1.How To Test:

1. You can check the up ports with command “***netstat -an | find “LISTEN”***”

A screenshot of a computer screen

Description automatically generated

1. Start Eureka Server under (on port 8761)  
   “**D:\IdeaProjects\EurekaServerApp**”

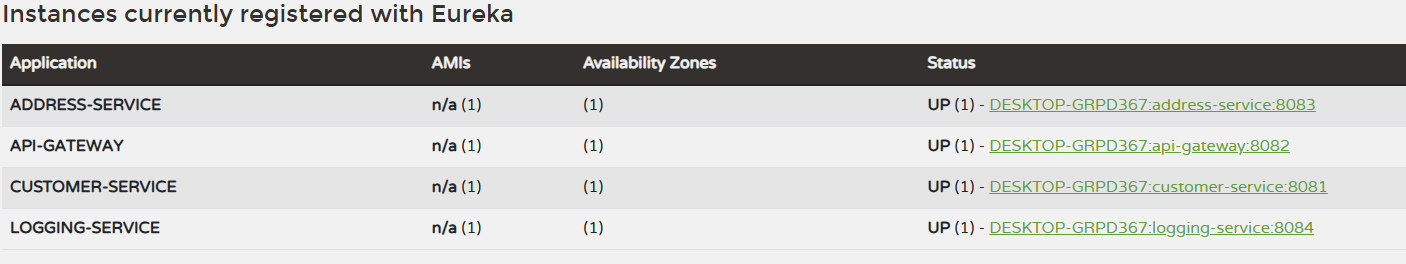
**java -jar target/eureka-server-0.0.1-SNAPSHOT.jar**

NOTE: if there is no related jar file in related path (/target/) create it first by “mvn clean install)

1. Start Keycloak (on port 8080)

**“D:\Keycloak\keycloak-26.0.7\keycloak-26.0.7\bin”  
kc.bat start-dev**

1. Start api-gateway under (on port 8082) “**D:\IdeaProjects\CWSMicroservicesUpd\api-gateway**”
2. Start address-service under (on port 8083)   
   “**D:\IdeaProjects\ CWSMicroservicesUpd \address-service**”
3. Start customer-service under (on port 8081)   
   “**D:\IdeaProjects\ CWSMicroservicesUpd \customer-service**”
4. Start logging-service under (on port 8084) (if working)  
   “**D:\IdeaProjects\ CWSMicroservicesUpd \logging-service**”
5. Check Eureka Server if you have related microservices. Open a browser, go to <http://localhost:8761/>. See related instances on the page:



1. Open Postman. Now you are able to send below requests:

|  |
| --- |
| IMPORTANT NOTE: For all of client requests, whether to address-service or customer-service, from now on, the customer web service requests only be sent over api-gateway, which has URL: **http://localhost:8082** |

## 1.1.Controls in Eureka and Keycloak:

If there is any problem occurs due to Eureka or Keycloak, to test both servers:

1. **Eureka:**

Check Eureka Server if you have related microservices. Open a browser, go to http://localhost:8761/. See related instances on the page:

A screenshot of a computer

Description automatically generated

If the instances are in DOWN status, run the services on idea or with **mvn spring boot:run:**

A screenshot of a computer

Description automatically generated

1. **Keycloak:**

On any browser try to go [**http://localhost:8080/realms/customer-app-realm/.well-known/openid-configuration**](http://localhost:8080/realms/customer-app-realm/.well-known/openid-configuration)

If the page loads successfully and return JSON, then Keycloak is running properly:

A screenshot of a computer screen

Description automatically generated

If the page does not load, then Keycloak may not be running correctly. Restart Keycloak and try again.

1. **Run the service in this order:** api-gateway, address-service, customer-service, logging-service

# 2.Updates in Keycloak:

## 2.1.Creating Clients:

There shoult be two different clients for this method we are using in the project. api-gateway-client and customer-service-client:

* api-gateway-client: This client is configured in the api-gateway, because it acts as an entry point for authentication and routing.
* Customer-service-client: customer-service needs a token to communicate with address-service. Ideally, it should use its own client (customer-service-client) to get an access token via getServiceToken().

## 2.1.Creating a Client for API-GW:

1. Go to **"Clients"** → Click **"Create Client"**
2. **Client ID**: api-gateway-client
3. **Client Type**: OpenID Connect
4. Click **Next**
5. **Client Authentication**: ON
6. **Authorization**: ON
7. **Redirect URIs**: http://localhost:8082/\*
8. Click **Save**

## 2.2.Configuring AP-GW Cleint:

1. Inside api-gateway-client, go to **"Credentials"**
2. Copy the **Client Secret** (you’ll need this later)
3. Go to **"Settings"**:
   * **Access Type**: confidential
   * **Service Accounts Enabled**: ON
   * **Authorization Enabled**: ON
   * **Standard Flow Enabled**: ON
   * **Direct Access Grants Enabled**: ON
   * **Valid Redirect URIs**: http://localhost:8082/\*
4. Click **Save**

## 2.3.Create a Role:

1. Go to **"Roles"** → Click **"Create Role"**
2. **Role Name**: USER
3. Click **Save**

## 2.4.Creating a Test User:

1. Go to **"Users"** → Click **"Add User"**
2. **Username**: testuser
3. **Email**: testuser@example.com
4. **First Name**: Test
5. **Last Name**: User
6. **Enabled**: ON
7. Click **Save**

## 2.5.Set Password for Test User:

1. Click on **"Credentials"** tab
2. Set **New Password**: testpassword
3. Set **Confirm Password**: testpassword
4. Set **Temporary?**: OFF
5. Click **Set Password** → Confirm

## 2.6.Assign Role to User:

1. Go to **Users** → Select testuser
2. Click **"Role Mappings"**
3. Under **Available Roles**, find USER → Click **Add Selected**

## 2.7.Create Client for Customer Service:

1. Go to **"Clients"** → Click **"Create Client"**
2. **Client ID**: customer-service-client
3. **Client Type**: OpenID Connect
4. Click **Next**
5. **Client Authentication**: ON
6. **Authorization**: ON
7. **Redirect URIs**: http://localhost:8081/\*
8. Click **Save**
9. Copy the **Client Secret** from the "Credentials" tab

# 3.Get JWT Token from Keycloak:

To access API Gateway, the client need to get a JWT token.

## 3.1.JWT Token(JSON Web Token):

### 3.1.1.General Information:

JWT is an open standard (RFC 7519) that defines a compact and self-contained way to represent information between two parties. It’s primarily used for securely transmitting information as a JSON object. This information can be verified and trusted because it is digitially signed (using a secret or public/private key pair).

JWT tokens are typically used in the context of **authentication and authorization** systems. They allow servers to verify the identity of users and provide them with access to certain resources.

### 3.1.2.Structure of a JWT:

A JWT consists of 3 parts seperated by periods (“.”):

1. **Header:** Contains metadata about the token, such as the signing algorithm used.
2. **Payload:** Contains the claims. Claims are statements about an entity (typically, the user) and additional metadata. It’s important to note that this section is not encrypted, so it should not contain sensitive information. There are 3 type of claims:
   1. **Registered Claims:** These are predefined claims that are not mandatorybut recommended.(iat (issued at), exp (expiration), sub (subject), etc)
   2. **Public Claims:** These are claims that can be used by anyone but should be defined to avoid collision( çakışma). (email)
   3. **Private Claims:** Customer claims created to share information between parties that agree on them (role, user\_id, etc.)
3. **Signature:** Used to verify that the sender of the JWT is who it says it is and to ensure that the message wasn’t tampered with. It’s created by signing the header and payload with a secret key or private key.

### 3.1.3.Explanation of a JWT Token:

|  |
| --- |
| {      "access\_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICI5NFRWWjlyeFdFWi14dEtwemVtaktUWFd4TERjLTNfOE9VQ2lpd051YWhBIn0..Co6vFbomMtmdneBQDlCVUpfl1Ra8YS02gNlD9R38R3XQkhZ4OKsbo9I3Z-XC5IOxIw2JbTKEgg8xjISYVQucQmDAxSzVTmY58fs1CRzg7DsD6CAfvPcZB5U9bODPZYeh0sjAdAGPycm4NhcavfXeDcJcRIU7g2B7Gi-xOnIbPKuEtw-uaMRnMVPQTJEaJdTRK6FEKqgUwcZf5-AGSNvzT-ZM4eP1T6a2w8wOTKmMStsdx2PfgtZioy-6G8s5oEgRiM6TKmH332EvUdkghC-d-0c3-SRFNjwU55wboD3x3\_2bkHshtsykQFnoRA4PXuNgWy58XNqpwN9\_aS9hz87V7A",      "expires\_in": 180,      "refresh\_expires\_in": 420,      "refresh\_token": "eyJhbGciOiJIUzUxMiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJjNTVlYjA4OS1lMGMwLTRmZTEtYTkzMy0xYjliM2U1MWM5MDQifQ..vIiZg1XlYEH-ZFKZqL2dLXNz9gl4J49j10wnLifyk57k8qNh7xth1QMF287ngsZGVPIF2xIUQdC8ylV9nrDoHw",      "token\_type": "Bearer",      "not-before-policy": 0,      "session\_state": "76279a3e-7a96-4ca3-95c1-b2bddfd0be84",      "scope": "profile email"  } |

The details of the JWT:

1. **access\_token:** The JWT that the client uses to authenticate API requests. It contains information about the user, their roles, and other claims.
2. **expires\_in:** The time (in seconds) until the access token expires. (180 secs = 3 mins)
3. **refresh\_expires\_in:** The time until the refresh token expires. (420 secs = 7 mins)
4. **refresh\_token:** A long-lived token used to obtain a new access token when the current one expires. It allows the user to stay logged in without requiring them to authenticate again.
5. **token\_type:** The type of token. Typically, this is Bearer, which indicates that the token is used for Bearer Authentication.
6. **not-before-policy:** This parameter indicates the time before which the token cannot be used, expressed as a UNIX timestamp. A value of 0 means that it can be used immediately.
7. **session\_state:** An identifier for the session, useful for tracking the user's authentication session across systems.
8. **scope:** Defines the level of access granted by the token. In this case, the user is allowed to access profile and email.

### 3.1.4.How to Decode an Access\_Token:

Open in a browser and go to “**https://jwt.io/”** page. Copy and paste the access\_token string to **Encoded** table of the page. Check the content from the **Decoded** table:

|  |
| --- |
| **HEADER:ALGORITHM & TOKEN TYPE** |
| {  "alg": "RS256", 🡺 *The signing algorithm used. RS256 stands for RSA with SHA-256*  "typ": "JWT",🡺 *The type of the token, generally JWT*  "kid": "94TVZ9rxWEZ-xtKpzemjKTXWxLDc-3\_8OUCiiwNuahA"🡺 *The key id*  } |
| **PAYLOAD:DATA 🡺** *(contains the claims-actual data)* |
| {  "exp": 1738739087, 🡺 *expiration time of the token in Unix epoch time (seconds since 1970-01-01)*  "iat": 1738738907, 🡺 *issued at time in Unix epoch format, indicating when the token was created.*  "jti": "56b8ea0e-fd91-46ac-a049-7e0db8914fe9",🡺*JWT ID. It’s a unique identifier for the JWT, used to prevent replay attacks*  "iss": "http://localhost:8080/realms/customer-app-realm", 🡺*The issuer of the token, i.e., the entity that issued the JWT (in this case, Keycloak server).*  "aud": "account", 🡺*The audience of the token, i.e., the intended recipient. This typically corresponds to the service or resource that will accept the token (again, here it refers to the same Keycloak realm).*  "sub": "9d96f80a-874c-4fec-bcdb-7a3b2b305493", 🡺*he subject claim, which typically represents the user or entity the token is issued for (here, it’s a unique identifier for the user).*  "typ": "Bearer", 🡺*Type of the token, Bearer Token*  "azp": "customer-service-client", 🡺 *The authorized party claim indicates the client application that is authorized to use the token (the customer-service-client in this case).*  "sid": "76279a3e-7a96-4ca3-95c1-b2bddfd0be84", 🡺 *The session ID, which ties the token to a specific user session.*  "acr": "1",  "allowed-origins": [  "http://localhost:8082"  ],  "realm\_access": {  "roles": [  "default-roles-customer-app-realm",  "offline\_access",  "uma\_authorization"  ]  },  "resource\_access": {  "account": {  "roles": [  "manage-account",  "manage-account-links",  "view-profile"  ]  }  },  "scope": "profile email",  "email\_verified": true,  "name": "Test User",  "preferred\_username": "testuser",  "given\_name": "Test",  "family\_name": "User",  "email": "testuser@example.com"  } |
| **VERIFY SIGNATURE** |
| RSASHA256(  base64UrlEncode(header) + "." +  base64UrlEncode(payload),  ,  {  "e": "AQAB",  "kty": "RSA",  "n": "wS1pzojDtNENG\_5oMyxjcVxaTmN8TchCJXYze-0eI2xMBfKyesREQjzSAE\_klWmzfv4CKhdHB5acmYZeDyZyRY-oGrVXONBpEUuAPwbB6sv1cIaus3C0HzCV4aSQ-eQ6kcEJfB\_FLjh5VAutvzOaN7RjqoJS408B6ke6VGXYXVsm\_9ddQrgcOKkMSEVHla-LF6esR3ksg3EXzeniWFRBgrgvf7p3kBWC5huYd6CAIByBWuHye4fc3mon\_cvVvQdJpSfDp6a7tZu0Fxc1xEoQcveP9TWa23O2ANDayRxDcOp639IOMQ4Lz39\_2NC4tG7v5CizeB7NhsOdzbRluhi2fw"  }  ) |

### 3.1.5.How to Decode a Refresh\_Token:

Open in a browser and go to “**https://jwt.io/”** page. Copy and paste the refresh\_token string to **Encoded** table of the page. Check the content from the **Decoded** table:

|  |
| --- |
| **HEADER:ALGORITHM & TOKEN TYPE** |
| {  "alg": "HS512",  "typ": "JWT",  "kid": "c55eb089-e0c0-4fe1-a933-1b9b3e51c904"  } |
| **PAYLOAD:DATA** |
| {  "exp": 1738739327,  "iat": 1738738907,  "jti": "60c9564a-94a3-4dd6-a541-63dd4bd272bc",  "iss": "http://localhost:8080/realms/customer-app-realm",  "aud": "http://localhost:8080/realms/customer-app-realm",  "sub": "9d96f80a-874c-4fec-bcdb-7a3b2b305493",  "typ": "Refresh",  "azp": "customer-service-client",  "sid": "76279a3e-7a96-4ca3-95c1-b2bddfd0be84",  "scope": "roles acr profile basic email web-origins",  "reuse\_id": "ed9bc479-4a67-4254-8aaa-f293757beb97"  } |
| **VERIFY SIGNATURE** |
| HMACSHA512(  base64UrlEncode(header) + "." +  base64UrlEncode(payload),    ) |

## 3.2.Token Mechanism in SecuredRBAC API-GW:

Now, we have two-level-access token mechanisms in the updated customer web service project.

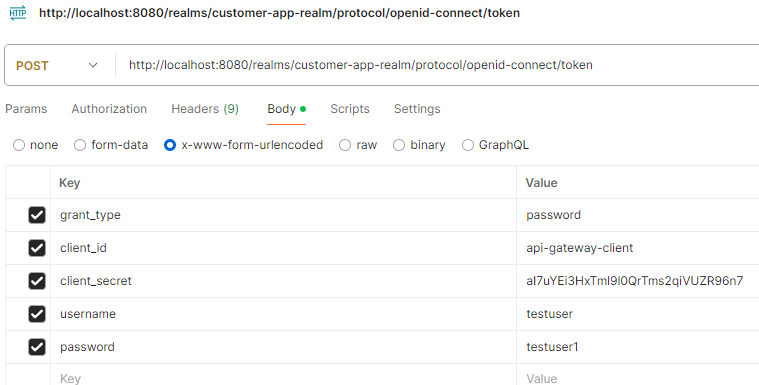
### 3.2.1.User Access Token without Client Credentials:

This is the access token that is obtained from POSTMAN when logging in with a username and password (OAuth2 Password Grant):

1. Open **Postman**.
2. Create a **POST request** to:

|  |
| --- |
| http://localhost:8080/realms/customer-app-realm/protocol/openid-connect/token |

1. **Go to "Body" → "x-www-form-urlencoded"** and add:
   * grant\_type: password
   * client\_id: api-gateway-client
   * client\_secret: aI7uYEi3HxTml9l0QrTms2qiVUZR96n7
   * username: testuser
   * password: testuser1



1. Click **Send**.
   * If successful, you get a JWT token (access\_token).

|  |
| --- |
| {      "access\_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICI5NFRWWjlyeFdFWi14dEtwemVtaktUWFd4TERjLTNfOE9VQ2lpd051YWhBIn0..IK-NiJ0YpOtUFPi1MnCHp83OBTYiqtAiMS0DOoGMxMPe3rD4gvcn9l9hnXJ5621ykypCAa1yCMtgQKJYRosIP2Lkal7RHXU0Nt3gZ6t8pKOvQgwa-Ho-pQ3sTkcfUuvvs2qOCFFMXUsjqWENyUgwfJHsRTpCPZi3KXULZ42bXY5OZKXQWJ5QyjoX6a4hdTIaA9NRAg21JotTotpjkyqr8KrOmtzDhGwniTU83dwG9WnjgoqeHOgkJGmP7W-0u8pZUIwPonvJF\_yAGstXNWxAPIRrO7xC6y8czkaGmfnV0ASeZxlmfx0NgsDGUKYuJ\_gDe9vXcDSKBsltUgYxRvVFLw",      "expires\_in": 180,      "refresh\_expires\_in": 420,      "refresh\_token": "eyJhbGciOiJIUzUxMiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJjNTVlYjA4OS1lMGMwLTRmZTEtYTkzMy0xYjliM2U1MWM5MDQifQ..lDeu0a8Lb0eo2MByXgaZYAZbXRJ7-jn3llhd0bhblFG5UcWE4pwqdQoPnvsXiQPB5z7cAtk3BLR\_NV49wNKwBg",      "token\_type": "Bearer",      "not-before-policy": 0,      "session\_state": "169ba22f-8192-453a-a12f-cefe9034cab2",      "scope": "profile email"  } |

* + **Copy the token** for the next step.

We will use this token when making direct calls as a user (e.g., calling GET <http://localhost:8082/customers> from Postman)

This token contains the user’s **roles** and **permissions**. Check decoded jwt token:

|  |
| --- |
| HEADER: ....  PAYLOAD:  "iat": 1738922475,  "jti": "5b48d29c-e482-45ef-b213-9d0d0f5df2ec",  "iss": "http://localhost:8080/realms/customer-app-realm",  "aud": "account",  "sub": "9d96f80a-874c-4fec-bcdb-7a3b2b305493",  "typ": "Bearer",  "azp": "api-gateway-client",  "sid": "169ba22f-8192-453a-a12f-cefe9034cab2",  "acr": "1",  "allowed-origins": [  "http://localhost:8082"  ],  "realm\_access": {  "roles": [  "default-roles-customer-app-realm",  "offline\_access",  "uma\_authorization"  ]  },  "resource\_access": {  "api-gateway-client": {  "roles": [  "USER"  ]  },  "account": {  "roles": [  "manage-account",  "manage-account-links",  "view-profile"  ]  }  },  "scope": "profile email",  "email\_verified": true,  "name": "Test User",  "preferred\_username": "testuser",  "given\_name": "Test",  "family\_name": "User",  "email": "testuser@example.com"  }  VERIFY SIGNATURE:... |

### 3.2.2.Service-to-Service Token with Client Credentials Grandted:

When customer-service calls address-service (by using previous access\_token in a e.g., GET <http://localhost:8082/customers>), this time, it’s not a user (external client) making the request – it is another service(customer-service) that is calling an internal different microservice (address-service). In such cases, we use Client Credentials Flow, which does not involve a user but uses a **client ID and secret** to get a token. And since the getServiceToken() is implemented in related methods in CustomerController of customer-service, customer-service must authenticate itself before calling any of address-service. So that means, for every service-to-service token customer-service should request a new access\_token.

|  |
| --- |
| **private** String getServiceToken() {  *//RestTemplate restTemplate = new RestTemplate();* String url = **tokenEndpoint**; 🡺 ${KEYCLOAK\_SERVER\_URL}/realms/${KEYCLOAK\_SERVER\_URL}/protocol/openid-connect/token   HttpHeaders headers = **new** HttpHeaders();  headers.setContentType(MediaType.***APPLICATION\_FORM\_URLENCODED***);   MultiValueMap<String, String> body = **new** LinkedMultiValueMap<>();  body.add(**"client\_id"**, **clientId**); 🡺 client\_id as “customer-service-client”  body.add(**"client\_secret"**, **clientSecret**); 🡺 client\_secret of “customer-service-client”  body.add(**"grant\_type"**, **grantType**); 🡺client\_credentials   HttpEntity<MultiValueMap<String, String>> entity = **new** HttpEntity<>(body, headers);   ResponseEntity<Map<String, Object>> response = **restTemplate**.exchange(  url, HttpMethod.***POST***, entity, **new** ParameterizedTypeReference<>() {}  );   **return** (String) response.getBody().get(**"access\_token"**); } |

Check decoded new access\_token:

|  |
| --- |
| HEADER: ....  PAYLOAD: {  "exp": 1738924159,  "iat": 1738923979,  "jti": "e7034a46-57c4-40f8-affa-24bb83eb4897",  "iss": "http://localhost:8080/realms/customer-app-realm",  "aud": "account",  "sub": "4a6229f5-6d5e-47d8-8087-a9cd51ac3098",  "typ": "Bearer",  "azp": "customer-service-client",  "acr": "1",  "allowed-origins": [  "http://localhost:8082"  ],  "realm\_access": {  "roles": [  "default-roles-customer-app-realm",  "offline\_access",  "uma\_authorization"  ]  },  "resource\_access": {  "customer-service-client": {  "roles": [  "uma\_protection"  ]  },  "account": {  "roles": [  "manage-account",  "manage-account-links",  "view-profile"  ]  }  },  "scope": "profile email",  "email\_verified": false,  "clientHost": "127.0.0.1",  "preferred\_username": "service-account-customer-service-client",  "clientAddress": "127.0.0.1",  "client\_id": "customer-service-client"  }  VERIFY SIGNATURE:.... |

# 43.Test Without a Token Case:

These tests without token should fail:

|  |  |
| --- | --- |
| GET | http://localhost:8082/customers |
| Response | 401 Unauthorized |

# 5.Tests With JWT Token:

## 5.1.Get All Customers (GET):

|  |  |
| --- | --- |
| **GET** | http://localhost:8082/customers |
| **Header** | Authorization: Bearer your-access-token |
| **Response** | [      {          "id": 1,          "name": "Kaptan Mağara Adamı ",          "email": "kaptan.magara.adami@deneme.com",          "phoneNumber": "1234567890",          "addresses": [              {                  "id": 3,                  "addressName": "Work Foolish",                  "street": "Handan Sokak",                  "city": "İstanbul",                  "state": "Türkiye",                  "zipCode": "10001",                  "customerId": 1              }          ]      },      {          "id": 2,          "name": "İdil Biret",          "email": "idilbiret@gmail.com",          "phoneNumber": "05545909896",          "addresses": []      },      {          "id": 3,          "name": "Seda Çelebi",          "email": "sedaclbi@gmail.com",          "phoneNumber": "05545909896",          "addresses": [              {                  "id": 4,                  "addressName": "Home",                  "street": "Zuhal Sokak",                  "city": "İstanbul",                  "state": "Türkiye",                  "zipCode": "356660",                  "customerId": 3              },              {                  "id": 5,                  "addressName": "İş",                  "street": "Zuhaliş Sokak",                  "city": "İstanbul",                  "state": "Türkiye",                  "zipCode": "356660",                  "customerId": 3              },              {                  "id": 6,                  "addressName": "Home Music",                  "street": "Muzisyes Sokak",                  "city": "İstanbul",                  "state": "Türkiye",                  "zipCode": "10001",                  "customerId": 3              }          ]      },      {          "id": 11,          "name": "Roei Morgen ghgdf1",          "email": "roei.morgen.ghdfg1@deneme.com",          "phoneNumber": "1234567890",          "addresses": [              {                  "id": 7,                  "addressName": "Work",                  "street": "Handan Sokak",                  "city": "İstanbul",                  "state": "Türkiye",                  "zipCode": "10001",                  "customerId": 11              }          ]      }  ] |

## 5.2.Get Customer by ID (GET):

|  |  |
| --- | --- |
| **GET** | [http://localhost:8082/customers/{customerId}](http://localhost:8082/customers/%7bcustomerId%7d)  http://localhost:8082/customers/3 |
| **Header** | Authorization: Bearer your-access-token |
| **Response** | {      "id": 3,      "name": "Seda Çelebi",      "email": "sedaclbi@gmail.com",      "phoneNumber": "05545909896",      "addresses": [          {              "id": 4,              "addressName": "Home",              "street": "Zuhal Sokak",              "city": "İstanbul",              "state": "Türkiye",              "zipCode": "356660",              "customerId": 3          },          {              "id": 5,              "addressName": "İş",              "street": "Zuhaliş Sokak",              "city": "İstanbul",              "state": "Türkiye",              "zipCode": "356660",              "customerId": 3          },          {              "id": 6,              "addressName": "Home Music",              "street": "Muzisyes Sokak",              "city": "İstanbul",              "state": "Türkiye",              "zipCode": "10001",              "customerId": 3          }      ]  } |

If the customerId is not valid in DB, it returns “**404 – Not Found:Customer not found!**”

## 5.3.Create a Customer (POST):

Without address

|  |  |
| --- | --- |
| **POST** | http://localhost:8082/customers |
| **Headers** | Content-Type: application/json |
| **Header** | Authorization: Bearer your-access-token |
| **Body** | {      "name": "Freddy Krugger",      "email": "freddy.krugger@deneme.com",      "phoneNumber": "1234567890",      "addresses": [          {              "addressName": "Work",              "street": "Elm Street",              "city": "New York",              "state": "USD",              "zipCode": "10001"          }      ]  } |
| **Response** | **201 - Created** {      "id": 12,      "name": "Freddy Krugger",      "email": "freddy.krugger@deneme.com",      "phoneNumber": "1234567890",      "addresses": [          {              "id": **null**,              "addressName": "Work",              "street": "Elm Street",              "city": "New York",              "state": "USD",              "zipCode": "10001",              "customerId": 12          }      ]  } |
| **POST** | http://localhost:8082/customers |
| **Headers** | Content-Type: application/json |
| **Header** | Authorization: Bearer your-access-token |
| **Body** | {      "name": "Burhan Kuzu",      "email": "burhan.kuzu@deneme.com",      "phoneNumber": "1234567890"  } |
| **Response** | **201 - Created** {      "id": 13,      "name": "Burhan Kuzu",      "email": "burhan.kuzu@deneme.com",      "phoneNumber": "1234567890",      "addresses": **null**  } |

## 5.4.Delete Customer (DELETE):

|  |  |
| --- | --- |
| **DELETE** | http://localhost:8082/customers/{customerId} |
| **Header** | Authorization: Bearer your-access-token |
| **Response** | {  "message": "Customer deleted successfully"  } |
| **DELETE** | http://localhost:8082/customers/58 |
| **Header** | Authorization: Bearer your-access-token |
| **Response** | 200 – OK: Customer and related addresses deleted successfully. |
| **DELETE** | http://localhost:8082/customers/61 |
| **Header** | Authorization: Bearer your-access-token |
| **Response** | 200 – OK: Customer and related addresses deleted successfully. |

If the URL contains a customerId that is not valid in customer table in DB, it will return “**404 – Not Found:Customer not found!**”

## 5.5.Get All Addresses (GET):

|  |  |
| --- | --- |
| **GET** | http://localhost:8082/addresses |
| **Header** | Authorization: Bearer your-access-token |
| **Response** | [      {          "id": 3,          "addressName": "Work Foolish",          "street": "Handan Sokak",          "city": "İstanbul",          "state": "Türkiye",          "zipCode": "10001",          "customerId": 1      },      {          "id": 4,          "addressName": "Home",          "street": "Zuhal Sokak",          "city": "İstanbul",          "state": "Türkiye",          "zipCode": "356660",          "customerId": 3      },      {          "id": 5,          "addressName": "İş",          "street": "Zuhaliş Sokak",          "city": "İstanbul",          "state": "Türkiye",          "zipCode": "356660",          "customerId": 3      },      {          "id": 6,          "addressName": "Home Music",          "street": "Muzisyes Sokak",          "city": "İstanbul",          "state": "Türkiye",          "zipCode": "10001",          "customerId": 3      },      {          "id": 7,          "addressName": "Work",          "street": "Handan Sokak",          "city": "İstanbul",          "state": "Türkiye",          "zipCode": "10001",          "customerId": 11      },      {          "id": 8,          "addressName": "Work",          "street": "Elm Street",          "city": "New York",          "state": "USD",          "zipCode": "10001",          "customerId": 12      }  ] |

## 5.6.Get All Addresses of a Customer (GET):

|  |  |
| --- | --- |
| **GET** | [http://localhost:8082/addresses/customers/{customerId}](http://localhost:8082/addresses/customers/%7bcustomerId%7d)  http://localhost:8082/addresses/customers/3 |
| **Header** | Authorization: Bearer your-access-token |
| **Response** | [      {          "id": 4,          "addressName": "Home",          "street": "Zuhal Sokak",          "city": "İstanbul",          "state": "Türkiye",          "zipCode": "356660",          "customerId": 3      },      {          "id": 5,          "addressName": "İş",          "street": "Zuhaliş Sokak",          "city": "İstanbul",          "state": "Türkiye",          "zipCode": "356660",          "customerId": 3      },      {          "id": 6,          "addressName": "Home Music",          "street": "Muzisyes Sokak",          "city": "İstanbul",          "state": "Türkiye",          "zipCode": "10001",          "customerId": 3      }  ] |

If the customerId is not valid in DB, it returns “**404 – Not Found:Customer not found!**”

## 5.7.Delete an Address of a Customer (DELETE):

|  |  |
| --- | --- |
| **DELETE** | http://localhost:8082/customers/{customerId} |
| **Header** | Authorization: Bearer your-access-token |
| **Response** | {  "message": "Customer deleted successfully"  } |
| **DELETE** | http://localhost:8082/customers/58 |
| **Header** | Authorization: Bearer your-access-token |
| **Response** | 200 – OK: Customer and related addresses deleted successfully. |
| **DELETE** | http://localhost:8082/customers/61 |
| **Header** | Authorization: Bearer your-access-token |
| **Response** | 200 – OK: Customer and related addresses deleted successfully. |

If the URL contains a customerId that is not valid in customer table in DB, it will return “**404 – Not Found:Customer not found!**”

# 6. Refresh Token Mechanism:

Asd will check later