebar titled 'My Workspace' with a tree view of API collections and environments. The main area shows a request for 'Banking.API.Project / AccountOperations Get Account By User Id'. The method is 'GET', the URL is 'http://localhost:5154/api/accounts/account-information', and the 'Authorization' header is set to 'Bearer Token' with the value 'eyJhbGciOiJIUzI1NiJ9.R5cI6lkpXVCj9.eyJ...'. The response status is '200 OK', time is '17 ms', size is '312 B', and the body contains a JSON object representing an account record.

And in the db:

The screenshot shows the DBeaver database interface. The 'BankingDB' database is selected, and the 'Accounts' table is open. The table has columns: Id, Balance, CreatedAt, UserId, DailySpend, DailyLimit, OperationLimit. There are 10 rows of data. The last row (Id: 10) has a Balance of 210, a CreatedAt timestamp of '2024-01-07 14:47:17.632', and an OperationLimit of 250.

	Id	Balance	CreatedAt	UserId	DailySpend	DailyLimit	OperationLimit
1	1	250	2024-01-07 14:42:59.424 +0300	3	0	500	250
2	2	15,000	2024-01-07 14:42:59.479 +0300	4	0	500	250
3	3	450	2024-01-07 14:42:59.489 +0300	5	0	500	250
4	4	1,500	2024-01-07 14:42:59.499 +0300	8	0	500	250
5	5	1,000	2024-01-07 14:42:59.511 +0300	9	0	500	250
6	6	400	2024-01-07 14:42:59.521 +0300	10	0	500	250
7	7	1,000	2024-01-07 14:42:59.532 +0300	11	0	500	250
8	8	1,000	2024-01-07 14:42:59.545 +0300	12	0	500	250
9	9	1,000	2024-01-07 14:42:59.558 +0300	13	0	500	250
10	10	210	2024-01-07 14:47:17.632 +0300	15	0	500	250

In records table:

SQl Server Management Studio (SSMS) screenshot showing the results of a query on the 'BankingDb' database. The query retrieves data from the 'Records' table.

ID	Timestamp	OperationType	Amount	AccountId	ReceiverAccountId	IsSuccessful	ErrorMessage	IsPending	Userid
1	2024-01-07 14:42:59.450 +0300	0	250	1	[NULL]	[NULL]	[NULL]	[]	3
2	2024-01-07 14:42:59.493 +0300	0	15,000	2	[NULL]	[NULL]	[NULL]	[]	4
3	2024-01-07 14:42:59.493 +0300	0	450	3	[NULL]	[NULL]	[NULL]	[]	5
4	2024-01-07 14:42:59.504 +0300	0	1,500	4	[NULL]	[NULL]	[NULL]	[]	8
5	2024-01-07 14:42:59.515 +0300	0	1,000	5	[NULL]	[NULL]	[NULL]	[]	9
6	2024-01-07 14:42:59.526 +0300	0	400	6	[NULL]	[NULL]	[NULL]	[]	10
7	2024-01-07 14:42:59.539 +0300	0	1,000	7	[NULL]	[NULL]	[NULL]	[]	11
8	2024-01-07 14:42:59.552 +0300	0	1,000	8	[NULL]	[NULL]	[NULL]	[]	12
9	2024-01-07 14:42:59.563 +0300	0	1,000	9	[NULL]	[NULL]	[NULL]	[]	13
10	2024-01-07 14:42:59.603 +0300	6	35	2	[NULL]	[NULL]	[NULL]	[]	4
11	2024-01-07 14:42:59.622 +0300	6	45	2	[NULL]	[NULL]	[NULL]	[]	4
12	2024-01-07 14:42:59.637 +0300	6	105	2	[NULL]	[NULL]	[NULL]	[]	4
13	2024-01-07 14:42:59.651 +0300	6	60	3	[NULL]	[NULL]	[NULL]	[]	5
14	2024-01-07 14:47:17.742 +0300	0	100	10	[NULL]	[NULL]	[NULL]	[]	15
15	2024-01-07 15:09:19.121 +0300	2	10	10	[NULL]	[NULL]	[NULL]	[]	15
16	2024-01-07 15:10:50.017 +0300	2	100	10	[NULL]	[NULL]	[NULL]	[]	15

Operation Type:

Visual Studio Code (VS Code) screenshot showing the code editor with C# files open. The current file is 'Record.cs'.

```

public enum OperationType
{
    CreateAccount,
    Payment,
    Deposit,
    Withdrawal,
    Transfer,
    CreditApplication,
    AutomaticPaymentSetup,
    SupportRequest
}

```

The code editor shows several errors in the 'PROBLEMS' panel:

- Dereference of a possibly null reference. (CS8602) [Ln 44, Col 33]
- Dereference of a possibly null reference. (CS8602) [Ln 53, Col 42]

Withdraw:

The screenshot shows the Postman interface with a project named "Banking API Project". A GET request is being made to `http://localhost:5154/api/accounts/account-information`. The "Authorization" header is set to "Bearer Token" with the value `eyJhbGciOiJIUzI1NiJ9.R5cI6lkpXVCj9.eyJ...`. The response status is 200 OK, time 88 ms, size 313 B. The response body is a JSON object:

```
1 [  
2   {"id": 10,  
3    "balance": 190,  
4    "createdAt": "2024-01-07T11:47:17.632Z",  
5    "userId": 15,  
6    "dailySpend": 20,  
7    "dailyLimit": 500,  
8    "operationLimit": 250,  
9    "bills": []  
10 }]
```

Before withdrawing the amount is 190.

After withdrawing 10 it is now 180.

The screenshot shows the Postman interface with a POST request to `http://localhost:5154/api/accounts/withdraw`. The "Body" tab is selected with the value `{"amount": 10}`. The response status is 200 OK, time 75 ms, size 313 B. The response body is identical to the previous one:

```
1 [  
2   {"id": 10,  
3    "balance": 180,  
4    "createdAt": "2024-01-07T11:47:17.632Z",  
5    "userId": 15,  
6    "dailySpend": 30,  
7    "dailyLimit": 500,  
8    "operationLimit": 250,  
9    "bills": []  
10 }]
```

The record is below. OperationType 3 means withdraw.

DBeaver 23.3.1 - Records

File Edit Navigate Search SQL Editor Database Window Help

Database Navigator Projects BankingDb public@BankingDb

Enter a part of object name here

BankingDb - localhost:6477

Properties Data ER Diagram

BankingDb Databases BankingDb Schemas BankingSchema Tables Records

Grid Text

1 2024-01-07 14:42:59.450 +0300 0 250 1 0 [NULL] [v] [NULL] [] 3 25

2 2024-01-07 14:42:59.483 +0300 0 15,000 2 0 [NULL] [v] [NULL] [] 4

3 2024-01-07 14:42:59.493 +0300 0 450 3 0 [NULL] [v] [NULL] [] 5

4 2024-01-07 14:42:59.504 +0300 0 1,500 4 0 [NULL] [v] [NULL] [] 8

5 2024-01-07 14:42:59.515 +0300 0 1,000 5 0 [NULL] [v] [NULL] [] 9

6 2024-01-07 14:42:59.526 +0300 0 400 6 0 [NULL] [v] [NULL] [] 10

7 2024-01-07 14:42:59.539 +0300 0 1,000 7 0 [NULL] [v] [NULL] [] 11

8 2024-01-07 14:42:59.552 +0300 0 1,000 8 0 [NULL] [v] [NULL] [] 12

9 2024-01-07 14:42:59.563 +0300 0 1,000 9 0 [NULL] [v] [NULL] [] 13

10 2024-01-07 14:42:59.603 +0300 6 35 2 0 [NULL] [v] [NULL] [] 4

11 2024-01-07 14:42:59.622 +0300 6 45 2 0 [NULL] [v] [NULL] [] 4

12 2024-01-07 14:42:59.637 +0300 6 105 2 0 [NULL] [v] [NULL] [] 4

13 2024-01-07 14:42:59.651 +0300 6 60 3 0 [NULL] [v] [NULL] [] 5

14 2024-01-07 14:42:59.742 +0300 0 100 10 0 [NULL] [v] [NULL] [] 15

15 2024-01-07 14:59:51.121 +0300 2 10 0 [NULL] [v] [NULL] [] 15

16 2024-01-07 15:10:50.017 +0300 2 100 0 [NULL] [v] [NULL] [] 15

17 2024-01-07 15:10:51.171 +0300 3 15 0 [NULL] [v] [NULL] [] 15

18 2024-01-07 15:23:17.389 +0300 3 240 10 0 [NULL] [v] [] Insufficient funds for [] 15

19 2024-01-07 15:23:16.569 +0300 3 5 10 0 [NULL] [v] [NULL] [] 15

20 2024-01-07 15:23:20.320 +0300 3 240 10 0 [NULL] [v] [] Insufficient funds for [] 15

21 2024-01-07 15:23:20.166 +0300 3 260 2 0 [NULL] [v] [] Operation limit exceed [v] 4

22 2024-01-07 15:23:20.206 +0300 3 260 2 0 [NULL] [v] [] Operation limit exceed [v] 4

23 2024-01-07 15:23:20.206 +0300 3 260 2 0 [NULL] [v] [] Operation limit exceed [v] 4

24 2024-01-07 15:33:45.966 +0300 3 260 2 0 [NULL] [v] [NULL] [] 4

25 2024-01-07 19:00:17.128 +0300 3 10 10 0 [NULL] [v] [NULL] [] 15

Project - General

Name: DataSou

Record

Refres... Save Cancel Export data 200 25 25 row(s) fetched - 1ms, on 2024-01-07 at 19:00:54

TRT en

5°C Sicaklık tahmini

Search

7:01 PM 1/7/2024

21 references

```
public enum OperationType
{
    CreateAccount,
    Payment,
    Deposit,
    Withdrawal,
    Transfer,
    CreditApplication,
    AutomaticPaymentSetup,
    SupportRequest
}
```

You, 4 days ago • Account & Record entities

If balance is less than amount: returns a bad request.

The screenshot shows the Postman interface with a POST request to `http://localhost:5154/api/accounts/withdraw`. The request body is JSON with the key `Amount` set to 240. The response status is 400 Bad Request, with the message "Insufficient funds for the withdrawal."

```
1 {
2   "Amount": 240
3 }
```

```
1 {
2   "StatusCode": 400,
3   "Message": [
4     "Insufficient funds for the withdrawal."
5   ]
6 }
```

If the amount exceeds OperationLimit it returns a bad request.

The screenshot shows the Postman interface with a POST request to `http://localhost:5154/api/accounts/withdraw`. The request body is JSON with the key `Amount` set to 260. The response status is 400 Bad Request, with the message "Operation limit exceeded for account with id 2".

```
1 {
2   "Amount": 260
3 }
```

```
1 {
2   "StatusCode": 400,
3   "Message": [
4     "Operation limit exceeded for account with id 2"
5   ]
6 }
```

Editing DailySpend value for account id 10. Now sending the same request.

DBeaver 23.3.1 - Accounts

BankingDB - localhost:6477

Properties Data LR Diagram

BankingDb Databases BankingDb Schemas BankingSchema Tables Accounts

Enter a part of object name here

Grid

	Id	Balance	CreatedAt	UserId	DailySpend	DailyLimit	OperationLimit	Value
1	1	259	2024-01-07 19:08:58.213 +0300	3	0	500	250	495
2	2	15,000	2024-01-07 19:08:58.260 +0300	4	0	500	250	
3	3	495	2024-01-07 19:08:58.377 +0300	5	0	500	250	
4	4	1,505	2024-01-07 19:08:58.286 +0300	8	0	500	250	
5	5	1,000	2024-01-07 19:08:58.295 +0300	9	0	500	250	
6	6	403	2024-01-07 19:08:58.303 +0300	10	0	500	250	
7	7	1,000	2024-01-07 19:08:58.312 +0300	11	0	500	250	
8	8	1,000	2024-01-07 19:08:58.321 +0300	12	0	500	250	
9	9	1,000	2024-01-07 19:08:58.329 +0300	13	0	500	250	
10	10	100	2024-01-07 19:15:09.469 +0300	15	495	500	250	

Project - General

Name: DataSou

Bookmarks Diagrams Scripts

Refresh Save Cancel Export data 200 10 10 row(s) fetched - 2ms, on 2024-01-07 at 19:16:16

TRT en 7:16 PM 1/7/2024

Postman

Banking API Project / AccountOperations Create Withdraw DailySpend

POST http://localhost:5154/api/accounts/withdraw

Params Authorization Headers Body Pre-request Script Tests Settings

Body

```
1 {
2   ... "Amount": 15
3 }
```

Body Cookies Headers (5) Test Results

Status: 400 Bad Request Time: 47 ms Size: 251 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "StatusCode": 400,
3   "Message": [
4     "Daily spending limit exceeded for account with id 10"
5   ]
6 }
```

Postbot Runner Start Proxy Cookies Trash

The screenshot shows a DBVisualizer interface with a grid of transaction records. The columns include Id, TimeStamp, OperationType, Amount, AccountId, ReceiverAccountId, IsSuccessful, ErrorMessage, IsPending, and UserId. Record 16 is highlighted, showing an error message in the ErrorMessage column: "Daily spending limit exceeded for account with id 10".

ID	TimeStamp	OperationType	Amount	AccountId	ReceiverAccountId	IsSuccessful	ErrorMessage	IsPending	UserId
1	2024-01-07 19:08:58.240 +0300	0	250	1	[NULL]	✓	[NULL]	[]	3
2	2024-01-07 19:08:58.272 +0300	0	15000	2	[NULL]	✓	[NULL]	[]	4
3	2024-01-07 19:08:58.281 +0300	0	450	3	[NULL]	✓	[NULL]	[]	5
4	2024-01-07 19:08:58.290 +0300	0	1500	4	[NULL]	✓	[NULL]	[]	8
5	2024-01-07 19:08:58.298 +0300	0	1000	5	[NULL]	✓	[NULL]	[]	9
6	2024-01-07 19:08:58.307 +0300	0	400	6	[NULL]	✓	[NULL]	[]	10
7	2024-01-07 19:08:58.316 +0300	0	1000	7	[NULL]	✓	[NULL]	[]	11
8	2024-01-07 19:08:58.325 +0300	0	1000	8	[NULL]	✓	[NULL]	[]	12
9	2024-01-07 19:08:58.333 +0300	0	1000	9	[NULL]	✓	[NULL]	[]	13
10	2024-01-07 19:08:58.372 +0300	6	35	2	[NULL]	✓	[NULL]	[]	4
11	2024-01-07 19:08:58.387 +0300	6	45	2	[NULL]	✓	[NULL]	[]	4
12	2024-01-07 19:08:58.402 +0300	6	105	2	[NULL]	✓	[NULL]	[]	4
13	2024-01-07 19:08:58.416 +0300	6	60	3	[NULL]	✓	[NULL]	[]	5
14	2024-01-07 19:10:58.443 +0300	3	650	2	[NULL]	[]	Daily spending limit reached	[]	4
15	2024-01-07 19:15:09.498 +0300	0	100	10	[NULL]	✓	[NULL]	[]	15
16	2024-01-07 19:16:55.508 +0300	3	15	10	[NULL]	[]	Daily spending limit exceeded for account with id 10	[]	15

Record has been created too with an error message.

Transfer:

If the receiver account id does not exist.

The screenshot shows a Postman interface with a request to "Create Transfer DTO Valid". The request method is POST, URL is http://localhost:5154/api/accounts/transfer, and the Body is set to JSON. The response status is 400 Bad Request, and the response body is: "StatusCode": 400, "Message": ["Account with id 42 does not exist in the database"].

```

{
  "StatusCode": 400,
  "Message": [
    "Account with id 42 does not exist in the database"
  ]
}

```

DBeaver 23.3.1 - Accounts

BankingDB - public@BankingDb

Properties Data DLR Diagram

	Id	Balance	CreatedAt	UserId	DailySpend	DailyLimit	OperationLimit
1	1	259	2024-01-07 19:09:58.213 +0300	3	0	500	259
2	2	15,000	2024-01-07 19:09:58.269 +0300	4	0	500	259
3	3	450	2024-01-07 19:09:58.277 +0300	5	0	500	259
4	4	1,500	2024-01-07 19:09:58.286 +0300	8	0	500	259
5	5	1,000	2024-01-07 19:09:58.295 +0300	9	0	500	259
6	6	400	2024-01-07 19:09:58.303 +0300	10	0	500	259
7	7	1,000	2024-01-07 19:09:58.312 +0300	11	0	500	259
8	8	1,000	2024-01-07 19:09:58.321 +0300	12	0	500	259
9	9	1,000	2024-01-07 19:09:58.329 +0300	13	0	500	259
10	10	100	2024-01-07 19:15:09.469 +0300	15	495	500	259

Project - General

Record

Refresh Save Cancel Export data 200 10 rows fetched - 2ms, on 2024-01-07 at 19:16:16 7:28 PM 1/7/2024

Before sending a valid transfer from account 2 to account 5.

After sending a valid transfer:

Postman

Banking API Project / AccountOperations Create Transfer DTO Valid

POST http://localhost:5154/api/accounts/transfer

Body

```
{
  "amount": 25,
  "receiverAccountId": 5
}
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

Status: 200 OK Time: 91 ms Size: 473 B Save as example

Postbot Runner Start Proxy Cookies Trash

Updated accounts:

DBeaver 23.3.1 - Accounts

BankingDB - localhost:6477

Properties Data LR Diagram

Enter a part of object name here

BankingDb Databases BankingDb Schemas BankingSchema Tables Accounts

BankingDb - localhost:6477

Grid Text

#	Id	Balance	CreatedAt	Userid	DailySpend	DailyLimit	OperationLimit
1	1	250	2024-01-07 19:08:58.213 +0300	3	0	500	250
2	3	450	2024-01-07 19:08:58.277 +0300	5	0	500	250
3	4	1,500	2024-01-07 19:08:58.286 +0300	8	0	500	350
4	6	400	2024-01-07 19:08:58.303 +0300	10	0	500	250
5	7	1,000	2024-01-07 19:08:58.311 +0300	11	0	500	250
6	8	1,000	2024-01-07 19:08:58.321 +0300	12	0	500	250
7	9	1,000	2024-01-07 19:08:58.329 +0300	13	0	500	250
8	10	100	2024-01-07 19:15:09.460 +0300	15	495	500	250
9	2	14,975	2024-01-07 19:08:58.264 +0300	4	25	500	250
10	5	1,025	2024-01-07 19:08:58.295 +0300	9	0	500	250

Project - General > Name: DataSources

Bookmarks Diagrams Scripts

Refresh Save Cancel Export data 200 10 10 row(s) fetched - 2ms, on 2024-01-07 at 19:29:08 TRT en 7:29 PM 1/7/2024

Çok bulutlu 2°C

Created record:

DBeaver 23.3.1 - Records

BankingDB - localhost:6477

Properties Data LR Diagram

Enter a part of object name here

BankingDb Databases BankingDb Schemas BankingSchema Tables Records

BankingDb - localhost:6477

Grid Text

#	Id	TimeStamp	OperationType	Amount	AccountId	ReceiverAccountId	IsSuccessful	ErrorMessage	IsPending	Userid
1	1	2024-01-07 19:08:58.240 +0300	0	250	1	[NULL]	[v]	[NULL]	[]	3
2	2	2024-01-07 19:08:58.272 +0300	0	15,000	2	[NULL]	[v]	[NULL]	[]	4
3	3	2024-01-07 19:08:58.281 +0300	0	450	3	[NULL]	[v]	[NULL]	[]	5
4	4	2024-01-07 19:08:58.290 +0300	0	1,500	4	[NULL]	[v]	[NULL]	[]	8
5	5	2024-01-07 19:08:58.296 +0300	0	1,000	5	[NULL]	[v]	[NULL]	[]	9
6	6	2024-01-07 19:08:58.307 +0300	0	400	6	[NULL]	[v]	[NULL]	[]	10
7	7	2024-01-07 19:08:58.316 +0300	0	1,000	7	[NULL]	[v]	[NULL]	[]	11
8	8	2024-01-07 19:08:58.325 +0300	0	1,000	8	[NULL]	[v]	[NULL]	[]	12
9	9	2024-01-07 19:08:58.333 +0300	0	1,000	9	[NULL]	[v]	[NULL]	[]	13
10	10	2024-01-07 19:08:58.337 +0300	6	35	2	[NULL]	[v]	[NULL]	[]	4
11	11	2024-01-07 19:08:58.338 +0300	6	25	2	[NULL]	[v]	[NULL]	[]	4
12	12	2024-01-07 19:08:58.402 +0300	6	105	2	[NULL]	[v]	[NULL]	[]	4
13	13	2024-01-07 19:08:58.416 +0300	6	60	3	[NULL]	[v]	[NULL]	[]	5
14	14	2024-01-07 19:08:58.443 +0300	3	650	2	[NULL]	[v]	[]	Daily spending limit reached	4
15	15	2024-01-07 19:15:09.498 +0300	0	100	10	[NULL]	[v]	[NULL]	[]	15
16	16	2024-01-07 19:16:55.508 +0300	3	15	10	[NULL]	[v]	Daily spending limit reached	[]	15
17	17	2024-01-07 19:26:52.320 +0300	4	25	2	[v]	[NULL]	[v]	[]	4

Project - General > Name: DataSources

Bookmarks Diagrams Scripts

Refresh Save Cancel Export data 200 17 17 row(s) fetched - 2ms, on 2024-01-07 at 19:29:42 TRT en 7:29 PM 1/7/2024

Çok bulutlu 2°C

Support Role Operations

Get All Record By Querying:

Supports can query records. They can get records that are pending, or not pending. They can sort records, or not sort.

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'My Workspace' containing collections like 'API Gateway Requests', 'Banking API Project' (which has a 'User Operations' folder and a 'Support Operations' folder with a 'Get All Records' request), and other service collections. The main workspace shows a 'GET Get All Records' request under 'Banking API Project / Support Operations / Get All Records'. The request URL is `http://localhost:5154/api/support/records?PageNumber=1&PageSize=50`. The 'Params' tab shows parameters: PageNumber (1), PageSize (50), IsPending (true), UserId (5), and OrderBy (account.id asc, id desc). The 'Body' tab shows the response body as JSON:

```
1 {
2   "id": 1,
3   "timeStamp": "2024-01-07T16:08:58.240Z",
4   "operationType": "CreateAccount",
5   "amount": 200,
6   "userId": 3,
7   "accountId": 1,
8   "receiverAccountId": null,
9   "isSuccessful": true,
10  "errorMessage": null,
11  "isPending": false
12 },
13 {
14   "id": 2,
15   "timeStamp": "2024-01-07T16:08:58.272Z"
16 }
```

The screenshot shows the Postman application interface. The left sidebar is titled "My Workspace" and lists various collections and environments. The main area displays a POST request to "Get All Records" with parameters: PageNumber=1, PageSize=50, OrderBy=account.Id desc, id desc. The response body is shown as JSON, detailing two withdrawal operations. The bottom status bar indicates the request was successful with a status of 200 OK, took 19 ms, and had a size of 3.82 KB.

```
[{"id": 1, "timeStamp": "2024-01-07T16:16:55.508263Z", "operationType": "Withdrawal", "amount": 100, "userId": 15, "account": null, "receiverAccountId": null, "isSuccessful": false, "errorMessage": "Daily spending limit exceeded for account with id 10", "isPending": false}, {"id": 2, "timeStamp": "2024-01-07T16:15:09.498981Z", "operationType": "CreateAccount", "amount": 100, "userId": 15, "account": {"id": 1, "name": "Savings Account", "balance": 100, "type": "Savings"}, "receiverAccountId": null, "isSuccessful": true, "errorMessage": null, "isPending": false}], [{"id": 1, "timeStamp": "2024-01-07T16:16:55.508263Z", "operationType": "Withdrawal", "amount": 100, "userId": 15, "account": null, "receiverAccountId": null, "isSuccessful": false, "errorMessage": "Daily spending limit exceeded for account with id 10", "isPending": false}, {"id": 2, "timeStamp": "2024-01-07T16:15:09.498981Z", "operationType": "CreateAccount", "amount": 100, "userId": 15, "account": {"id": 1, "name": "Savings Account", "balance": 100, "type": "Savings"}, "receiverAccountId": null, "isSuccessful": true, "errorMessage": null, "isPending": false}]]
```

Now sending a transfer request that will exceed the operation limit that is 200.

The screenshot shows the Postman interface with a collection named "Banking.API.Project". A POST request to `/accounts/transfer` is being tested. The body contains a JSON object with `Amount: 260` and `ReceiverAccountId: 5`. The response status is 400 Bad Request, with the message: `"Operation limit exceeded for account with id 2"`.

Support can get this record via querying.

The screenshot shows the Postman interface with a collection named "Banking.API.Project". A GET request to `/support/records? pageNumber=1& pageSize=50& isPending=true` is being tested. The query parameters are `pageNumber: 1`, `pageSize: 50`, and `isPending: true`. The response status is 200 OK, returning a list of records, one of which is a pending transfer.

In records table:

	<code>Id</code>	<code>TimeStamp</code>	<code>OperationType</code>	<code>Amount</code>	<code>AccountId</code>	<code>ReceiverAccountId</code>	<code>IsSuccessful</code>	<code>ErrorMessage</code>	<code>IsPending</code>	<code>UserId</code>
1	1	2024-01-07 19:37:56.455 +0300	0	250	1	[NULL]	[NULL]	[NULL]	[]	3
2	2	2024-01-07 19:37:56.487 +0300	0	15000	2	[NULL]	[NULL]	[NULL]	[]	4
3	3	2024-01-07 19:37:56.498 +0300	0	-450	3	[NULL]	[NULL]	[NULL]	[]	5
4	4	2024-01-07 19:37:56.504 +0300	0	1500	4	[NULL]	[NULL]	[NULL]	[]	8
5	5	2024-01-07 19:37:56.514 +0300	0	1000	5	[NULL]	[NULL]	[NULL]	[]	9
6	6	2024-01-07 19:37:56.524 +0300	0	400	6	[NULL]	[NULL]	[NULL]	[]	10
7	7	2024-01-07 19:37:56.533 +0300	0	1000	7	[NULL]	[NULL]	[NULL]	[]	11
8	8	2024-01-07 19:37:56.542 +0300	0	1000	8	[NULL]	[NULL]	[NULL]	[]	12
9	9	2024-01-07 19:37:56.550 +0300	0	1000	9	[NULL]	[NULL]	[NULL]	[]	13
10	10	2024-01-07 19:37:56.585 +0300	6	35	2	[NULL]	[NULL]	[NULL]	[]	4
11	11	2024-01-07 19:37:56.597 +0300	6	45	2	[NULL]	[NULL]	[NULL]	[]	4
12	12	2024-01-07 19:37:56.610 +0300	6	105	2	[NULL]	[NULL]	[NULL]	[]	4
13	13	2024-01-07 19:37:56.623 +0300	6	60	3	[NULL]	[NULL]	[NULL]	[]	5
14	14	2024-01-07 19:40:19.566 +0300	4	260	2	5	[]	Operation limit exceed	[]	4

Record with id has been created in the last request. And IsPending = true. Support can execute this transfer that is blocked by the system.

Accounts right before executing transfer by support

	<code>Id</code>	<code>Balance</code>	<code>CreatedAt</code>	<code>UserId</code>	<code>DailySpend</code>	<code>DailyUnit</code>	<code>OperationLimit</code>	<code>Value</code>
1	1	250	2024-01-07 19:37:56.429 +0300	3	0	500	250	
2	2	15.000	2024-01-07 19:37:56.487 +0300	4	0	500	250	
3	3	450	2024-01-07 19:37:56.498 +0300	5	0	500	250	
4	4	1.500	2024-01-07 19:37:56.501 +0300	8	0	500	250	
5	5	1.000	2024-01-07 19:37:56.510 +0300	9	0	500	250	
6	6	400	2024-01-07 19:37:56.519 +0300	10	0	500	250	
7	7	1.000	2024-01-07 19:37:56.529 +0300	11	0	500	250	
8	8	1.000	2024-01-07 19:37:56.538 +0300	12	0	500	250	
9	9	1.000	2024-01-07 19:37:56.547 +0300	13	0	500	250	

Now executing pending record.

The screenshot shows the Postman interface. On the left, the 'My Workspace' sidebar lists various API collections and environments. The main workspace shows a 'Banking.API.Project / Support Operations Execute Pending Request' collection. A 'GET' request is selected with the URL `http://localhost:5154/api/support/pending-withdrawal/14`. The 'Authorization' tab is active, showing a 'Bearer Token' type with a token value. The response body shows a single character '1'. At the bottom, the status bar indicates 'Status: 200 OK'.

It returned 200.

And in db:

A new record has been created for the transfer. Old record that is pending is now not pending.

The screenshot shows the DBeaver database interface. The central window displays a table named 'Records' with columns: Id, OperationType, Amount, AccountId, ReceiverAccountId, IsSuccessful, ErrorMessage, IsPending, and UserId. The table contains 15 rows of data. Row 14 is highlighted in blue, showing a transfer from account 1 to account 5 for 200 units. The 'ErrorMessage' column for this row contains the message 'Operation limit exceed'. The bottom status bar shows '15 row(s) fetched - 2ms, on 2024-01-07 at 19:47:43'.

Also Accounts:

DBeaver 23.3.1 - Accounts

BankingDB - localhost:6477

Properties Data LR Diagram

#	Id	Balance	CreatedAt	UserId	DailySpend	DailyLimit	OperationLimit
1	1	250	2024-01-07 19:37:56.429 +0300	3	0	500	250
2	2	14740	2024-01-07 19:37:56.490 +0300	4	260	500	250
3	3	450	2024-01-07 19:37:56.490 +0300	5	0	500	250
4	4	1500	2024-01-07 19:37:56.501 +0300	8	0	500	350
5	7	1000	2024-01-07 19:37:56.529 +0300	11	0	500	250
6	8	1000	2024-01-07 19:37:56.530 +0300	12	0	500	250
7	9	1000	2024-01-07 19:37:56.547 +0300	13	0	500	250
8	2	14740	2024-01-07 19:37:56.490 +0300	4	260	500	250
9	5	1260	2024-01-07 19:37:56.510 +0300	9	0	500	250

Project - General > DataSources

Record

Refresh Save Cancel Export data 200 9 9 rows(fetched - 1ms. on 2024-01-07 at 19:48:34) TRT en 7:48 PM 1/7/2024

And support tries to get pending records:

Postman

My Workspace

Banking API Project / Support Operations Get All Records

GET http://localhost:5154/api/support/records?PageNumber=1&PageSize=50&IsPending=true

Params

Key	Value	Description
PageNumber	1	
PageSize	50	
IsPending	true	

Body

Status: 200 OK Time: 20 ms Size: 287 B

1

And if try to execute the same record one more time:

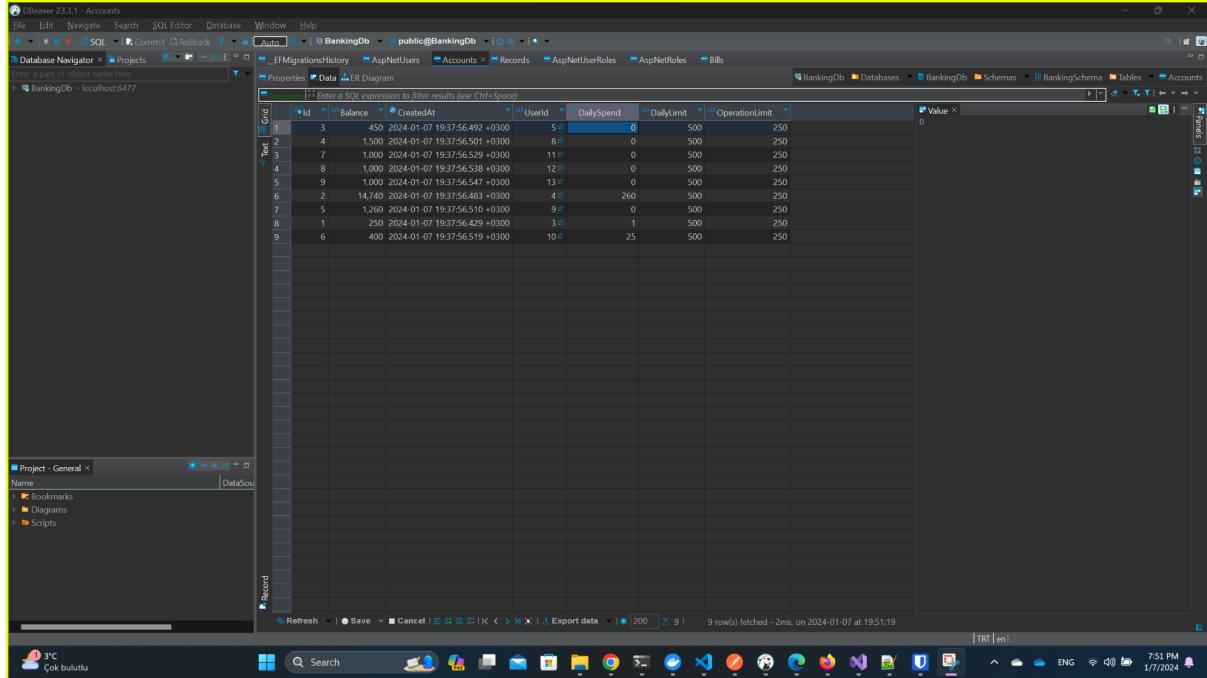
The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar lists various collections and environments. The main workspace displays a 'Banking.API.Project / Support Operations' collection with a 'Execute Pending Request' endpoint selected. The request details show a GET method to 'http://localhost:5154/api/support/pending-withdrawal/14'. The 'Authorization' tab is active, showing a 'Bearer Token' type with a token value. The response body is displayed in JSON format, indicating a 400 Bad Request error with the message: "No pending record with id '14'". The status bar at the bottom shows the date and time as 1/7/2024 7:49 PM.

Gets 400 bad request.

Background Services

Reset Daily Spend:

Accounts daily spends before background service.



	#Id	Balance	CreatedAt	UserId	DailySpend	DailyLimit	OperationLimit	Value
1	3	450	2024-01-07 19:37:56.409 +0300	5	0	500	250	0
2	4	1,500	2024-01-07 19:37:56.501 +0300	8	0	500	250	0
3	7	1,000	2024-01-07 19:37:56.529 +0300	11	0	500	250	0
4	8	1,000	2024-01-07 19:37:56.539 +0300	12	0	500	250	0
5	9	1,000	2024-01-07 19:37:56.541 +0300	13	0	500	250	0
6	2	14,740	2024-01-07 19:37:56.483 +0300	4	260	500	250	0
7	5	1,260	2024-01-07 19:37:56.510 +0300	9	0	500	250	0
8	1	250	2024-01-07 19:37:56.429 +0300	3	1	500	250	0
9	6	400	2024-01-07 19:37:56.519 +0300	10	25	500	250	0

At 19.55, daily spend background service will be executed.

```
1 reference
public static IServiceCollection ConfigureQuartzJobs(this IServiceCollection services)
{
    // Add Quartz services
    services.AddQuartz(q =>
    {
        q.UseMicrosoftDependencyInjectionJobFactory();

        var dailySpendJobKey = JobKey.Create(nameof(DailySpendResetService));
        var payBillsJobKey = JobKey.Create(nameof(PayBillsBackgroundService));

        q.AddJob<DailySpendResetService>(dailySpendJobKey)
            .AddTrigger(trigger =>
                trigger.ForJob(dailySpendJobKey)
                    .WithSchedule(CronScheduleBuilder.DailyAtHourAndMinute(19, 55)));

        q.AddJob<PayBillsBackgroundService>(payBillsJobKey)
            .AddTrigger(trigger =>
                trigger.ForJob(payBillsJobKey)
                    .WithSchedule(CronScheduleBuilder.DailyAtHourAndMinute(18, 57)));
    });

    services.AddQuartzHostedService(options =>
    {
        options.WaitForJobsToComplete = true;
    });

    return services;
}
```

```

PowerShell PowerShell PowerShell
2024-01-07 19:53:43 [INF] Executed DbCommand ("1"ms) [Parameters=[], CommandType='Text', CommandTimeout='30']"r\n"SELECT EXISTS (r\n      SELECT 1r\n      FROM `Bank
ingsSchema`.\"Bills\" AS b)"
2024-01-07 19:53:43 [DBG] TaskSchedulingThreadPool configured with max concurrency of 10 and TaskScheduler ThreadPoolTaskScheduler.
2024-01-07 19:53:43 [INF] Initialized Scheduler Signaller of type: Quartz.Core.SchedulerSignalerImpl
2024-01-07 19:53:43 [INF] Quartz Scheduler created
2024-01-07 19:53:43 [INF] JobFactory set to: Quartz.Simpl.MicrosoftDependencyInjectionJobFactory
2024-01-07 19:53:43 [INF] RAMJobStore initialized.
2024-01-07 19:53:43 [INF] Quartz Scheduler 3.6.3.0 - 'QuartzScheduler' with instanceId 'NON_CLUSTERED' initialized
2024-01-07 19:53:43 [INF] Using thread pool 'Quartz.Simpl.DefaultThreadPool', size: 10
2024-01-07 19:53:43 [INF] Using job store 'Quartz.Simpl.RAMJobStore', supports persistence: False, clustered: False
2024-01-07 19:53:43 [INF] Adding 2 jobs, 2 triggers.
2024-01-07 19:53:43 [INF] Adding job: DEFAULT.DailySpendResetService
2024-01-07 19:53:43 [DBG] Scheduling job: "DEFAULT.DailySpendResetService" with trigger: "DEFAULT.b185b74b-a7eb-4aae-9658-4e540bef3003"
2024-01-07 19:53:43 [INF] Adding job: DEFAULT.PayBillsBackgroundService
2024-01-07 19:53:43 [DBG] Scheduling job: "DEFAULT.PayBillsBackgroundService" with trigger: "DEFAULT.54fc5a0f-01be-428b-bad4-8250a0323048"
2024-01-07 19:53:43 [DBG] Rescheduling job: DEFAULT.DailySpendResetService with updated trigger: DEFAULT.b185b74b-a7eb-4aae-9658-4e540bef3003
2024-01-07 19:53:43 [DBG] Rescheduling job: DEFAULT.PayBillsBackgroundService with updated trigger: DEFAULT.54fc5a0f-01be-428b-bad4-8250a0323048
2024-01-07 19:53:43 [INF] Now listening on: "http://localhost:5154"
2024-01-07 19:53:43 [INF] Application started. Press Ctrl+C to shut down.
2024-01-07 19:53:43 [INF] Hosting environment: "Development"
2024-01-07 19:53:43 [INF] Content root path: "C:\Users\veyse\Desktop\Patika\FinalCase\source\Banking.API"
2024-01-07 19:53:43 [INF] Scheduler QuartzScheduler$_NON_CLUSTERED started.
2024-01-07 19:53:43 [DBG] Batch acquisition of 0 triggers
2024-01-07 19:54:12 [DBG] Batch acquisition of 0 triggers
2024-01-07 19:54:38 [DBG] Batch acquisition of 1 triggers
2024-01-07 19:55:00 [DBG] Batch acquisition of 0 triggers
2024-01-07 19:55:00 [DBG] Calling Execute on job DEFAULT.DailySpendResetService
2024-01-07 19:55:00 [INF] Processing daily spend reset at midnight...
2024-01-07 19:55:00 [INF] Executed DbCommand ("6"ms) [Parameters=[], CommandType='Text', CommandTimeout='30']"r\n"SELECT a.\"Id\", a.\"Balance\", a.\"CreatedAt\", a
.\"DailyLimit\", a.\"DailySpend\", a.\"OperationLimit\", a.\"UserId\", a.xmin\r\nFROM `BankingsSchema`.\"Accounts\" AS a"
2024-01-07 19:55:00 [INF] Executed DbCommand ("6"ms) [Parameters=[@p1=? (DbType = Int32), @p0=? (DbType = Int32), @p2=? (DbType = Object), @p4=? (DbType = Int32),
@p3=? (DbType = Int32), @p5=? (DbType = Object), @p7=? (DbType = Int32), @p6=? (DbType = Int32), @p8=? (DbType = Object)]]", CommandType='Text', CommandTimeout='30']"r\n"UPDATE `BankingsSchema`.\"Accounts\" SET \"DailySpend\" = @p0\r\nWHERE \"Id\" = @p1 AND xmin = @p2\r\nRETURNING xmin;r\nUPDATE `BankingsSchema`.\"Accounts\" SET \"DailySpend\" = @p6\r\nWHERE \"Id\" = @p7 AND xmin = @p8\r\nRETURNING xmin;"r\n
2024-01-07 19:55:00 [INF] Daily spend reset successfully completed.
2024-01-07 19:55:00 [DBG] Trigger instruction : NoInstruction
2024-01-07 19:55:00 [DBG] Batch acquisition of 0 triggers

```



Accounts after execution:

Screenshot of DBeaver 23.3.1 showing the Accounts table after execution.

Properties:

- Grid
- Value

Columns:

- Id
- Balance
- CreatedAt
- UserId
- DailySpend
- DailyLimit
- OperationLimit
- Value

Table Data:

Grid	Value
1	3
2	4
3	7
4	8
5	9
6	5
7	1
8	2
9	6

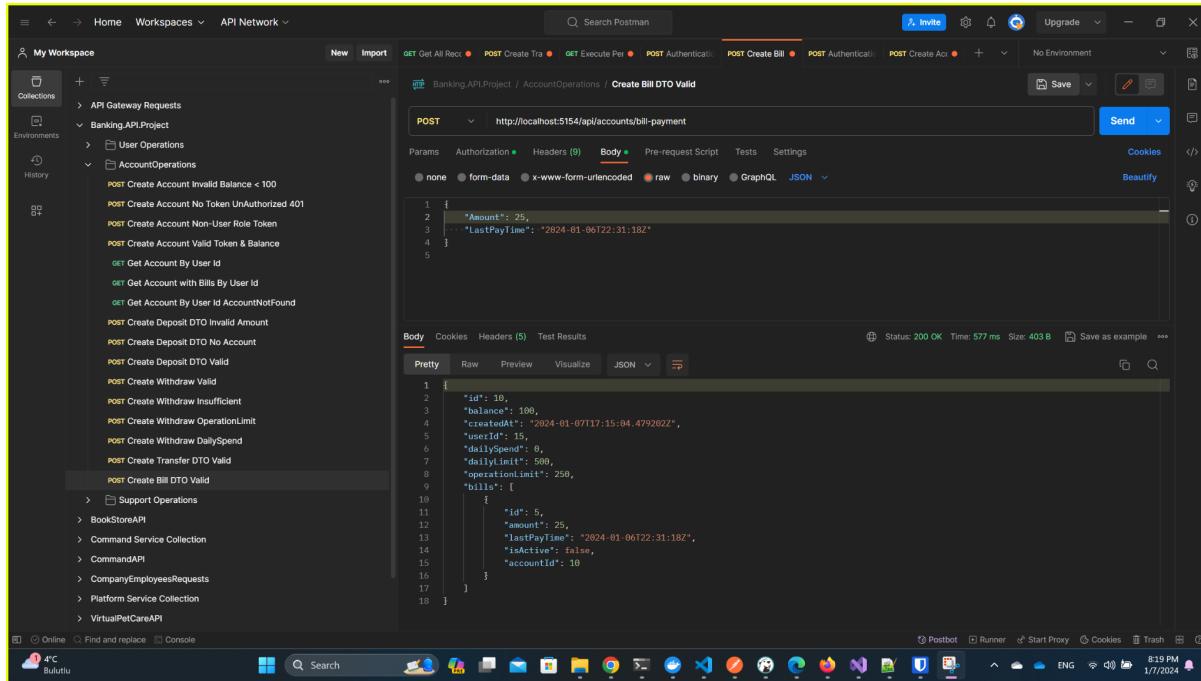
Project - General:

- Names: Bills
- Bookmarks
- Diagrams
- scripts

Bottom Taskbar:

Pay Bills BackgroundService:

Users can add bill.



POST http://localhost:5154/api/accounts/bill-payment

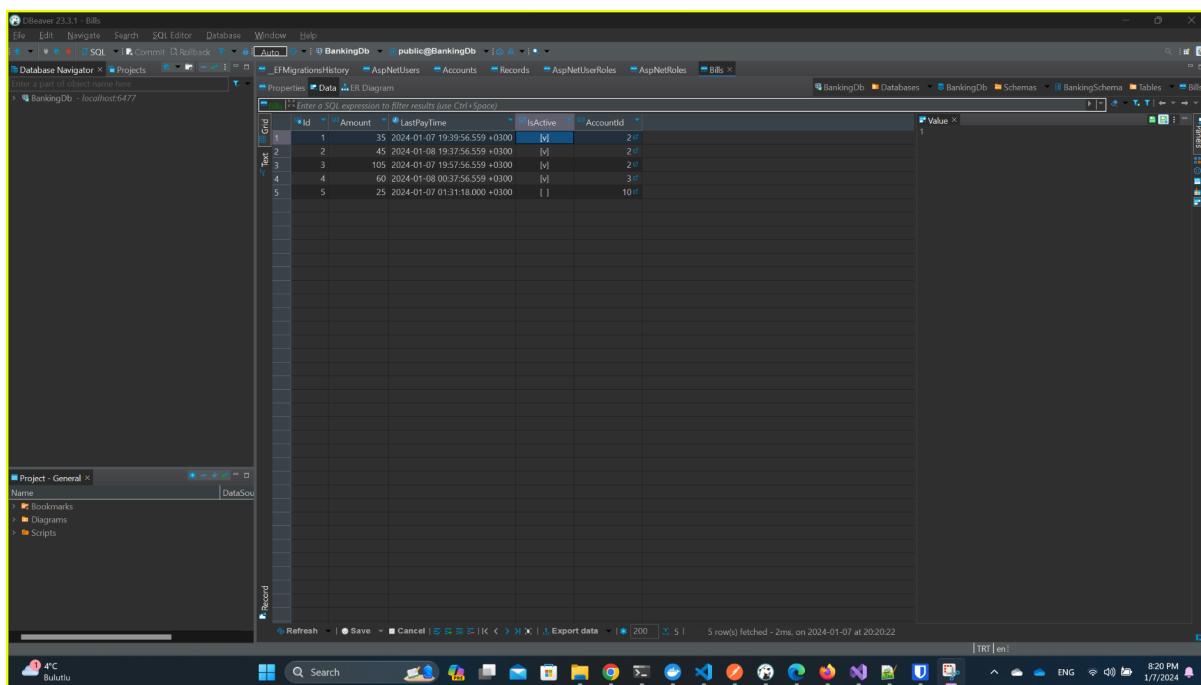
Body (JSON)

```
1 {
2     "Amount": 25,
3     "LastPayTime": "2024-01-06T22:31:18Z"
4 }
```

Body Cookies Headers (5) Test Results

Status: 200 OK Time: 577 ms Size: 403 B Save as example

In Bills table:



Id	Amount	LastPayTime	IsActive	AccountId
1	35	2024-01-07 19:39:56.559 +0300	M	2
2	45	2024-01-08 19:37:56.559 +0300	M	2
3	105	2024-01-07 19:57:56.559 +0300	M	2
4	60	2024-01-08 00:37:56.559 +0300	M	3
5	25	2024-01-07 01:31:18.000 +0300		10

Firing the background service at 20:27

```

1 reference
public static IServiceProvider ConfigureQuartzJobs(this IServiceProvider services)
{
    // Add Quartz services
    services.AddQuartz(q =>
    {
        q.UseMicrosoftDependencyInjectionJobFactory();

        var dailySpendJobKey = JobKey.Create(nameof(DailySpendResetService));
        var payBillsJobKey = JobKey.Create(nameof(PayBillsBackgroundService));

        q.AddJob<DailySpendResetService>(dailySpendJobKey)
            .AddTrigger(trigger =>
                trigger.ForJob(dailySpendJobKey)
                    .WithSchedule(CronScheduleBuilder.DailyAtHourAndMinute(19, 55)));

        q.AddJob<PayBillsBackgroundService>(payBillsJobKey)
            .AddTrigger(trigger =>
                trigger.ForJob(payBillsJobKey)
                    .WithSchedule(CronScheduleBuilder.DailyAtHourAndMinute(20, 27)));
    });
}

You, 5 seconds ago • Uncommitted changes
services.AddQuartzHostedService(options =>
{
    options.WaitForJobsToComplete = true;
});

```

```

PowerShell          PowerShell          PowerShell          + 
M \\BankingSchema\".\"Bills\" AS b\r\n      WHERE a.\"Id\" = b.\"AccountId\" IS NULL)\r\nORDER BY a.\"Id\""
2024-01-07 20:27:00 [DBG] Account ID: 2, Balance: 14748, DailySpend: 0
2024-01-07 20:27:00 [DBG] Account new balance: 14705
2024-01-07 20:27:00 [DBG] Bill new date: 02/07/2024 16:39:56
2024-01-07 20:27:00 [DBG] Bill ID: 1, Amount: 35, LastPayTime: 2/7/2024 4:39:56 PM
2024-01-07 20:27:00 [DBG] Account new balance: 14600
2024-01-07 20:27:00 [DBG] Bill new date: 02/07/2024 16:57:56
2024-01-07 20:27:00 [DBG] Bill ID: 3, Amount: 105, LastPayTime: 2/7/2024 4:57:56 PM
2024-01-07 20:27:00 [DBG] Account ID: 3, Balance: 450, DailySpend: 0
2024-01-07 20:27:00 [DBG] Account new balance: 390
2024-01-07 20:27:00 [DBG] Bill new date: 02/07/2024 21:37:56
2024-01-07 20:27:00 [DBG] Bill ID: 4, Amount: 60, LastPayTime: 2/7/2024 9:37:56 PM
2024-01-07 20:27:00 [DBG] Account ID: 10, Balance: 100, DailySpend: 0
2024-01-07 20:27:00 [DBG] Account new balance: 75
2024-01-07 20:27:00 [DBG] Bill new date: 02/06/2024 22:31:18
2024-01-07 20:27:00 [DBG] Bill ID: 5, Amount: 25, LastPayTime: 2/6/2024 10:31:18 PM
2024-01-07 20:27:00 [INF] Executed SqlCommand ("6ms") [Parameters='{"@p1=?' (DbType = Int32), "@p0=?' (DbType = Object), "@p4=?' (DbType = Int32), "@p3=?' (DbType = Int32), "@p5=?' (DbType = Object), "@p7=?' (DbType = Int32), "@p6=?' (DbType = Object), "@p8=?' (DbType = Int32), "@p9=?' (DbType = DateTime), "@p10=?' (DbType = Int32), "@p11=?' (DbType = DateTime), "@p12=?' (DbType = Int32), "@p13=?' (DbType = Int32), "@p14=?' (DbType = Int32), "@p15=?' (DbType = Boolean), "@p16=?' (DbType = DateTime), "@p17=?' (DbType = Int32), "@p18=?' (DbType = Boolean), "@p19=?' (DbType = Boolean), "@p20=?' (DbType = Boolean), "@p21=?' (DbType = Boolean), "@p22=?' (DbType = Int32), "@p23=?' (DbType = Int32), "@p24=?' (DbType = DateTime), "@p25=?' (DbType = Int32), "@p26=?' (DbType = Int32), "@p27=?' (DbType = Int32), "@p28=?' (DbType = Boolean), "@p29=?' (DbType = Boolean), "@p30=?' (DbType = Boolean), "@p31=?' (DbType = Boolean), "@p32=?' (DbType = DateTime), "@p33=?' (DbType = Int32), "@p34=?' (DbType = Int32), "@p35=?' (DbType = Boolean), "@p36=?' (DbType = Boolean), "@p37=?' (DbType = Boolean), "@p38=?' (DbType = Boolean), "@p39=?' (DbType = Boolean), "@p40=?' (DbType = Boolean), "@p41=?' (DbType = Int32), "@p42=?' (DbType = DateTime), "@p43=?' (DbType = Int32), "@p44=?' (DbType = Int32), "@p45=?' (DbType = Boolean), "@p46=?' (DbType = Boolean), "@p47=?' (DbType = Boolean), "@p48=?' (DbType = Boolean), "@p49=?' (DbType = Boolean), "@p50=?' (DbType = Boolean), "@p51=?' (DbType = DateTime), "@p52=?' (DbType = Int32)", CommandType='Text', CommandTimeout='30']\r\n""UPDATE \"BankingSchema\".\"Accounts\" SET \"Balance\" = @p1 AND xmin = @p2\r\nWHERE \"Id\" = @p4 AND xmin = @p3\r\nRETURNING xmin;\r\nUPDATE \"BankingSchema\".\"Accounts\" SET \"Balance\" = @p6\r\nWHERE \"Id\" = @p5 AND xmin = @p7 AND xmin = @p8\r\nRETURNING xmin;\r\nUPDATE \"BankingSchema\".\"Bills\" SET \"LastPayTime\" = @p9\r\nWHERE \"Id\" = @p10 AND xmin = @p11\r\nRETURNING xmin;\r\nUPDATE \"BankingSchema\".\"Bills\" SET \"LastPayTime\" = @p12\r\nWHERE \"Id\" = @p13\r\nINSERT INTO \"BankingSchema\".\"Records\" ((\"AccountId\", \"Amount\", \"ErrorMessage\", \"IsPending\", \"IsSuccessful\", \"OperationType\", \"ReceiverAccountId\", \"TimeStamp\", \"UserId\")\r\nVALUES (@p17, @p18, @p19, @p20, @p21, @p22, @p23, @p24, @p25)\r\nRETURNING \"Id\";\r\nINSERT INTO \"BankingSchema\".\"Records\" ((\"AccountId\", \"Amount\", \"ErrorMessage\", \"IsPending\", \"IsSuccessful\", \"OperationType\", \"ReceiverAccountId\", \"TimeStamp\", \"UserId\")\r\nVALUES (@p26, @p27, @p28, @p29, @p30, @p31, @p32, @p33, @p34)\r\nRETURNING \"Id\";\r\nINSERT INTO \"BankingSchema\".\"Records\" ((\"AccountId\", \"Amount\", \"ErrorMessage\", \"IsPending\", \"IsSuccessful\", \"OperationType\", \"ReceiverAccountId\", \"TimeStamp\", \"UserId\")\r\nVALUES (@p35, @p36, @p37, @p38, @p39, @p40, @p41, @p42, @p43)\r\nRETURNING \"Id\";\r\nINSERT INTO \"BankingSchema\".\"Records\" ((\"AccountId\", \"Amount\", \"ErrorMessage\", \"IsPending\", \"IsSuccessful\", \"OperationType\", \"ReceiverAccountId\", \"TimeStamp\", \"UserId\")\r\nVALUES (@p44, @p45, @p46, @p47, @p48, @p49, @p50, @p51, @p52)\r\nRETURNING \"Id\"";
2024-01-07 20:27:00 [DBG] Trigger instruction : NoInstruction
2024-01-07 20:27:00 [DBG] Batch acquisition of 0 triggers

```

The screenshot shows the DBVisualizer 23.3.1 interface. The main window displays a database grid titled 'Bills' with columns: Id, Amount, LastPayTime, IsActive, and AccountId. The data shows five rows of bills with various amounts and payment times. A filter bar at the top right allows for filtering results using a SQL expression. The bottom right corner of the screen shows a Windows taskbar with various icons and system status.

	Id	Amount	LastPayTime	IsActive	AccountId
1	2	45	2024-01-08 19:37:56.559 +0300	[v]	2 [v]
2	1	35	2024-01-07 19:39:55.559 +0300	[v]	3 [v]
3	3	105	2024-02-07 19:57:56.559 +0300	[v]	2 [v]
4	4	60	2024-02-08 00:37:56.559 +0300	[v]	3 [v]
5	5	25	2024-02-07 01:31:18.000 +0300	[v]	10 [v]

Now LastPayTime has been added one month. Only dates with less than today's has been paid.