

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
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
1 package com.mehmetvasfi.service;
2
3 import java.util.List;
4
5 import com.mehmetvasfi.entites.Activity;
6 import com.mehmetvasfi.entites.User;
7
8 public interface IActivityService {
9
10     public List<Activity> getAllActivity();
11
12     public Activity saveActivity(Activity activity);
13
14     public Activity getActivityById(Integer id);
15
16     public boolean deleteActivity(Integer id);
17
18     public Activity updateActivity(Integer id,Activity activity);
19
20     You, 1 saat önce • Uncommitted changes
21
22 }
```

Şekil 2

Şekil 2'de projemizde kullandığımız interface kullanımı mevcuttur.

```
J ActivityServiceImpl.java 1 X J RequestServiceImpl.java 2 J IUserController.java
src > main > java > com > mehmetvasfi > service > impl > J ActivityServiceImpl.java
1 package com.mehmetvasfi.service.impl;
2
3
4 import java.util.List;
5 import java.util.Optional;
6
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.stereotype.Service;
9 import com.mehmetvasfi.entites.Activity;
10
11 import com.mehmetvasfi.repository.ActivityRepository;
12 import com.mehmetvasfi.repository.UserRepository;
13 import com.mehmetvasfi.service.IActivityService;
14
15 @Service
16 public class ActivityServiceImpl implements IActivityService {
17
18     @Autowired
19     private ActivityRepository activityRepository;
20
21     You, 53 dakika önce • Uncommitted changes
22
23 }
```

Şekil 3

Şekil 2'de implement özelliğimizin kullanımı mevcuttur.

```

1 package com.mehmetvasfi.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4 import org.springframework.stereotype.Repository;
5
6 import com.mehmetvasfi.entites.Activity;
7
8 @Repository
9 public interface ActivityRepository extends JpaRepository<Activity, Integer> {
10
11 }

```

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.

```

@RestController
@RequestMapping("/rest/api/user")
public class UserControllerImpl implements IUserController

```

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.

```

@Override
@GetMapping(path="/list")
public List<User> getAllUser() {

    return userServices.getAllUser();
}

```

Şekil 6

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

```

@PostMapping(path = "/save")
@Override
public User saveUser(@RequestBody @Valid User user){
    return userServices.saveUser(user);
}

```

Şekil 7

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

```

@PutMapping(path="/update/{id}")
@Override
public User updateUser(@PathVariable(name="id") Integer id,@RequestBody User updateUser){
    return userServices.updateUser(id, updateUser);
}

```

Şekil 8

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

```

@DeleteMapping(path = "/delete/{id}")
@Override
public boolean deleteUser(@PathVariable(name = "id") Integer id){

    return userServices.deleteUser(id);
}

```

Şekil 9

Şekil 9’da projemizde bulunan User kullanıcısı @DeleteMapping de bulunan DELETE metodu 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 doğru ise kullanıcıya iki saat geçerli bir token verir bu onaylamadan sonra apinin tüm özelliklerine erişebilir.

```

1 package com.nehmetvasfi.config;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4
5 @Configuration
6 @EnableWebSecurity
7 public class SecurityConfig {
8
9     public static final String AUTHENTICATE = "/authenticate";
10    public static final String REGISTER = "/register";
11    public static final String REFRESH_TOKEN = "/refreshToken";
12
13    @Autowired
14    private AuthenticationProvider authenticationProvider;
15
16    @Autowired
17    private AuthenticationFilter authenticationFilter;
18
19    @Autowired
20    private AuthEntryPoint authEntryPoint;
21
22    @Bean
23    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
24        http.csrf().disable()
25            .authorizeHttpRequests(request -> {
26                request.requestMatchers(AUTHENTICATE, REGISTER, REFRESH_TOKEN)
27                    .permitAll()
28                .anyRequest()
29                    .authenticated()
30            })
31            .exceptionHandling().authenticationEntryPoint(authEntryPoint).and()
32            .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
33            .authenticationProvider(authenticationProvider)
34            .addFilterBefore(authenticationFilter, UsernamePasswordAuthenticationFilter.class);
35        return http.build();
36    }
37 }

```

Şekil 10

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

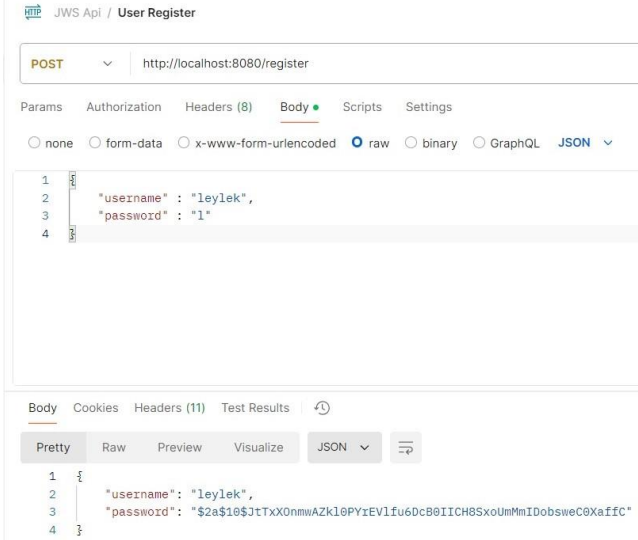
```

src > main > resources > application.properties
myturkmen, 3 gün önce | 1 author (myturkmen)
1 spring.application.name=spor-event
2 spring.datasource.url=jdbc:postgresql://localhost:5432/postgres
3 spring.jpa.properties.hibernate.default_schema=spor_event
4 spring.datasource.username=postgres
5 spring.datasource.password=1
6
7 spring.jpa.hibernate.ddl-auto=update
8 spring.jpa.show-sql=true
9 spring.jpa.properties.hibernate.format_sql=true
10
11 spring.mail.host=smtp.gmail.com
12 spring.mail.port=587
13 spring.mail.username=turkmenmehmetvasfi7@gmail.com
14 spring.mail.password=l r q v l h u y l y f b u c x d
15 spring.mail.properties.mail.smtp.auth=true
16 spring.mail.properties.mail.smtp.starttls.enable=true
17

```

Şekil 12

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



Şekil 11

Şekil 10'da kullanıcı kayıt olarak hashletiyor.

```

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

```

Şekil 13

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

C. Veri Tabanı Entegrasyonu

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

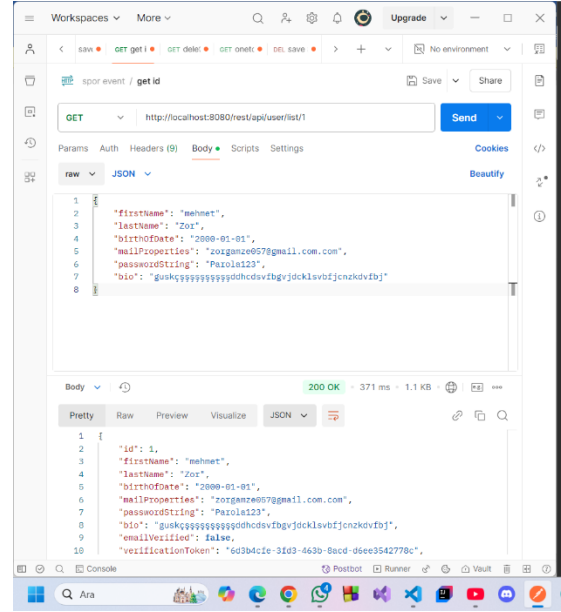
Projemizde veritabanı olarak PostgreSQL kullanılmıştır.

```

86 public class User {
87
88     @Id
89     @GeneratedValue(strategy = GenerationType.IDENTITY)
90     @Column(name = "id")
91     private Integer id;
92
93     @NotBlank(message = "First name cannot be blank")
94     @Size(max = 50, message = "First name cannot exceed 50 characters")
95     @Column(name = "first_name")
96     private String firstName;
97
98     @NotBlank(message = "Last name cannot be blank")
99     @Size(max = 50, message = "Last name cannot exceed 50 characters")
100    @Column(name = "last_name")
101    private String lastName;
102
103    @NotNull(message = "Birth date is required")
104    @Past(message = "Birth date must be in the past")
105    @JsonFormat(pattern = "yyyy-MM-dd")
106    @Column(name = "birth_of_date")
107    private Date birthOfDate;
108
109    @NotBlank(message = "Email is required")
110    @Email(message = "Email must be valid")
111    @Column(name = "mail")
112    private String mailProperties;
113
114    @NotBlank(message = "Password cannot be blank")
115    @Size(min = 8, max = 100, message = "Password must be between 8 and 100 characters")
116    @Column(name = "password")
117    private String passwordString;
118
119    @Size(max = 250, message = "Bio cannot exceed 250 characters")
120    @Column(name = "bio")
121    private String bio;
122
123    @Column(name = "email_verified")
124    private boolean emailVerified = false;
125
126    @Column(name = "verification_token")
127    private String verificationToken;
128
129    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL)
130    private List<Activity> activity;
131
132

```

Şekil 14



Şekil 16

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

Şekil 14’te projemizde bulunan User classının fotoğrafı [vardır](#). [@Column](#) anotasyonu kullanarak veritabanında oluşacak sütunun adı verilir.

The screenshot shows a table named 'users' with the following columns: id, birth_of_date, email_verified, first_name, last_name, mail, password, and verification_token. The table contains one row of data for a user with id 1.

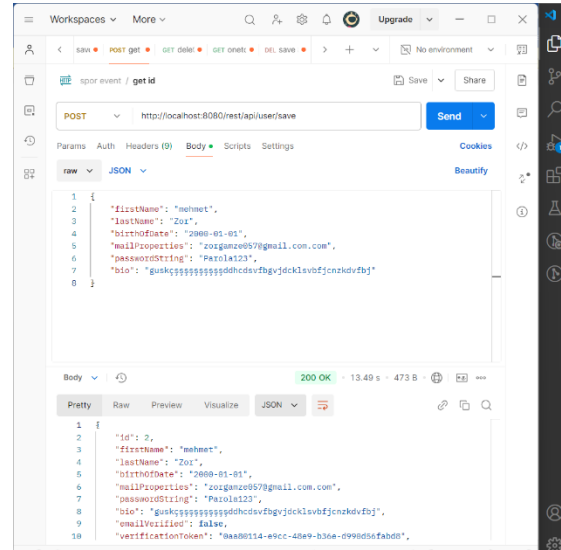
id	birth_of_date	email_verified	first_name	last_name	mail	password	verification_token
1	2000-01-01 00:00:00	0	mehmet	Zor	zoiganze097@gmail.com	Pazola123	6d3b4cfe-3f03-463b-8ecd-d6ee3542778c

Şekil 15

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

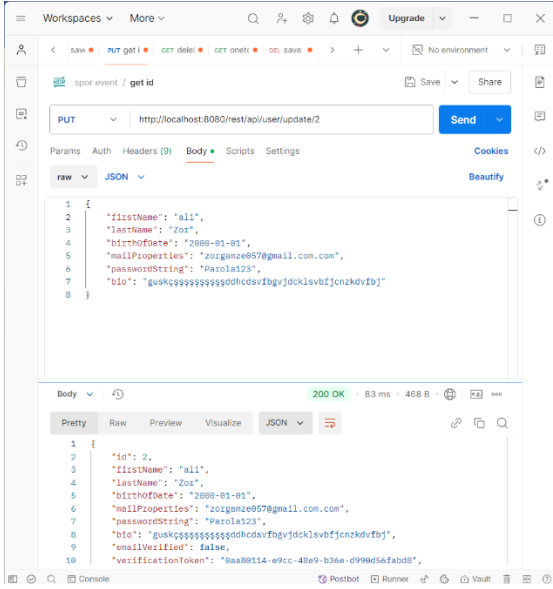
- Veriler üzerinden Create (Oluştur), Read (Okuma), Update (Güncelleme), Delete (Silme) (CRUD) işlemlerinin eksiksiz yapılabilmesi

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



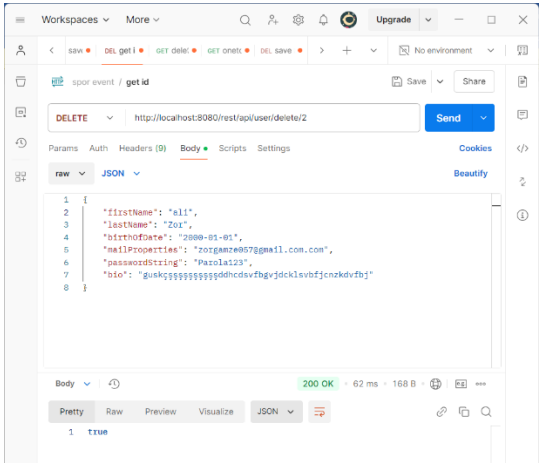
Şekil 17

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

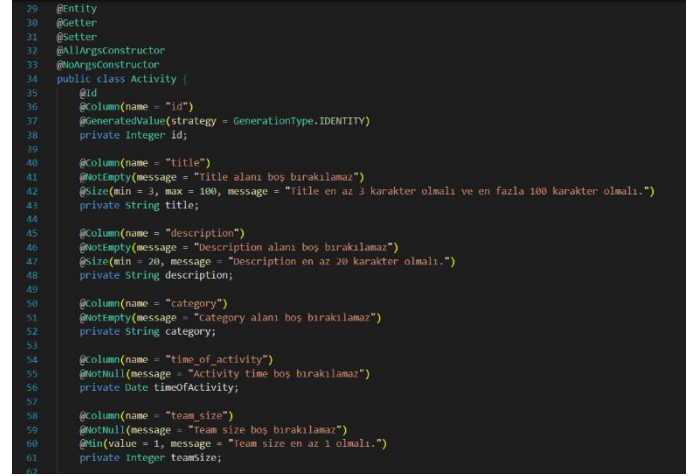
- Kodun verimli, düzenli, okunabilir olması, anlamlı değişken isimleri kullanılması ve gerekli yerlerde yorum satırlarıyla açıklamaların yapılması



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



Şekil 21

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

```

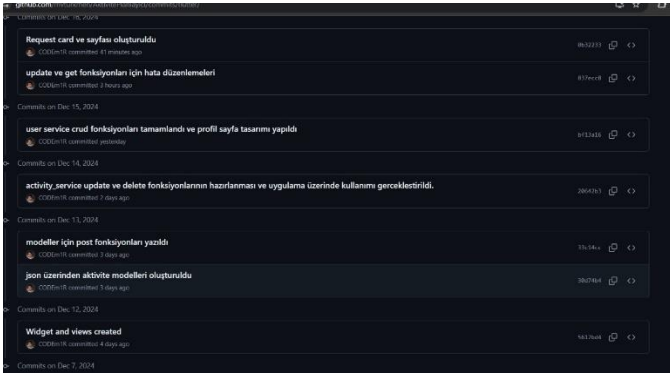
19 @ControllerAdvice
20 public class GlobalExceptionHandler {
21     //Spring validation dan fırlatılan hataları alıp yönetmek
22
23     private List<String> addUpValue(List<String> list, String newValue){
24         list.add(newValue);
25         return list;
26     }
27
28     @ExceptionHandler({MethodArgumentNotValidException.class})
29     public ResponseEntity<ApiError> handleMethodArgumentNotValidException(MethodArgumentNotValidException ex) {
30         Map<String, List<String>> errorsMap = new HashMap<>();
31         for (ObjectError objectError : ex.getBindingResult().getAllErrors()) {
32             String fieldName = ((FieldError) objectError).getField();
33             if (errorsMap.containsKey(fieldName)) {
34                 errorsMap.put(fieldName, addUpValue(errorsMap.get(fieldName), objectError.getDefaultMessage()));
35             }
36             else {
37                 errorsMap.put(fieldName, addUpValue(new ArrayList<>(), objectError.getDefaultMessage()));
38             }
39         }
40         return ResponseEntity.badRequest().body(createApiError(errorsMap));
41     }
42
43     private <T> ApiError<T> createApiError(T errors) {
44         ApiError<T> apiError = new ApiError<>();
45         apiError.setUuid(UUID.randomUUID().toString());
46         apiError.setErrors(errors);
47         apiError.setErrorTimestamp(new Date());
48         return apiError;
49     }
50 }

```

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.

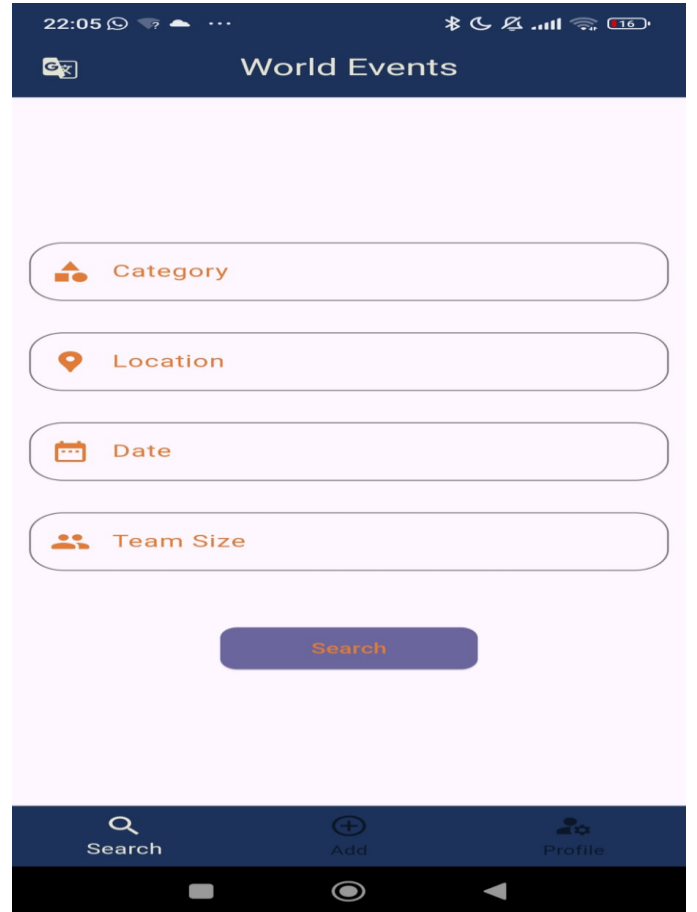


Şekil 24

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 uygulamamız kullanıcı dostu ve kullanılabilir bir uygulamadır.

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

Şekil 27

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

```
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), tfLabel: // description
          InputField(tfController: tfDescriptionController, tfIcon: const Icon(Icons.description), tfLabel: // category
          InputField(tfController: tfCategoryController, tfIcon: const Icon(Icons.category), tfLabel: // date
          InputField(tfController: tfDateController, tfIcon: const Icon(Icons.calendar_month), tfLabel: // teamSize
          InputField(tfController: tfTeamSizeController, tfIcon: const Icon(Icons.group_round), tfLabel:
          AppButton(
            label: "ADD",
            function: (){
              // Add activity function
              createActivity(context);
            }
          ) // AppButton
        ],
      ), // Column
    ), // Center
  );
```

```
// 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(jsonItem));
      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 için 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>