

GraphQL, ilk olarak 2012 yılında mobil uygulamalar için kullanmaya başlayan Facebook tarafından geliştirilmiştir.

Geliştirdiğimiz bir projede, çok sayıda anlık istek alan sayfalar nedeniyle Restful mimarisinin ve MVC’nin getirdiği bazı zorluklarla karşılaştık. Artık veriye erişim ve veriyi sorgulama modelimizin yeni alternatifler içinde yeniden tanımlanmaya başlandığını görmeye başladık.

REST tabanlı web servislerinde birçok alana sahip ve büyük boyuttaki veriler client'a gönderildiğinde trafiğin fazla kullanılmasına neden olur.

GraphQL; “resource bazlı bir istek modeli yerine(/api/products gibi)”, client'ın ihtiyacı olan veriyi kendinin tanımladığı bir modeldir.

GraphQL'in çözmeye çalıştığı problem, uygulama seviyesinde(application level) client'lar için bir sorgulama arayüzü oluşturmak.

Application level'den kastımız, server tarafında oluşturulan Query Interface ile client tarafında bu interface uygun olarak client'ın kendi sorgularını oluşturabilme imkanını kazanması yeteneği diye düşünebiliriz.

GraphQL içeren çeşitli open source projeler vardır. Bunlardan bazıları aşağıdaki gibidir.

- Apollo
- Offix
- Graphback
- OpenAPI-to-GraphQL

1. Client'ın, veri tabanımızdan x ürününü istediğini düşünün.

/api/product/x

2. Ayrıca bu ürünün stok durumu ve bu ürünü de kapsayan son 20 satışa da ihtiyacımız olsun.

/api/product/x/stok
/api/product/x/sales?limit=20

3. Sonra, ürünle ilgili resimleri de ilgili tablodan sorgulayalım

/api/product/x/pictures

Yukarıda, 4 ayrı istekde bulunduk.

Her bir kaynak veya o kaynağın ilişkisi olan diğer kaynak için ayrı ayrı endpoint'ler tanımlayıp, sorgulamalar yaptığımız bu model yerine; yukarıdaki işlemi bir GraphQL sorgusu ile eşleştirmeye çalışalım.

```
{
  urun(x){
    _id,
    name,
    price,
    stok {
      count
    },
    sales(limit:20) {
      customer,
      total
    },
    pictures{
      thumbs,
      name
    }
  }
}
```

GraphQL sorgumuza baktığımızda, sorgunun kendini açıklayan ve kolay bir yapıda olduğunu görmekteyiz.

Ayrıca client'ın örnekteki gibi “**ürün**” alanı için sadece “**id, name ve price**” alanlarını istediğini ve yine ürünle ilgili son 20 satışı ve ürünün resimlerini de beklediğini açıkca anlamaktayız.

İlk modelimizde(REST/MVC), ilgili ürünleri almanız sonrasında bu ürün için tekrar tekrar istekde bulunmamız gerekecek.

REST/MVC mimarisiyle ilgili başka bir sıkıntıyı ve GraphQL’in bu konudaki avantajını pratik bir karşılaştırma yaparak anlamaya çalışalım; Uygulamamızın bir yerinde, ürünümüzün “ismi,fiyatı ve markasını” göstermemiz gereksin.

Örnek request:

/api/products/x

Aşağıdaki gibi bir response alalım;

```
{
  id:22,
  name:"televizyon",
  price:"12.32",
  brand:"Vestel",
  model:"y1233",
  isOnSale:false,
  isOnHomePage:false,
  lastUpdatedByUserId:232,
  created_at:"132132133",
  updated_at:"132142135"
}
```

Bu tür bir response yerine(ürünün sadece şu 3 alanına ihtiyacımız varken “ismi,fiyatı ve markası”) client ürün bilgisinin tamamını alacaktır. Uygulamanın başka bir noktasında ise ürünün sadece id alanına ihtiyaç duyabiliriz. GraphQL örneğinde, client neye ihtiyacı olduğunu kendisi belirteceği için server tarafında aynı alt yapı ile client'ın hemen hemen tüm ihtiyaçlarına karşılayacaktır.

Örnek;

```
{
  urun(x) {
    name,
    price,
    brand
  }
}
```

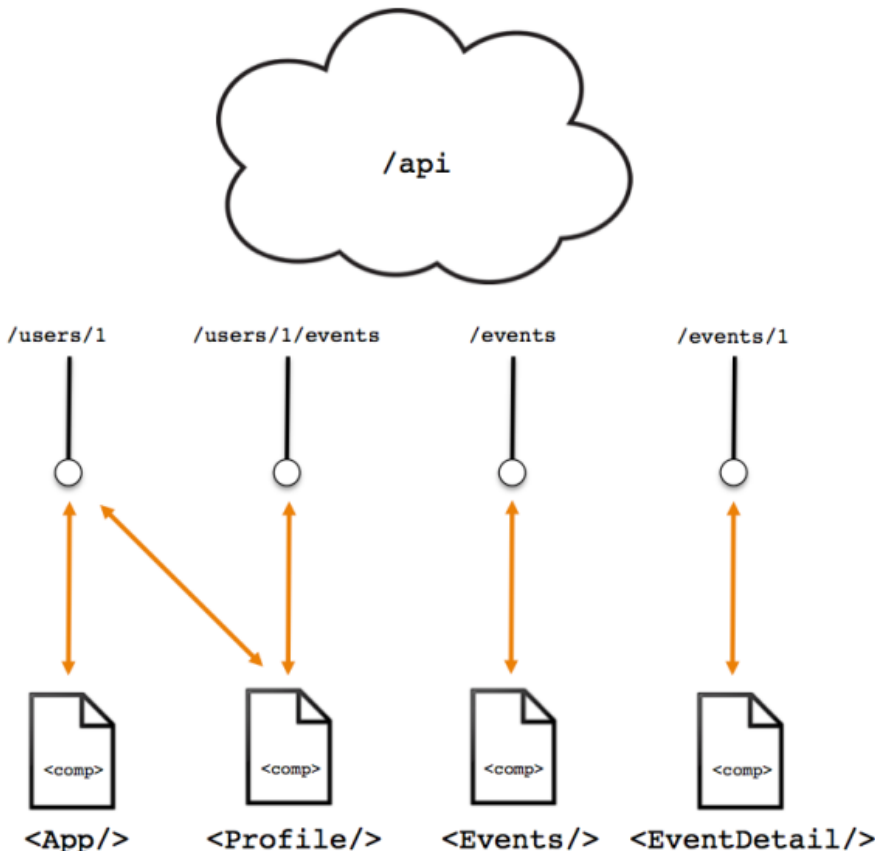
Veya

```
urun(x) {
  _id
}
```

Üstelik, GraphQL framework'lerin bir çoğunun dahili olarak sağladığı caching(önbellekleme) ile zaten client'ta bulunan verileri tekrar sorgulamadan, mevcut veriyi cache'den kolayca alma yeteneği var. Aşağıdaki iki resme bakarak GraphQL ve REST/MVC yapıları arasındaki ilk farkı görebiliriz.

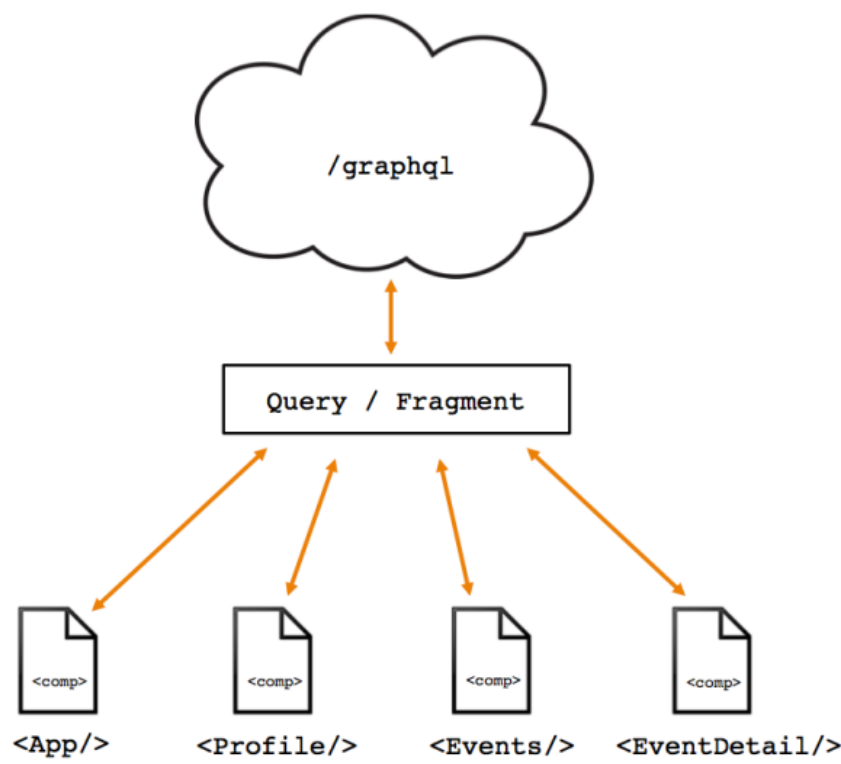
MVC/Rest

Many Components Making Requests To Many Endpoints



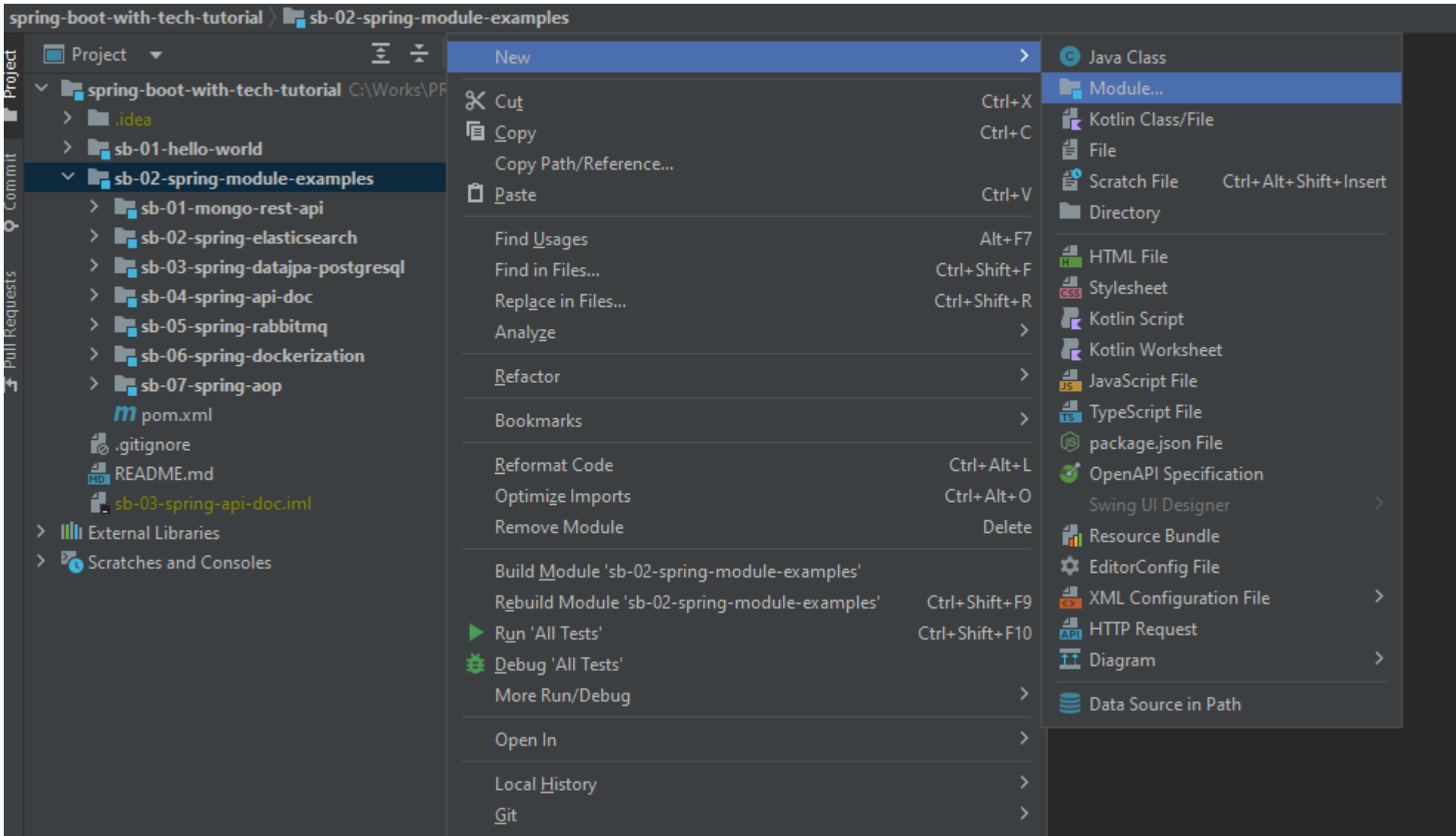
GraphQL

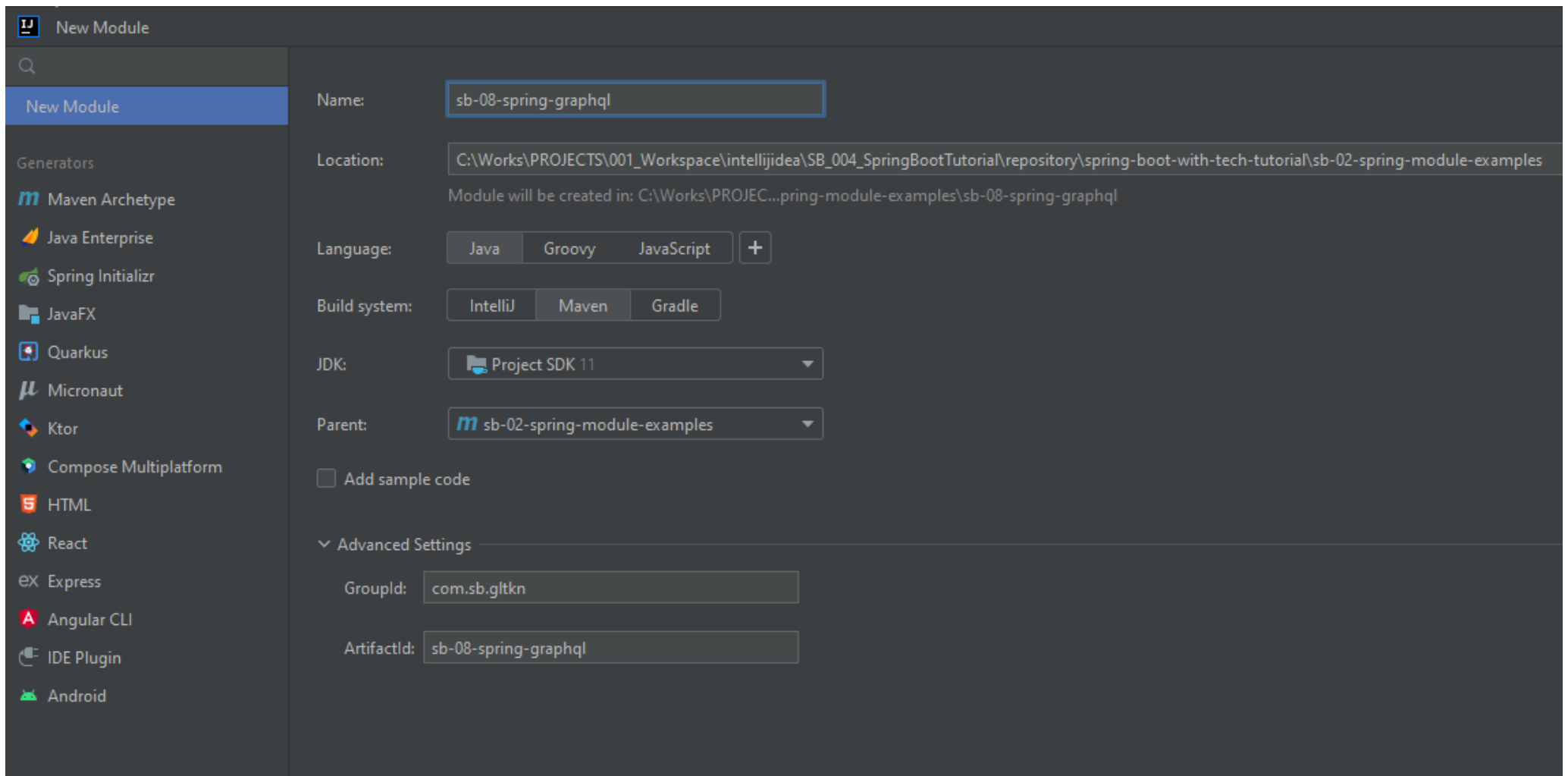
Many Components Making
Requests to One GraphQL Endpoint



Yukarıdaki görüntüde GraphQL sorgumuzun ne kadar anlaşılır ve “declarative” olduğunu görüyoruz.

<https://docs.github.com/en/graphql/overview/explorer>





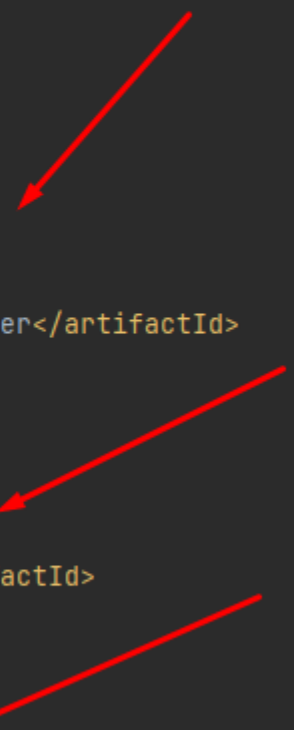
```
m pom.xml (sb-08-spring-graphql) x
15  <properties>
16      <maven.compiler.source>11</maven.compiler.source>
17      <maven.compiler.target>11</maven.compiler.target>
18  </properties>
19
20  <dependencies>
21      <!-- spring-boot web -->
22      <dependency>
23          <groupId>org.springframework.boot</groupId>
24          <artifactId>spring-boot-starter-web</artifactId>
25          <version>2.7.0</version>
26      </dependency>
27      <!-- spring-boot data-jpa -->
28      <dependency>
29          <groupId>org.springframework.boot</groupId>
30          <artifactId>spring-boot-starter-data-jpa</artifactId>
31      </dependency>
32      <!-- spring-boot h2database -->
33      <dependency>
34          <groupId>com.h2database</groupId>
35          <artifactId>h2</artifactId>
36          <version>1.4.197</version>
37      </dependency>
38      <!-- lombok -->
39      <dependency>
40          <groupId>org.projectlombok</groupId>
41          <artifactId>lombok</artifactId>
42          <optional>true</optional>
43      </dependency>
44  </dependencies>
45
46  </project>
```

spring-boot-with-tech-tutorial > sb-02-spring-module-examples > sb-08-spring-graphql > pom.xml

Project
Commit
Pull Requests
Bookmarks
Structure

spring-boot-with-tech-tutorial C:\Works\PROJECTS\0
> .idea
> sb-01-hello-world
sb-02-spring-module-examples
> sb-01-mongo-rest-api
> sb-02-spring-elasticsearch
> sb-03-spring-datajpa-postgresql
> sb-04-spring-api-doc
> sb-05-spring-rabbitmq
> sb-06-spring-dockerization
> sb-07-spring-aop
sb-08-spring-graphql
src
main
java
com.sb.gltkn
Application
com.sb.gltkn.controller
VehicleMutationResolver
VehicleQueryResolver
com.sb.gltkn.dto
VehicleDto
com.sb.gltkn.entity
Vehicle
com.sb.gltkn.repository
VehicleRepository
resources
application.properties
vehicle.graphqls
test
target
pom.xml
pom.xml
.gitignore

```
30 </dependency>
31 <!-- spring-boot h2database -->
32 <dependency>
33     <groupId>com.h2database</groupId>
34     <artifactId>h2</artifactId>
35     <version>1.4.197</version>
36 </dependency>
37 <!-- lombok -->
38 <dependency>
39     <groupId>org.projectlombok</groupId>
40     <artifactId>lombok</artifactId>
41     <optional>true</optional>
42 </dependency>
43 <!-- spring-boot graphql-->
44 <dependency>
45     <groupId>com.graphql-java</groupId>
46     <artifactId>graphql-spring-boot-starter</artifactId>
47     <version>5.0.2</version>
48 </dependency>
49 <!-- graphql-java-tools -->
50 <dependency>
51     <groupId>com.graphql-java</groupId>
52     <artifactId>graphql-java-tools</artifactId>
53     <version>5.2.4</version>
54 </dependency>
55 <!-- spring-boot graphiql -->
56 <dependency>
57     <groupId>com.graphql-java</groupId>
58     <artifactId>graphiql-spring-boot-starter</artifactId>
59     <version>5.0.2</version>
60 </dependency>
61 </dependencies>
```



spring-boot-with-tech-tutorial > sb-02-spring-module-examples > sb-08-spring-graphql > src > main > java > com > sb > gltkn > entity > Vehicle

Project

- sb-01-hello-world
- sb-02-spring-module-examples
 - sb-01-mongo-rest-api
 - sb-02-spring-elasticsearch
 - sb-03-spring-datajpa-postgresql
 - sb-04-spring-api-doc
 - sb-05-spring-rabbitmq
 - sb-06-spring-dockerization
 - sb-07-spring-aop
 - sb-08-spring-graphql
 - src
 - main
 - java
 - com.sb.gltkn
 - Application
 - com.sb.gltkn.controller
 - VehicleMutationResolver
 - VehicleQueryResolver
 - com.sb.gltkn.dto
 - VehicleDto
 - com.sb.gltkn.entity
 - Vehicle
 - com.sb.gltkn.repository
 - VehicleRepository
 - resources
 - application.properties
 - vehicle.graphqls
 - test
- pom.xml
- .gitignore
- README.md
- sb-03-spring-api-doc.iml

Vehicle.java

```
5
6  import javax.persistence.*;
7  import java.io.Serializable;
8  import java.util.Date;
9
10  @Entity
11  @Table(name = "vehicle")
12  @Getter
13  @Setter
14  public class Vehicle implements Serializable {
15
16      @Id
17      @GeneratedValue(strategy = GenerationType.AUTO)
18      private Long id;
19
20      @Column(length = 100, name = "vehicle_type")
21      private String type;
22
23      @Column(length = 100, name = "model_code")
24      private String modelCode;
25
26      @Column(length = 100, name = "brand_name")
27      private String brandName;
28
29      @Column(length = 100, name = "launch_date")
30      private Date launchDate;
31  }
32
```

spring-boot-with-tech-tutorial > sb-02-spring-module-examples > sb-08-spring-graphql > src > main > java > com > sb > gltkn

Project

- sb-01-hello-world
- sb-02-spring-module-examples
 - sb-01-mongo-rest-api
 - sb-02-spring-elasticsearch
 - sb-03-spring-datajpa-postgresql
 - sb-04-spring-api-doc
 - sb-05-spring-rabbitmq
 - sb-06-spring-dockerization
 - sb-07-spring-aop
 - sb-08-spring-graphql
 - src
 - main
 - java
 - com.sb.gltkn
 - Application
 - com.sb.gltkn.controller
 - VehicleMutationResolver
 - VehicleQueryResolver
 - com.sb.gltkn.dto
 - VehicleDto
 - com.sb.gltkn.entity
 - Vehicle
 - com.sb.gltkn.repository
 - VehicleRepository
 - resources
 - application.properties
 - vehicle.graphqls
 - test
 - pom.xml
 - .gitignore
 - README.md
 - sb-03-spring-api-doc.iml

Vehicle.java

VehicleDto.java

```
1  package com.sb.gltkn.dto;
2
3  import lombok.Data;
4
5  @Data
6  public class VehicleDto {
7
8      private String type;
9      private String modelCode;
10     private String brandName;
11 }
12
```



```

VehicleRepository.java x
1 package com.sb.gltkn.repository;
2
3 import com.sb.gltkn.entity.Vehicle;
4 import org.springframework.data.jpa.repository.JpaRepository;
5 import org.springframework.stereotype.Repository;
6
7 import java.util.List;
8
9 4 usages
10 @Repository
11 public interface VehicleRepository extends JpaRepository<Vehicle, Long> {
12     1 usage
13     List<Vehicle> getByTypeLike(String type);
14 }
15

```

```

VehicleQueryResolver.java x
1 package com.sb.gltkn.controller;
2
3 import com.coxautodev.graphql.tools.GraphQLQueryResolver;
4 import com.sb.gltkn.entity.Vehicle;
5 import com.sb.gltkn.repository.VehicleRepository;
6 import lombok.RequiredArgsConstructor;
7 import org.springframework.stereotype.Component;
8
9 import java.util.List;
10 import java.util.Optional;
11
12 @Component
13 @RequiredArgsConstructor
14 public class VehicleQueryResolver implements GraphQLQueryResolver {
15
16     2 usages
17     private final VehicleRepository vehicleRepository;
18
19     public List<Vehicle> getVehicles(String type){
20         return vehicleRepository.getByTypeLike(type);
21     }
22
23     public Optional<Vehicle> getById(Long id){
24         return vehicleRepository.findById(id);
25     }
26 }

```

```

VehicleMutationResolver.java x
3  import com.coxautodev.graphql.tools.GraphQLMutationResolver;
4  import com.sb.gltkn.dto.VehicleDto;
5  import com.sb.gltkn.entity.Vehicle;
6  import com.sb.gltkn.repository.VehicleRepository;
7  import lombok.RequiredArgsConstructor;
8  import org.springframework.stereotype.Component;
9
10 import java.util.Date;
11
12 @Component
13 @RequiredArgsConstructor
14 public class VehicleMutationResolver implements GraphQLMutationResolver {
15
16     1 usage
17     private final VehicleRepository vehicleRepository;
18
19     public Vehicle createVehicle(VehicleDto vehicleDto){
20         return vehicleRepository.save(dtoToEntity(vehicleDto));
21     }
22
23     1 usage
24     private Vehicle dtoToEntity(VehicleDto vehicleDto) {
25         Vehicle vehicle = new Vehicle();
26         vehicle.setBrandName(vehicleDto.getBrandName());
27         vehicle.setModelCode(vehicleDto.getModelCode());
28         vehicle.setType(vehicleDto.getType());
29         vehicle.setLaunchDate(new Date());
30         return vehicle;
31     }
32 }

```

spring-boot-with-tech-tutorial > sb-02-spring-module-examples > sb-08-spring-graphql > src > main > resources

Project

- sb-01-hello-world
- sb-02-spring-module-examples
 - sb-01-mongo-rest-api
 - sb-02-spring-elasticsearch
 - sb-03-spring-datajpa-postgresql
 - sb-04-spring-api-doc
 - sb-05-spring-rabbitmq
 - sb-06-spring-dockerization
 - sb-07-spring-aop
 - sb-08-spring-graphql
 - src
 - main
 - java
 - com.sb.gltkn
 - Application
 - com.sb.gltkn.controller
 - VehicleMutationResolver
 - VehicleQueryResolver
 - com.sb.gltkn.dto
 - VehicleDto
 - com.sb.gltkn.entity
 - Vehicle
 - com.sb.gltkn.repository
 - VehicleRepository
 - resources
 - application.properties
 - vehicle.graphqls
 - test

application.properties x

```

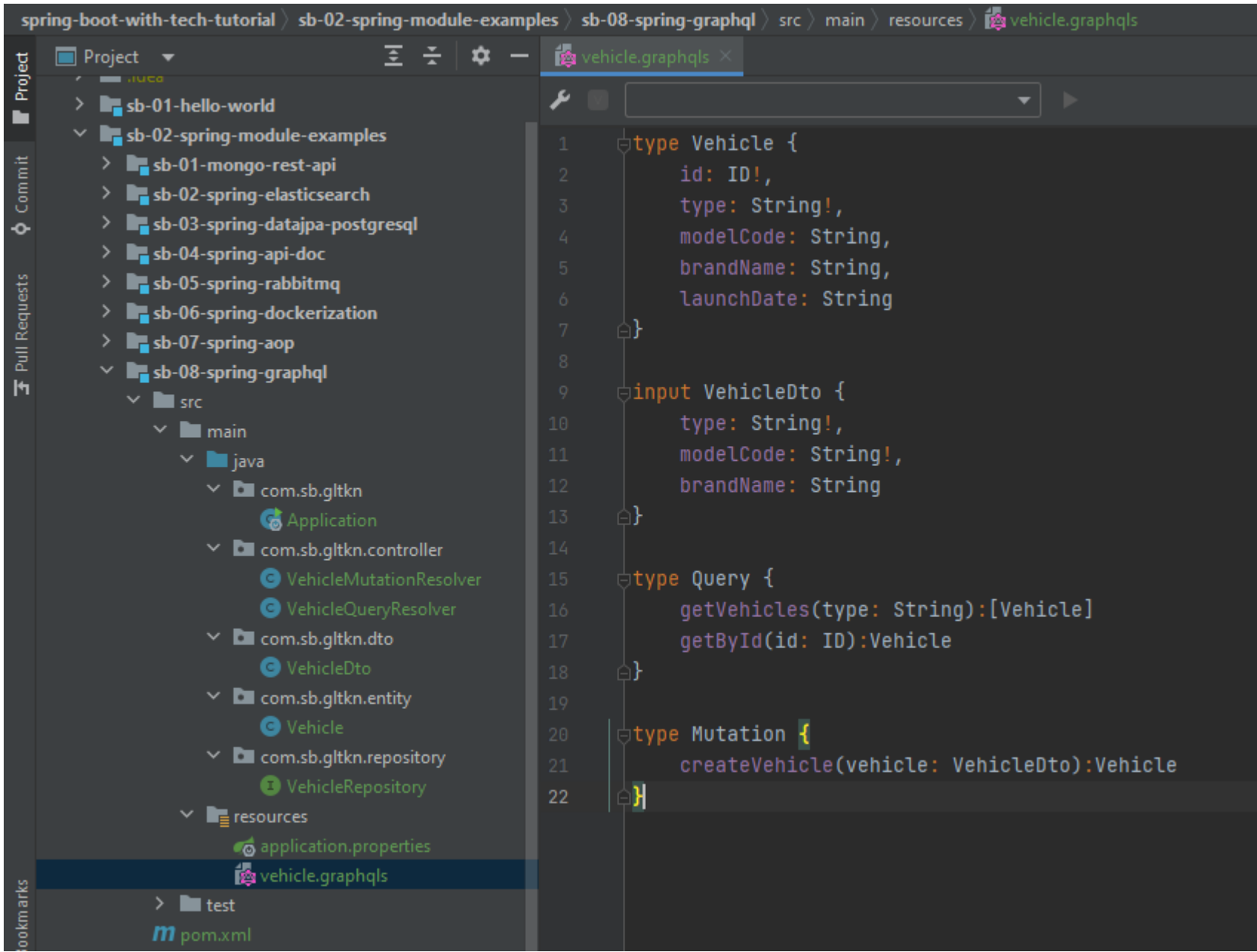
1 server.port=8090

```

Bookmarks

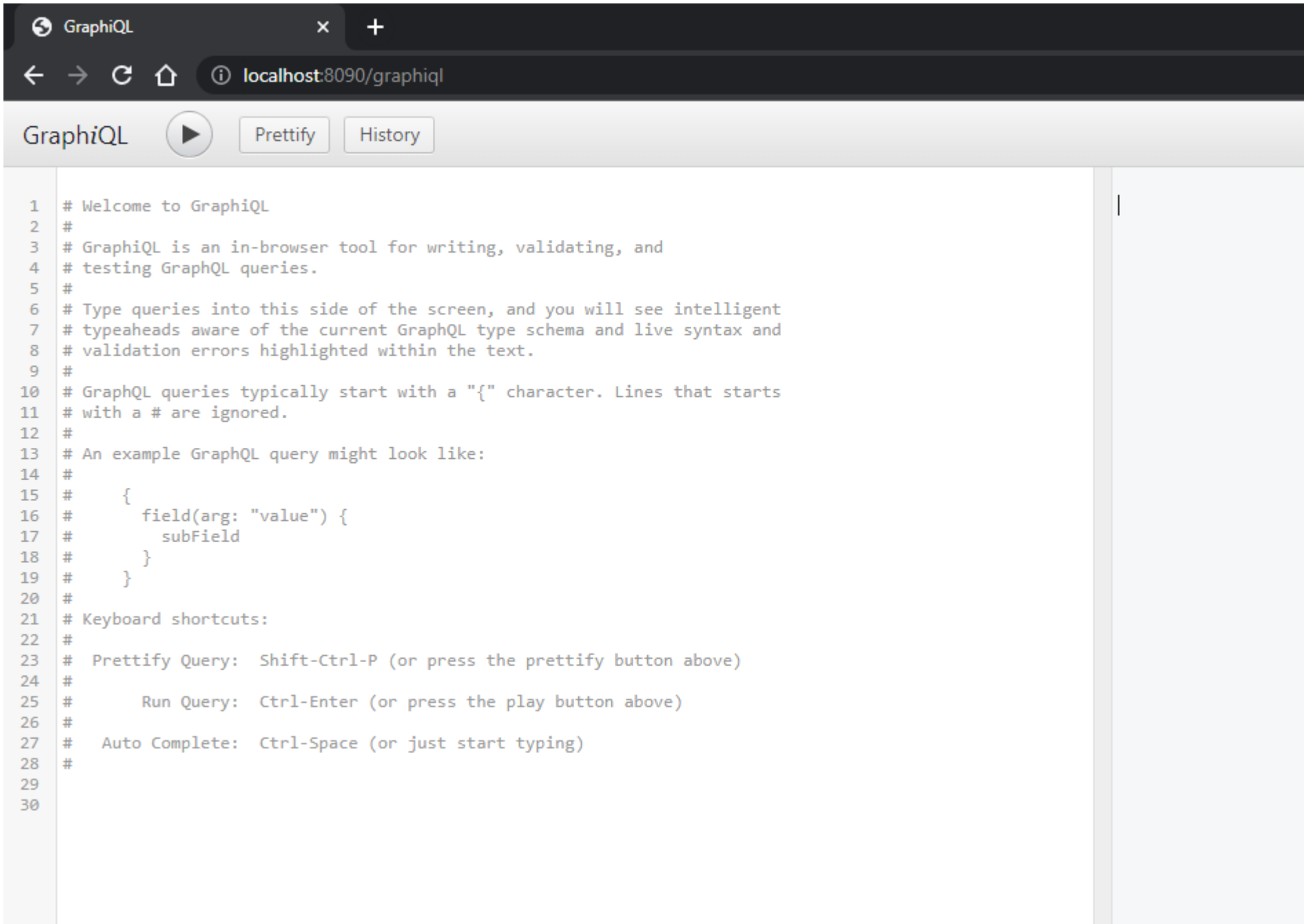
m pom.xml

Server'a **input** ile belirtilen veriler gönderilebilir, **type** ile belirtilenler ise client'a dönebilir. Yanında ünlem işareti olan field'ların zorunlu olduğunu belirtmiş olduk.



Uygulamamızı çalıştıralım.

<http://localhost:8090/graphql>



Şu input'u graphql üzerinden girerek data oluşturalım.

```
mutation{
  createVehicle(vehicle:{type:"SW", modelCode:"2019", brandName:"Audi"}){
    id,
    launchDate,
    brandName
  }
}
```

Aşağıda 3 veri oluşturduk.

GraphiQL

localhost:8090/graphql?query=m...

GraphiQL

1 2 3 4 5 6 7 8

mutation{
 createVehicle(vehicle:{type:"SW", modelCode:"2019", brandName:"Audi"}){
 id,
 launchDate,
 brandName
 }
}

1 2 3 4 5 6 7 8

{
 "data": {
 "createVehicle": {
 "id": "1",
 "launchDate": "Sun Jun 26 22:28:59 EET 2022",
 "brandName": "Audi"
 }
 }
}

Schema

Mutation

Search Mutation...

No Description

FIELDS

createVehicle(vehicle: VehicleDto): Vehicle

GraphiQL

localhost:8090/graphql?query=m...

GraphiQL

1 2 3 4 5 6 7 8

mutation{
 createVehicle(vehicle:{type:"SW", modelCode:"2019", brandName:"Mercedes"}){
 id,
 launchDate,
 brandName
 }
}

1 2 3 4 5 6 7 8

{
 "data": {
 "createVehicle": {
 "id": "3",
 "launchDate": "Sun Jun 26 22:32:15 EET 2022",
 "brandName": "Mercedes"
 }
 }
}

Schema

Mutation

Search Mutation...

No Description

FIELDS

createVehicle(vehicle: VehicleDto): Vehicle

GraphiQL

localhost:8090/graphql?query=m...

GraphiQL

1 2 3 4 5 6 7 8

mutation{
 createVehicle(vehicle:{type:"Sedan", modelCode:"2019", brandName:"Mercedes"}){
 id,
 launchDate,
 brandName
 }
}

1 2 3 4 5 6 7 8

{
 "data": {
 "createVehicle": {
 "id": "2",
 "launchDate": "Sun Jun 26 22:31:27 EET 2022",
 "brandName": "Mercedes"
 }
 }
}

Schema

Mutation

Search Mutation...

No Description

FIELDS

createVehicle(vehicle: VehicleDto): Vehicle

GraphiQL

localhost:8090/graphql?query=m...

GraphiQL

1 2 3 4 5 6 7 8

mutation{
 createVehicle(vehicle:{type:"SW", modelCode:"2019", brandName:"Mercedes"}){
 id,
 launchDate,
 brandName
 }
}

1 2 3 4 5 6 7 8

{
 "data": {
 "createVehicle": {
 "id": "3",
 "launchDate": "Sun Jun 26 22:32:15 EET 2022",
 "brandName": "Mercedes"
 }
 }
}

Schema

Mutation

Search Mutation...

No Description

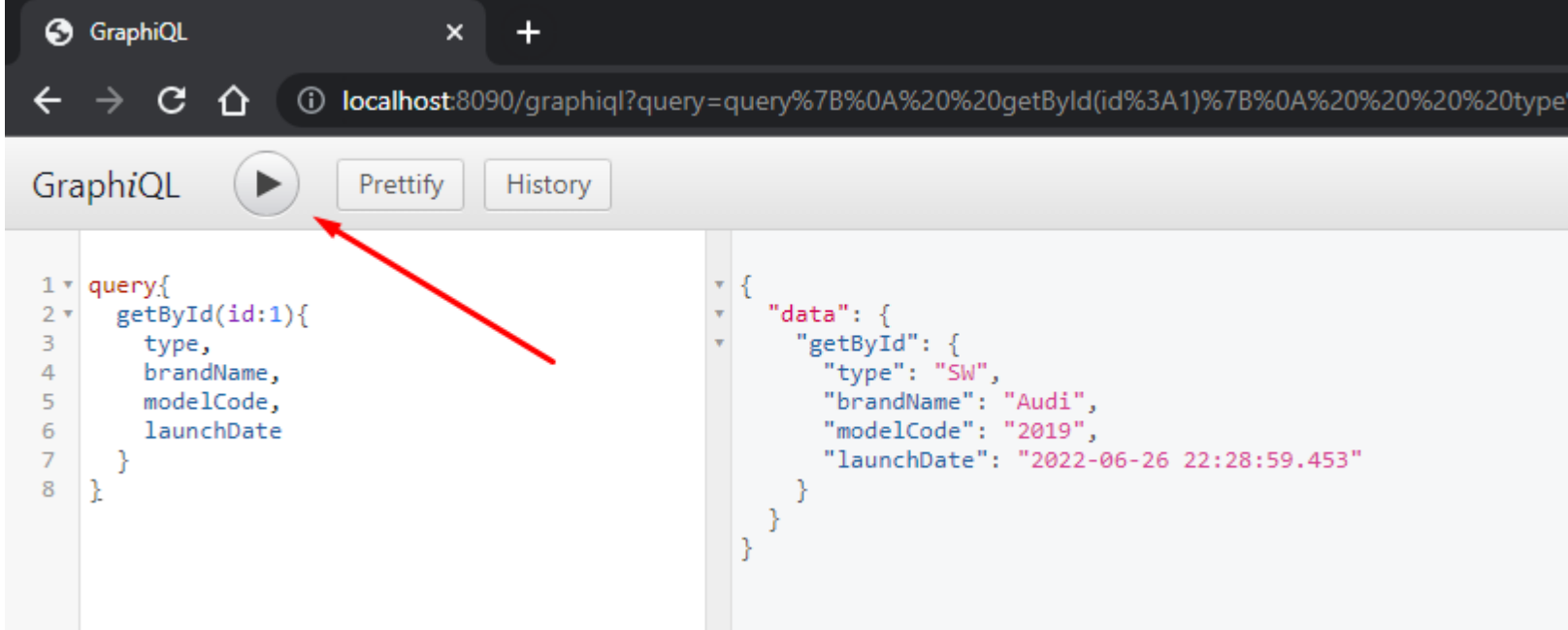
FIELDS

createVehicle(vehicle: VehicleDto): Vehicle

Şimdi bu oluşturduğumuz datalar için query işlemlerimizi gerçekleştirelim.

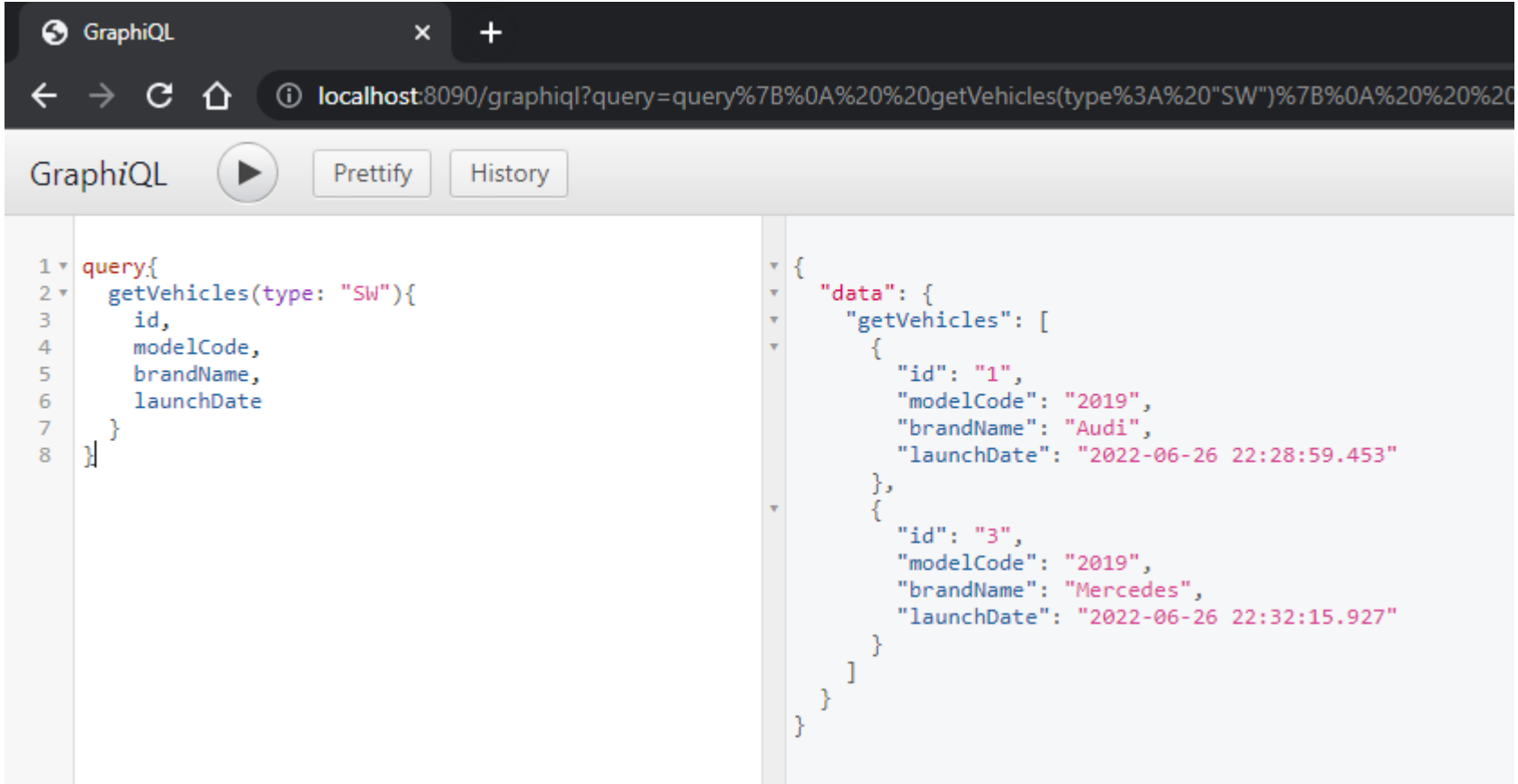
Şu sorgu ile id'si 1 olan kayıtlı çekeceğiz ve geriye *type, brandName, modelCode, launchDate* bilgilerini döndüreceğiz.

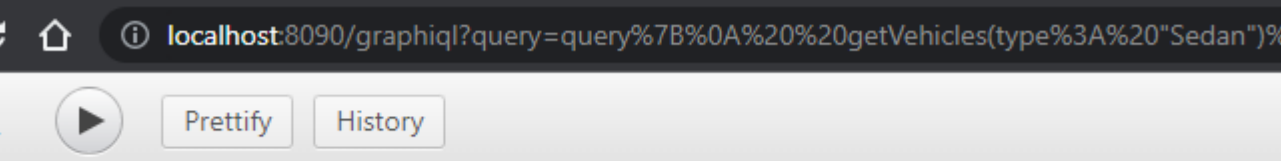
```
query{
  getById(id:1){
    type,
    brandName,
    modelCode,
    launchDate
  }
}
```



Şimdi **getVehicles** fonksiyonundan **type** bilgisi **SW** olanları sorgulayalım.

```
query{
  getVehicles(type: "SW"){
    id,
    modelCode,
    brandName,
    launchDate
  }
}
```





The screenshot shows the GraphQL Playground interface. The top bar includes navigation icons and the URL `localhost:8090/graphql?query=query%7B%0A%20%20getVehicles(type%3A%20%22Sedan%22)%7B%0A%20%20%20id%2C%20%20modelCode%2C%20%20brandName%2C%20%20launchDate%7D%7D`. Below the URL bar, there are buttons for "Prettify" and "History". The main area is split into two panels. The left panel shows the query:

```
1 query{
2   getVehicles(type: "Sedan"){
3     id,
4     modelCode,
5     brandName,
6     launchDate
7   }
8 }
```

The right panel shows the JSON response:

```
{
  "data": {
    "getVehicles": [
      {
        "id": "2",
        "modelCode": "2019",
        "brandName": "Mercedes",
        "launchDate": "2022-06-26 22:31:27.439"
      }
    ]
  }
}
```

The screenshot shows the GraphQL IDE interface. The left pane contains the following query:

```
1 query{  
2   getVehicles(type: "S"){  
3     id,  
4     type,  
5     modelCode,  
6     brandName,  
7     launchDate  
8   }  
9 }
```

The right pane shows the JSON response:

```
{  
  "data": {  
    "getVehicles": [  
      {  
        "id": "1",  
        "type": "SW",  
        "modelCode": "2019",  
        "brandName": "Audi",  
        "launchDate": "2022-06-26 22:28:59.453"  
      },  
      {  
        "id": "2",  
        "type": "Sedan",  
        "modelCode": "2019",  
        "brandName": "Mercedes",  
        "launchDate": "2022-06-26 22:31:27.439"  
      },  
      {  
        "id": "3",  
        "type": "SW",  
        "modelCode": "2019",  
        "brandName": "Mercedes",  
        "launchDate": "2022-06-26 22:32:15.927"  
      }  
    ]  
  }  
}
```