

DefineX Bootcamp Homework Week 2 - Hakan AFAT

1- Senkron iletişimde veri alışverişi sıralı olarak çalışır. Yazdığımız kod bloğu sırası ile işleme alınır ve komut tamamlandıktan sonra diğer satıra geçer. Ancak UI, data transferi vb. gibi birçok alanda işlemi tamamlanması uzun süreler almaktadır. Bunun önüne geçmek için async/await keywordler, future kavramlarının yardımıyla Asenkron programlama yapmak gerekir.

Asenkron programlamada başlatılan komutlar runtime ve kullandığımız sistemin izin verdiği ölçülerde (maksimum açılacak soket sayısı, maksimum thread sayısı, memory vb.) aynı anda çalıştırabiliriz. Bu sayede örneğin bir frontend uygulamasında hem backend gerekli API istekleri atılırken UI elementlerini çizdirebiliriz veya birden fazla grafiğin verisini aynı anda güncelleyebiliriz. Günümüzün çok çekirdekli sistemlerini daha verimli olarak kullanmış oluruz.

Kod örnekleri repoda verilecektir.

2-RabbitMQ çalışma sistemi olarak message queue (AMQP) sistemini kullanır. Mesajların alındı bilgisinin önemli olduğu (ACK) durumlarda kullanılır. Mesaj alındığında queue dan silinir. Bu mesajların tesliminin önceliklendirilmesi mümkündür. Çeşitli routing mekanizmaları yapılabilir. Birçok yazılım dilini ve protokolü destekler. Örnek vermek gerekirse;

- Bir e-ticaret platformu üzerinden, müşteri bir sipariş oluşturabilmesi sırasında çeşitli mekanizmalar gerekmektedir. Siparişin ödemesinin kontrol edilmesi için gerekli bilgiler message queue a yazılır. Queue dan siparişler çekilir ve onaylanmaması durumunda mail servisi oluşan hatayı gönderebilir veya olumlu olması durumunda siparişin hazırlanması için sipariş hazırlama depo servisine gerekli mesajı gönderebilir.
- Sağlık alanında kullanılacak sensörlerin olası hata durumlarında raporları message queue a yazılır. Görevli personelin kullandığı monitörlere veya mobil cihazlarında bu queue a subscribe olması sağlanır.
- Pos(Point of Sale) cihazlarının iletişimde kullanılır. Pos cihazı ile backend servisi arasında transactionların ve stok durumunun takibi açısından tercih edilebilir. Birçok mağazası olan bir firmanın stok durumları bu sayede senkronize olur. Özel müşterilerin işlemleri önceliklendirilebilir.

Kafka ise farklı olarak pub-sub sistemini kullanır. Yüksek veri ve düşür gecikmenin gerekli olduğu durumlarda tercih edilir. Çok daha fazla veri yükünü kaldırabilir. Mesaj bir subscriber tarafından tüketilse dahi bir süre boyunca (retention period) saklanmaya devam eder. Bu sayede servisleri reprocess ihtiyacı varsa burdan tekrar erişebilir. Temel olarak real-time big data stream lerini handle etmek için kullanılır. Buradaki stream, client eventları veya microservice lerin loglarının toplanması olarak kullanılabilir. Sadece kendi binary protokolünü destekler. Örnek vermek gerekirse;

- Bir e-ticaret sitesinde müşterilerin yaptığı tıklamalar, incelenen ürünler, aranan kategoriler gibi detaylar Kafka sayesinde bir event olarak toplanabilir. Backend servisi real-time olarak streamı alır ve bu datalar işlenerek kişiselleştirilmiş reklamlar, kampanyalar oluşturulabilir.
- Bir otomobil üzerindeki sensörlerinin verileri IoT (Internet of Things) cihazlar ile toplanarak Kafka streamleri üzerinden yapılacak analizlerle aracın ilerideki

verebileceği sorunlar ve sürüş karakteri hakkında tahmin yapılabilir. Bu sayede bakımlarda araç arıza yapmadan parçalara gerekli müdahaleler yapılır.

3-Docker bir container bazlı sanallaştırma teknolojisidir. Linux çekirdeğinden gelen özellik sayesinde çalışmaktadır. Windows da HyperV teknolojisi ve MacOS üzerinde xhyve kullanılarak Docker containerları çalıştırılabilir. Bir host işletim sistemine ihtiyaç duyar ve işletim sistemi kaynaklarını paylaşarak üzerinde containeri çalıştırır. Daha az kaynak kullanarak hazırlanan imajları; örneğin bir database, bir backend ve frontend servisini; çağırıp sistem üzerinde bir container üzerinde ayağa kaldırarak kolayca kullanabiliriz.

Virtual Machine ise çeşitli donanımsal sanallaştırma (VT-d, VT-x veya AMD-V) özellikleri kullanılarak bir sistem üzerinde birçok işletim sistemini aynı anda çalıştırmaya yarar. Bu sistemler arasında tam bir izolasyon da sağlamış olur. Ancak dedicated bir kaynak kullanımı (core, thread, memory vb.) vardır. İnternet üzerinde kiraladığımız VPS ler de aslında serverlar üzerinde çalışan kaynakları seçeceğimiz tarife göre belirlenmiş bir virtual machine dir.

4- Docker üzerinde RabbitMQ ve PostgreSQL kurulumunu gösteren ekran görüntüleri;

RabbitMQ 5672 nolu port üzerinden, yönetim paneli ise 15672 nolu port üzerinden çalışmaktadır.

```
m3@m3-workstation:~$ docker run -it --rm --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:3.13-management
Unable to find image 'rabbitmq:3.13-management' locally
3.13-management: Pulling from library/rabbitmq
bccd10f490ab: Pull complete
f1000ca5c91d: Pull complete
15fd8d52bc6c: Pull complete
d9b381d4c87a: Pull complete
497a9eb5f435: Pull complete
dd2bd0cede52: Pull complete
bed3197826ba: Pull complete
0c1d09ec487d: Pull complete
a89de7acba35: Pull complete
74d11b8768c5: Pull complete
b8a34a89caa8: Pull complete
853ab4a97c91: Pull complete
Digest: sha256:18d7104751b66c882c109349f537108c7cd979d87fe9020ef4dc4d773d37691e
Status: Downloaded newer image for rabbitmq:3.13-management
2024-03-20 20:40:47.576658+00:00 [notice] <0.44.0> Application syslog exited with reason: stopped
2024-03-20 20:40:47.582137+00:00 [notice] <0.248.0> Logging: switching to configured handler(s); following messages may not
be visible in this log output
2024-03-20 20:40:47.583156+00:00 [notice] <0.248.0> Logging: configured log handlers are now ACTIVE
```

```
## ##      RabbitMQ 3.13.0
## ##
##### Copyright (c) 2007-2024 Broadcom Inc and/or its subsidiaries
##### ##
##### Licensed under the MPL 2.0. Website: https://rabbitmq.com

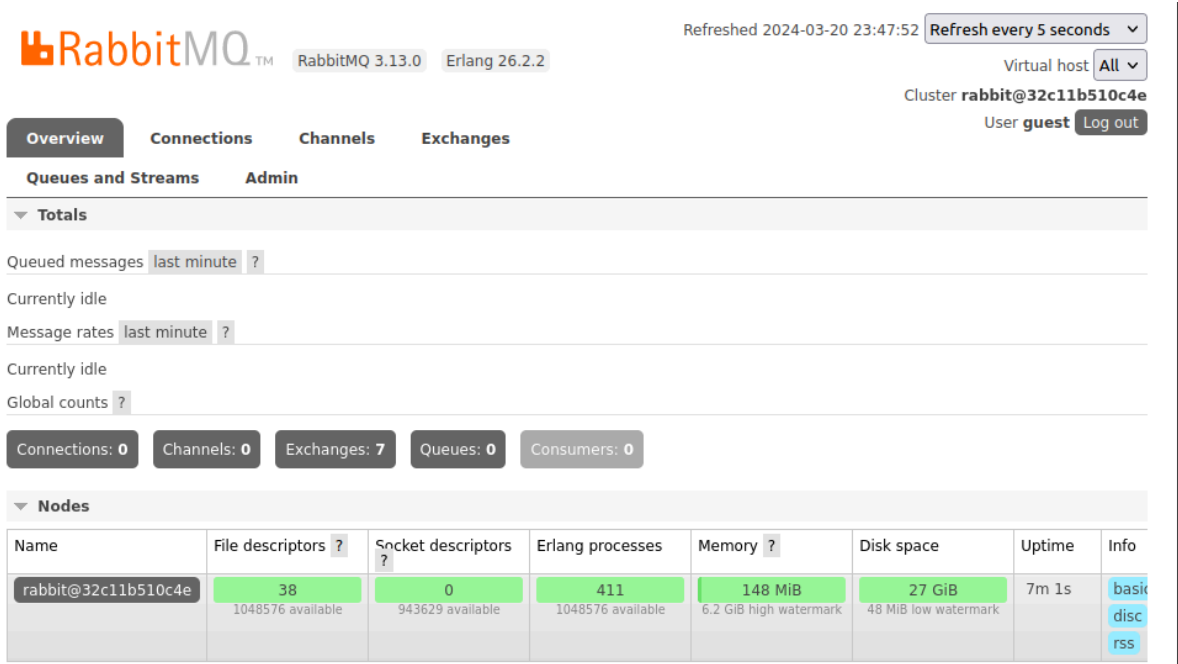
Erlang:      26.2.2 [jit]
TLS Library: OpenSSL - OpenSSL 3.1.5 30 Jan 2024
Release series support status: supported

Doc guides:  https://www.rabbitmq.com/docs/documentation
Support:     https://www.rabbitmq.com/docs/contact
Tutorials:   https://www.rabbitmq.com/tutorials
Monitoring:  https://www.rabbitmq.com/docs/monitoring

Logs: <stdout>

Config file(s): /etc/rabbitmq/conf.d/10-defaults.conf
```

RabbitMQ yönetim konsolu;



PostgreSQL ve PostgreSQL Client kurulumu (image önceden indirildiği için direk oluşturdu);

```
m3@m3-workstation:~$ docker run --name some-postgres -e POSTGRES_PASSWORD=mysecretpassword -d -p 5432:5432 postgres
ecf0783b872689dd027d27a9a1345c99f3fbf499c10bf101eff07a6ebc718546
m3@m3-workstation:~$ sudo apt install postgresql-client
```

PostgreSQL Client üzerinden containerdaki database a bağlantı kurulumu;

```
m3@m3-workstation:~$ psql -h localhost -U postgres -p 5432 -W
Password:
psql (14.11 (Ubuntu 14.11-0ubuntu0.22.04.1), server 16.2 (Debian 16.2-1.pgdg120+2))
WARNING: psql major version 14, server major version 16.
        Some psql features might not work.
Type "help" for help.

postgres=# help
You are using psql, the command-line interface to PostgreSQL.
Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help with psql commands
       \g or terminate with semicolon to execute query
       \q to quit
```

5- Çeşitli Docker komutlarının aşağıdaki listede görebiliriz.

- docker images | Bu komut makinadaki kayıtlı imajları listeler.
- docker ps | Bu komut makinadaki aktif containerları listeler.
docker ps -a | "-a" komutu eklenirse tüm containerlar listelenir.
- docker run <image> | Bu komut verilen docker image ını çalıştırmamızı sağlar,
docker run -it <image> | "-it" eklenerek çağırılırsa interactive moda a geçer.

docker run -p 5001:8080 | "-p" ve ilgili port numaraları eklenerek çağrılırsa container içerisindeki 8080 nolu port ile host makina arasında 5001 nolu port match edilmiş olur.

- docker stop <id> | Bu komut verdiğimiz id ye göre ilgili containerı bulup durdurur.
- docker rm <id> | Bu komut verdiğimiz id ye göre ilgili containerı siler.
- docker exec <id> <cmd> | Bu komut dizisi verdiğimiz id ye göre ilgili containerı bulur ve eklediğimiz komutu bu container içinde çalıştırır.
- docker pull <image> | Bu komut verdiğimiz image ı Docker Hub üzerinden indirmemizi sağlar.
- docker push <image> | Bu komut localdeki image i Docker Hub a göndermemizi sağlar.

6- Monolith mimaride uygulamamız tek bir ünite şeklindedir (database, frontend, backend). Microservice mimaride ise her servis için farklı bir teknoloji seçimi yapabiliriz ve bu servislerin toplamı bir uygulamayı ortaya koyar. Microservice mimaride ölçeklendirme daha kolay yapılabilirken örneğin belirli bir endpoint e fazla istek gelirse o kısmın server kapasitesi artırılır ve bu sayede zaman hemde maliyet olarak avantajlıdır. Monolith mimaride iste uygulamanın tümünü dikkate almak gerekir bu da duruma göre dezavantajlıdır. Olası atak durumlarında microservice mimaride sadece belirli bir servis devre dışı kalırken, monolith de ise tüm uygulama devre dışı kalır. Monolith mimaride uygulamanın geliştirilmesi tek bir codebase olduğundan daha basittir ancak microservice mimaride her servisin kendi codebase i olduğundan daha karmaşık olabilir. Microservice mimaride servislerin kendi arasında konuşması gereken durumlarda çeşitli teknolojiler ile bağlanması gerekir. Örneğin, RESTful API, gRPC veya çeşitli message queue'leri (RabbitMQ gibi). Ancak monolith mimaride bunlara gerek yoktur.

7- API Gateway: Microservice mimarilerinde kullanılan bir proxy server diyebiliriz. Çeşitli tercih sebepleri bulunmaktadır. Örneğin; elimizdeki her microservice üzerinde authentication ve authorization gibi güvenlik mekanizmalarını tekrar tekrar geliştirmek istemeyiz. Bu durumda API Gateway bizim için bu yönlendirmeyi sağlayacağından yetkileri ve kullanıcıları önden kontrol edecek ve yetkisiz erişimi engelleyecektir.

Logging olarak da kullanılabilir. Bu durumda microservice lere gelen istekleri ve kimden geldiği gibi çeşitli bilgileri tek bir noktada toplayabiliriz.

Caching kullanımı da mümkündür. Microservice a atılan benzer istekleri çok daha kısa bir şekilde yanıtlayarak hem sunucu yükü azaltılır. Hem de isteklere verilen yanıtlama süresi düşmüş olur.

Service Discovery: Servisimiz başka servislerin kendisini kolayca bulabilmesi için belirli bir porta yayın yapar. Örneğin UPnP protokolü üzerinde yayın yapan bir ip kameraya erişmek istedik. Ancak kameraya bir statik ip verilmediği için ip adresini bilmiyoruz veya ip adresi değişti. UPnP üzerinden bir service discovery yaptığımızda ağımda istediğimiz özelliklerdeki ve bilinen seri numaralarına kayıtlı cihazları bulabiliriz ve bulduğumuz ip adresi üzerinden erişiriz.

Load Balancer: Servislerimize gelen istekler arttıkça kullandığımız sistemler yetersiz kalabilir. Kullanıcı sayımızın artması ile bu durum oluşabileceği gibi saldırganlar da çeşitli DoS atakları ile servisleri zorlayabilirler. Bu gibi durumların önüne geçmek için load balancer sistemleri kullanılır.

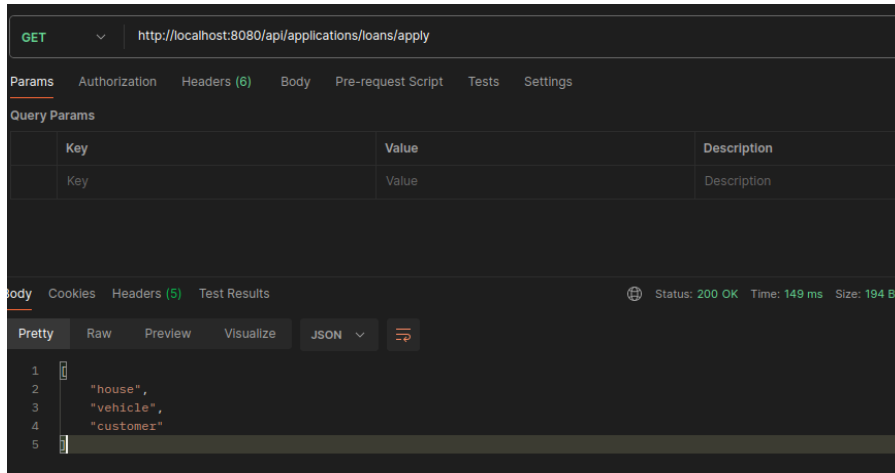
Client side load balancer; Örneğin sitenin ana sayfası cacheden yüklendiğinde frontend üzerinde çalışan kod, server ip listesinden kişinin bulunduğu bölge veya ülkeye göre en yakınındaki adresi seçer. Bundan sonraki yapılacak istekler doğrudan o sunucuya gönderilir.

Server side load balancer; Örneğin bir siteye istek geldiğinde öncelikle proxy sunucu karşılar. Proxy sunucu; ip bilgisinden çeşitli bilgileri toplar (ülke, bölge) veya sunucu yüküne göre listesinden bir sunucuyu tercih eder ve gönderir. Gelen cevabı da response olarak gönderir.

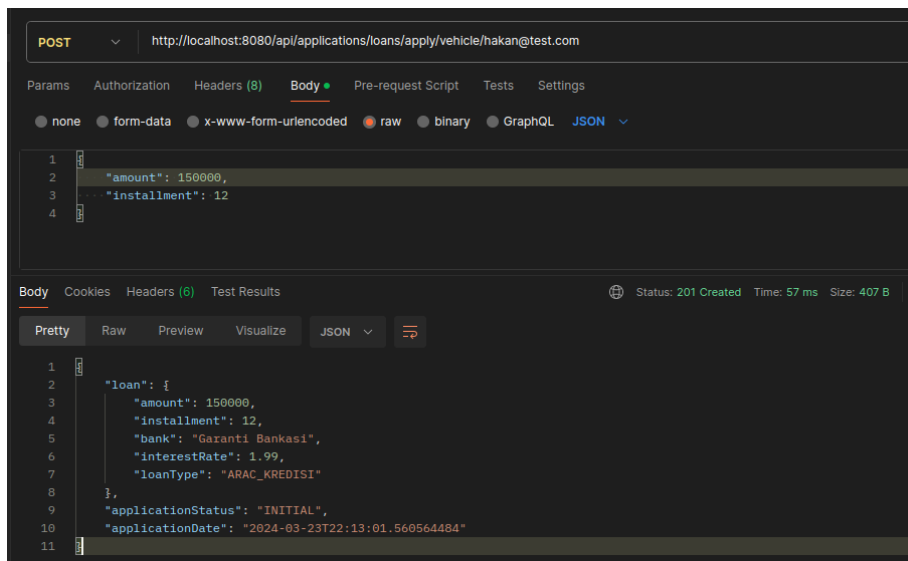
8- Hibernate, JPA, Spring Data frameworkleri için örnekler repoda görülebilir.

9- KredinBizde uygulamasına gerekli özellikler eklenmiştir. Repoda görülebilir. Eklenen endpointlerimiz şunlardır;

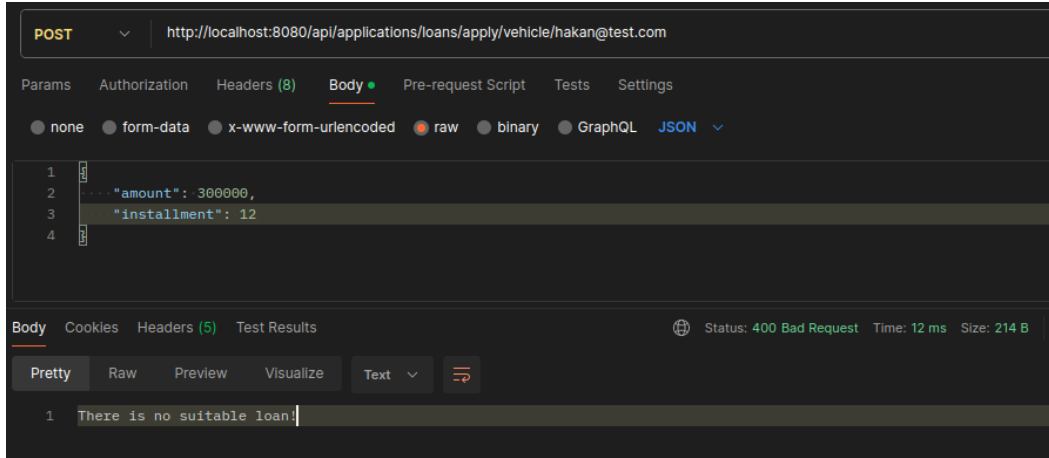
- GET /api/applications/loans/apply - Yapabildiğimiz kredi başvuru tiplerini döner.



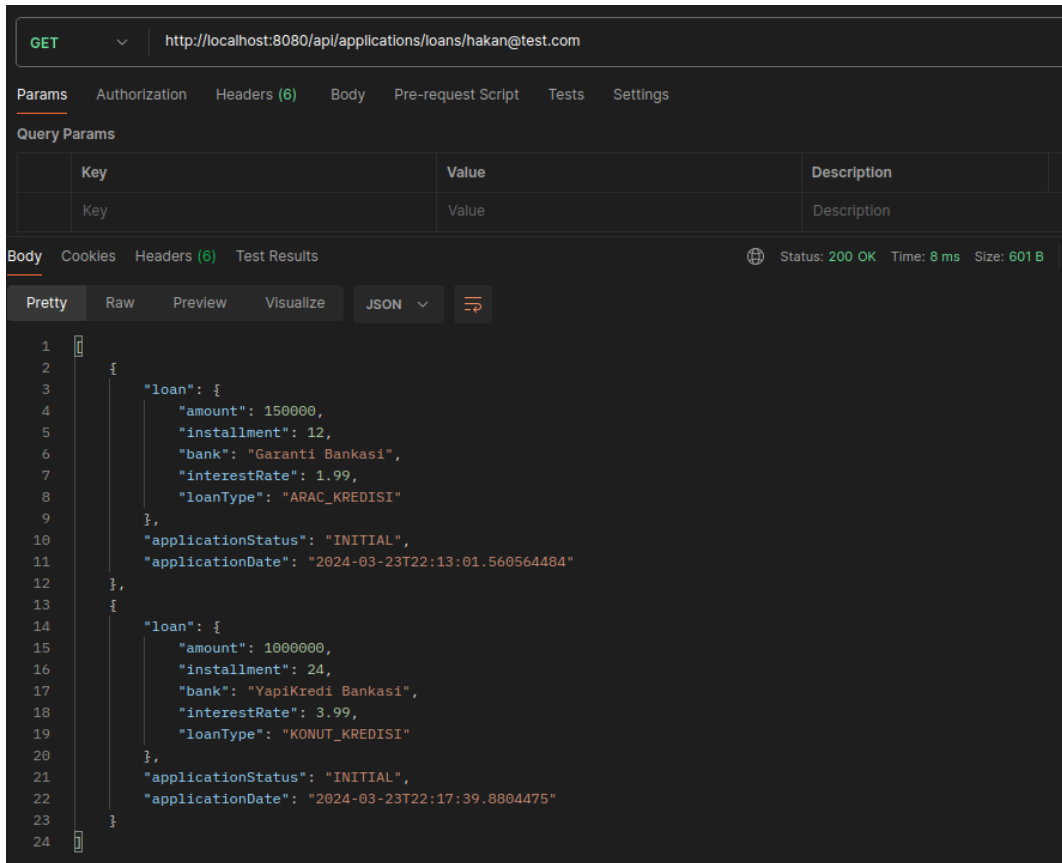
- POST /api/applications/loans/apply/{loanTypeName}/{email} - Path üzerinde kredi başvuru tipi ve müşterinin email adresini girmemiz gerekir. Body üzerine amount ve installment belirtmemiz gerekir. Uygun kredi bulunamazsa hata döner. Uygun kredi bulunduğunda yeni bir başvuru oluşturur ve kaydeder. Örnek uygun kredi durumu;



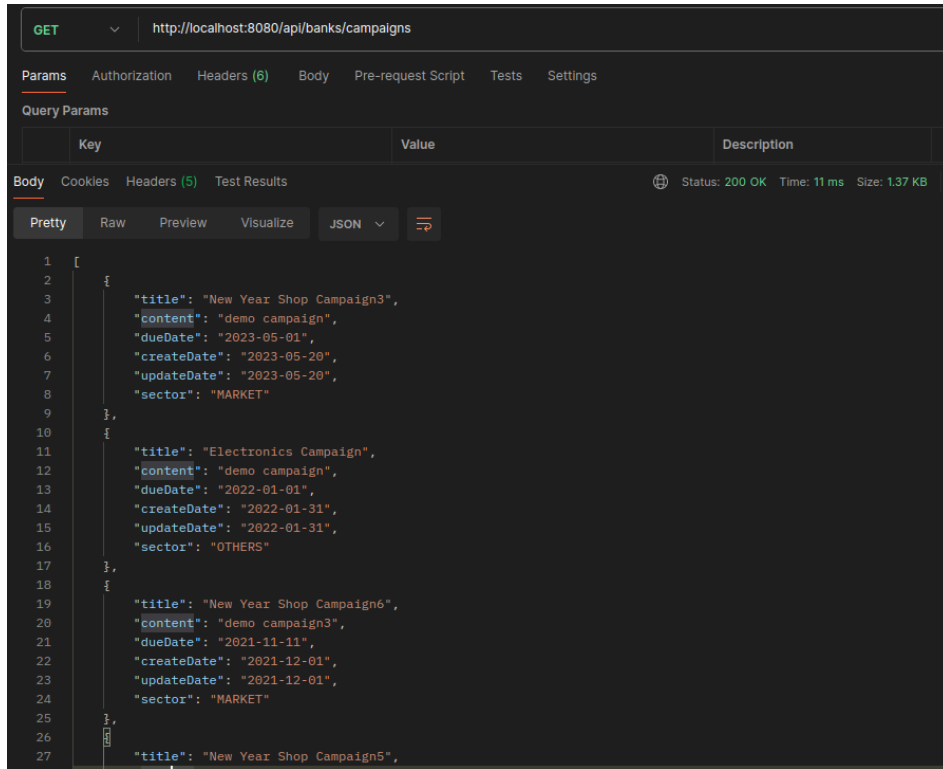
Kredi bulunamadığı durum;



- GET /api/applications/loans/{email} - Kayıtlı başvurularımızı döner.



- GET /api/banks/campaigns - Sistemdeki kayıtlı kampanyaları yeniden eskiye göre sıralı olarak verir.



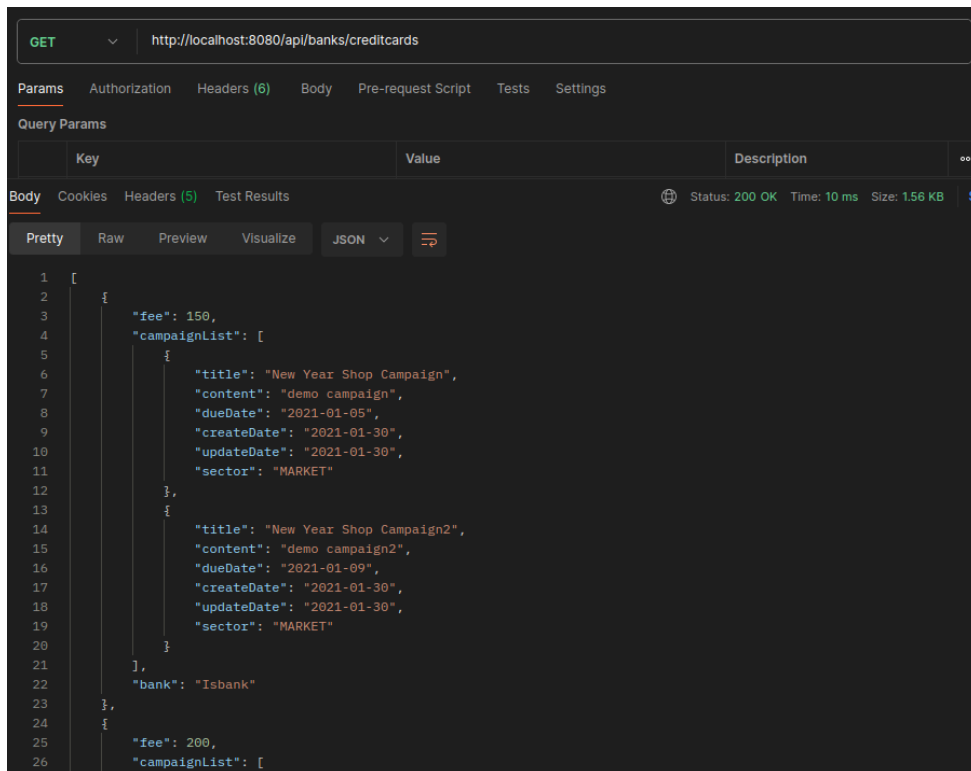
```
GET http://localhost:8080/api/banks/campaigns

Query Params

Body
Pretty Raw Preview Visualize JSON

1  [
2    {
3      "title": "New Year Shop Campaign3",
4      "content": "demo campaign",
5      "dueDate": "2023-05-01",
6      "createDate": "2023-05-20",
7      "updateDate": "2023-05-20",
8      "sector": "MARKET"
9    },
10   {
11     "title": "Electronics Campaign",
12     "content": "demo campaign",
13     "dueDate": "2022-01-01",
14     "createDate": "2022-01-31",
15     "updateDate": "2022-01-31",
16     "sector": "OTHERS"
17   },
18   {
19     "title": "New Year Shop Campaign6",
20     "content": "demo campaign3",
21     "dueDate": "2021-11-11",
22     "createDate": "2021-12-01",
23     "updateDate": "2021-12-01",
24     "sector": "MARKET"
25   },
26   {
27     "title": "New Year Shop Campaign5",
```

- GET /api/banks/creditcards - Bankaların kredi kartlarını ve kampanyalarını döner.



```
GET http://localhost:8080/api/banks/creditcards

Query Params

Body
Pretty Raw Preview Visualize JSON

1  [
2    {
3      "fee": 150,
4      "campaignList": [
5        {
6          "title": "New Year Shop Campaign",
7          "content": "demo campaign",
8          "dueDate": "2021-01-05",
9          "createDate": "2021-01-30",
10         "updateDate": "2021-01-30",
11         "sector": "MARKET"
12       },
13       {
14         "title": "New Year Shop Campaign2",
15         "content": "demo campaign2",
16         "dueDate": "2021-01-09",
17         "createDate": "2021-01-30",
18         "updateDate": "2021-01-30",
19         "sector": "MARKET"
20       }
21     ],
22     "bank": "Isbank"
23   },
24   {
25     "fee": 200,
26     "campaignList": [
```