Spor Event

1. Gizem Zor
Yazılım Mühendisliği
Kocaeli Üniversitesi
210229060

2. Emir Sağlam
Yazılım Mühendisliği
Kocaeli Üniversitesi
220229008

3. Mehmet Vasfi Türkmen
Yazılım Mühendisliği
Kocaeli Üniversitesi
210229032

***Özetçe*—Http Restfull Api , Java Oop prensipleri kullanılarak bir etkinlik planlayıcı uygulaması geliştirilmiştir. Katmanlı mimari yapısı kullanarak projenin daha düzenli ve okunabilir olması sağlanmıştır.**

Anahtar Kelimeler — Http , restful api, katmanlı mimari ,spor event,request,user,activity.

I. GİRİŞ

Spor event etkinlik uygulaması ,Java proglamlama dili ve flutter frameworkü kullanılarak bir uygulama geliştirilmiştir.

A. OOP (Nesne Yönelimli Programlama İlkeleri)

Projemizde class , extends ve implements gibi java oop özelliği olan yapılar kullanılarak nesne yönelimli ilkelere uyulmuştur.Projemizin tamamında metotlar class kullanılarak oluşturulmuştur.

main > java > com > mehmetvasfi > entites > J User.java

```java
import lombok.NoArgsConstructor;
import lombok.Setter;

@Entity
@Table(name = "userss")
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Integer id;

    @NotBlank(message = "First name cannot be blank")
    @Size(max = 50, message = "First name cannot exceed 50 characters")
    @Column(name = "first_name")
    private String firstName;

    @NotBlank(message = "Last name cannot be blank")
    @Size(max = 50, message = "Last name cannot exceed 50 characters")
    @Column(name = "last_name")
    private String lastName;
```

Şekil 1

Şekil 1’ de projemizde kullandığımız bir class yapısının bir ekran görüntüsü mevcuttur.

src > main > java > com > mehmetvasfi > service > J IActivityService.java

```java
package com.mehmetvasfi.service;

import java.util.List;

import com.mehmetvasfi.entites.Activity;
import com.mehmetvasfi.entites.User;

public interface IActivityService {

    public List<Activity> getAllActivity();

    public Activity saveActivity(Activity activity);

    public Activity getActivityById(Integer id);

    public boolean deleteActivity(Integer id);

    public Activity updateActivity(Integer id,Activity activity);

        You, 1 saat önce • Uncommitted changes

}
```

Şekil 2

Şekil 2’de projemizde kullandığımız interface kullanımı mevcuttur.

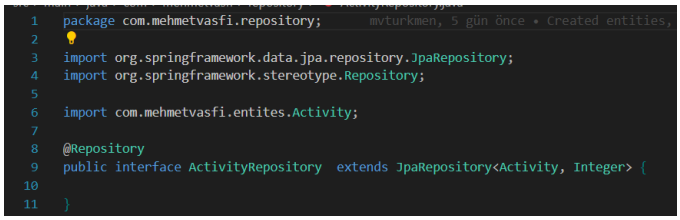
J ActivityServiceImpl.java 1 × J RequestServiceImpl.java 2 J IUserController.java

src > main > java > com > mehmetvasfi > service > impl > J ActivityServiceImpl.java

```java
package com.mehmetvasfi.service.impl;


import java.util.List;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.mehmetvasfi.entites.Activity;

import com.mehmetvasfi.repository.ActivityRepository;
import com.mehmetvasfi.repository.UserRepository;
import com.mehmetvasfi.service.IActivityService;

@Service
public class ActivityServiceImpl implements IActivityService {

    @Autowired
    private ActivityRepository activityRepository;

        You, 53 dakika önce • Uncommitted changes


```

Şekil 3

Şekil 2’de implement özelliğimizin kullanımı mevcuttur.



Şekil 4

Şekil 4'de projemizde kullanılan extends özelliği kullanılmıştır.

B. API Kullanılması

- Uygulamanın back-end kısmında API geliştirilmesi ve kullanılması.

API http protokolü aracılığıyla çalışmaktadır.Projemizde GET,POST,PUT,DELETE gibi http metotları işlenmiştir.

Şekil 5

Şekil 5'de projemizde kullandığımız User classımızla ilgili işlemleri yaptığımız ana API kodumuzun yazıldığı kısım.Bu ana API kodumuzu @RequestMapping anatasyonu kullanarak API işlemi yapıldı.

Şekil 6

Şekil 6'da projemizin veritabanında bulunan User listi @GetMapping anatasyonu kullanarak GET işlemi yapıldı.

Şekil 7

Şekil 7'de projemize @PostMapping anatasyonu kullanılarak POST işlemi ile User kaydı yapılmıştır.

Şekil 8

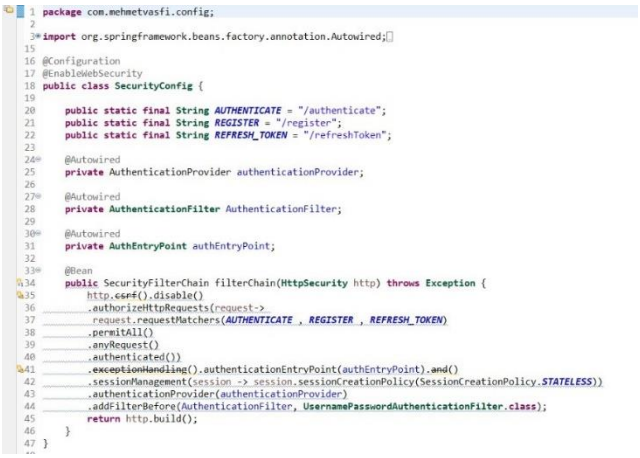
Şekil 8'de projemize @PutMpping anatasyonu kullanılarak PUT işlemi ile User kullanıcımız ile ilgili update işlemi yapılmıştır.

Şekil 9

Şekil 9'da projemizde bulunan User kullanıcısı @DeleteMapping de bulunan DELETE metotu kullnılarak veritabanından silinmiştir.

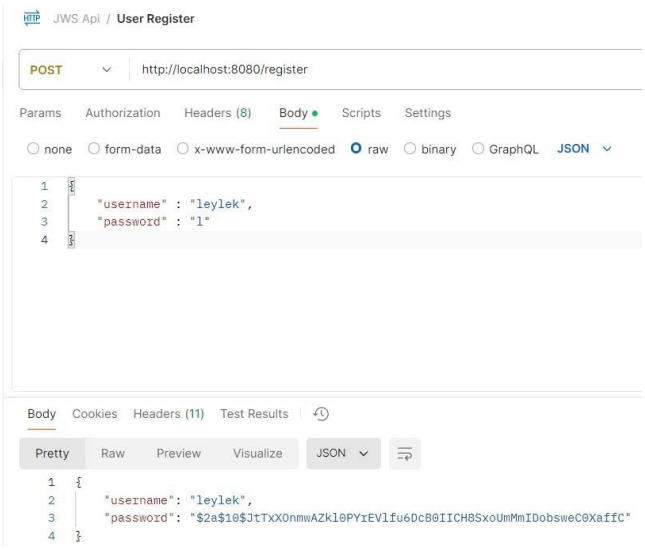
- API erişiminde güvenliğin sağlanması

Kullanıcı authenticate olmamışsa sisteme kayıt olabilir yada giriş yapabilir.Kullanıcı kayıt olarak şifresini sisteme ekleyebilir.Yazdığımız API bu şifreyi veritabanına hashleyerek saklar.Daha sonra logine yönlendirilir .Loginde girilen username ve şifre yine haslenerek karşılaştırılır doğrıı ise kullanıcıya iki saat geçerli bir token verir bu onaylamadan sonra apinin tüm özelliklerine erişebilir.



Şekil 10

Şekil 10'da register,authanticate uzantıları hariç hiçbir istek sisteme erişemiyor.Kullanıcı authenticate olursa API nin diğer servislerine ulaşabiliyor.



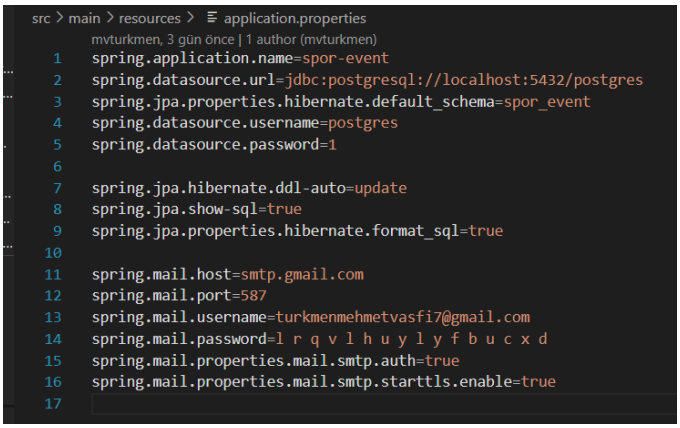
Şekil 11

Şekil 10'da kullanıcı kayıt olarak hashletiyor.

C. Veri Tabanı Entegrasyonu

- Verilerin düzgün ve organize bir şekilde tutulması, gerekli tabloların ve ilişkilerin doğru tanımlanması

Projemizde veritabanı olarak PostgreSql kullanılmıştır.



Şekil 12

Şekil 12'de projemize PostgreSql veritabanını nasıl bağladığımızı gösteren bir ekran görüntüsü vardır.

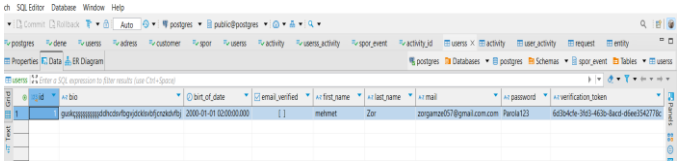
Şekil 13

Şekil 13'de @Entity anatasyonu kullanarak veritabanımızda User classımızın tablosu oluşuyor.@Table anatasyonu ile tablomuzun adını veriyoruz.

```java
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Integer id;

    @NotBlank(message = "First name cannot be blank")
    @Size(max = 50, message = "First name cannot exceed 50 characters")
    @Column(name = "first_name")
    private String firstName;

    @NotBlank(message = "Last name cannot be blank")
    @Size(max = 50, message = "Last name cannot exceed 50 characters")
    @Column(name = "last_name")
    private String lastName;

    @NotNull(message = "Birth date is required")
    @Past(message = "Birth date must be in the past")
    @JsonFormat(pattern = "yyyy-MM-dd")
    @Column(name = "birt_of_date")
    private Date birthOfDate;

    @NotBlank(message = "Email is required")
    @Email(message = "Email must be valid")
    @Column(name = "mail")
    private String mailProperties;

    @NotBlank(message = "Password cannot be blank")
    @Size(min = 8, max = 100, message = "Password must be between 8 and 100 characters")
    @Column(name = "password")
    private String passwordString;

    @Size(max = 250, message = "Bio cannot exceed 250 characters")
    @Column(name = "bio")
    private String bio;

    @Column(name = "email_verified")
    private boolean emailVerified = false;

    @Column(name = "verification_token")
    private String verificationToken;

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL)

    private List<Activity>activity;
```

Şekil 14

Şekil 14'te projemizde bulunan User classının fotoğrafı vardır. @Column anatasyonu kullanarak veritabanında oluşacak sütunun adı verilir.

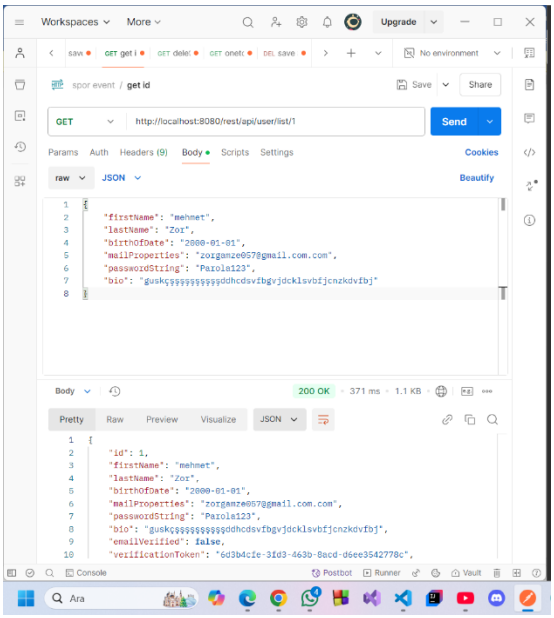


Şekil 15

Şekil 15'de projemizin veritabanında bulunan User bilgisinin bulunduğu bir fotoğraftır.

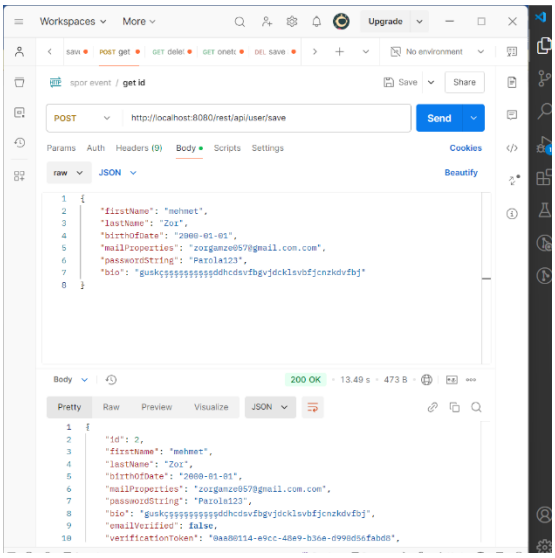
- Veriler üzerinden Create (Oluştur), Read (Okuma), Update (Güncelleme), Delete (Silme) (CRUD) işlemlerinin eksiksiz yapılabilmesi

Postman uygulaması kullarak Create ,Update,Delete,Read işlemleri yapılmıştır.



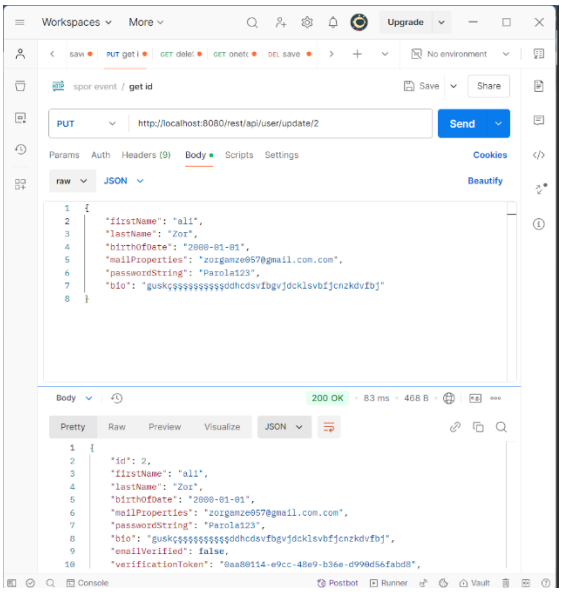
Şekil 16

Şekil 16'da Postman uygulaması ile GET metodu ile Read işlemi ile veritabanındaki ! nolu kullanıcıya ait veriler getirilmiştir.



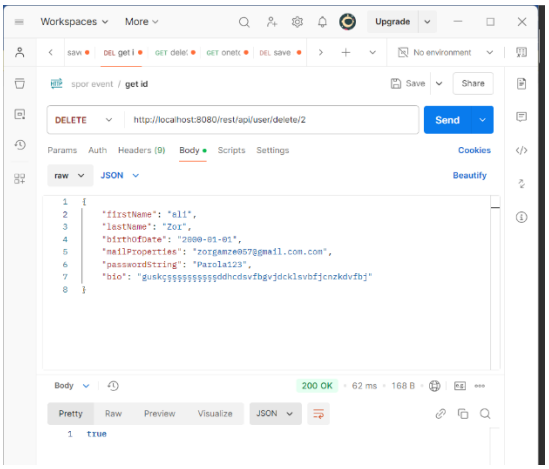
Şekil 17

Şekil 17'de Postman uygulaması ile POST metodu kullanarak Create işlemi yapılmıştır ve o kullanıcı veritabanımız kaydedilmiştir.



Şekil 18

Şekil 18'de Postmanda PUT metodu kullanarak 2 nolu kullanıcının veritabanında update edilmesi sağlanmıştır.



Şekil 19

Şekil 19'da Postmanda DELETE metodu kullanılarak 2 nolu kullanıcının veritabanından silinmesi sağlanmıştır.

D. . Genel Programlama Kalitesi

- Kodun verimli, düzenli, okunabilir olması, anlamlı değişken isimleri kullanılması ve gerekli yerlerde yorum satırlarıyla açıklamaların yapılması

```java
@Override
public boolean deleteActivity(Integer id) {
    Optional<Activity> optional = activityRepository.findById(id);
    if(optional.isPresent()) {
        activityRepository.delete(optional.get());
        return true;
    }
    return false;
}
```

Şekil 20

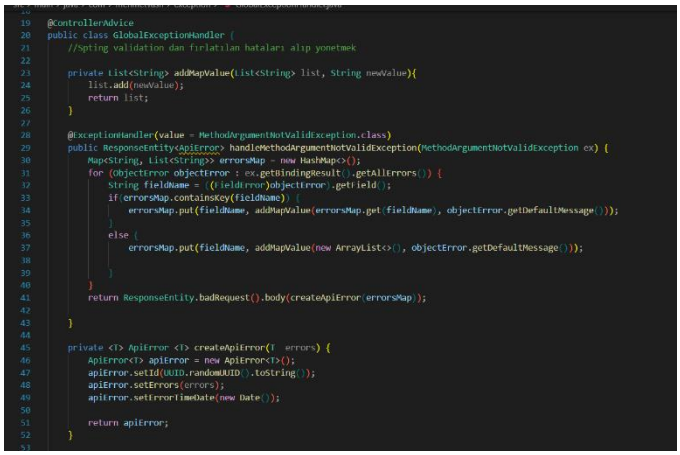
Şekil 20'de projemizde bulunan deleteActivity metodu gösterilmiştir.Burada aktivite silme işlemi yapılmıştır .Projemizin tamamında o metotta hangi işlem yapıldıysa o işlemi belirten bir adı vardır.

- Programda olası hataların yakalanması ve uygun hata yönetimi (exception handling) kullanılması

```java
@Entity
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class Activity {
    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(name = "title")
    @NotEmpty(message = "Title alanı boş bırakılamaz")
    @Size(min = 3, max = 100, message = "Title en az 3 karakter olmalı ve en fazla 100 karakter olmalı.")
    private String title;

    @Column(name = "description")
    @NotEmpty(message = "Description alanı boş bırakılamaz")
    @Size(min = 20, message = "Description en az 20 karakter olmalı.")
    private String description;

    @Column(name = "category")
    @NotEmpty(message = "Category alanı boş bırakılamaz")
    private String category;

    @Column(name = "time_of_activity")
    @NotNull(message = "Activity time boş bırakılamaz")
    private Date timeOfActivity;

    @Column(name = "team_size")
    @NotNull(message = "Team size boş bırakılamaz")
    @Min(value = 1, message = "Team size en az 1 olmalı.")
    private Integer teamSize;
```

Şekil 21

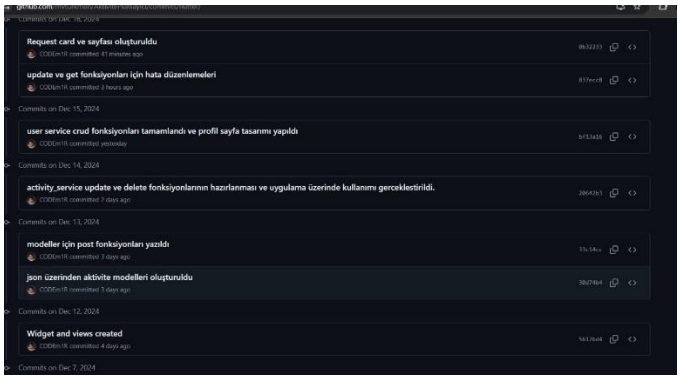
Şekil 21'de validation kullanarak post işlemlerinde bilgiler doldururken bir eksiklik varsa onun dönmesi sağlanmıştır.



Şekil 22

Şekil 22'de exception handling işlemleri yapılmıştır.

- SVN kullanılarak düzenli geliştirme yapılması

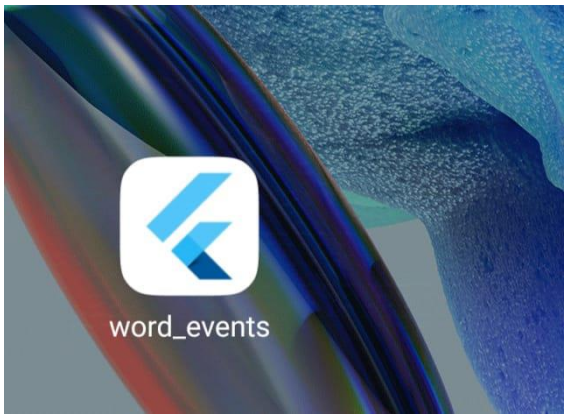


Şekil 23

Şekil 23'te github üzerinde düzenli commitler atarak projenin kontrolü sağlanmıştır.

E. Projenin farklı platformlarda (örneğin: masaüstü, web) çalışabilmesi veya mobil uygulama yapılması

Projemizi mobil uygulama olarak tasarladık.

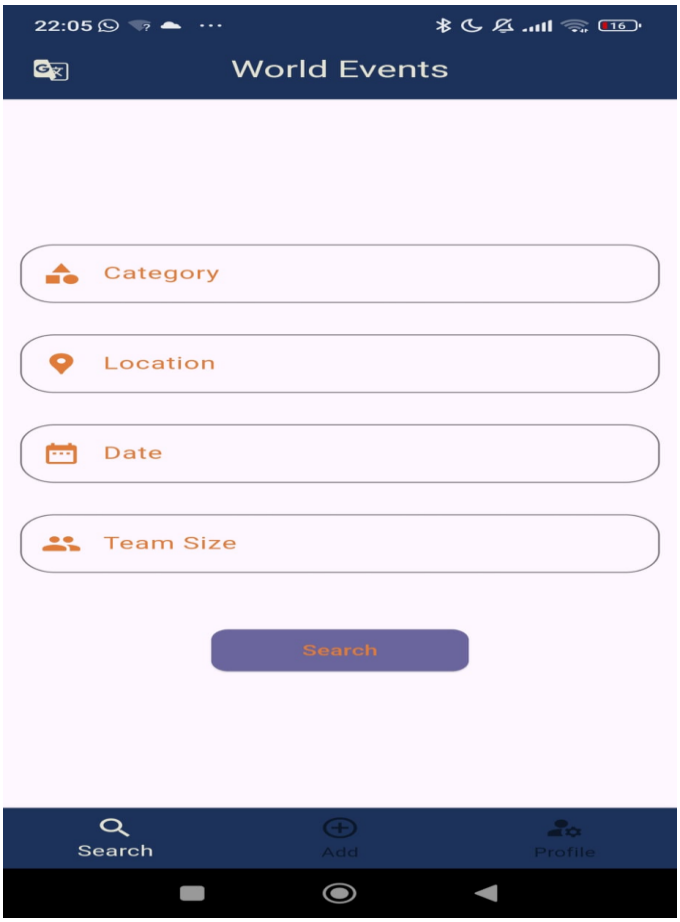


Şekil 24

Şekil 24'de telefonumuzda projemizin mobil uygulama olarak gösterimi mevcut.

F. GUI (Graphical User Interface)

- Tasarımın basit ve anlaşılır olması, kullanıcı deneyimi açısından iyi düşünülmüş kullanıcı dostu bir arayüz



Şekil 25

Şekil 25'te görüldüğü gibi uygulamaız kullanıcı dostu ve kullanabilir bir uygulamadır.

- Arayüzdeki bileşenlerin (butonlar, menüler, form elemanları vb.) doğru çalışması ve projenin barındırdığı tüm fonksiyonlarını yansıtması

```
Future<void> createActivity(BuildContext context) async{
  Activity activity = Activity(
    title: tfTitleController.text,
    description: tfDescriptionController.text,
    category: tfCategoryController.text,
    timeOfActivity: DateTime.parse(tfDateController.text),
    teamSize: int.parse(tfTeamSizeController.text),
  );

  final resp = await activityService.postActivity(activity);

  if(resp.statusCode == HttpStatus.ok){
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(content: Text("New Activity Added !"))
    );
    tfTitleController.clear();
    tfDescriptionController.clear();
    tfCategoryController.clear();
    tfDateController.clear();
    tfTeamSizeController.clear();
  }
  else{
    ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text("Failed to create ."))
    );
  }
}
```

Şekil 26

Şekil 26'da Future void senkronizasyonu sağlar dolayısıyla api dan yanıt gelmedikçe işlem yapılamaz.Böyle veri tutarlılığı sağlanamaz.

```
Widget addUI(BuildContext context) {
  return SingleChildScrollView(
    child: Center(
      child: Column(
        mainAxisAlignment:  MainAxisAlignment.center,
        children: [
          // title
          InputField(tfController: tfTitleController, tfIcon: const Icon(Icons.title), tfLab
          // description
          InputField(tfController: tfDescriptionController, tfIcon: const Icon(Icons.descrip
          // category
          InputField(tfController: tfCategoryController, tfIcon: const Icon(Icons.category),
          // date
          InputField(tfController: tfDateController, tfIcon: const Icon(Icons.calendar_month
          // teamSize
          InputField(tfController: tfTeamSizeController, tfIcon: const Icon(Icons.group_roun

          AppButton(
            label: "ADD",
            function: (){
              // Add activity function
              createActivity(context);
            }
          ) // AppButton
        ],
      ), // Column
    ) // Center
```

Şekil 27

Şekil 27'de butono tıklanıldığında yukarıdaki inputlara göre nesne oluşturulur ve API a istek atılır.

```
// Create (POST)
Future<http.Response> postActivity(Activity activity) async {
  try {
    final response = await http.post(
      Uri.parse(activitiePostUrl),
      headers: <String, String>{
        'Content-Type' : 'application/json; charset=UTF-8',
      },
      body: jsonEncode(
        activity.toJson(),
      ),
    );
    print(response.body);
    return response;
  } catch (error){
    throw Exception("Failed when create a json data ! Error: $error");
  }

}
```

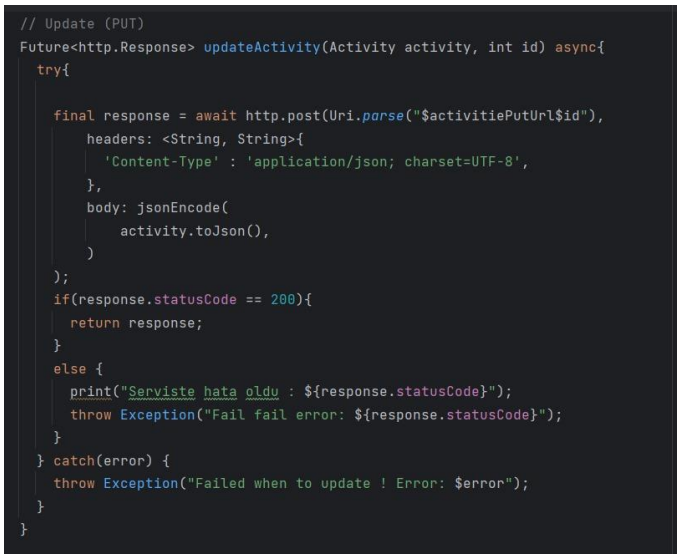
Şekil 28

Şekil 28'de flutter ile HTTP POST isteği ile bir sunucuya veri göndermeyi sağlar .

```
// Read (GET)
Future<List<Activity>> getActivities() async {
  try {
    final response = await http.get(Uri.parse(activitiesGetUrl));

    if(response.statusCode == 200) {
      print(response.body);
      List<dynamic> responseList = json.decode(response.body);

      List<Activity> activities = responseList.map((jsonItem) => Activity.fromJson(jsonIt
      return activities;
    }
    else {
      throw Exception("Failed to connect");
    }
  } catch (error) {
    print('Error: $error');
    throw Exception('Failed to load activities ! Error: $error');
  }

}
```

Şekil 29

Şekil 29'da URL üzerinden GET isteği atarak JSON formatındaki aktiviteleri alır ve Activity nesneleri listesine dönüştürür.



```
// Update (PUT)
Future<http.Response> updateActivity(Activity activity, int id) async{
  try{

    final response = await http.post(Uri.parse("$activitiePutUrl$id"),
        headers: <String, String>{
          'Content-Type' : 'application/json; charset=UTF-8',
        },
        body: jsonEncode(
            activity.toJson(),
        )
    );
    if(response.statusCode == 200){
      return response;
    }
    else {
      print("Serviste hata oldu : ${response.statusCode}");
      throw Exception("Fail fail error: ${response.statusCode}");
    }
  } catch(error) {
    throw Exception("Failed when to update ! Error: $error");
  }
}
```

Şekil 30

Şekil 30'da flutter uygulaması ile bir PUT isteği atarak sunucudaki bir aktiviteyi güncellemek içi kullanılır.

```
// Delete
Future<http.Response> deleteActivity(int id) async{
  try {
    final response = http.delete(Uri.parse("$activitieDeleteUrl$id"));
    return response;
  } catch(error) {
    throw Exception("Failed when delete json data ! Error: $error");
  }
}
```

Şekil 31

Şekil 31 'de flutter uygulaması ile DELETE isteği atarak sunucudaki bir aktiviteyi güncellemek için kullanılır.

KAYNAKLAR

[1] https://medium.com/@sureyyakaanperktas/spring-boot-e5ae1c8a1298

[2] https://medium.com/deviniti-technology-driven-blog/implementing-multitenancy-architecture-spring-boot-jpa-hibernate-flyway-8fb19b312a10

[3] https://medium.com/towardsdev/spring-boot-validations-71970f3bb1ef

[4] https://medium.com/@AlexanderObregon/integration-of-spring-boot-security-with-spring-data-jpa-8a1ed11200ce