SPRINGBOOT, HIBERNATE, MYSQL CRUD REST API

Bu projede springboot, hibernate ve mySQL veritabanını kullanarak web servisleri oluşturacağız.

CRUD(Create, Read, Update, Delete) işlemleri gerçekleştireceğiz. Çalışan ve şirket bilgilerinin yönetildiği sistem şeklinde geliştireceğimiz bu projede gerçekleştirilecek işlemler;

1)Sistemde yeni çalışan veya şirket kaydı oluşturma

2)Kayıtlı tüm çalışan veya şirket verilerini görüntüleme

3)Belirli bir id'ye sahip çalışan veya şirket verisini görüntüleme

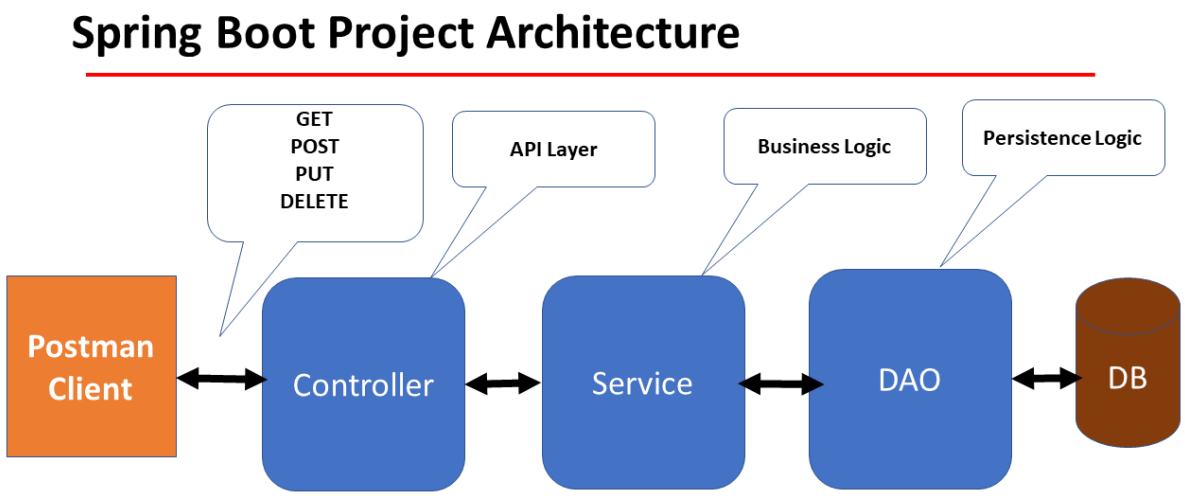
4) Belirli bir id'ye sahip çalışan veya şirket verisini güncelleme

5) Belirli bir id'ye sahip çalışan veya şirket verisini silme

Kullaanılan Tool ve Teknolojiler

- Spring Boot-2.7.0 version
- JDK 18.0.1
- Spring Data JPA
- IDE: Intelij Idea
- Hibernate
- Postman
- MySQL

Projede Kullanılan Mimari Yapı



Bu yapı temelde 3 katmanın olduğu Spring Boot uygulamalarında yaygın kullanılan bir mimaridir.

- **Controller Layer:** İstemci(client) tarafından gönderilen tüm istekleri işler. Aynı zamanda API Layer da denilir çünkü API'lar controller layerda tanımlanır.
- **Service Layer:** Verileri işleyen katmandır. Tüm uygulama düzeyindeki business logic service layerda tutulur. Business logic; uygulamanın veri işleme, yorumlama kurallarının, algoritmalarının bütününe verilen isimdir.
- **DAO(Data Access Object) Layer:** Repository(Depo) katmanı da denir. Veritabanıyla konuşmaktan sorumlu olan katmandır. Repository layer veritabanıyla ilgili tüm mantığı barındırır buna Persistence Logic denilir.

Bu üç katman birbirinden bağımsızdır, bağımlılıklar sonradan enjekte edilecektir.

Tüm REST Endpoints(get, post, put, delete) test etmek için Postman Client kullanacağız.

Proje Adımları:

1- Create Spring Boot Project

İlk olarak Spring Boot projesi oluşturmamız gerekiyor. Projeyi oluştururken Spring Initializr kullanıldı. Proje bilgilerini ve dependency'leri ekledikten sonra proje oluşturulur.

Kullanılan Dependency'ler:

- Spring Web
- Spring Data JPA
- MySQL Driver
- Lombok

2- Create Packaging Structure

Spring Boot projemiz için paketleme yapısı oluşturuldu.

Oluşturulan Paketler:

- Controller
- Service -> ServiceImpl
- Exception
- Repository
- Model

3- Configure MySQL Database

Spring Boot projemiz için MySQL veritabanını yapılandıracağız. Öncelikle bir veritabanı oluşturmamız gerekiyor bunun için MySQL Workbench kullanıyoruz. MySQL Workbench, MySQL sunucusuna bağlanmak için popüler bir istemcidir. MySQL Workbench'de "create database ems;" komutu ile veritabanını oluşturduk. Veritabanını kullanabilmemiz için gerekli konfigürasyonları yapmalıyız. Bunun için projede "application.properties" dosyasına aşağıdaki özellikleri ekleyerek yapılandırmamız gerekiyor.

```
spring.datasource.url=jdbc:mysql://localhost:3306/ems?useSSl=false
spring.datasource.username=root
spring.datasource.password=1234

#Hibernate properties
spring.jpa.open-in-view=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect
#create, create-drop
spring.jpa.hibernate.ddl-auto=update
```

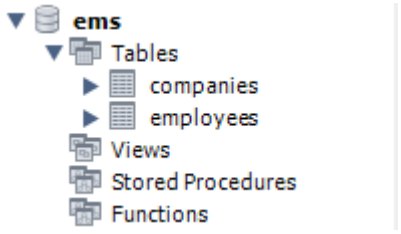
4- Create Employee and Company JPA Entity

Şimdi Employee ve Company varlıkları için JPA entity oluşturmamız gerekiyor. Bunun için model paketi içerisinde varlık isimlerinde yani Employee ve Company isminde 2 class oluşturduk ve varlık özelliklerini tanımlıyoruz.

Anotasyon(annotation) kullanımı için Lombok kütüphanesini kullanacağız. Lombok javada sürekli yaptığımız işlemleri kısa yoldan yapmamızı sağlayan bir kütüphanedir. Getters, setters, constructors, toString gibi metotları @Getters gibi anotasyonlar kullanarak daha temiz ve az kod yazarak gerçekleştirmemizi sağlar.

```
29 usages
@Data
@Entity
@Table(name = "employees")
public final class Employee {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    @Column(name = "firmaId")
    private long firmaId;
    @Column(name = "firstname", nullable = false)
    private String firstname;
    @Column(name = "lastname")
    private String lastname;
    @Column(name = "email")
    private String email;
}
```

```java
29 usages
@Data
@Entity
@Table(name = "companies")
public class Company {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    @JoinColumn(name = "firmaId")
    private long firmaId;
    @Column(name = "companyName", nullable = false)
    private String companyName;
    @Column(name = "address")
    private String address;
}
```



5- Create Employee and Company Repository Interface

Employee ve Company için repository oluşturmamız gerekiyor. Repository bizim tüm veritabanı işlemlerimizin gerçekleştirileceği katmandır. Repository paketi altında Employee Repository ve Company Repository isimlerinde Interface oluşturuyoruz ve bu Interface'leri JPARepository sınıfından extend ediyoruz. Böylece Spring bootun bize sağlamış olduğu bu kolaylıkla JpaRepositoryde tanımlı database işlemlerini doğrudan kullanabiliyoruz. Burada Bağlı tablolar kullandığımız için EmployeeRepository içerisinde bağlılığı belirten bir Query tanımlıyoruz.

```java
@Repository
public interface EmployeeRepository extends JpaRepository<Employee, Long>{
    @Query(value = "select e.id, e.firmaId, e.firstname, e.lastname, e.email from employees as e " +
            "INNER JOIN companies as c on e.firmaId=c.firmaId", nativeQuery=true)
    List<Object[]> findEmployee();
}
```

```java
@Repository
@Component
public interface CompanyRepository extends JpaRepository<Company, Long> {

}
```

6- Create ResourceNotFoundException Class

Kullanıcı isteğinde belirtilen kimliğe sahip kaynağın veritabanında bulunmaması durumunda kullanıcıya hata mesajı döndürülmelidir. Bunun için Exception paketi altında ResourceNotFoundException adında bir class oluşturuldu. Burada kaynak adı(ResourceName), alan adı(fieldName) ve alan değeri(fieldValue) kullanarak veritabanında olmayan bir kaynak istemciye döndürülür. Bu class RuntimeException classından extend edilir.

```java
@ResponseStatus(value = HttpStatus.NOT_FOUND)
public class ResourceNotFoundException extends RuntimeException {
    2 usages
    private String resourceName;
    2 usages
    private String fieldName;
    2 usages
    private Object fieldValue;

    6 usages
    public ResourceNotFoundException(String resourceName, String fieldName, Object fieldValue) {
        super(String.format("%s not found with %s : '%s' ", resourceName, fieldName, fieldValue));
        this.resourceName= resourceName;
        this.fieldName= fieldName;
        this.fieldValue= fieldValue;
    }

    public String getResourceName() { return resourceName; }

    public String getFieldName() { return fieldName; }

    public Object getFieldValue() { return fieldValue; }
}
```

7- Build Service Layer

Sırada REST API yapısını oluşturmak var. Öncelikle Service paketi içerisinde EmployeeService ve CompanyService isimli Interface'leri oluşturduk ve Interface içindeki mettotların implement edileceği EmployeeImpl ve CompanyImpl classlarını ServiceImpl paketi içerisinde oluşturduk. Bu classları Interface'lerinden implement ettik. Business sürecin ilerlediği katman Service katmanıdır. Service katmanı, Controller ve Repository arasındaki iletişimi sağlayan katman olarak düşünülebilir. Controllera gelen requestlere göre Repositorydeki database işlemlerini yapmak için gerekli olan metodları bu katmanda tanımlıyoruz.

```java
public interface EmployeeService {
    1 usage  1 implementation
    Employee saveEmployee(Employee employee);
    1 usage  1 implementation
    List<Employee> getAllEmployee();
    1 usage  1 implementation
    Employee getEmployeeById(long id);
    1 usage  1 implementation
    Employee updateEmployee(Employee employee, long id);
    1 usage  1 implementation
    void deleteEmployee(long id);
}
```

```java
public interface CompanyService {
    1 usage  1 implementation
    Company saveCompany(Company company);
    1 usage  1 implementation
    List<Company> getAllCompany();
    1 usage  1 implementation
    Company getCompanyById(long id);
    1 usage  1 implementation
    Company updateCompany(Company company, long id);
    1 usage  1 implementation
    void deleteCompany(long id);
}
```

```java
@Service
public class EmployeeServiceImpl implements EmployeeService {
    8 usages
    private EmployeeRepository employeeRepository;

    public EmployeeServiceImpl(EmployeeRepository employeeRepository) {
        super();
        this.employeeRepository = employeeRepository;
    }

    1 usage
    @Override
    public Employee saveEmployee(Employee employee) { return employeeRepository.save(employee); }

    1 usage
    @Override
    public List<Employee> getAllEmployee() { return employeeRepository.findAll(); }

    1 usage
    @Override
    public Employee getEmployeeById(long id) {
        return employeeRepository.findById(id).orElseThrow(() ->new ResourceNotFoundException("Employee", "Id", "id
    }
```

```java
    @Override
    public Employee updateEmployee(Employee employee, long id) {
        //we need to check whether employee with given id is exist in DB or not
        Employee existingEmployee= employeeRepository.findById(id).orElseThrow(() -> new ResourceNotFoundExceptio
        existingEmployee.setFirstname(employee.getFirstname());
        existingEmployee.setLastname(employee.getLastname());
        existingEmployee.setFirmaId(employee.getId());
        existingEmployee.setEmail(employee.getEmail());
        //save existing employee to DB
        employeeRepository.save(existingEmployee);
        return existingEmployee;
    }

    1 usage
    @Override
    public void deleteEmployee(long id) {
        //check whether a employee exiist in a DB or not
        employeeRepository.findById(id).orElseThrow(() -> new ResourceNotFoundException("Employee", "Id", "id"));

        employeeRepository.deleteById(id);
    }
```

```java
@Service
public class CompanyServiceImpl implements CompanyService {
    8 usages
    private CompanyRepository companyRepository;

    public CompanyServiceImpl(CompanyRepository companyRepository) {
        super();
        this.companyRepository = companyRepository;
    }

    1 usage
    @Override
    public Company saveCompany(Company company) { return companyRepository.save(company); }

    1 usage
    @Override
    public List<Company> getAllCompany() { return companyRepository.findAll(); }

    1 usage
    @Override
    public Company getCompanyById(long id) {
        return companyRepository.findById(id).orElseThrow(() ->new ResourceNotFoundException("Company", "Id", "id")
    }
```

```java
    @Override
    public Company updateCompany(Company company, long id) {
        //we need to check whether employee with given id is exist in DB or not
        Company existingCompany= companyRepository.findById(id).orElseThrow(() -> new ResourceNotFoundException("C
        existingCompany.setFirmaId(company.getFirmaId());
        existingCompany.setCompanyName(company.getCompanyName());
        existingCompany.setAddress(company.getAddress());
        //save existing employee to DB
        companyRepository.save(existingCompany);
        return existingCompany;
    }

    1 usage
    @Override
    public void deleteCompany(long id) {
        //check whether a employee exiist in a DB or not
        companyRepository.findById(id).orElseThrow(() -> new ResourceNotFoundException("Company", "Id", "id"));

        companyRepository.deleteById(id);
    }
```

8- Build Controller Layer

Controller katmanımızı oluşturalım. Controller en dış katmandır ve requestler ilk olarak controllera gelir. Controller paketi altında EmployeeController ve CompanyController adında iki class oluşturulur. Controller sınıfı olduğunu belirtmek için **@**RestController anotasyonunu ekliyoruz. Kullanılan Service'in bağımlılığı enjekte edilir. Ardından create, getAll, getById, update ve delete REST API oluşturulur. Bu işlemler Post, get, put ve delete annotation'lar ile gerçekleştirilir.

- **POST** : Post isteği ile requestte body içerisinde gönderilen veriler sunucuya iletilir ve yeni bir kaynak oluşturulur. Biz örneğimizde POST metodu ile yeni bir çalışan veya şirket kaydı oluşturacağız.

- **GET** : Sunucudan veri almak için kullandığımız yöntemdir. Get isteği yapılırken request içerisinde body kullanılmaz. Uygulamamızda tüm çalışan veya şirket bilgilerini ve belirli bir id'ye sahip çalışan veya şirket bilgisini almak için kullanacağız.
- **PUT** : Sunucudaki belirli bir kaynağı güncellemek için kullanılır. Biz de belirli bir id'ye sahip çalışan veya şirket bilgisini update etmek için kullanacağız.
- **DELETE** : Sunucudaki veriyi silmek için kullanılır. Örneğimizde belirli bir id'ye göre çalışan veya şirket bilgisini sunucudan silmek için kullanacağız.

```java
@Controller
@RequestMapping("/api/employees")
public class EmployeeController {
    6 usages
    private EmployeeService employeeService;

    public EmployeeController(EmployeeService employeeService) {
        super();
        this.employeeService = employeeService;
    }
    // build create employee REST API
    @PostMapping()
    public ResponseEntity<Employee>saveEmployee(@RequestBody Employee employee){
        return new ResponseEntity<Employee>(employeeService.saveEmployee(employee), HttpStatus.CREATED);
    }

    //build get all employees REST API
    @GetMapping
    public ResponseEntity<List<Employee>> getAllEmployee(){
        List<Employee> allEmployee= employeeService.getAllEmployee();
        return new ResponseEntity<List<Employee>>(allEmployee, HttpStatus.OK);
    }
```

```java
    //build get employee by ID REST API
    //http://localhost:8080/api/employees/1
    @GetMapping("{id}")
    public ResponseEntity<Employee> getEmployeeById(@PathVariable("id") long employeeId){
        return new ResponseEntity<Employee>(employeeService.getEmployeeById(employeeId),HttpStatus.OK);
    }

    //build update employee REST API
    //http://localhost:8080/api/employees/1
    @PutMapping("{id}")
    public ResponseEntity<Employee> updateEmployee(@PathVariable("id") long id,@RequestBody Employee employee){
        return new ResponseEntity<Employee>(employeeService.updateEmployee(employee, id), HttpStatus.OK);
    }

    //build delete employee REST API
    //http://localhost:8080/api/employees/1
    @DeleteMapping("{id}")
    public ResponseEntity<String> deleteEmployee(@PathVariable("id") long id){
        //delete employee from DB
        employeeService.deleteEmployee(id);
        return new ResponseEntity<String>( body: "Employee deleted successfully!", HttpStatus.OK);
    }
```

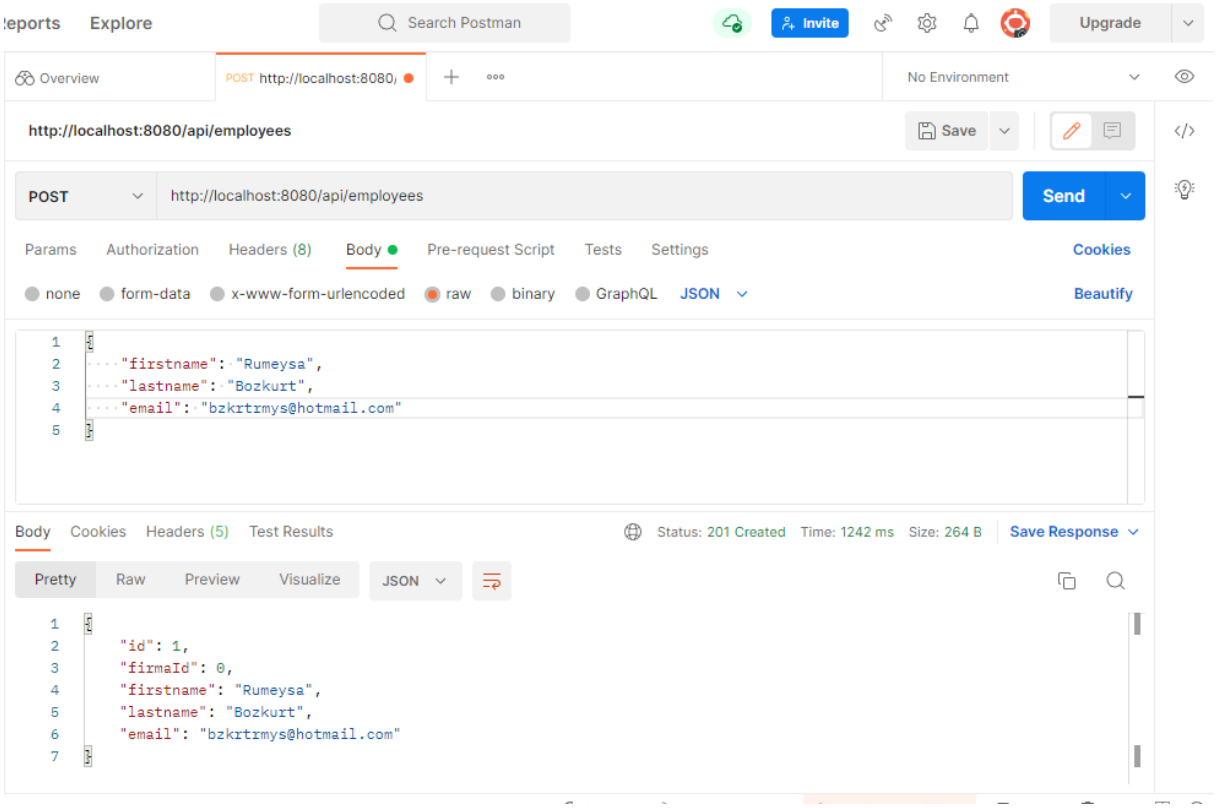
```java
@Controller
@RequestMapping("/api/companies")
public class CompanyController {
    6 usages
    private CompanyService companyService;

    public CompanyController(CompanyService companyService) {
        super();
        this.companyService = companyService;
    }

    // build create company REST API
    @PostMapping()
    public ResponseEntity<Company> saveCompany(@RequestBody Company company){
        return new ResponseEntity<Company>(companyService.saveCompany(company), HttpStatus.CREATED);
    }

    //build get all company REST API
    @GetMapping
    public ResponseEntity<List<Company>> getAllCompany(){
        List<Company> allCompany= companyService.getAllCompany();
        return new ResponseEntity<List<Company>>(allCompany, HttpStatus.OK);
    }
```

```java
    //build get company by ID REST API
    //http://localhost:8080/api/companies/1
    @GetMapping("{id}")
    public ResponseEntity<Company> getCompanyById(@PathVariable("id") long companyId){
        return new ResponseEntity<Company>(companyService.getCompanyById(companyId),HttpStatus.OK);
    }

    //build update company REST API
    //http://localhost:8080/api/companies/1
    @PutMapping("{id}")
    public ResponseEntity<Company> updateCompany(@PathVariable("id") long id,@RequestBody Company company){
        return new ResponseEntity<Company>(companyService.updateCompany(company, id), HttpStatus.OK);
    }

    //build delete company REST API
    //http://localhost:8080/api/companies/1
    @DeleteMapping("{id}")
    public ResponseEntity<String> deleteCompany(@PathVariable("id") long id){
        //delete employee from DB
        companyService.deleteCompany(id);
        return new ResponseEntity<String>( body: "Company deleted successfully!", HttpStatus.OK);
    }
```

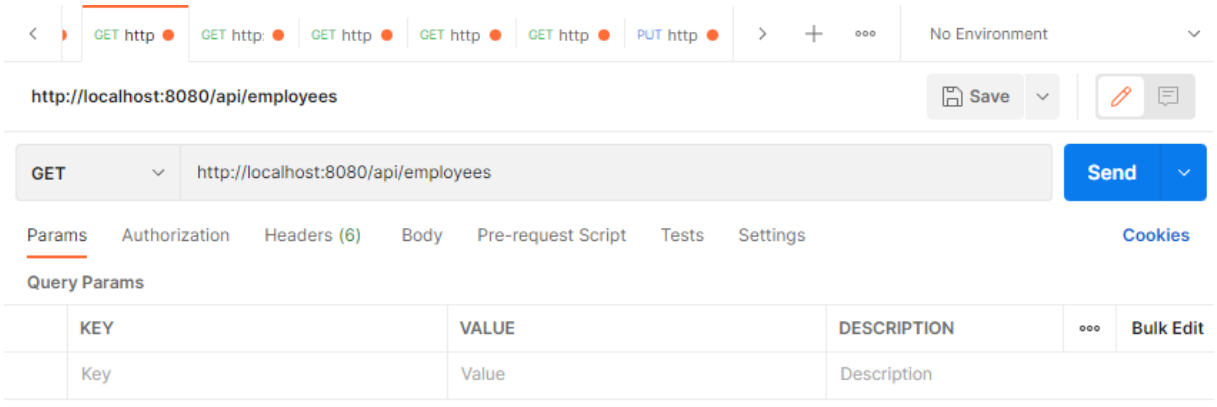
Proje Test İşlemleri:

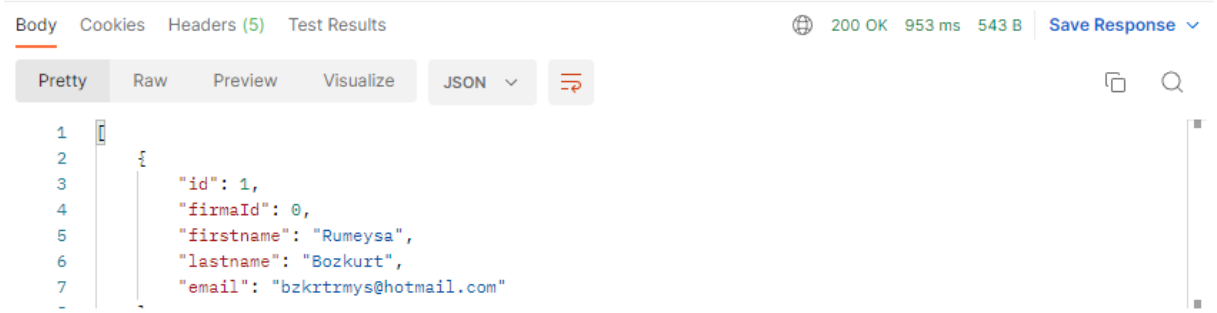
Uygulamamızı tamamladığımıza göre şimdi projemizi ayağa kaldırıp Postman üzerinden testlerini yapmaya başlayabiliriz. İlk olarak **SpringbootBackendApplication.java** classına gidip run diyerek projemizi ayağa kaldırmamız gerekiyor. Postmanı açıp test etmeye başlayabiliriz.

- İlk olarak post metodunu test edip veritabanına yeni bir çalışan kaydı oluşturduğumuzu görelim. Http metodunu POST olarak seçip body'de gerekli alanları request içerisinde gönderdiğimizde başarıyla kayıt oluşturduğumuzu görebiliriz.

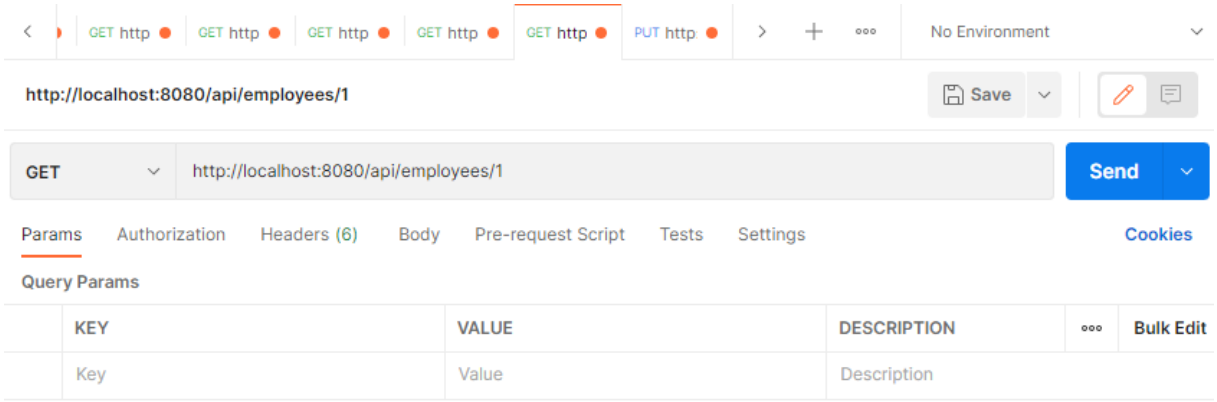


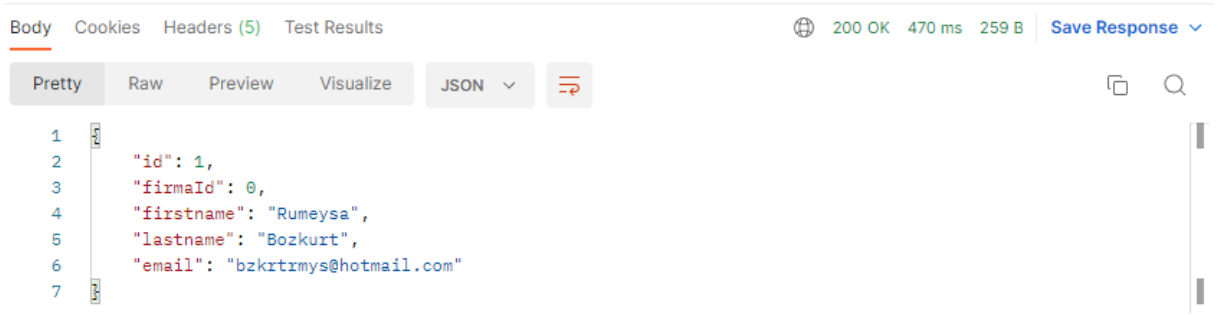
- GET işlemini test edelim. Önceden oluşturduğumuz çalışan bilgilerinin hepsini görüntüleyelim.



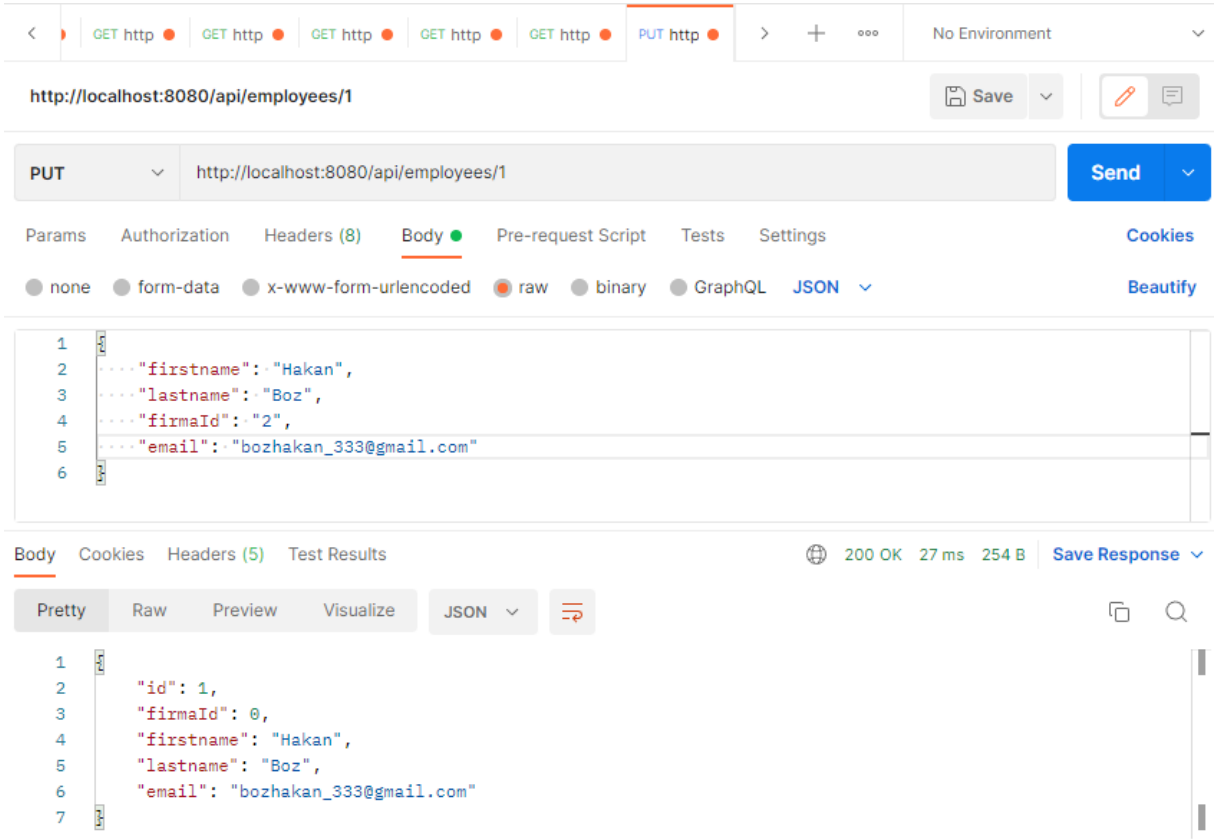


- Belirli bir id'ye göre GET işlemini test edelim. Örneğin kayıtlı verilerden id'si 1 olan kaydı çekmek istediğimizde aldığımız response'a bakalım.

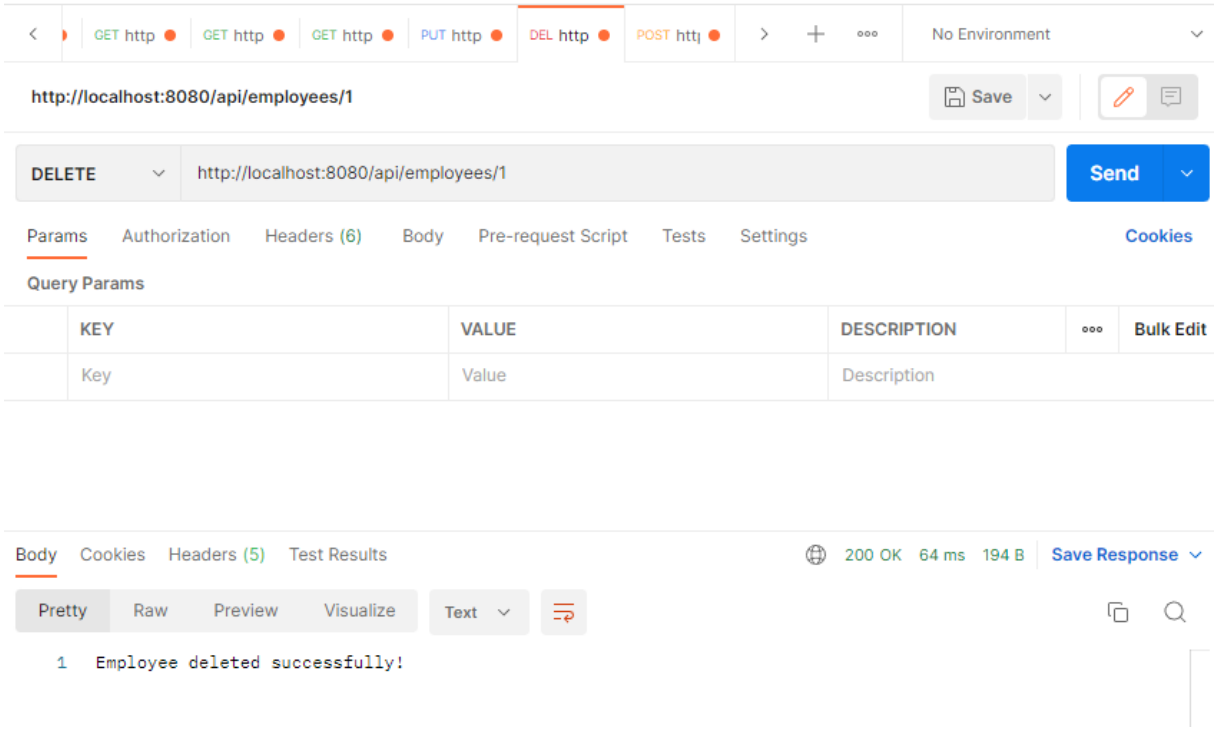




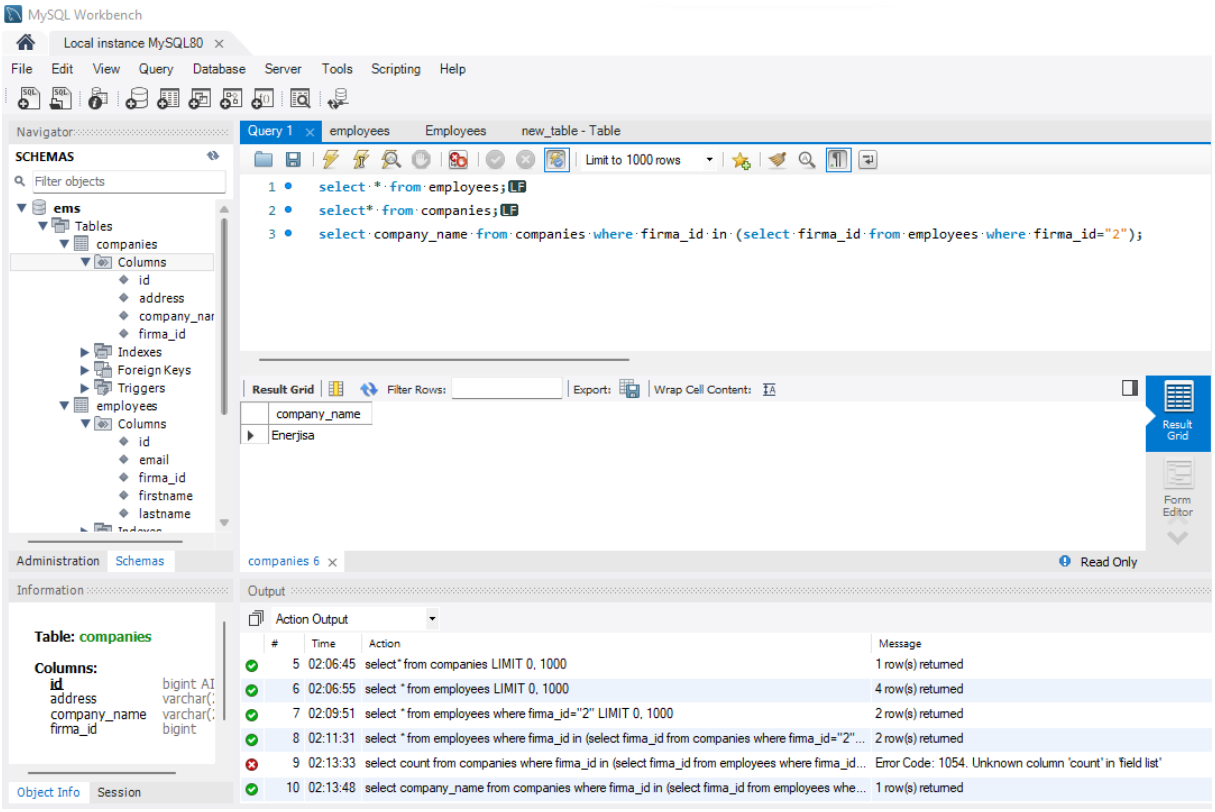
- id'si 1 olan çalışanın email bilgisini PUT metodu ile update edelim. Http metodunu PUT seçip body kısmında gerekli verileri girdikten sonra request oluşturalım.



- Son olarak DELETE işleminin testini yapalım. Employee tablosundan id'si 1 olan kaydı silelim.



Postman dışında MySQL Workbench üzerinden sorgu denemesi yapalım.



Test işlemlerini de tamamladık proje sorunsuz çalışmaktadır.