

Final Projesi: Tepe Market “Market Stok Takip Sistemi” Web Servis (API)

Tepe Market müşterimiz için geliştirilen Market Stok Takip Sistemi yazılımının Web Servis Kodlama projesinin bahsi geçen raporudur. Proje'de Spring Boot teknolojisi, Java programlama dili, Maven proje yönetimi gibi teknolojiler kullanılmış olup sektör standarlarında kaliteye sahip bir web servis geliştirilmiştir.

Web Servis Projesi içerisinde Sisteme Ürün Ekleme, Sistemdeki Ürünleri Listelemek, Sistemden Seri Numarasına Göre Ürün Bilgilerini Görmek, Sistemden Ürün silmek gibi temel işlemler yapılmaktadır

Proje Adımları ve Proje İçeriği

Spring Boot Proje Oluşturmak

<https://start.spring.io/> sitesinden oluşturacağımız projenin bilgilerini girip “GENERATE” fonksiyonu ile projemizi .zip formatında olusturulup indirme baslatılıyor. Proje kullanılan IDE nin proje dosyaları içerisinde çıkartılır ve IDE içerisinde proje çalıştırılıp default projeye erişim sağlanıyor.

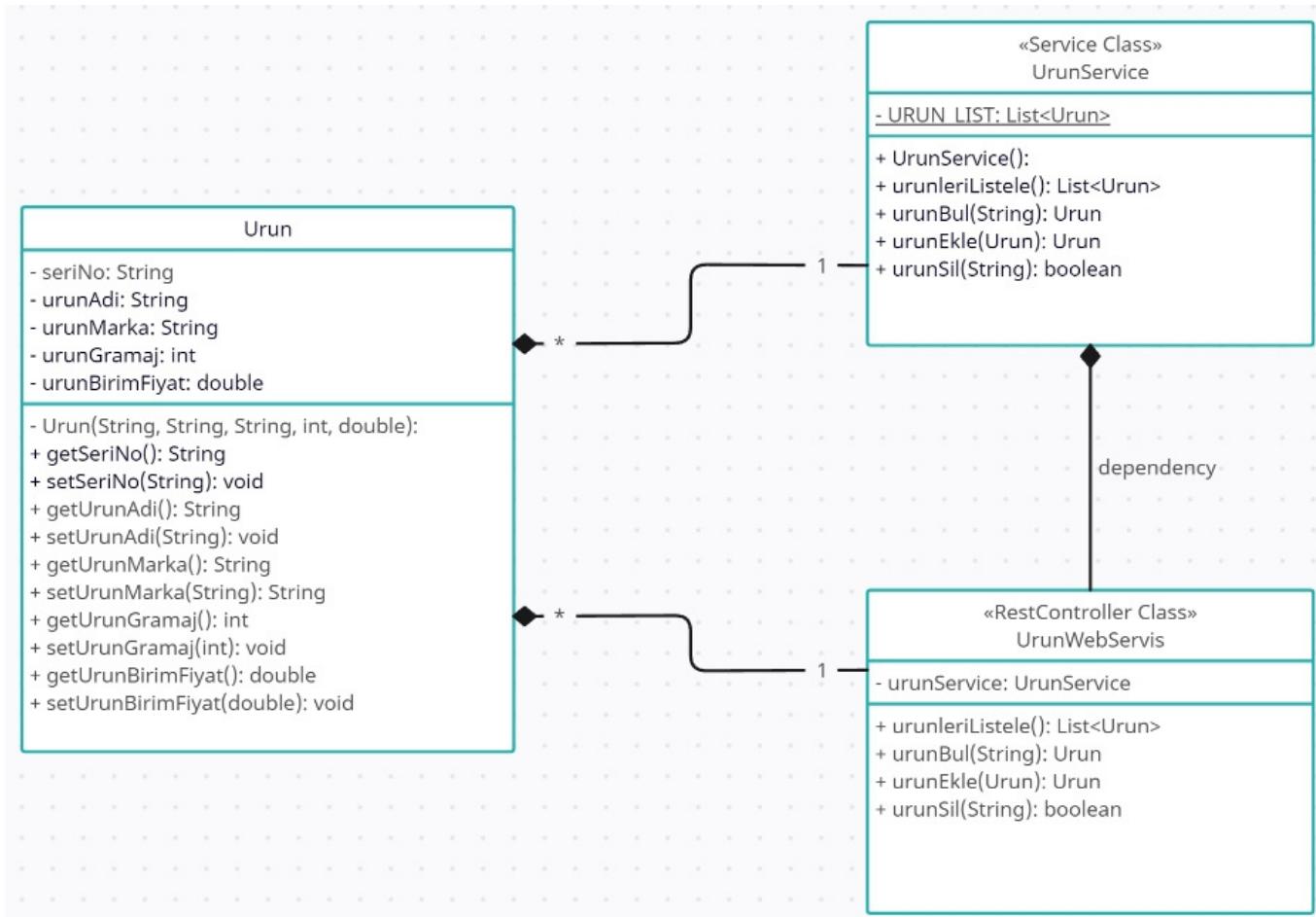
The screenshot shows the Spring Initializr interface for creating a new Spring Boot project. The configuration is as follows:

- Project:** Maven (selected)
- Language:** Java (selected)
- Spring Boot:** 3.2.1 (selected)
- Dependencies:** Spring Web (selected)
- Project Metadata:**
 - Group: stoktakip.market.tepe
 - Artifact: StokTakipRestApi
 - Name: StokTakipRestApi
 - Description: REST API of "Stok Takip Sistemi" Software
 - Package name: stoktakip.market.tepe.restapi
 - Packaging: Jar (selected)
 - Java: 21 (selected)
- Buttons at the bottom:** GENERATE (CTRL + F), EXPLORE (CTRL + SPACE), SHARE...

Proje kemik hali ile spring boot ile API geliştirmek için gereken kutupanelerin pom.xml içerisinde dependency olarak eklenmiş halinde, kodlamaya hazır bi şekilde default olarak aciliyor. Bu projede biz Tepe Market Stok Takip Sistemi yazılımımızın web servisini yazmak istediğimiz için ek sınıflar oluşturup bu sınıflar içerişine kodlama yaptık

Proje UML Sınıf Diyagramları ve Use Case Diyagramlar Oluşturulması

"Market Stok Takip Sistemi" yazılımının tasarımları için oluşturulan UML Sınıf Diyagramı, projede yer alacak sınıfları, bu sınıfların içinde bulunan değişkenleri ve değişken tiplerini, methodları, ayrıca erişim özelliklerini belirtmektedir. Ayrıca, bu sınıflar arasındaki iletişim net bir şekilde gösteren bir yapı sunulmuştur.

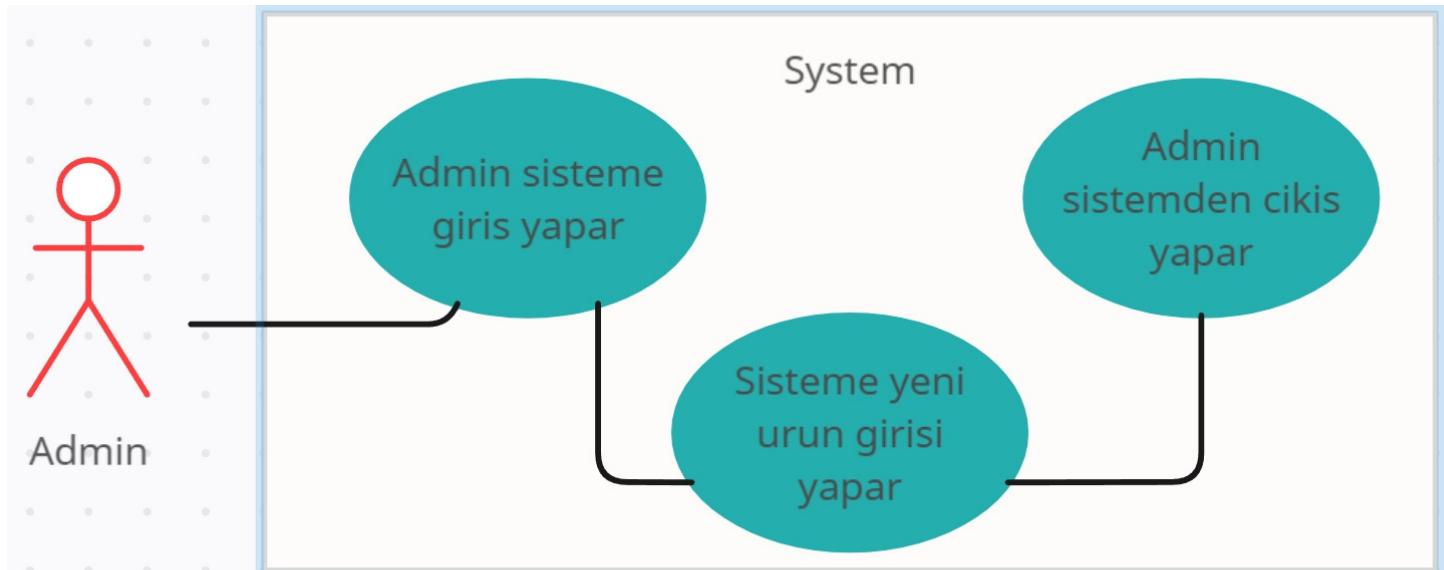


Yukarıdaki UML sınıf diyagramında Sınıflar, sınıfların içerisindeki method ve değişkenler gibi bilgiler yer almaktadır. Urun sınıfı her iki sınıfı da kullanılan, nesne oluşturuluran ve bu sınıfların ihtiyaç duyduğu bir classtır dolayısıyla bir dependency söz konusudur. Bir ürüne birden fazla servis uygulanabilir, bir servis de birden fazla ürüne uygulanabilir dolayısıyla çoktan çoka ilişki bulunmaktadır

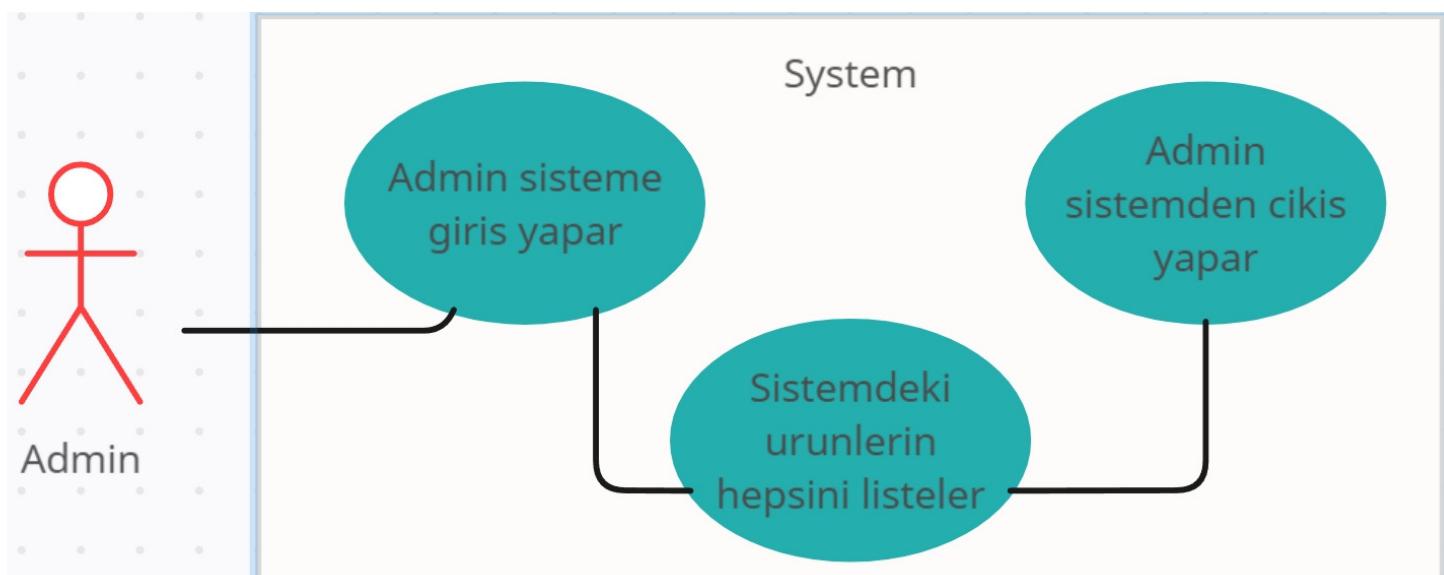
Use Case (Kullanım Senaryosu) Diyagramları

Kullanım senaryosu diyagramları, ürünün kullanımı halinde oluşacak senaryolar, ürünün karşılaşması beklenen gereklilikleri içeren adı üstünde kullanım senaryolarını gösteren diyagamlardır. Bu diyagamlar, ürünün gerçek hayatı nasıl bir senaryoda bir probleme çözüm olacağını somut olarak gösterir ve uygulamanın tasarımına olumlu etkisi çok büyütür

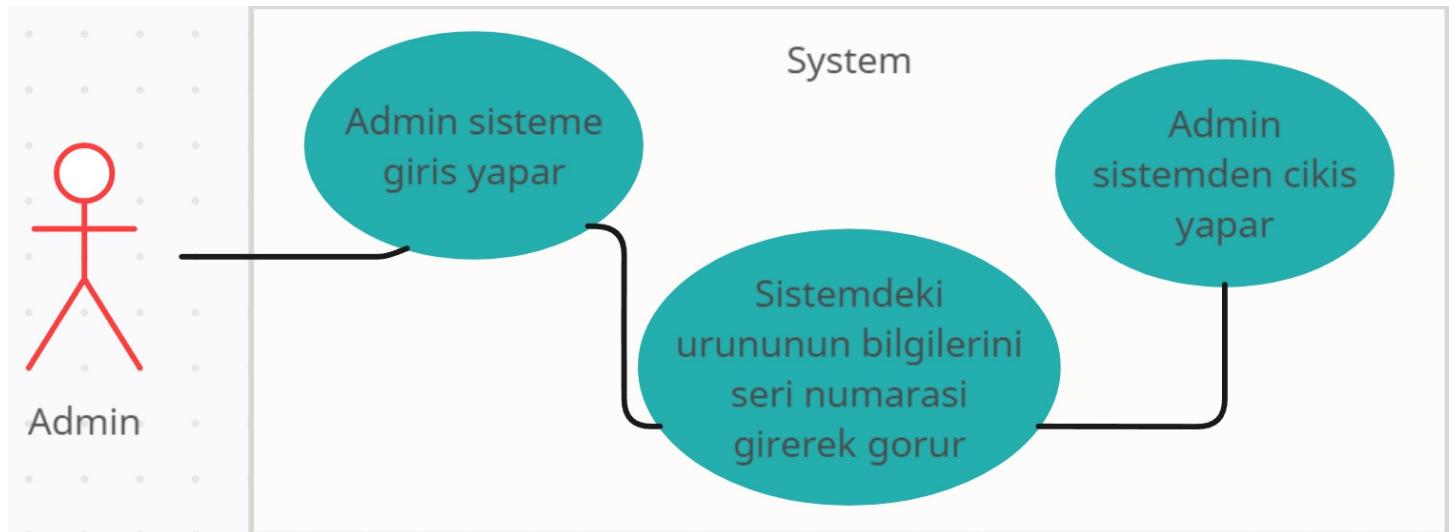
Kullanım Senaryosu 1: Sisteme Ürün Giriş Yapmak



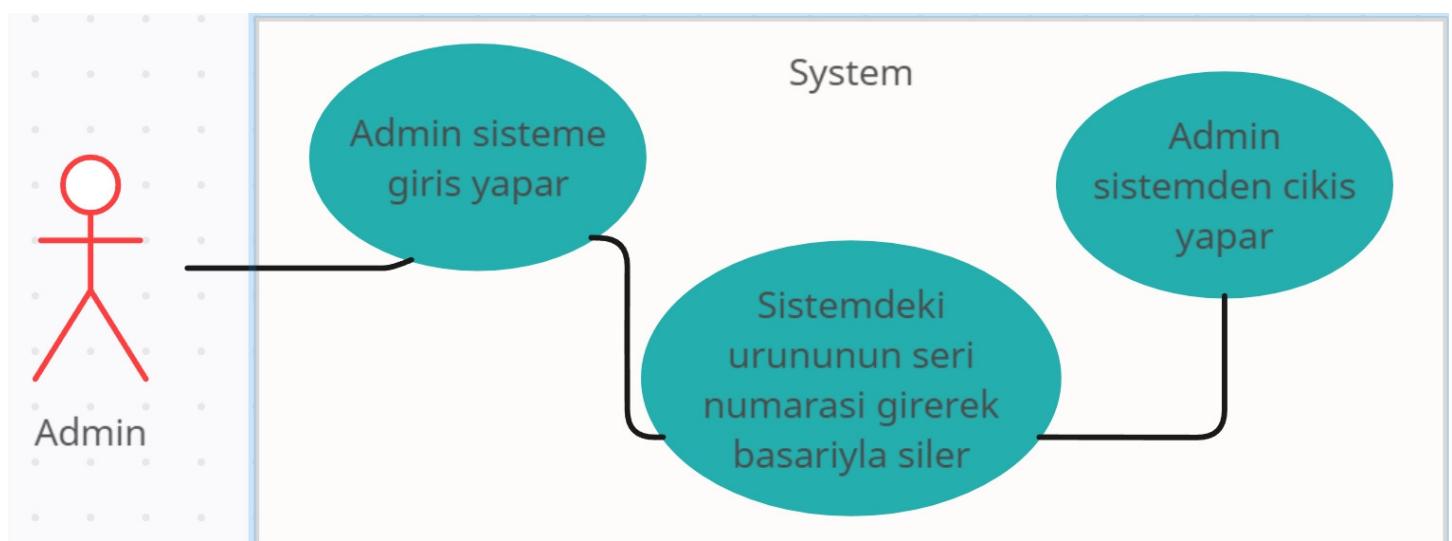
Kullanım Senaryosu 2: Sistemdeki Ürünlerin Hepsini Listelemek



Kullanım Senaryosu 1: Sisteme Ürün Giriş Yapmak

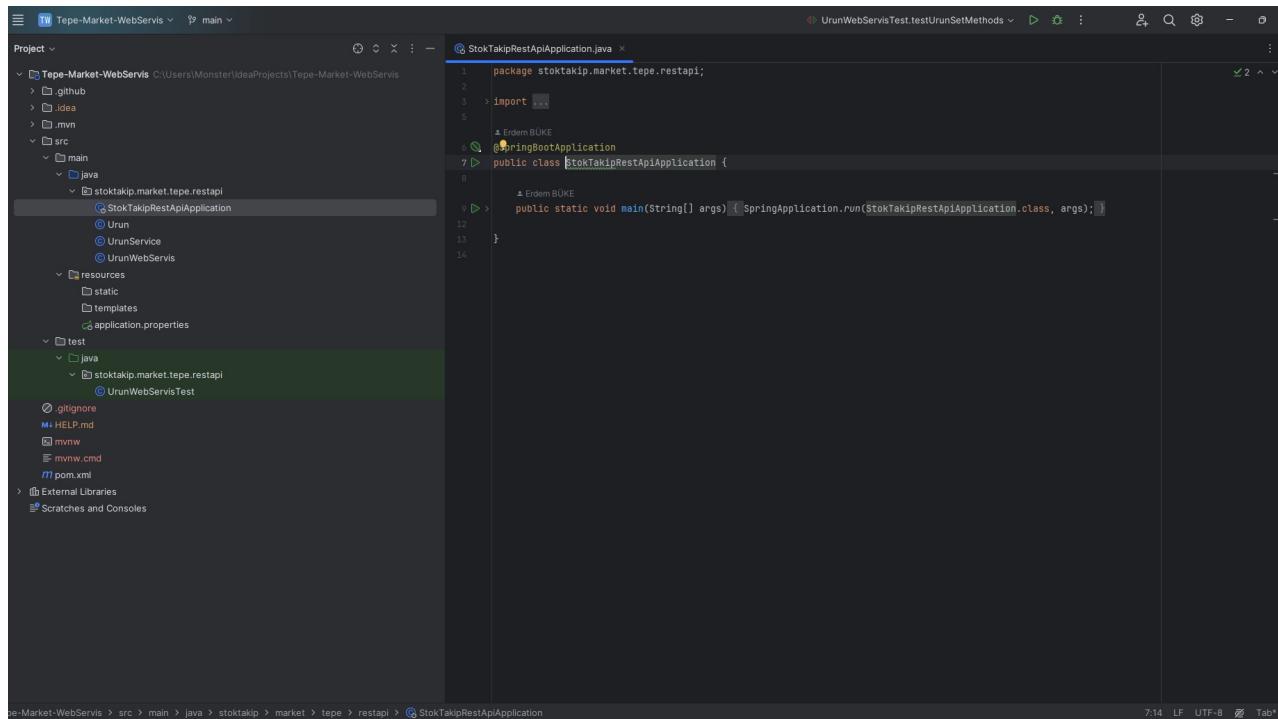


Kullanım Senaryosu 2: Sistemdeki Ürünlerin Hepsini Listelemek



Web Servisler İçin Gerekli Sınıfları Oluşturmak

Proje okunaklığını artırmak ve unit test yazmayı kolaylaştırmak için kullanılan sınıflar, nesne oluşturacağımız Urun sınıfı, bir **@Service** sınıfı olan, Web Servis metodlarının yazıldığı UrunService sınıfı ve UrunService sınıfındaki metodları kullanarak **@RestController** annotation u altında web servislerin çalıştığı ana sınıf UrunWebServis sınıfı. Bu sınıflar sayesinde hem daha bir clean code yazılmış oldu ve Mockito kütüphanesi ile unit testlerin yazılma kolaylığı artırıldı. Web Servis 8082 portunda çalışmaktadır



a) Proje Kaynak Kodlarının Yazılması ve Kullanılan Sınıfların Açıklanması

Bu aşamada proje kodlanması başlandı. **Urun sınıfı** UrunService içerisinde webservis metodlarını yazarken, nesne oluşturarak, oluşturduğumuz nesneleri mock veri olarak veritabanına statik olarak eklemek gibi işlemlerde kullanıldı. Aynı zamanda unit test yazarken de kullanıldı. **Sınıf içerisindeki kod ve metodlara** yakından bakmak gereklidir;

```
30 usages ▲ Erdem BÜKE
public class Urun {
    // Attributes
    4 usages
    private String seriNo;
    4 usages
    private String urunAdi;
    4 usages
    private String urunMarka;
    4 usages
    private int urunGramaj;
    4 usages
    private double urunBirimFiyat;

    // Constructor
    10 usages ▲ Erdem BÜKE
    public Urun(String seriNo, String urunAdi, String urunMarka, int urunGramaj, double urunBirimFiyat) {
        this.seriNo = seriNo;
        this.urunAdi = urunAdi;
        this.urunMarka = urunMarka;
        this.urunGramaj = urunGramaj;
        this.urunBirimFiyat = urunBirimFiyat;
    }

    // Getter - Setters
}
```

The screenshot shows the code editor with the 'Urun' class definition. The code is annotated with 'usages' and 'Erden BÜKE' comments. It includes private attributes for 'seriNo', 'urunAdi', 'urunMarka', 'urunGramaj', and 'urunBirimFiyat'. A constructor is defined to initialize these attributes. Below the class definition, a note indicates that getters and setters are generated.

Urun sınıfı içerisinde urun seri numarası, urun adı, urun markası, urun gramajı, urun birim fiyatı gibi veritabanında tutulacak olan bilgileri bulunmaktadır. Devamında ise sınıfın Constructor methodu ve getter - setter methodları ile birlikte toString() method'u bulunmaktadır.

UrunService sınıfı içerisinde web servislerin işlemlerinin kodları, methodlarını barındırır. bu sınıf @Service annotation una sahiptir. İçerisinde veritabanındaki ürünlerin hepsini listeleyen urunleriListele(), Ürün seri numarasına göre ürün getiren urunBul(), Sisteme ürün ekleyen urunEkle(), sistemden seri numarasına göre ürün silme işlemi gerçekleştiren urunSil() methodları bulunur.

```

1 package stoktakip.market.tepe.restapi;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.web.bind.annotation.*;
5 import java.util.List;
6
7 // server.port = 8082
8
9 @RestController
10 @RequestMapping("/v1/urun")
11 public class UrunService {
12
13     @Autowired
14     private UrunService urunService;
15
16     @GetMapping("/{id}")
17     public List<Urun> urunleriListele() {
18         return urunService.urunleriListele();
19     }
20
21     @GetMapping("/{id}/{seriNo}")
22     public Urun urunBul(@PathVariable String seriNo) {
23         return urunService.urunBul(seriNo);
24     }
25
26     @PostMapping("/")
27     public Urun urunEkle(@RequestBody Urun urun) {
28         return urunService.urunEkle(urun);
29     }
30
31     @DeleteMapping("/{id}/{seriNo}")
32     public boolean urunSil(@PathVariable String seriNo) {
33         return urunService.urunSil(seriNo);
34     }
35 }

```

Sınıf içerisinde tanımlı, aynı zamanda veritabanı olarak kullanılan ve içerisinde Urun tipinde nesne barındıran bir ArrayList tanımlıdır. Bu sınıf constructor methodunda ise veritabanına statik olarak mock veri girişi yapılmıştır. Projede API test kontrolü ve unit test yazımında mock veriler büyük fayda sağlamıştır

UrunWebServis @RestController sınıfı içerisinde çağrıların yapıldığı methodları barındıran sınıfır. Bu sınıfın UrunService turundan bir nesne oluşturulup @Autowired ile sınıfı bağıllılık kazanmıştır.

```

1 package stoktakip.market.tepe.restapi;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.web.bind.annotation.*;
5 import java.util.List;
6
7 // server.port = 8082
8
9 @RestController
10 @RequestMapping("/v1/urun")
11 public class UrunWebServis {
12
13     @Autowired
14     private UrunService urunService;
15
16     @GetMapping("/{id}")
17     public List<Urun> urunleriListele() {
18         return urunService.urunleriListele();
19     }
20
21     @GetMapping("/{id}/{seriNo}")
22     public Urun urunBul(@PathVariable String seriNo) {
23         return urunService.urunBul(seriNo);
24     }
25
26     @PostMapping("/")
27     public Urun urunEkle(@RequestBody Urun urun) {
28         return urunService.urunEkle(urun);
29     }
30
31     @DeleteMapping("/{id}/{seriNo}")
32     public boolean urunSil(@PathVariable String seriNo) {
33         return urunService.urunSil(seriNo);
34     }
35 }

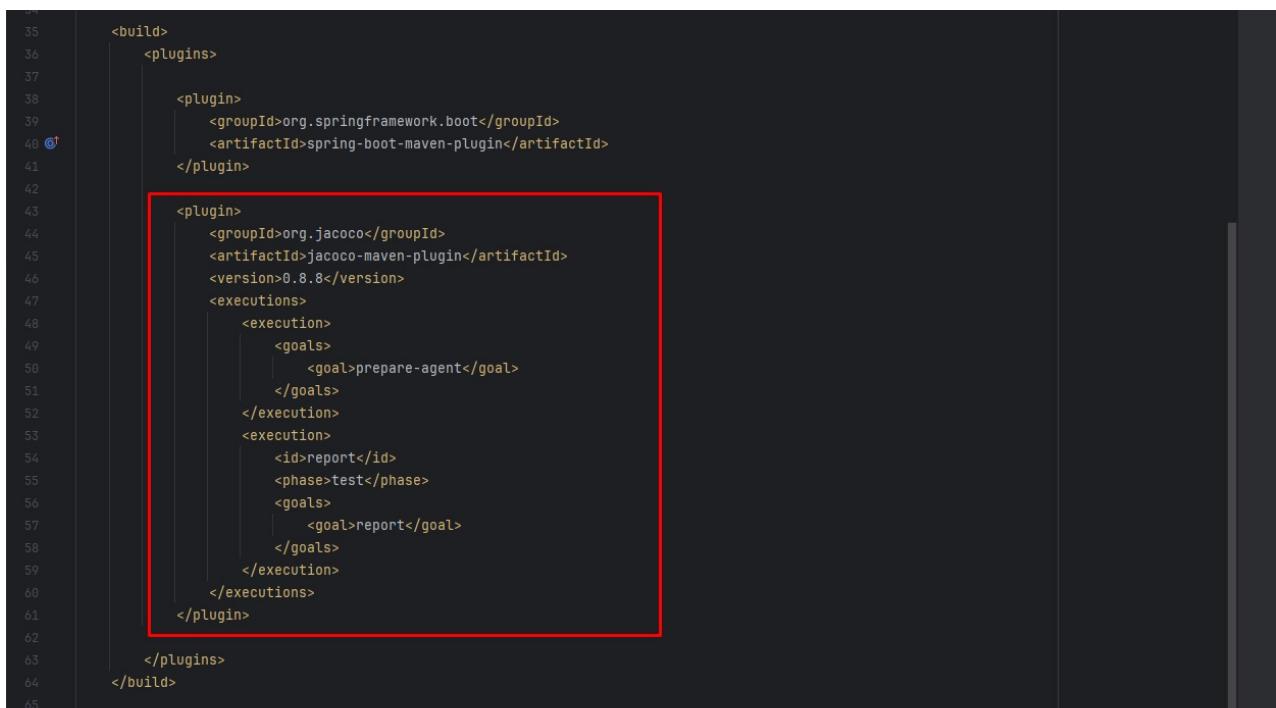
```

Proje Unit (Birim) Testleri

Projede başta Web Servis (API) olmak üzere, neredeyse bütün methodlara birim test yazılmıştır, bu sayede test kod kapsamı mümkün olduğunda fazlalaştırılmaya yönelik çalışma yapılmıştır. code coverage olarak %97 gibi çok yüksek bir orana ulaşılmış ve proje kalitesinin üst standartlarda olması hedeflenmiştir. Bu kod test kapsamı JaCoCo (Java Code Coverage) dependency'nin projeye eklenmesiyle gözlemlenmiştir. **Unit test yazma adımlarını** daha yakından incelersek

1- JaCoCo (Java Code Coverage) Kütüphanesinin Projeye Dahil Edilmesi

Projedeki unit testlerin code coverage olarak yüzde kaç tekabül ettiğini hesaplayan kütüphane, Maven proje yönetimi kullanıldığı için kolaylıkla pom.xml e dependency eklenerek projeye dahil edilmiştir. Bu sayede maven test ile bütün projedeki test methodları çalıştığında, target/site/jacoco klasörü altında index.html olarak test coverage raporu otamatik olarak oluşturulmaktadır



```
35     <build>
36         <plugins>
37
38             <plugin>
39                 <groupId>org.springframework.boot</groupId>
40                 <artifactId>spring-boot-maven-plugin</artifactId>
41             </plugin>
42
43             <plugin>
44                 <groupId>org.jacoco</groupId>
45                 <artifactId>jacoco-maven-plugin</artifactId>
46                 <version>0.8.8</version>
47                 <executions>
48                     <execution>
49                         <goals>
50                             <goal>prepare-agent</goal>
51                         </goals>
52                     </execution>
53                     <execution>
54                         <id>report</id>
55                         <phase>test</phase>
56                         <goals>
57                             <goal>report</goal>
58                         </goals>
59                     </execution>
60                 </executions>
61             </plugin>
62
63         </plugins>
64     </build>
65 
```

2- Proje Test Class (UrunWebServisTest) Oluşturulması ve Mimari

Projede bulunan methodların testinin yazıldığı Class olan UrunWebServisTest Classı oluşturuldu. Class @WebMvcTest annotation sahiptir ve mimari olarak, MockMvc class ini @Autowired annotation ile kullanır. MockMvc API'ların hareketlerini taklit eder, fake request gönderir tipki bir api gibi davranışarak API unit testlerimizi yazmakta katkı sağlar. bunun yanında @MockBean annotationu ile UrunService classından bir obje oluşturuldu ve bu nesne annotation sayesinde class içindeki api methodlarına erişim sağladı. Assertion'lar hem MockMvc kullanarak hem de MockBean kullanarak yazıldı.

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure for "Tepe-Market-WebServis".
- Code Editor:** Displays the content of `UrunWebServisTest.java`.
- Code Block:** A specific section of the code is highlighted with a red box.

```

12 import static org.mockito.Mockito.when;
13 import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*;
14 import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*;
15
16 /**
17  * Erdem BÜKE
18  */
19
20 @RunWith(MockMvcTest.class)
21 @AutoWired
22 private MockMvc mockMvc;
23
24 @MockBean
25 private UrunService urunService;
26
27 /**
28  * API Unit Tests
29  */
30 @ErdemBUEKE
31 @Test
32 void testUrunleriListele() throws Exception {
33     when(urunService.urunleriListele())
34         .thenReturn(List.of(
35             // we expect mock datas we put as static list in UrunService.class
36             new Urun(seriNo: "908861", urunAdi: "Cikolatali Gofret", urunMarka: "Ulker", urunGramaj: 40, urunBirimFiyat: 5.50),
37             new Urun(seriNo: "908862", urunAdi: "Kelebek Makarna", urunMarka: "Nuh Ankara", urunGramaj: 500, urunBirimFiyat: 10.75)
38         ));
39
40     mockMvc.perform(get(urlTemplate: "/urun"))
41         .andExpect(status().isOk())
42         // also here we expect mock datas we put as static list in UrunService.class
43         .andExpect(content().json(jsonContent: "[{\\"seriNo\\":\\"908861\\",\\"urunAdi\\":\\"Cikolatali Gofret\\",\\"urunMarka\\":\\"Ulker\\",\\"urunGramaj\\":40,\\"urunBirimFiyat\\":5.5}, {\\"seriNo\\":\\"908862\\",\\"urunAdi\\":\\"Kelebek Makarna\\",\\"urunMarka\\":\\"Nuh Ankara\\",\\"urunGramaj\\":500,\\"urunBirimFiyat\\":10.75}]))";
44
45 }
46
47 /**
48  * Erdem BÜKE
49  */
50 @Test
51 void testUrunBul() throws Exception {
52     when(urunService.urunBul(seriNo: "908861"))
53         .thenReturn(
54             // Got the values from Postman Get call with raw json and copy-pasted here
55         );
56 }
57 
```

3- Unit Testler Yazılması

a) testUrunleriListele()

`testUrunleriListele()` methodu, `UrunService` classında eklediğimiz mock (sahte) verileri kullanarak `UrunService` Service class in `UrunWebServis` RestController class ile başarılı şekilde iletişime geçtigini, içindeki `urunleriListele()` methodunun calistiginin testini gerçekleştirir. eklenen veriler Ülker Çikolatalı Gofret ve Nuh Ankara Kelebek Makarnayıdır. Bu verileri kullanıcıya getirip getirmedenin testi `when` kullanılarak `MockBean` ile liste formatında kontrol edilir, sonrasında `MockMvc` ile json formatında doğru getirdiğinin assertion u yapılır.

The screenshot shows the expanded code for `testUrunleriListele()` in the code editor.

```

// API Unit Tests
@ErdemBUEKE
@Test
void testUrunleriListele() throws Exception {
    when(urunService.urunleriListele())
        .thenReturn(List.of(
            // we expect mock datas we put as static list in UrunService.class
            new Urun(seriNo: "908861", urunAdi: "Cikolatali Gofret", urunMarka: "Ulker", urunGramaj: 40, urunBirimFiyat: 5.50),
            new Urun(seriNo: "908862", urunAdi: "Kelebek Makarna", urunMarka: "Nuh Ankara", urunGramaj: 500, urunBirimFiyat: 10.75)
        ));

    mockMvc.perform(get(urlTemplate: "/urun"))
        .andExpect(status().isOk())
        // also here we expect mock datas we put as static list in UrunService.class
        .andExpect(content().json(jsonContent: "[{\\"seriNo\\":\\"908861\\",\\"urunAdi\\":\\"Cikolatali Gofret\\",\\"urunMarka\\":\\"Ulker\\",\\"urunGramaj\\":40,\\"urunBirimFiyat\\":5.5}, {\\"seriNo\\":\\"908862\\",\\"urunAdi\\":\\"Kelebek Makarna\\",\\"urunMarka\\":\\"Nuh Ankara\\",\\"urunGramaj\\":500,\\"urunBirimFiyat\\":10.75}]))";
    // Got the values from Postman Get call with raw json and copy-pasted here
}
 
```

b) testUrunBul()

testUrunBul() methodu, UrunService classında eklediğimiz mock (sahte) verileri kullanarak UrunService Service class in UrunWebServis RestController class ile başarılı sekilde iletişimde getetigini, icindeki urunBul() methodunun calistiginin testini gerçekleştirir. 908861 seri numaralı Ülker Çikolatalı Gofret in bilgisi girilir ve ürün bilgilerini doğru getirdiginin testi gerçekleştirilir

```
▲ Erdem BÜKE
@Test
void testUrunBul() throws Exception {
    when(urunService.urunBul( seriNo: "908861"))
        .thenReturn(
            new Urun( seriNo: "908861", urunAdi: "Cikolatali Gofret", urunMarka: "Ulker", urunGramaj: 40, urunBirimFiyat: 5.5)
        );

    mockMvc.perform(get( urlTemplate: "/urun/908861"))
        .andExpect(status().isOk())
        .andExpect(jsonPath( expression: "$.seriNo").value( expectedValue: "908861"))
        .andExpect(jsonPath( expression: "$.urunAdi").value( expectedValue: "Cikolatali Gofret"))
        .andExpect(jsonPath( expression: "$.urunMarka").value( expectedValue: "Ulker"))
        .andExpect(jsonPath( expression: "$.urunGramaj").value( expectedValue: 40))
        .andExpect(jsonPath( expression: "$.urunBirimFiyat").value( expectedValue: 5.5));
}
```

c) testUrunEkle()

testUrunEkle() methodu, UrunService classında eklediğimiz mock (sahte) verileri kullanarak UrunService Service class in UrunWebServis RestController class ile başarılı sekilde iletişimde getetigini, icindeki urunEkle() methodunun calistiginin testini gerçekleştirir. Testte Mock veri ile ürün ekleme requesti gönderilir, response bodydeki Json verilerin gönderilenlerle aynı olduğunu assertionunu yaparak testi gerçekleştirir

```
▲ Erdem BÜKE
@Test
void testUrunEkle() throws Exception {
    Urun urun = new Urun( seriNo: "9088621", urunAdi: "Mock Product", urunMarka: "Mock Brand", urunGramaj: 250,
        urunBirimFiyat: 147.26);

    // any yazmadan response bos geldigi icin any matcher ile sorunu cozduk
    when(urunService.urunEkle(any(Urun.class)))
        .thenReturn(urun);

    mockMvc.perform(post( urlTemplate: "/urun/")
        .contentType(MediaType.APPLICATION_JSON)
        .content("{\"seriNo\":\"9088621\", \"urunAdi\":\"Mock Product\", \"urunMarka\":\"Mock Brand\", \" +
            \"urunGramaj\":250, \"urunBirimFiyat\":147.26}")
        .andExpect(status().isOk())
        .andExpect(jsonPath( expression: "$.seriNo").value( expectedValue: "9088621"))
        .andExpect(jsonPath( expression: "$.urunAdi").value( expectedValue: "Mock Product"))
        .andExpect(jsonPath( expression: "$.urunMarka").value( expectedValue: "Mock Brand"))
        .andExpect(jsonPath( expression: "$.urunGramaj").value( expectedValue: 250))
        .andExpect(jsonPath( expression: "$.urunBirimFiyat").value( expectedValue: 147.26));
}
```

d) testUrunSil()

testUrunSil() methodu, UrunService classında eklediğimiz mock (sahte) verileri kullanarak UrunService Service class in UrunWebServis RestController class ile başarılı sekilde iletişimde getetigini, icindeki urunSil() methodunun calistiginin testini gerçekleştirir. Testte veritabanında kayitli mock veri olan Ülker Çikolatalı Gofret ürününün seri numarası gönderilir, ve return olarak response body de true döndüğünü assertler.

```
▲ Erdem BÜKE
@Test
void testUrunSil() throws Exception {
    when(urunService.urunSil(seriNo: "908861"))
        .thenReturn(true);

    mockMvc.perform(delete(urlTemplate: "/urun/908861"))
        .andExpect(status().isOk())
        .andExpect(content().string(expectedContent: "true"));

}
```

e) testUrunEkleUrunService()

testUrunEkleUrunService() methodu, test içinde UrunService class indan nesne olusturarak, kendi içinde izole çalışlığında urunEkle() methodunun calistiginin testini gerçekleştirir. Testte yeni ürün olusturulur ve ürün UrunService.urunEkle() methodu ile eklenir, eklenen ürünün bilgileri bu sefer getter methodlar ile alınır ve assertion yapılır. UrunService sınıfındaki methodların ve getter - setter methodların izole test edilmesinin sebebi birim olarak doğru çalışlığından emin olmak ve test kod kapsamını artırmaktır.

```
▲ Erdem BÜKE
@Test
public void testUrunEkleUrunService() {
    UrunService urunService1 = new UrunService();
    Urun urun = new Urun(seriNo: "01908862", urunAdi: "Mock", urunMarka: "Mock", urunGramaj: 20, urunBirimFiyat: 150);
    Urun eklenenUrun = urunService1.urunEkle(urun);

    // Assertions
    Assertions.assertEquals(expected: "01908862", eklenenUrun.getSeriNo());
    Assertions.assertEquals(expected: "Mock", eklenenUrun.getUrunAdi());
    Assertions.assertEquals(expected: "Mock", eklenenUrun.getUrunMarka());
    Assertions.assertEquals(expected: 20, eklenenUrun.getUrunGramaj());
    Assertions.assertEquals(expected: 150, eklenenUrun.getUrunBirimFiyat());
}
```

f) testUrunleriListeleUrunService()

testUrunEkleUrunService() methodu, test içinde UrunService class indan nesne olusturarak, kendi içinde izole çalışlığında urunleriListele() methodunun calistiginin testini gerçekleştirir. Testte listeleme methodu çağırılarak boş bir listeye ürünler eklenir. Listedede en az 1 ürün olup olmadığı assertion u yapılır böylece methodun çalışlığının assertion'u yapılmış olur.

```
// Unit tests for the methods in UrunService class and Urun class

▲ Erdem BÜKE
@Test
public void testUrunleriListeleUrunService() {
    UrunService urunService1 = new UrunService();
    List<Urun> urunList = urunService1.urunleriListele();
    // Assertion of list size more than 1
    Assertions.assertTrue(condition: urunList.size() > 1);

}
```

g) testUrunBulUrunService()

testUrunBulService() methodu, test içinde UrunService class indan nesne olusturarak, kendi içinde izole çalıştığında urunBul() methodunun calistiginin testini gerçekleştirir. UrunService.urunBul() method çağırılır ve parametre olarak veritabanında mock veri olarak bulunan Ülker Çikolatalı Gofret'in seri numarası girilir. İlgili ürünün bilgilerinin hepsinin doğru getirildiğinin assertion'ları get-set methodlar çağırılarak doğrulanır

```
└─ Erdem BÜKE
  └─ @Test
    public void testUrunBulUrunService() {
        UrunService urunService1 = new UrunService();
        Urun urun = urunService1.urunBul(seriNo: "908861"); // Ulker Cikolatali Gofret Mock Data

        // Assertions
        Assertions.assertEquals(expected: "908861", urun.getSeriNo());
        Assertions.assertEquals(expected: "Cikolatali Gofret", urun.getUrunAdi());
        Assertions.assertEquals(expected: "Ulker", urun.getUrunMarka());
        Assertions.assertEquals(expected: 40, urun.getUrunGramaj());
        Assertions.assertEquals(expected: 5.50, urun.getUrunBirimFiyat());
    }
```

h) testUrunSilUrunService()

testUrunSilService() methodu, test içinde UrunService class indan nesne olusturarak, kendi içinde izole çalıştığında urunSil() methodunun calistiginin testini gerçekleştirir. UrunService.urunEkle() method çağırılır ve veritabanına yeni mock veri ile ürün eklenir. Silme işlemi UrunService.urunSil() method kullanılarak iki kere gerçekleştirilir ve ilk requestte response body de “true”, ikinci requestte ise zaten silindiği için “false” olduğunun testi gerçekleştirilir

```
└─ Erdem BÜKE
  └─ @Test
    public void testUrunSilUrunService() {
        UrunService urunService1 = new UrunService();
        Urun urun = new Urun(seriNo: "123456789", urunAdi: "mock", urunMarka: "mock", urunGramaj: 24, urunBirimFiyat: 24);
        urunService1.urunEkle(urun);

        boolean deleted = urunService1.urunSil(seriNo: "123456789"); // should be true
        boolean deleted2 = urunService1.urunSil(seriNo: "123456789"); // should be false , deleted already

        // Assertion
        Assertions.assertTrue(deleted);
        Assertions.assertFalse(deleted2);
    }
```

i) Urun Class İçerisindeki Methodların Birim Testleri

Urun sınıfı içerisindeki getter - setter ve toString methodlarının birim olarak doğru çalıştığını doğrulamak için testler yazılmıştır. Kod kapsama oranını ve yazılımın kalitesinin artırılması amaçlanmıştır

```

    @Test
    public void testUrunToString() {
        Urun urun = new Urun(seriNo: "12345", urunAdi: "mock", urunMarka: "mock", urunGramaj: 1, urunBirimFiyat: 1);

        Assertions.assertEquals(expected: "Urun{seriNo='12345', urunAdi='mock', urunMarka='mock', urunGramaj=1, urunBirimFiyat=1.0}", urun.toString());
    }

    ▲ Erdem BÜKE
    @Test
    public void testUrunSetMethods() {
        Urun urun = new Urun(seriNo: "123456", urunAdi: "mock", urunMarka: "mock", urunGramaj: 1, urunBirimFiyat: 1);

        // Change values with set methods
        urun.setSeriNo("102030");
        urun.setUrunAdi("modifiedName");
        urun.setUrunMarka("modifiedBrand");
        urun.setUrunGramaj(2);
        urun.setUrunBirimFiyat(2);

        // Assertions
        Assertions.assertEquals(expected: "102030", urun.getSeriNo());
        Assertions.assertEquals(expected: "modifiedName", urun.getUrunAdi());
        Assertions.assertEquals(expected: "modifiedBrand", urun.getUrunMarka());
        Assertions.assertEquals(expected: 2, urun.getUrunGramaj());
        Assertions.assertEquals(expected: 2, urun.getUrunBirimFiyat());
    }
}

```

4- JaCoCo ile Code Coverage Raporunun Oluşması

Projede birim test yazımını tamamladıktan sonra, maven test komutu ile projedeki testlerin hepsini koşuldu. Test koşumundan sonra bahsedilen path'da (target/site/jacoco) "index.html" adıyla test raporomuz oluşturuldu. Test koşumundan sonra oluşan rapor ve rapor içeriği dökümanı aşağıda bulunmaktadır

```

    // Assertion
    Assertions.assertTrue(deleted);
    Assertions.assertFalse(deleted2);
}

▲ Erdem BÜKE
@Test
public void testUrunToString() {
    Urun urun = new Urun(seriNo: "12345", urunAdi: "mock", urunMarka: "mock", urunGramaj: 1, urunBirimFiyat: 1);

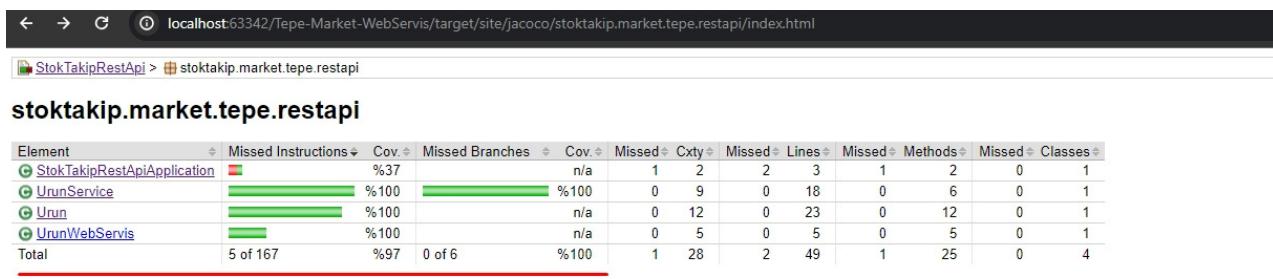
    Assertions.assertEquals(expected: "Urun{seriNo='12345', urunAdi='mock', urunMarka='mock', urunGramaj=1, urunBirimFiyat=1.0}", urun.toString());
}

▲ Erdem BÜKE
@Test
public void testUrunSetMethods() {
    Urun urun = new Urun(seriNo: "123456", urunAdi: "mock", urunMarka: "mock", urunGramaj: 1, urunBirimFiyat: 1);

    // Change values with set methods
    urun.setSeriNo("102030");
    urun.setUrunAdi("modifiedName");
    urun.setUrunMarka("modifiedBrand");
    urun.setUrunGramaj(2);
    urun.setUrunBirimFiyat(2);

    // Assertions
    Assertions.assertEquals(expected: "102030", urun.getSeriNo());
    Assertions.assertEquals(expected: "modifiedName", urun.getUrunAdi());
    Assertions.assertEquals(expected: "modifiedBrand", urun.getUrunMarka());
    Assertions.assertEquals(expected: 2, urun.getUrunGramaj());
    Assertions.assertEquals(expected: 2, urun.getUrunBirimFiyat());
}

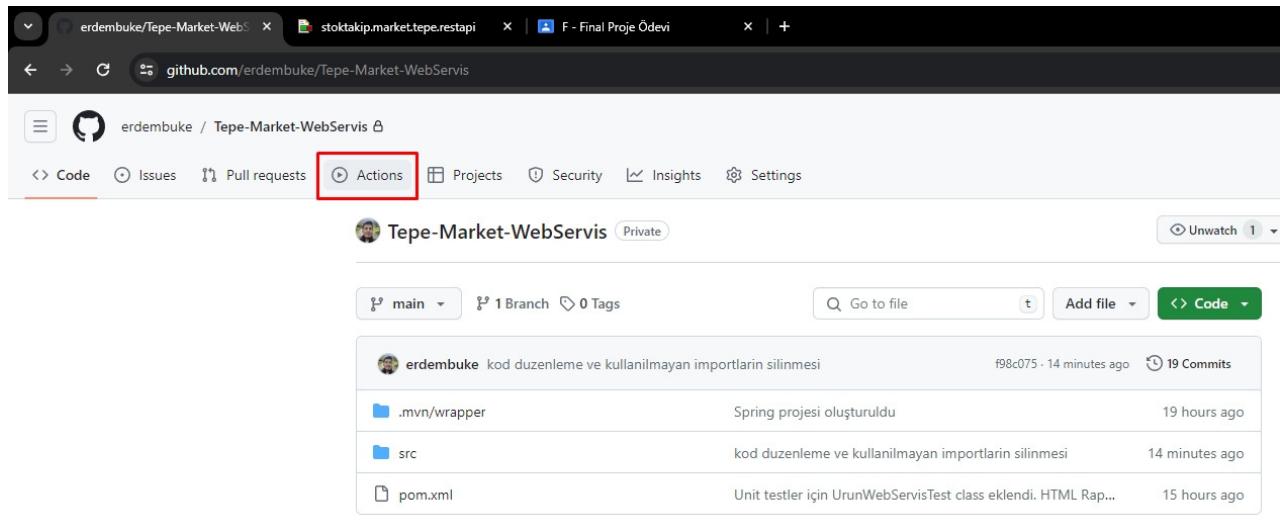
```



CI (Continuous Integration) ile Maven Build ve JaCoCo Raporu Ekleme

Projede GitHub Actions ile “Java with Maven” kullanılarak, “main” branch ile kod pushlarken ya da “main” branch a pull request yapıldığı zaman, Maven projenin buildinin gerçekleştiğini ve kodda hata olmadığını doğrulayan sistem eklendi. Bu sisteme ek olarak “maven.yml” dosyasına “Coverage” ismi ile JaCoCo madrapps/jacoco-report@v1.3 eklendi ve build sonrasında değiştirilen kod ile güncel kod kapsamını pull request üzerinde de görüntülenmesi sağlandı.

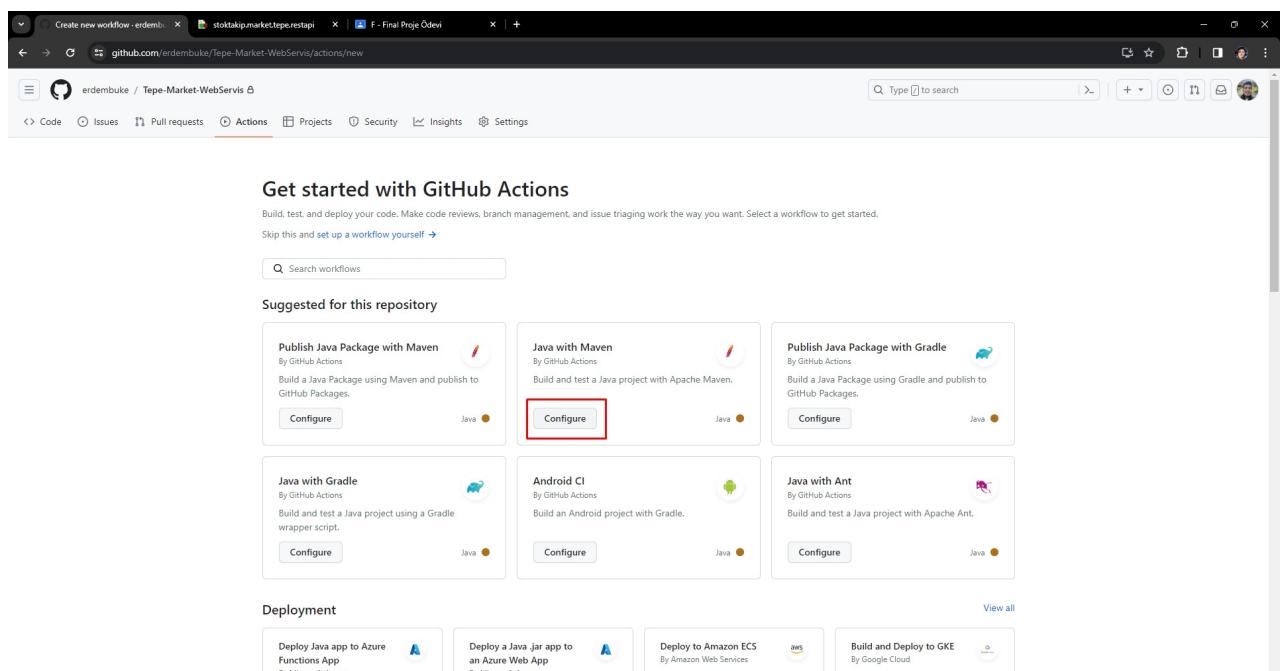
1- Repository’deki GitHub Actions Sekmesine Giriş



The screenshot shows a GitHub repository page for 'erdembuke/Tepe-Market-WebServis'. The 'Actions' tab is highlighted with a red box. The main content area displays recent commits:

- erdembuke kod düzenlemeye ve kullanılmayan importların silinmesi (f98c075 - 14 minutes ago)
- .mvn/wrapper (Spring projesi oluşturuldu - 19 hours ago)
- src (kod düzenlemeye ve kullanılmayan importların silinmesi - 14 minutes ago)
- pom.xml (Unit testler için UrunWebServisTest class eklendi. HTML Rap... - 15 hours ago)

2- Java with Maven Configure Düğmesine Tıklamak



The screenshot shows the 'Create new workflow' page for the 'erdembuke/Tepe-Market-WebServis' repository. The 'Actions' tab is selected. The 'Get started with GitHub Actions' section is visible, along with a search bar and a 'Skip this and set up a workflow yourself' link. Below this, there are sections for 'Suggested for this repository' and 'Deployment'. In the 'Suggested for this repository' section, the 'Java with Maven' card is highlighted with a red box around its 'Configure' button. Other cards include 'Publish Java Package with Maven', 'Publish Java Package with Gradle', 'Java with Gradle', 'Android CI', 'Java with Ant', and 'Deploy Java app to Azure Functions App', 'Deploy a Java jar app to an Azure Web App', 'Deploy to Amazon ECS', and 'Build and Deploy to GKE'.

3- maven.yml Dosyası İçerisindeki Alanları Projeye Göre Doldurup Commitlemek

The screenshot shows the GitHub workflow editor for the repository 'erdembuke/Tepe-Market-WebServis'. The file 'maven.yml' is open, displaying the following YAML code:

```
name: TepeMarketWebServis CI
on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Set up JDK 17
        uses: actions/setup-java@v3
        with:
          java-version: '17'
          distribution: 'temurin'
          cache: maven
      - name: Build with Maven
        run: mvn -B install --file pom.xml
      - name: Coverage
        id: jacoco
        uses: madnapse/jacoco-report@v1.3
        with:
          paths: ${{ github.workspace }}/target/site/jacoco/jacoco.xml
          token: ${{ secrets.GITHUB_TOKEN }}
```

The GitHub interface includes a sidebar with files like 'main', '.github/workflows', '.mvn', 'src', and 'pom.xml'. On the right, there are sections for 'Marketplace' and 'Featured Actions' (Cache, Upload a Build Artifact, Close stale issues, Setup .NET Core SDK, First interaction).

3- CI “Java with Maven” kullanarak build success ekranı ve PR’da Code Coverage

The screenshot shows the GitHub Actions interface for the repository 'erdembuke/Tepe-Market-WebServis'. It displays a single workflow run for the 'Create README.md' action, triggered by a pull request. The run was created 3 minutes ago and took 42s. The status is 'Success'.

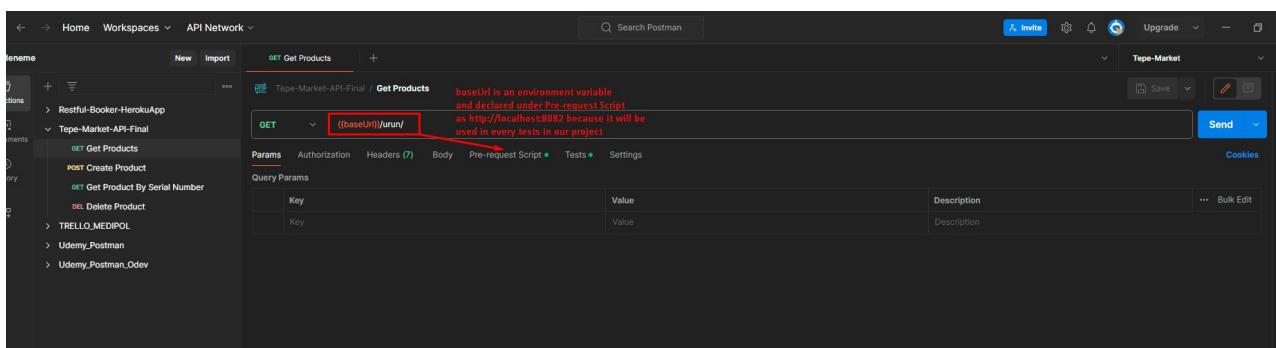
The screenshot shows a GitHub Pull Request for 'Create README.md #4'. The pull request has been merged into the 'main' branch. The code coverage section shows a total project coverage of 97.01%. The review summary indicates that all checks have passed and there are no conflicts with the base branch.

Postman Kullanılarak Projedeki Web Servislerin Test Edilmesi

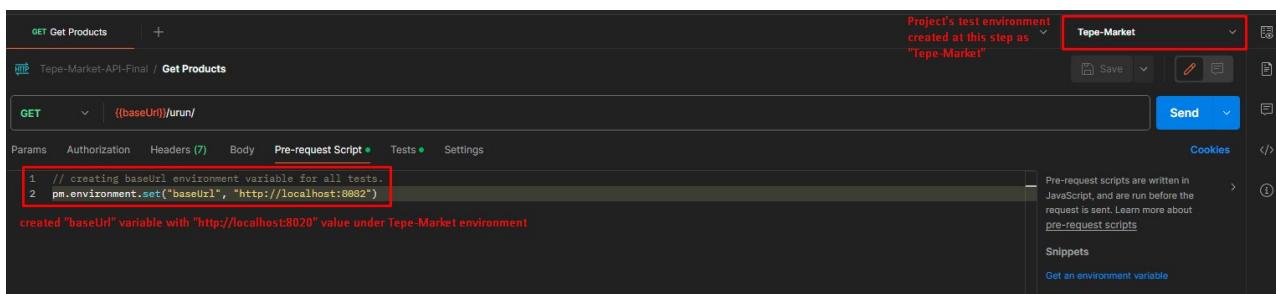
Postman Tool'u kullanılarak projede yazılan web servislerin detaylı testi gerçekleştirildi. Testler hazırlanırken tekrar tekrar sorunsuz koşulabilmesi için postman "environment" variables feature'i kullanıldı ve Tests kısmında gerekli requestlerde detaylı testler yazıldı. Test data'ların dinamik olarak değişmesi ve elle müdahale olmadan istenildiği kadar koşulabilmesi için Pre-request Script kullanılarak gerekli yerlerde dinamik değişkenler oluşturuldu.

a) Get Products

`{{baseUrl}}/urun/` URL sine HTTP GET request method gönderilerek market stok veritabanında bulunan bütün verilerin döndürünün testi yapıldı. `{{baseUrl}}` environment (Sağ üst köşede Tepe-Market environment) variable olarak Pre-request Script altında oluşturuldu ve bütün testlerde kullanıldı.



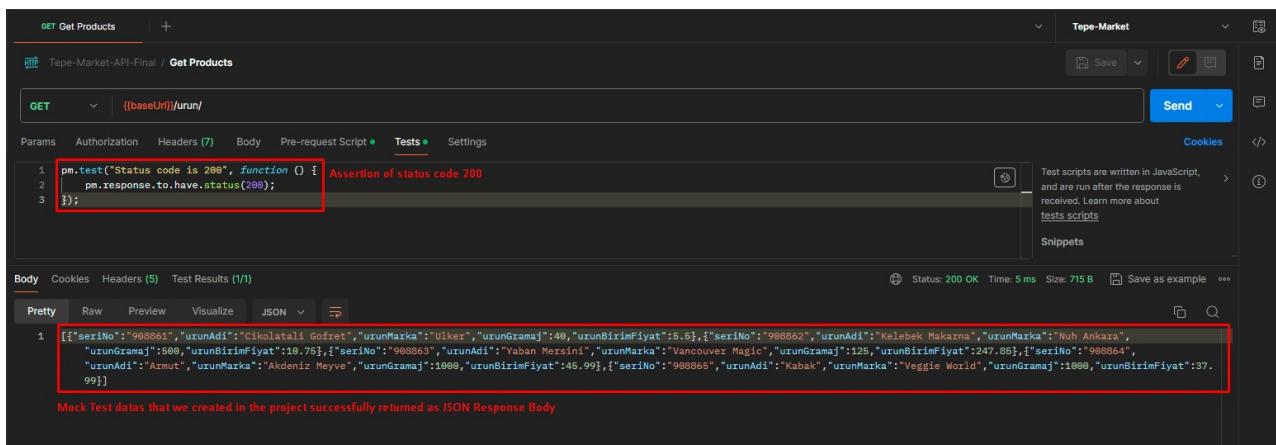
The screenshot shows the Postman interface with a 'GET Get Products' request selected. The URL field contains `http://{{baseUrl}}/urun/`. A tooltip on the right side of the URL field states: "baseUrl is an environment variable and declared under Pre-request Script as http://localhost:8082 because it will be used in every tests in our project". The 'Params' tab is selected, showing a single query parameter 'Key' with value 'Value'. The 'Pre-request Script' tab is also visible.



The screenshot shows the 'Pre-request Script' tab being selected. The script code is:

```
1 // creating baseUrl environment variable for all tests.
2 pm.environment.set('baseUrl', 'http://localhost:8020')
```

 A note below the script says: "created 'baseUrl' variable with 'http://localhost:8020' value under Tepe-Market environment". The 'Tepe-Market' environment is selected in the top right.



The screenshot shows the 'Tests' tab being selected. The test script is:

```
1 pm.test('Status code is 200', function () {
2   pm.response.to.have.status(200);
3});
```

 A note below the test says: "Test scripts are written in JavaScript, and are run after the response is received. Learn more about tests scripts". The 'Tepe-Market' environment is selected in the top right. The 'Body' tab is selected, showing the JSON response data.

b) Create Products

`{{baseUrl}}/urun/` URL sine HTTP POST request method gönderilerek market veritabanına yeni ürün eklenmesi testi yapıldı. Testin dinamik olması için pre-request scriptte bütün Json Request Body alanları otomatik olarak environment variable olarak atandı ve body kısmında kullanıldı. Testlerde ise oluşturulduktan sonra postman response body json olarak değişkene atandı ve detaylı assertionları yazıldı.

The screenshot shows the Postman interface with a POST request to `http://Tepe-Market-API-Final/Create Product`. The 'Pre-request Script' tab is selected. The script content is:

```
1 // random seri numarası api call öncesi olusacak
2 // ve cagrıda bu veri kullanılacak
3
4 pm.environment.set("productSerial", "" + parseInt(Math.random()*100000)) // "" for converting serial number to string
5 pm.environment.set("productName", "Product " + parseInt(Math.random()*1000))
6 pm.environment.set("productBrand", "Brand " + parseInt(Math.random()*1000))
7 pm.environment.set("productWeight", parseInt(Math.random()*1000))
8 pm.environment.set("productPrice", parseInt(Math.random()*200))
```

A note below the script says: "For improving overall test quality, saving time and effort, Tests have design with dynamic values and automated. before test executed, all of the body variables of our Product will be generated with random values under our test environment 'Tepe-Market' and we will use these values in other tests".

The screenshot shows the Postman interface with a POST request to `http://Tepe-Market-API-Final/Create Product`. The 'Body' tab is selected and set to JSON. The request body is:

```
{ "seriNo": "{{productSerial}}", "urunAdi": "{{productName}}", "urunMarka": "{{productBrand}}", "urunGramaj": {{productWeight}}, "urunBirimFiyat": {{productPrice}}}
```

A note above the body says: "Random values that created on Pre-request Script used here as environment variables".

The screenshot shows the Postman interface with a POST request to `http://Tepe-Market-API-Final/Create Product`. The 'Tests' tab is selected. The test script content is:

```
1 pm.test("Status code is 200", function () {
2     pm.response.to.have.status(200); status code assertion
3 });
4
5 let response = pm.response.json() response body taken as a variable (JSON FORMAT) for assertion test writing
6
7 pm.test("Created Product Attributes Match!", function () {
8     pm.expect(response.seriNo).is.eql(pm.environment.get("productSerial"))
9     pm.expect(response.urunAdi).is.eql(pm.environment.get("productName"))
10    pm.expect(response.urunMarka).is.eql(pm.environment.get("productBrand"))
11    pm.expect(response.urunGramaj).is.eql(pm.environment.get("productWeight"))
12    pm.expect(response.urunBirimFiyat).is.eql(pm.environment.get("productPrice"))
13 })
```

A note next to the script says: "Assertion of every attribute of the product is the same value with the values we've given before with Pre-Request Script".

The 'Test Results' section shows "all tests passed".

c) Get Product By Serial Number

`{{baseUrl}}/urun/{{productSerial}}` URL sine HTTP GET request method gönderilerek market veritabanından bir önceki teste eklediğimiz ürününün getirilmesini içeren test. path variable olarak productSerial environment variable olarak bir önceki teste oluşturulmuştu, burda kullanarak aynı şekilde testimizi dinamik hale getirdik ve assertionları yazarkende environment variable'lerimizi kullanmış olduk.

The screenshot shows the Postman interface for a GET request. The URL is `http://Tepe-Market-API-Final / Get Product By Serial Number`. The Pre-request Script contains the code:

```
productSerial is the variable that contains the serial number of the product. We created it as an environment variable in the Pre-request Script section of Create Product API Call
```

. The Tests tab shows the following assertions:

```
pm.test("Status code is 200", function () {  
    pm.response.to.have.status(200);  
});  
  
pm.test("Product informations fetched correctly!", function() {  
    pm.expect(response.seriNo).is.eql(pm.environment.get("productSerial"));  
    pm.expect(response.urunAdi).is.eql(pm.environment.get("productName"));  
    pm.expect(response.urunMarka).is.eql(pm.environment.get("productBrand"));  
    pm.expect(response.urunGramaj).is.eql(pm.environment.get("productWeight"));  
    pm.expect(response.urunBirimFiyat).is.eql(pm.environment.get("productPrice"));  
})
```

The screenshot shows the Test Results for the GET request. It indicates "all tests passed". The Body tab displays the response JSON:

```
{  
  "seriNo": "877367",  
  "urunAdi": "Product 5398",  
  "urunMarka": "Brand 592",  
  "urunGramaj": 117,  
  "urunBirimFiyat": 84.0  
}
```

c) Delete Product

`{{baseUrl}}/urun/{{productSerial}}` URL sine HTTP DELETE request method gönderilerek market veritabanından Create Product testinde oluşturduğumuz ürünün silinmesini içeren test. Status code 200 ve Response Body ise kodlandığı gibi true döndüğünün Assertion u yapılmıştır

The screenshot shows the Postman interface for a DELETE request. The URL is `http://Tepe-Market-API-Final / Delete Product`. The Pre-request Script contains the code:

```
pm.test("Status code is 200", function () {  
    pm.response.to.have.status(200);  
});
```

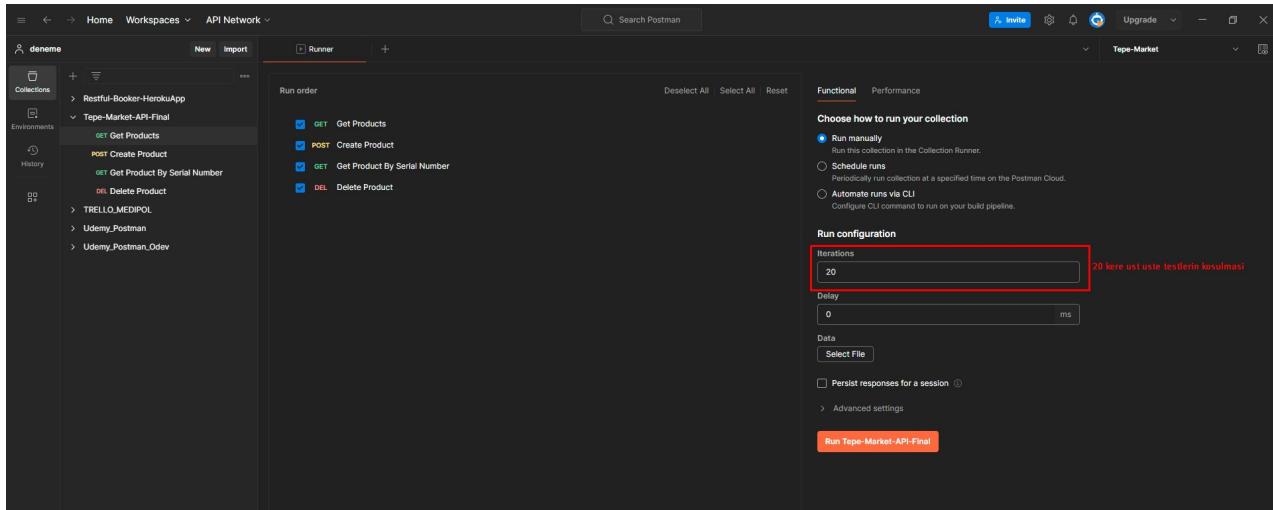
 and the Tests tab shows the following assertions:

```
pm.test("True in Response Body", function () {  
    pm.expect(pm.response.text()).is.eql('true');  
})
```

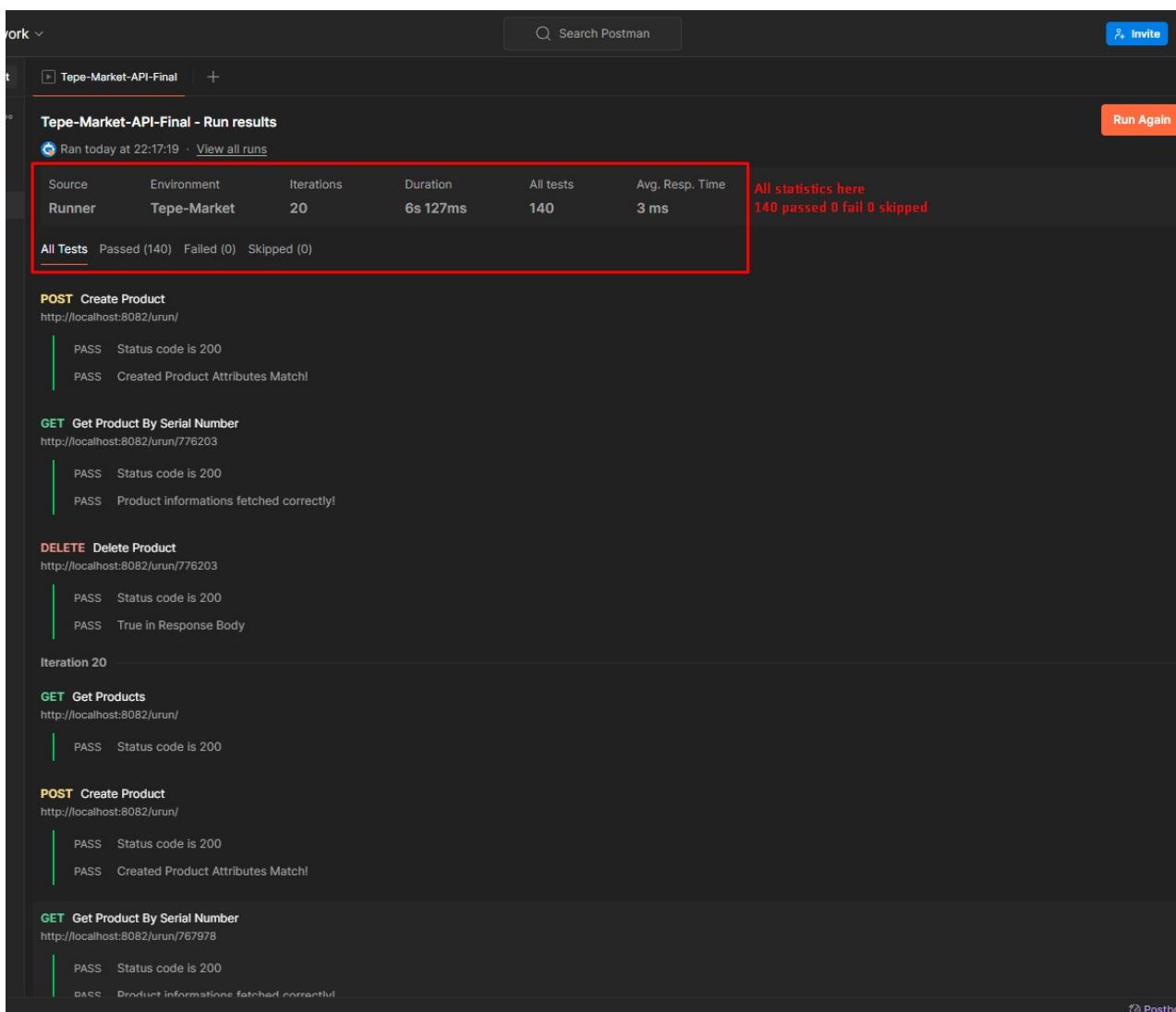
The Body tab displays the response body: `true`.

Postman API Testlerin Collection Runner ile Koşulması ve Test Sonuçları

Oluşturulan bütün testlerin dinamik olmasının bir sebebi de collection runner ile defalarca koşup test sonuçlarını gözlemlenmeyebilmekti. Oluşturulan testler collection ile “Iterations” yani koşma sayısı 20, Delay ise 0ms ayarlanarak koşuldu. Sonuçları içeren görseller aşağıda yer almaktadır



The screenshot shows the Postman interface with the 'Runner' tab selected. On the left, there's a sidebar with 'Collections' and 'Environments'. Under 'Collections', 'Tepe-Market-API-Final' is expanded, showing four test cases: 'Get Products', 'POST Create Product', 'GET Get Product By Serial Number', and 'DELETE Delete Product'. To the right, under 'Run order', all four tests are checked. In the 'Run configuration' section, the 'Iterations' field is set to '20', highlighted with a red box. A tooltip '20 kere üst üste testlerin koşulması' (Run 20 times in a row) appears next to it. Other settings like 'Delay' (0 ms), 'Data' (Select File), and 'Persist responses for a session' are also visible.

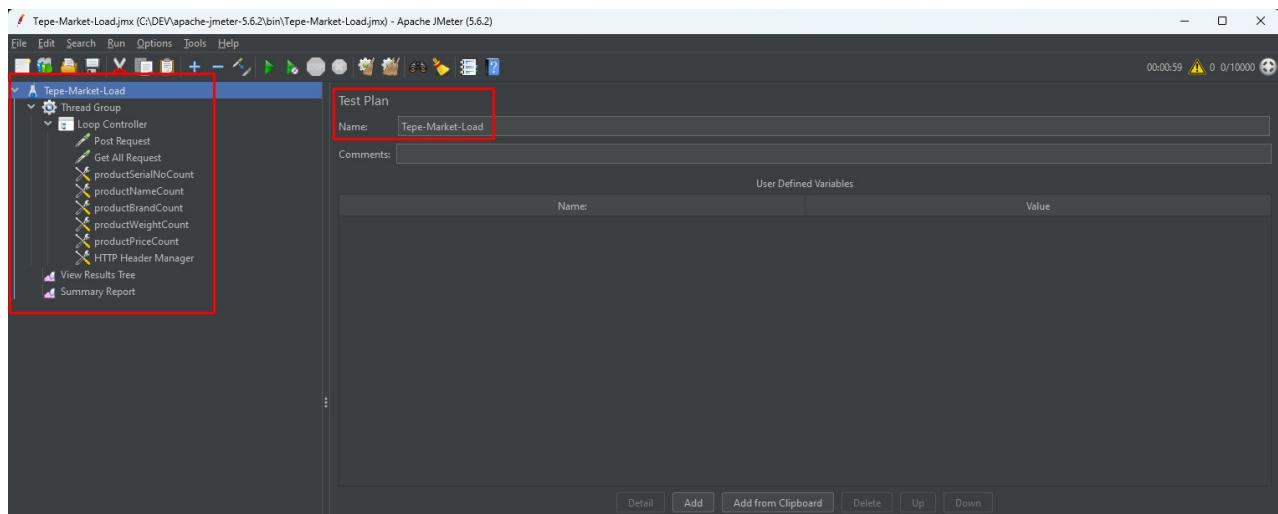


The screenshot shows the 'Run results' page for the 'Tepe-Market-API-Final' collection. At the top, it says 'Ran today at 22:17:19 · View all runs'. Below that, a summary table shows: Source (Runner), Environment (Tepe-Market), Iterations (20), Duration (6s 127ms), All tests (140), and Avg. Resp. Time (3 ms). To the right, a link 'All statistics here' and the message '140 passed 0 fail 0 skipped' are displayed. The main area lists individual test results for each iteration. For example, the first iteration shows results for 'POST Create Product', 'GET Get Product By Serial Number', and 'DELETE Delete Product'. The second iteration shows results for 'GET Get Products', 'POST Create Product', and 'GET Get Product By Serial Number'. All tests are marked as 'PASS' with status codes 200 and matching product information.

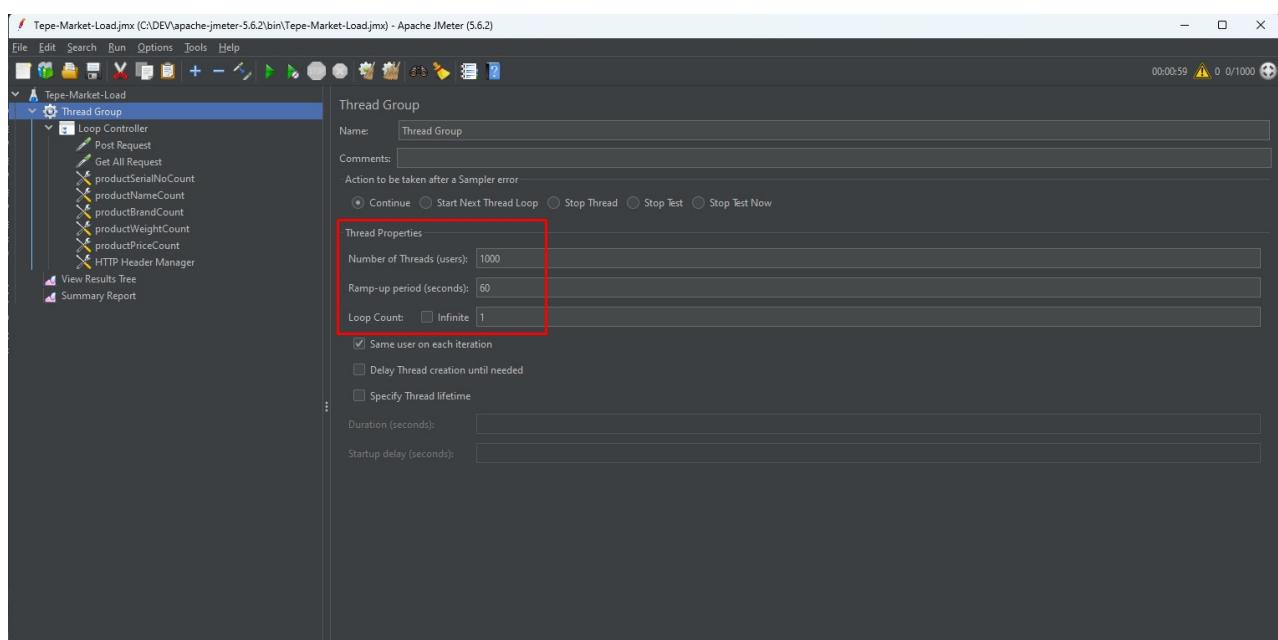
Apache JMeter kullanılarak Web Servislerin Performans Testlerinin Koşulması

Projede oluşturulan web servislerin performansını test etmek için non-functional test kategorisinde yer alan "Performans Testi" testleri yazıldı ve koşuldu. Performans testlerinin arasından ise tercih edilen yöntem stres testi yerine yük testi (load test) olmuştur. Geliştirilen yazılımın türü ve kullanıcı sayısı düşünüldüğünde web servisin maksimum kapasitesine yakın derecede zorlayacak bir test planı oluşturulmuştur. Test planı, 1000 adet Kullanıcının (Thread) 60 saniyelik zaman periyodu içerisinde sisteme yük uygulamasını, ve bunun 2 döngü halinde gerçekleşmesini içermektedir. İçerisinde sisteme en çok yük bindirecek, trafik oluşturacak olan 2 method seçilmiştir (Ürün Ekleme ve Bütün Ürünleri Listeleme). Her saniye sisteme iterative bir şekilde yük binip aynı zamanda her ürün eklendiğinde bütün ürünler de listelendiğinden dolayı ürün için kalite metriklerini karşılayan bir yük testi senaryosu gerçekleştirilmiştir. Testler koşulurken kullanılan veriler Counter ile oluşturulmuş ve json body de oluşturulan Counter'lar kullanılmıştır. Bu sayede duplicate veri kullanımının da önüne geçilmiştir, testler olabilecek en maksimum seviyede optimize edilmiştir.

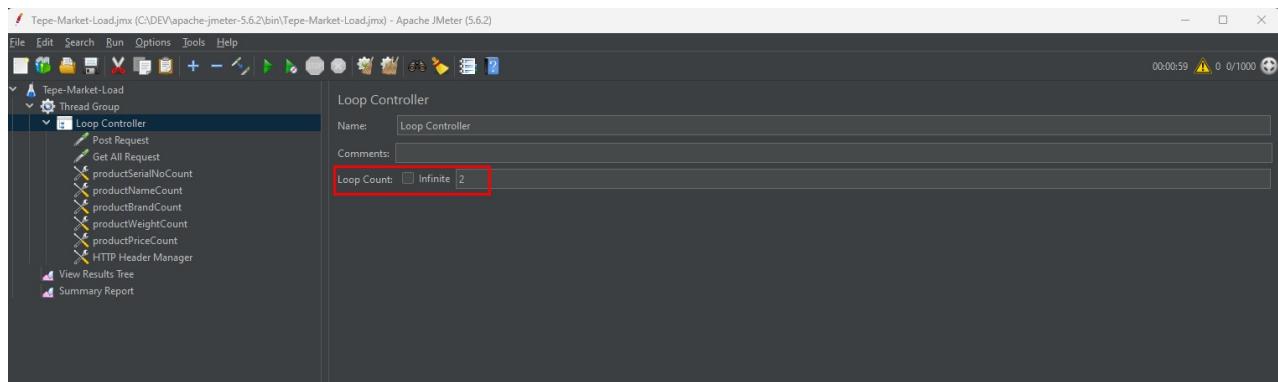
1- Test Plan Oluşturulması



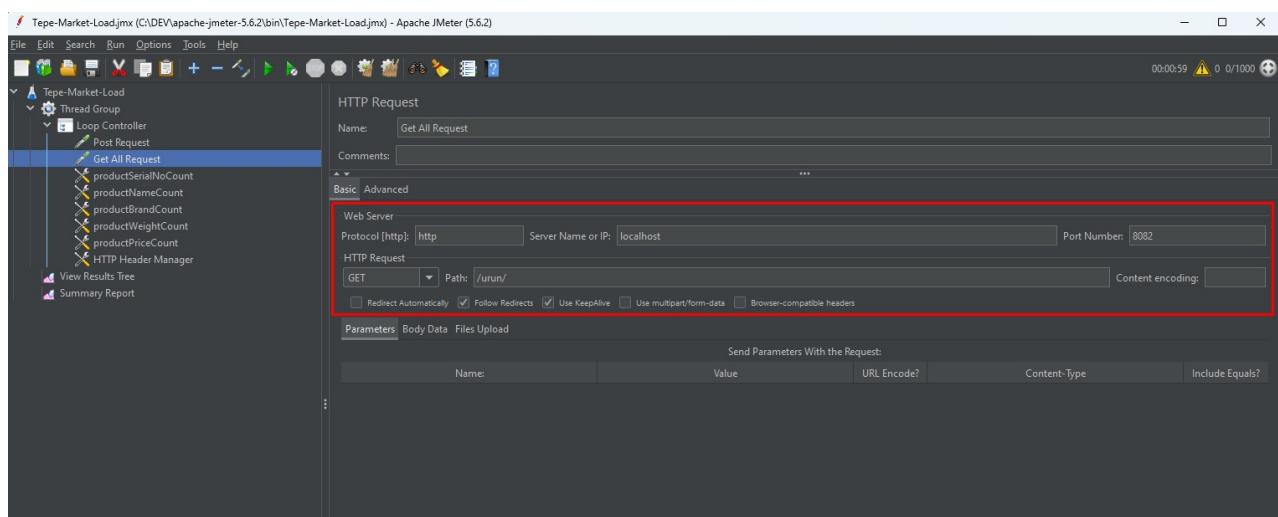
2- Thread (User) Group oluşturulması



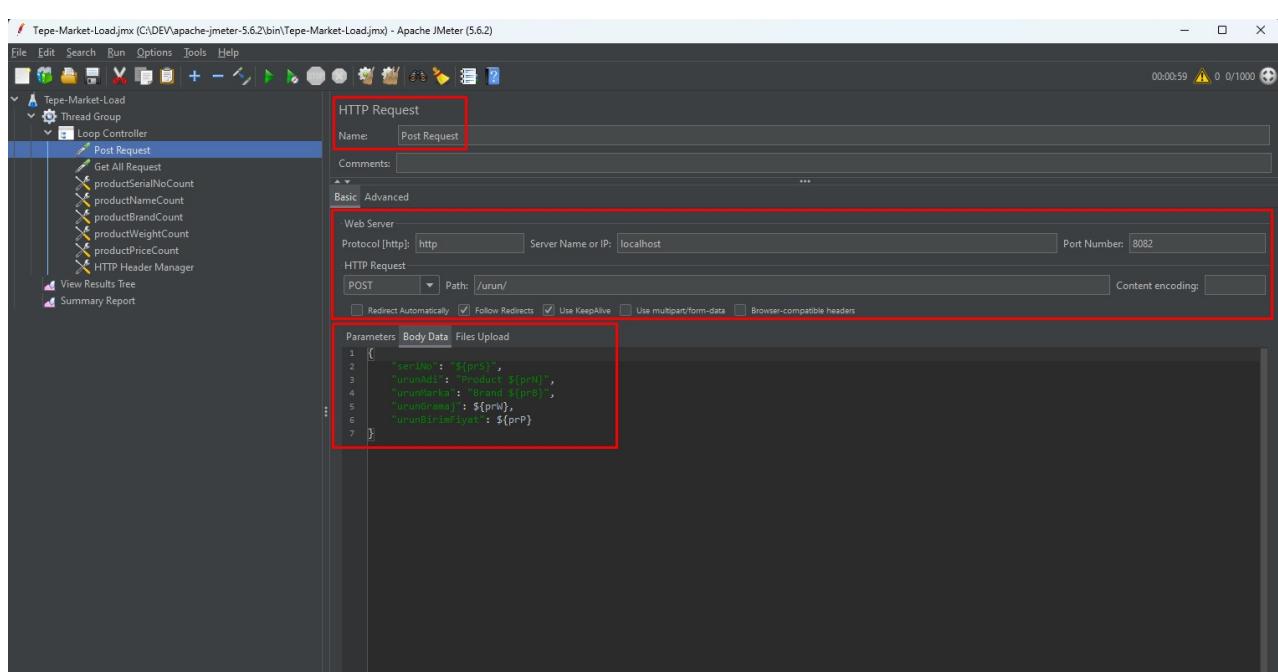
3- Loop Controller oluşturulması



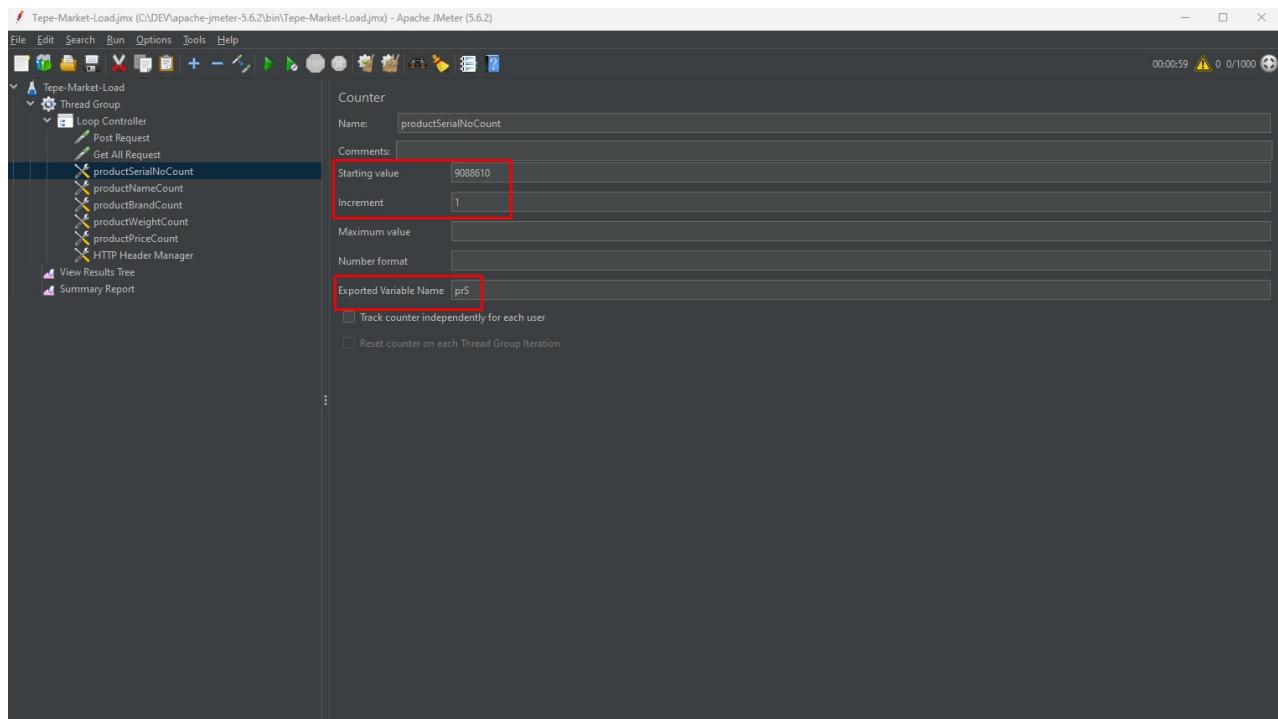
4- GET Request oluşturulması



5- POST Request oluşturulması

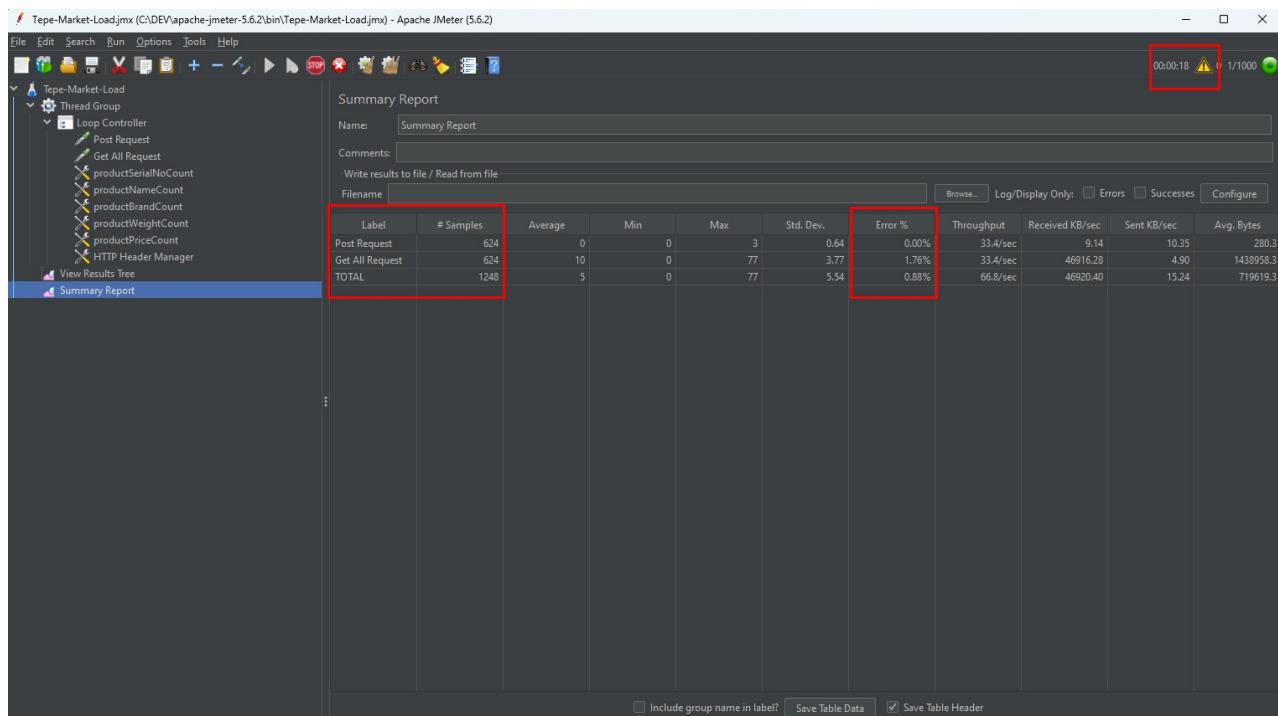


6- Controller ile Değişkenler Oluşturulması

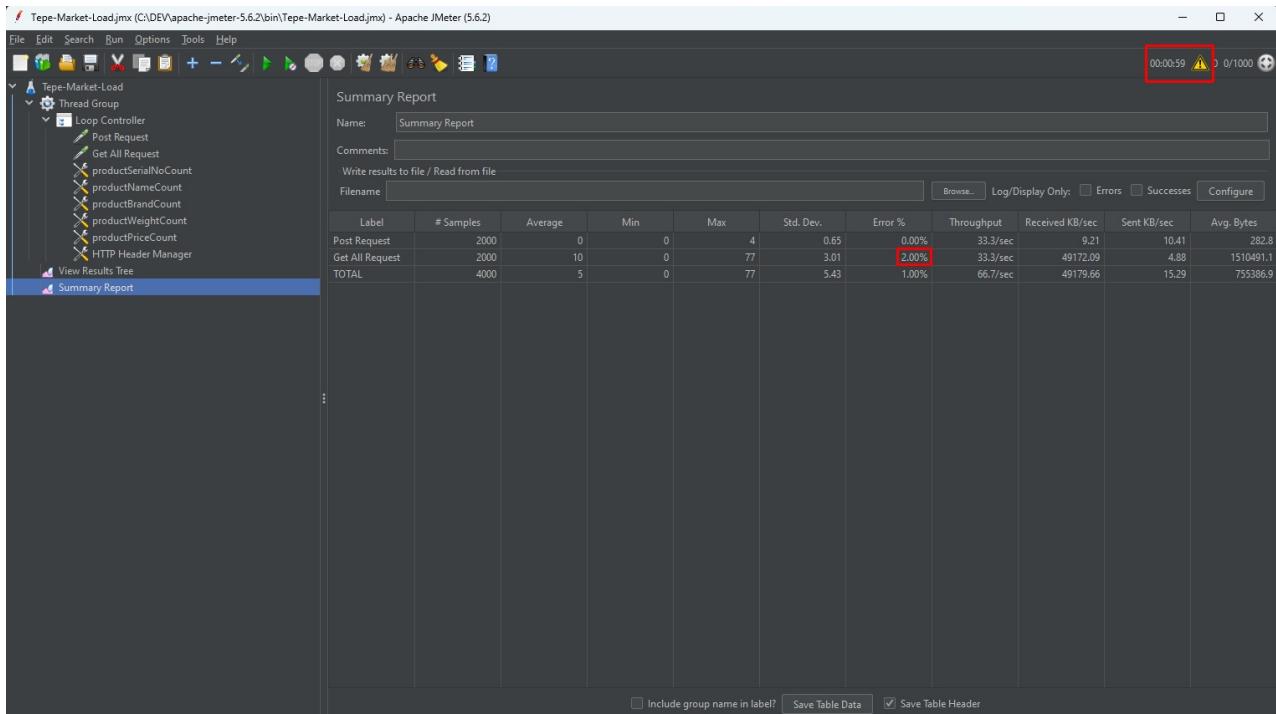


7- Testlerin Koşulması ve İlk 15 Saniyede Oluşan Sonuçlar

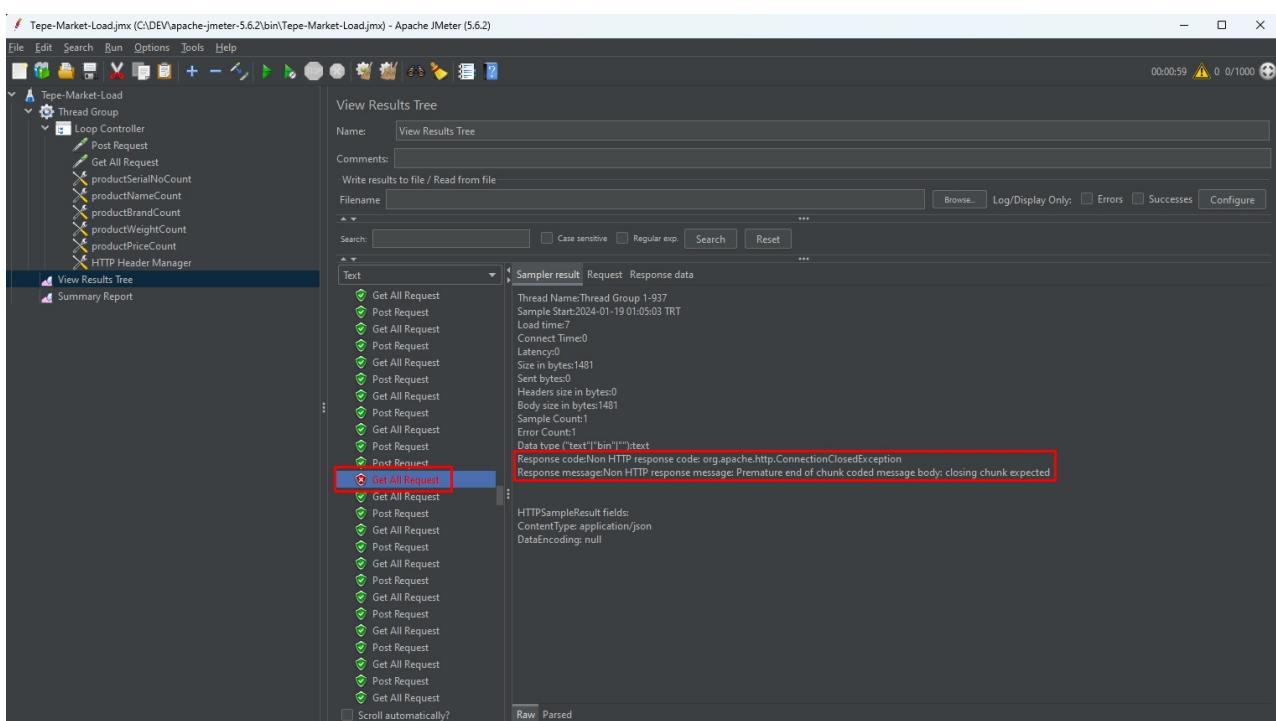
Aşağıdaki görselde görüldüğü gibi, oluşturulan test planında, 1000 userin 60 saniye içerisinde 2 döngü halinde, iki adet request ile toplama 4000 request oluşturmasını istedik. ilk 15 saniyedeki verilere bakılırsa hata oranı post requestte %0 iken Get All Request için %1.8 test tamamlandığında bu oranın artışı sonraki görselde görülmektedir.



8- Test Sonuçları ve Analizler



Test tamamlanınca oluşan grafiğe bakıldığında, hata oranının post request için %0, get all request için %2 olduğunu gözlemliyoruz. İlk 15 saniyeye göre analiz yaparsak, zaman ilerledikçe post request için herhangi ekstra hata oluşmamasına rağmen, get all requestte hata oranı artmıştır. Bunun sebebinin test ilerledikçe her saniye veritabanındaki verinin artması ve her saniye hepsinin yazdırmasının API için performans sınırlarında olması olabilir. Genel olarak senaryoya bakıldığı zaman, yük testinin de sınırlarını zorlayan bir test senaryosu olduğunu göz önünde bulundurursak web servisi performans metriklerine başarıyla ulaşmıştır. JMeter ile ayrıca "View Results Tree" kullanarak hata alan requestlere bakabilir, aşağıdaki görseldeki gibi detaylı logları görüntüleyebiliriz.



Proje'nin Private GitHub Repository Olarak Paylaşılması ve IDE üzerinden Git ile Commit Yapılırken İzlenen Adımlar

1- GitHub Üzerinde Private Repository Oluşturulması ve Collaborators Eklenmesi

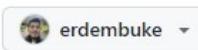
a) Private Repository oluşturulması

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (*).

Owner *



Repository name *

/ Tepe-Market-WebServis

Tepe-Market-WebServis is available.

Great repository names are short and memorable. Need inspiration? How about [friendly-couscous](#) ?

Description (optional)

Public

Anyone on the internet can see this repository. You choose who can commit.

Private

You choose who can see and commit to this repository.

Initialize this repository with:

Add a README file

This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: None ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: None ▾

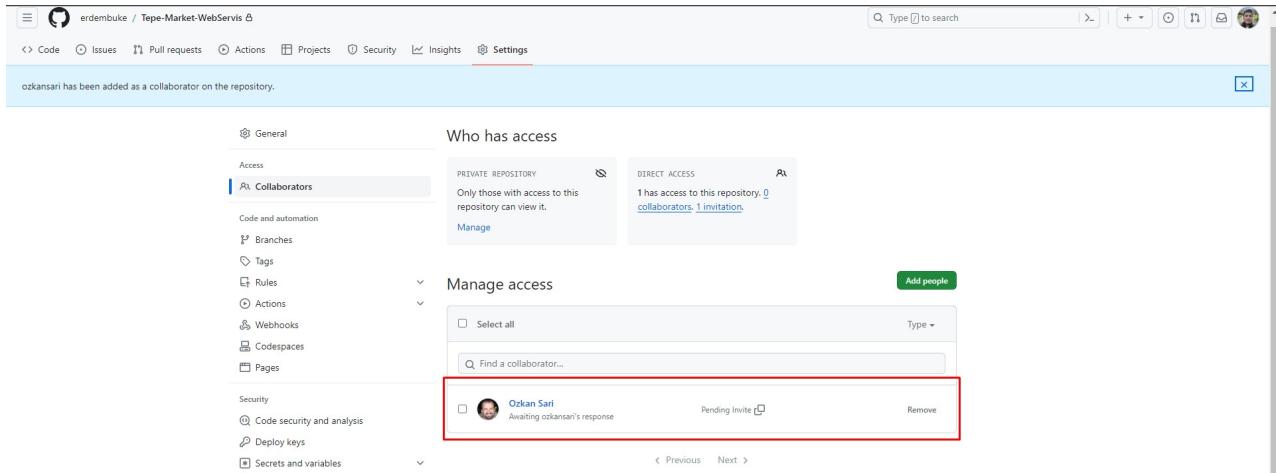
A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

You are creating a private repository in your personal account.

Create repository

b) Projeye “Ozkan Sarı” Collaborators Olarak Eklenmesi

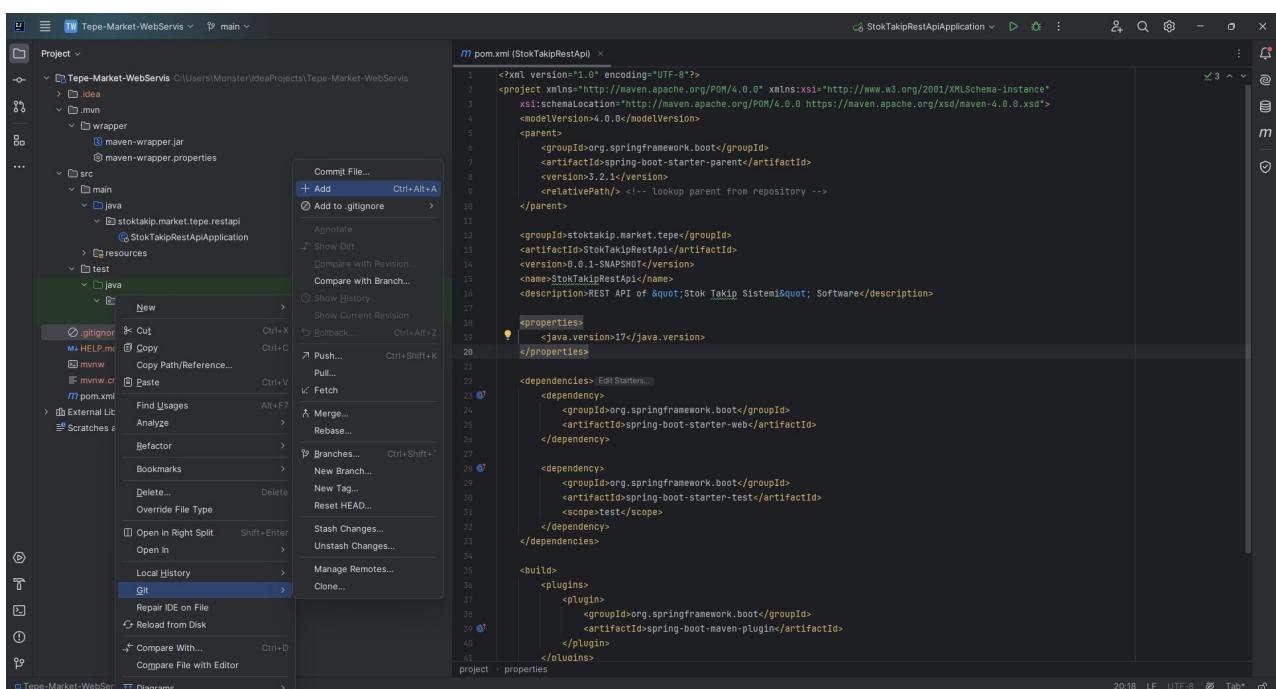
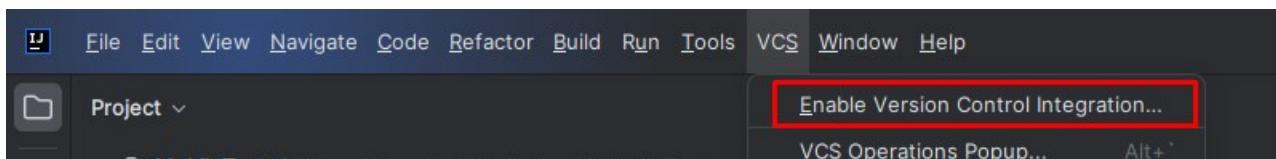
Oluşturulan private repository'e ulaştıktan sonra “Settings” e tıklayıp, sağda açılan menüden collaborators kısmına girilmeli. Bu alanda ekle butonuna basıp “ozkansari” kullanıcısına istek gonderilmeli. Kullanıcı mail üzerinden onayladığında projeye erişip katkıda bulunabilme yetkisine sahip olacak.



2- IntelliJ IDEA ile Oluşturulan Spring Projesinin Private Repository'e Eklenmesi

a) VCS Aktif Edilip Dosyaların Eklenmesi

Spring Projesini oluşturduktan ve zipen çikardıktan sonra IntelliJ IDEA ile proje açıldı. VCS/Enable Version Control Integration yolundan proje git e initialize edildi. Repository'e eklenecek dosyalar IDE yardımıyla manual sekilde git .add ile eklendi



b) Eklenen Dosyaların Terminal ile Oluşturulan Private Repository'e Gönderilmesi

Projede paylaşılacak dosyalar IDE yardımıyla eklendikten sonra;

- 1- Terminal açılarak git status ile mevcut durum paylaşıldı
- 2- git commit -m ile ilk commit yapıldı.
- 3- git branch -M main ile branch oluşturuldu
- 4- git remote add origin https://github.com/erdembuke/Tepe-Market-WebServis.git
- 5- git push -u origin main ile proje repository'e pushlandı

```
Terminal Local + ▾
src/ nothing added to commit but untracked files present (use "git add" to track)
PS C:\Users\Monster\IdeaProjects\Tepe-Market-WebServis> git status
On branch master

No commits yet

Changes to be committed:
(use "git rm --cached <file>..." to unstage)
  new file:   .mvn/wrapper/maven-wrapper.jar
  new file:   .mvn/wrapper/maven-wrapper.properties
  new file:   pom.xml
  new file:   src/main/java/stoktakip/market/tepe/restapi/StokTakipRestApiApplication.java
  new file:   src/test/java/stoktakip/market/tepe/restapi/StokTakipRestApiApplicationTests.java

Untracked files:
(use "git add <file>..." to include in what will be committed)
  .gitignore
  mvnw
  mvnw.cmd
  src/main/resources/

```

Above screenshot shows the terminal output of 'git status'. It lists untracked files under 'Changes to be committed' and untracked files under 'Untracked files'. A red box highlights the list of new files: '.mvn/wrapper/maven-wrapper.jar', '.mvn/wrapper/maven-wrapper.properties', 'pom.xml', 'src/main/java/stoktakip/market/tepe/restapi/StokTakipRestApiApplication.java', and 'src/test/java/stoktakip/market/tepe/restapi/StokTakipRestApiApplicationTests.java'. An arrow points from this box to the text 'eklenen dosyalar' (added files).

```
PS C:\Users\Monster\IdeaProjects\Tepe-Market-WebServis> git commit -m "Spring projesi oluşturuldu"
[master (root-commit) b340feb] Spring projesi oluşturuldu
  5 files changed, 72 insertions(+)
  create mode 100644 .mvn/wrapper/maven-wrapper.jar
  create mode 100644 .mvn/wrapper/maven-wrapper.properties
  create mode 100644 pom.xml
  create mode 100644 src/main/java/stoktakip/market/tepe/restapi/StokTakipRestApiApplication.java
  create mode 100644 src/test/java/stoktakip/market/tepe/restapi/StokTakipRestApiApplicationTests.java
PS C:\Users\Monster\IdeaProjects\Tepe-Market-WebServis> git branch -M main
PS C:\Users\Monster\IdeaProjects\Tepe-Market-WebServis> git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    .gitignore
    mvnw
    mvnw.cmd
    src/main/resources/

```

Above screenshot shows the terminal output of committing changes and creating a new branch. The commit message is 'Spring projesi oluşturuldu'. A red box highlights the command 'git commit -m "Spring projesi oluşturuldu"'. An arrow points from this box to the text 'Commit komutu ve commit message'. Another red box highlights the command 'git branch -M main'. An arrow points from this box to the text 'Branch oluşturma işlemi'.

```

nothing added to commit but untracked files present (use "git add" to track)
PS C:\Users\Monster\IdeaProjects\Tepe-Market-WebServis> git remote add origin https://github.com/erdembuке/Tepe-Market-WebServis.git
PS C:\Users\Monster\IdeaProjects\Tepe-Market-WebServis> git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    .gitignore
    mvnw
    mvnw.cmd
    src/main/resources/
    
```

↓
Projenin ilgili repository e entegre edilmesi

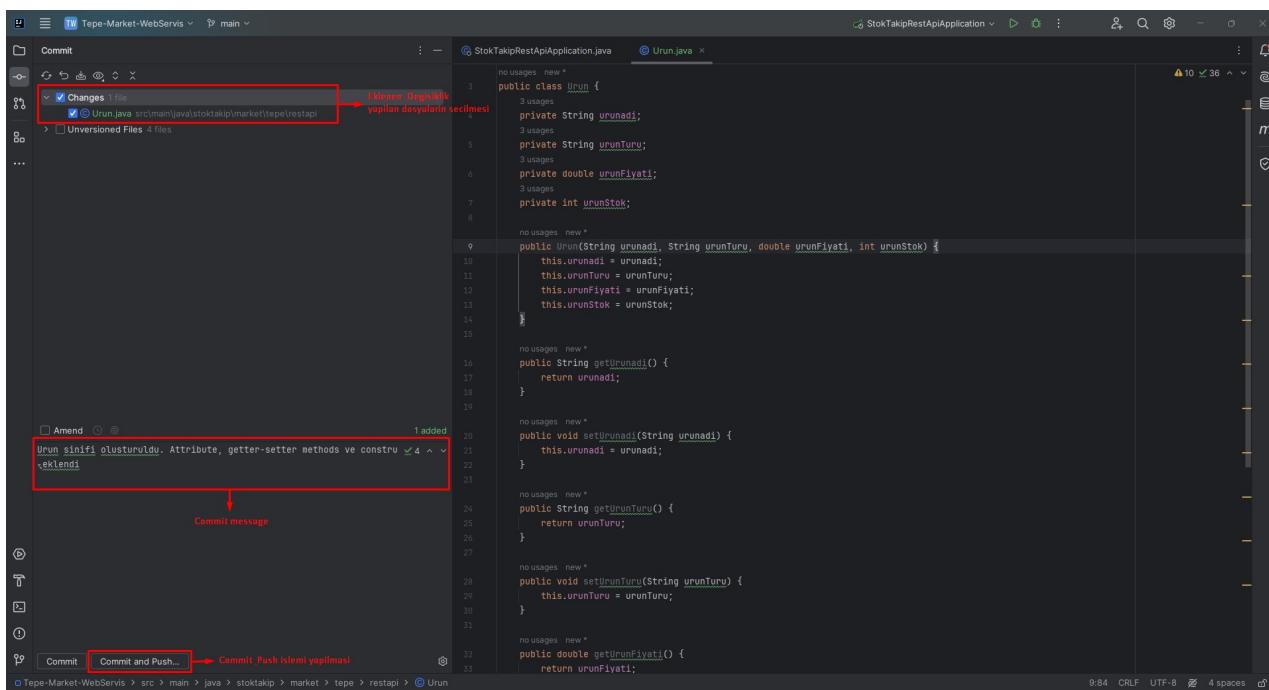
```

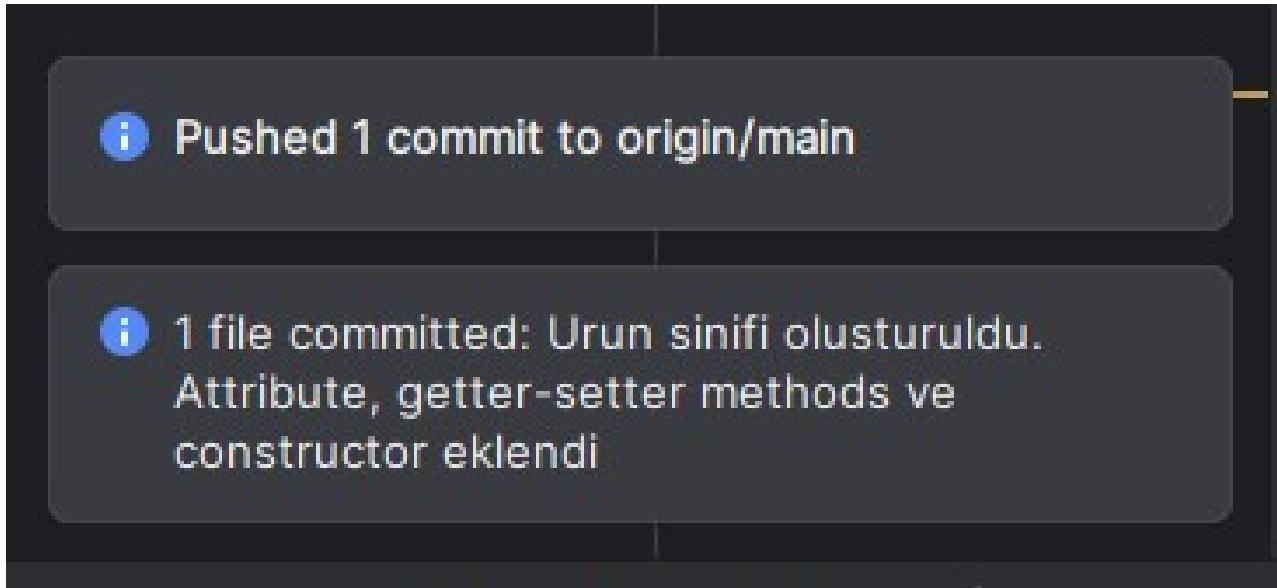
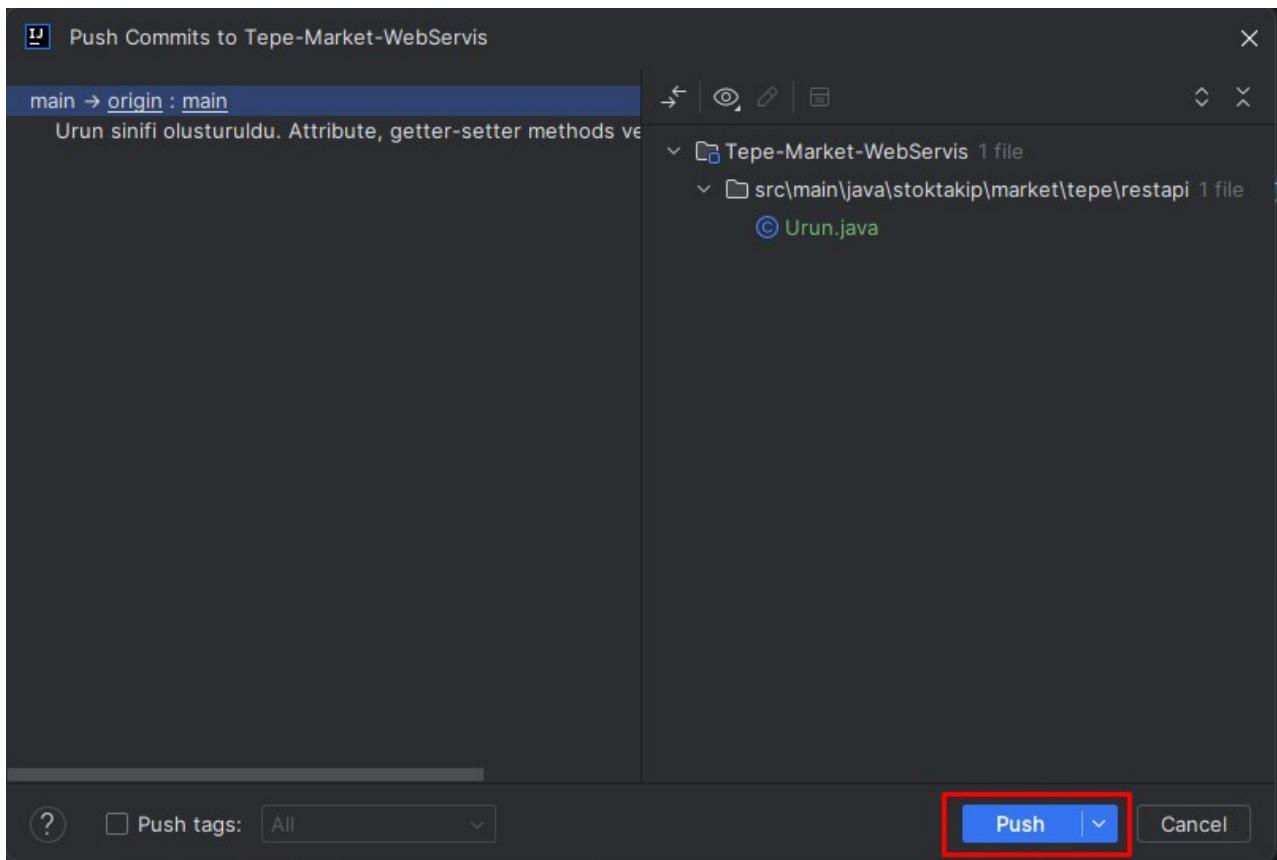
nothing added to commit but untracked files present (use "git add" to track)
PS C:\Users\Monster\IdeaProjects\Tepe-Market-WebServis> git push -u origin main → Commit edilenlerin repository'e pushlanması
Enumerating objects: 22, done.
Counting objects: 100% (22/22), done.
Delta compression using up to 16 threads
Compressing objects: 100% (11/11), done.
Writing objects: 100% (22/22), 56.27 KiB | 14.07 MiB/s, done.
Total 22 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/erdembuке/Tepe-Market-WebServis.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
PS C:\Users\Monster\IdeaProjects\Tepe-Market-WebServis> git status
On branch main
Your branch is up to date with 'origin/main'.
    
```

3- IntelliJ IDEA ile Kodların Repository'e Aktarılması, Commit İşlemi

Projede üzerinde yapılan değişiklikler aktarılmak istendiğinde;

- 1- Soldaki dikey sıralanmış menülerden Commit menu tab tıklanır
- 2- Changes adı altında istenen dosyalar seçilir
- 3- Commit message alttaki text alanına girilir
- 4- Commit and Push ile direkt kodlar aktarılır





Not: Proje'nin rapor dosyası olan ve şu an incelenen Tepe-Market.pdf dosyası, GitHub üzerinden commit yapılarak terminal kullanmadan projeye yüklenmiştir