



ÖZGÜR YAZILIM A.Ş.

STAJ PROJESİ

Spring, JPA, Maven Teknolojileri ile Laboratuvar Raporlama
Uygulaması

Doğacan Özdemir

ANKARA-2024

İçindekiler

TEŞEKKÜR.....	3
1. GİRİŞ	4
1.1. Java Tarihçesi.....	5
1.2. Java Kullanım Alanları.....	6
1.3. Katmanlı Mimari.....	7
1.4. MVC (Model-View-Controller) Yapısı	7
2. KULLANILAN TEKNOLOJİLER	9
2.1. Spring Framework.....	9
2.2. MySQL	14
2.3. Bootstrap.....	15
2.4. Thymeleaf	17
2.5. Apache Tomcat Server.....	17
2.6. Spring Data JPA.....	17
2.7. Lombok.....	18
2.8. Maven.....	18
3. UYGULAMA DETAYLARI	19
3.1. Dosya Yolları.....	19
3.2. Proje Mimarisi	20
3.3 Tomcat Server Üzerinde Çalışır Hali.....	30
3.4. Veritabanı İncelemesi (MySQL)	34
SONUÇLAR VE ÖNERİLER	35
KAYNAKÇA.....	37

TEŞEKKÜR

Öncelikle, staj başvurum kapsamında geliştirmem gereken projenin yürütülmesi sırasında bana yardımcı olan Dr. Öğretim Görevlisi Serkan Savaş'a teşekkürlerimi sunarım. Kendisi, proje süresince verdiği değerli bilgiler ve sağladığı rehberlikle, projenin başarılı bir şekilde tamamlanmasında büyük bir katkı sağlamıştır.

Ayrıca, bu staj fırsatını değerlendirme sürecinde bana destek veren ve katkı sağlayan Özgür Yazılım'a ve Sayın Doruk Fişek'e teşekkür ederim. Verdikleri bu değerli fırsat ve gösterdikleri ilgi için minnettarım.

Bu proje, benim için hem profesyonel hem de kişisel anlamda büyük bir öğrenme deneyimi oldu. Herkese katkılarından dolayı tekrar teşekkür ederim.

Saygılarımla,

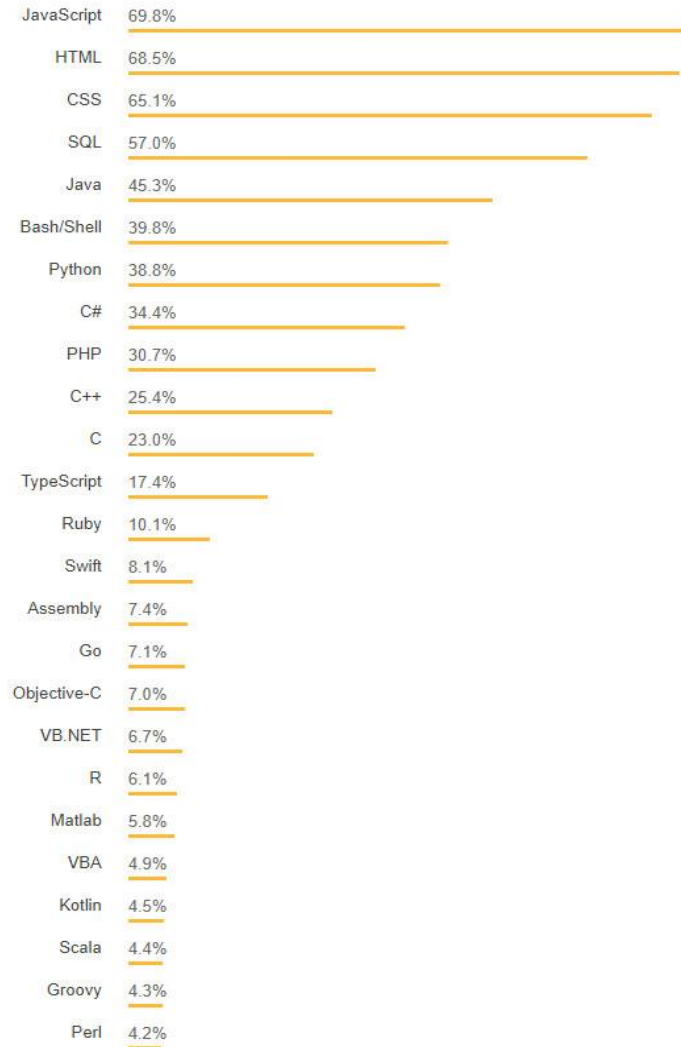
Doğacan Özdemir

1.GİRİŞ

Bilgisayarların hayatımıza girmesi ile birçok problem bilgisayarlar sayesinde çözülmeye başlamıştır. Bu duruma karşılık olarak ortaya birçok programlama dili çıkmıştır.

Programlama dili, yazılımcının bir algoritmayı ifade etmek amacıyla, bir bilgisayara ne yapmasını istediğini anlatmasının tek tipleştirilmiş yoludur. Programlama dilleri, yazılımcının bilgisayara hangi veri üzerinde işlem yapacağını, verinin nasıl depolanıp iletileceğini, hangi koşullarda hangi işlemlerin yapılacağını tam olarak anlatmasını sağlar.

Şu ana kadar 250'den fazla programlama dili ortaya çıkmıştır. Bunlardan bazıları C, C#, C++, Java, JavaScript, Python, Swift, Kotlin gibi dillerdir.



Tablo 1 – Dünyadaki Popüler Programlama Dilleri

1.1. Java Tarihçesi

Java, Sun Microsystems mühendislerinden James Gosling tarafından geliştirilmeye başlanmış açık kaynak kodlu, nesneye yönelik, platform bağımsız, yüksek verimli, çok işlevli, yüksek seviye, hem yorumlanan hem de derlenen bir dildir.

Java, Sun Microsystems'den James Gosling tarafından geliştirilen bir programlama dilidir ve 1995 yılında Sun Microsystems'in çekirdek bileşeni olarak piyasaya sürülmüştür. Bu dil C ve C++'dan birçok sözdizim türetmesine rağmen bu türevler daha basit nesne modeli ve daha az düşük seviye olanaklar içerir. Java uygulamaları bilgisayar mimarisine bağlı olmadan herhangi bir Java Sanal Makinesi (Java Virtual Machine - JVM) üzerinde çalışabilen tipik bytecode'dur (sınıf dosyası).

Versiyon	Yayın Tarihi
JDK Beta	1995
JDK1.0	23 Ocak 1996
JDK 1.1	19 Şubat 1997
J2SE 1.2	8 Aralık 1998
J2SE 1.3	8 Mayıs 2000
J2SE 1.4	6 Şubat 2002
J2SE 5.0	30 Eylül 2004
Java SE 6	11 Aralık 2006
Java SE 7	28 Temmuz 2011
Java SE 8 (LTS)	18 Mart 2014
Java SE 9	21 Eylül 2017
Java SE 10	20 Mart 2018
Java SE 11 (LTS)	25 Eylül 2018
Java SE 12	19 Mart 2019
Java SE 13	17 Eylül 2019
Java SE 14	17 Mart 2020
Java SE 15	15 Eylül 2020
Java SE 16	16 Mart 2021
Java SE 17 (LTS)	14 Eylül 2021
Java SE 18	22 Mart 2022
Java SE 19	13 Eylül 2022
Java SE 20	07 Haziran 2022 **
Java SE 21 (LTS)	07 Aralık 2022 **

Tablo 2 – Java Sürümleri

James Gosling ve Patrick Naughton Java projesini Haziran 1991'de başlattı. Java ilk olarak interaktif televizyonlar için tasarlandı ancak dijital kablo televizyon endüstrisi için o zamanlar çok gelişmişti. Java'nın ilk hali Oak ismini taşıyordu ve bu isimi Gosling'in ofisinin hemen yanında bulunan bir meşe ağacından almıştı. Daha sonra projenin ismi Green oldu ve en son Java adını aldı. Gosling, Java'yı C/C++'a benzer bir syntax ile tasarladı ve böylece programcılar için kolaylıkla öğrenilebilen bir dil oldu.

1.2. Java Kullanım Alanları

Kullanımı ücretsiz ve çok yönlü bir dil olması nedeniyle Java, yerelleştirilmiş ve dağıtılmış yazılımlar oluşturmada kullanılmaktadır. Java'nın yaygın kullanım alanları aşağıdakileri içerir:

1.2.1. Oyun Geliştirme

Mobil oyunlar ve bilgisayar oyunları dâhil birçok popüler video oyunu Java'da oluşturulmaktadır. Makine öğrenimi veya sanal gerçeklik gibi gelişmiş teknolojilerin kullanıldığı modern oyunlar bile Java teknolojisiyle oluşturulmaktadır.

1.2.2. Bulut Bilgi İşlem

Java, WORA [Write Once and Run Anywhere (Bir Kez Yazın ve Her Yerde Çalıştırın)] felsefesine uygun yapısı sayesinde, merkezi olmayan bulut tabanlı uygulamalar için ideal seçimdir. Bulut sağlayıcıları, programlarını çok çeşitli platformlarda çalıştırmak için Java dilini seçmektedir.

1.2.3. Büyük Veri

Java, karmaşık veri kümeleri ve devasa miktarda gerçek zamanlı veri ile birlikte çalışabilecek veri işleme altyapıları için kullanılır.

1.2.4. Yapay Zeka

Java, geniş kapsamlı makine öğrenimi kitaplıkları sunar. Kararlı ve hızlı bir programlama dili olması nedeniyle doğal dil işleme ve derin öğrenme gibi yapay zekâ uygulaması geliştirme çalışmaları için ideal seçimdir.

1.2.5. Nesnelerin İnterneti(IoT)

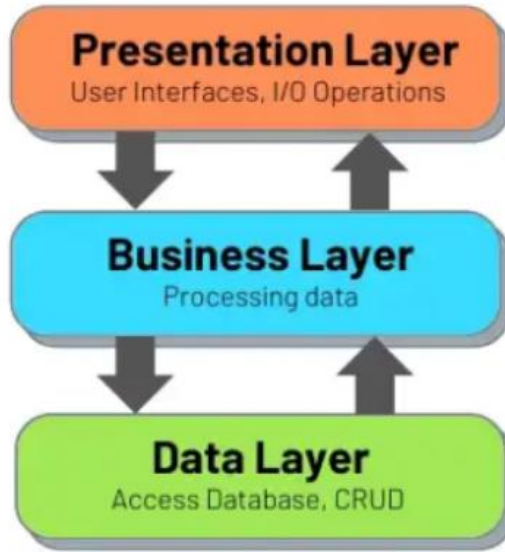
Java, bağımsız olarak internete bağlanabilen uç cihazlardaki sensörleri ve donanımları programlamak için kullanılmaktadır.

1.3. Katmanlı Mimari

Yazılım dünyasında sürdürülebilirlik çok önemlidir. Sürdürülebilirlikten kasıt kodumuzun değişime direnç göstermemesi, prensiplere uygun olması, temiz kod ile yazılması ve dokümantasyonu olmasıdır. Her ne kadar gelişigüzel kod yazarken asıl amacımız kodun çalışması olsa da aslında bir developer(geliştirici) için bu yanlıştır. Yazdığımız bir kodun, geliştirdiğimiz bir projenin okunabilirliği (readability), anlaşılabilirliği, tekrar kullanılabilirliği (reusability), bakım yapılabilirliği(maintainability) çok önemlidir. Bu yüzden projemizi belirli formatlarda geliştirmeliyiz. Bu açıdan mimari yaklaşımlardan yararlanırız.

Java ile uygulama geliştirilirken bazı durumlarda 3 katmanlı mimari kullanılır.

1.3.1. Katmanlı Mimarinin Katmanları



Şekil 1 - Katmanlı Mimari

1.3.2. MVC (Model-View-Controller) Yapısı

MVC (Model-View-Controller), katmanlı bir mimari desendir. Bu mimari desen, uygulamanın farklı sorumluluklarını ayrı bileşenlere böler ve genellikle üç ana bileşeni içerir:

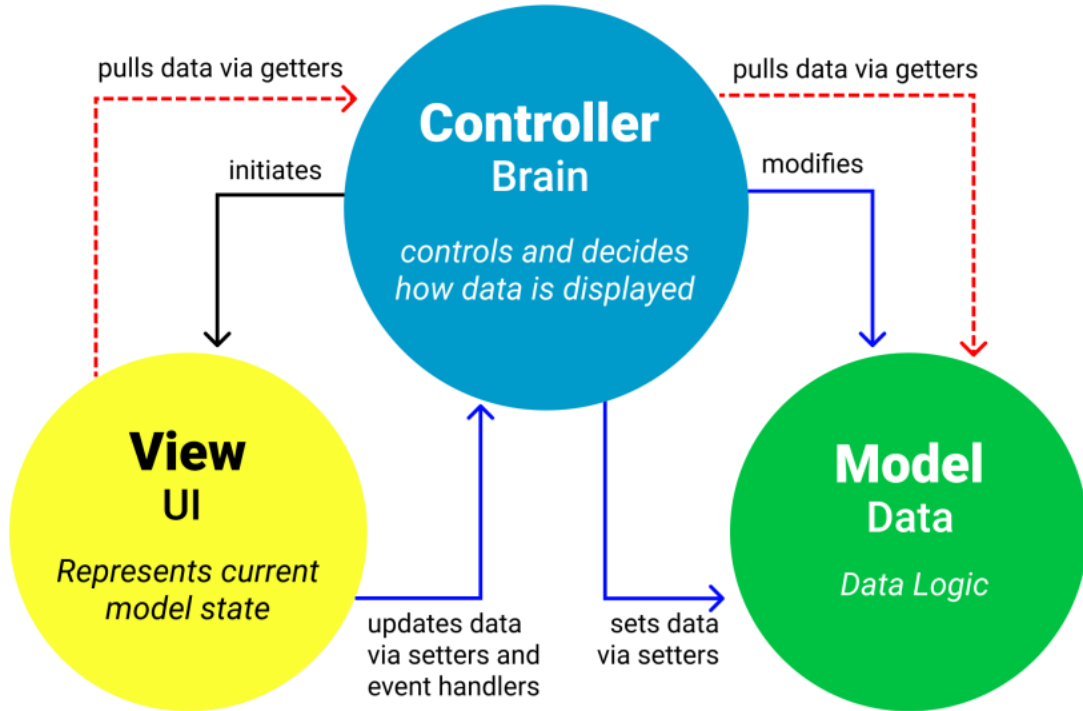
Model (Veri Katmanı): Uygulamanın veri ve iş mantığını içerir. Veritabanı işlemleri, iş kuralları ve veri yönetimi bu katmanda gerçekleştirilir.

View (Görünüm Katmanı): Kullanıcı arayüzünü temsil eder. Modelden gelen veriyi kullanıcıya sunar. Kullanıcı arayüzünün görsel tasarımı ve kullanıcı etkileşimi bu katmanda gerçekleştirilir.

Controller (Kontrolör Katmanı): Kullanıcı etkileşimini yönetir, gelen istekleri alır ve uygun Model ve View bileşenlerine yönlendirir. İşlemlerin yönlendirilmesi ve koordinasyonu bu katmanda gerçekleştirilir.

MVC, birçok modern web uygulamasında kullanılan popüler bir mimari desendir. Örneğin, bir web uygulamasında, Model, veritabanı etkileşimi ve iş mantığıyla ilgilenirken, View, kullanıcı arayüzünü oluşturur ve Controller, kullanıcıların isteklerini alır ve uygun Model ve View işlemlerini başlatır. Bu, uygulamanın daha modüler, bakımı kolay ve yeniden kullanılabilir olmasını sağlar.

MVC Architecture Pattern



Şekil 2- MVC Mimari Yapısı

2.Kullanılan Teknolojiler

2.1. Springboot

Spring Framework Java için geliştirilmiş, açık kaynak olan bir uygulama geliştirme framework'üdür.Spring Framework'ün temel özellikleri herhangi bir Java uygulaması tarafından kullanılabilir.



Şekil 3- Springboot

Eklentileri ile birlikte Java EE platformu üzerinde web uygulamaları geliştirmek için de kullanılabilir. Spring Framework Java toplulukları arasında Enterprise JavaBeans (EJB) modelinin yerine geçebilecek

popüler bir alternatif haline gelmiştir.

2.1.1. Springboot Anotasyonları

2.1.1.1. @SpringBootApplication Anotasyonu

@SpringBootApplication anotasyonu uygulamanın giriş metodunu belirtir. Yani halk arasındaki tabir ile main fonksiyondur. Uygulama bu metod ile başlar.

2.1.1.2. @Entity Anotasyonu

@Data ile belirtilen sınıf içerisindeki oluşturulan Getter ve Setter metotlarını veritabanı ile eşleştirir.

2.1.1.3. @Table Anotasyonu

İlgili veritabanındaki table ile eşleşmeyi sağlar. @Table(name = "tablename") şeklinde kullanılır.

2.1.1.4. @Id Anotasyonu

İlgilin verinin veritabanı içerisinde Primary Key olarak tanımlanmasını sağlar.

2.1.1.5. @GeneratedValue Anotasyonu

Primary Key olarak tanımlanan verinin Auto Increment (Otomatik değer alması) olarak tanımlanmasını sağlar.

2.1.1.6 @Controller Anotasyonu

İlgili sınıfın bir Controller sınıfı olduğunu belirten anotasyondur.

2.1.1.7. @Autowired Anotasyonu

Controller sınıfındaki metotlar ile nesnelerin bağımlılığını sağlar.

2.1.1.8. @GetMapping Anotasyonu

İlgili metotun bağlanacağı katman ile mapping işlemini gerçekleştirir. @GetMapping("/erişim yolu") şeklinde tanımlanır.

2.1.1.9. @PostMapping Anotasyonu

İlgili metotun bağlanacağı katman ile mapping işlemini gerçekleştirir.

2.1.1.10. @RequestParam Anotasyonu

İlgili metotun gerçekleşebilmesi için bir parametreye ihtiyacı olduğunu belirten anotasyondur. @RequestParam(veri_tipi veri_adı) şeklinde tanımlanabilir.

2.1.1.11. @Repository Anotasyonu

Sorgulama ve filtreleme ihtiyaçları karşısında ortaya çıkmış bir anotasyondur. findByAll() gibi hazır metotların kullanımında yararlanır.

2.1.1.12. @ModelAttribute Anotasyonu

Bir metodu veya Controller sınıfındaki bir metodu işaretlemek için kullanılır. Bu anotasyon, HTTP isteği sırasında model nesnelerini veya model özelliklerini metoda parametre olarak eklemek için kullanılır.

2.1.1.13. @PathVariable Anotasyonu

Bir Controller metodu içinde URL'de bulunan değişkenleri almak için kullanılır.

2.1.1.14. @Valid Anotasyonu

Genellikle HTTP isteklerinden gelen verilerin doğrulanması için kullanılır. Özellikle form verilerini işlerken,

gelen verilerin belirli kriterlere uymasını sağlamak için sıkça kullanılır. Bu projede hata yönetimini sağlamak için kullanılmıştır.

2.1.2. pom.xml Dosyası

POM, xml dosya formatında olan ve bir projenin build edilmesinden, nasıl edileceğine, bağımlılıklarından, packaging işlemine kadar proje ile ilgili olan her türlü bilgiyi içerisinde barındıran bir dosyadır. Maven'in son versionunda pom.xml olarak adlandırılır, önceki versiyonlarında farklı isimlendirmeler vardı.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.2.6-SNAPSHOT</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.doa</groupId>
  <artifactId>lab-master-v1.0</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>lab-master-v1.0</name>
  <description>Demo project for Spring Boot</description>
  <properties>
    <java.version>21</java.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jdbc</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
  </dependencies>
```

Şekil 4-pom.xml Dosyası Örneği

POM içerisinde bazı tag'ler kullanılır :

- **Packaging** : Paketleme tipini belirtir, default olarak jar'dır. Jar, War, Rar vs gibi tipler kullanılabilir.
- **Name** : Projeye isim vermek içindir.
- **URL** : Proje'nin adres niteliğindedir.
- **Dependencies** : Uygulama bağımlılıklarının tanımlandığı kısımdır.
- **Scope** : Proje ortamını belirler, compile, test, runtime olabilir.

2.1.3. application.properties Dosyası

Spring Boot varsayılan olarak yapılandırma ayarlarını *src/main/resources* dizini altındaki *application.properties* dosyasında tutar. Properties dosyaları kolay okunabilirlik ve yazım açısından oldukça kolay bir formattadır. Ancak, Spring Boot YAML formatına da destek verir ve iki açıdan da daha büyük kolaylık sağlar.

Properties için; *application.properties*, YAML için ise; *application.yml* dosyaları varsayılan olarak Spring tarafından tanınır.

Bu, konfigürasyonlarınızın ortam bazlı dinamik olarak yapılandırılabilmesi ve çoklu ortam kurulumları için çok kullanışlıdır.

```
spring.application.name=lab-master-v1.0
spring.datasource.url=jdbc:mysql://localhost:3306/lab_master_db
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.hibernate.ddl-auto=update

spring.servlet.multipart.enabled=true
spring.servlet.multipart.max-file-size=2MB
spring.servlet.multipart.max-request-size=2MB

file.upload-dir=/resources/uploads/

spring.security.user.name=user
spring.security.user.password=pass
spring.thymeleaf.cache=false
```

Şekil 5 - application.properties Dosyası

Şekil-5'deki bazı komutların amaçları:

- **spring.datasource.url** : Veritabanının yolunu gösterir.
- **spring.datasource.username** : Veritabana erişim için gerekli olan kullanıcı adını gösterir.
- **spring.datasource.password** : Veritabanına erişim için gerekli olan şifreyi gösterir.
- **server.port** : Tomcat Server'ın çalışacağı localhost adresini gösterir.
- **spring.security.user.name** : Spring Security'nin auth için kullandığı kullanıcı adını tanımlar.
- **spring.security.user.password** : Spring Security'nin auth için kullandığı şifreyi tanımlar. (Bu komut olmaması durumunda Spring Security rastgele bir şifre tanımlayacaktır.)

2.1.4. Spring Security

Spring framework kullanılarak geliştirilen doğrulama(Authentication), yetkilendirme(Authorization), şifreleme>Password Encoder) ve CSRF gibi güvenlik önlemleri sağlayan, Spring platformunda yer alan bir projedir.

Kullanıcılar uygulamalardan beklenen işlevin yanından kullanıcıların sahip olduğu yetkilere(yönetim, editör veya üye) göre işlem yapması istenebilir.

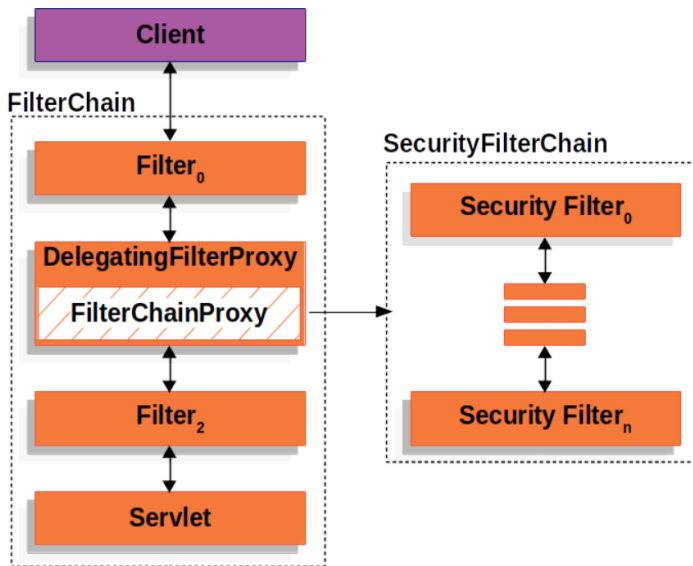
Uygulama bu ihtiyaç ile birlikte doğrulama, yetkilendirme ve saldırganın yetkileri elde ederek uygulama işlevini değiştirmesini önleme gibi güvenlik tedbirlerine gereksinim duyulur.

Her bir gereksinim içerisinde başka gereksinimlere ihtiyaç duyabilir.

Örneğin; Doğrulama sıradan bir dosyada yer alan kullanıcı bilgilerinin kontrolü, veritabanından kontrolü veya LDAP gibi protokollerden kontrolü olabilir.

Temel doğrulama ve yetkilendirmenin yanında saldırganlar sürekli olarak farklı saldırı yöntemleri kullanılarak güvenlik açıkları ortaya çıkarmaktadır.

Tüm bu güvenlik gereksinimleri uygulamadan beklenen işlevselliği yerine getirmek için kullanılan efor kadar efora neden olacaktır.



Şekil 6 - Spring Security Çalışma Katmanları

Spring Security doğrulama, yetkilendirme, şifreleme, güvenlik önlemlerini esnek, kolay ve sürekli olarak paketlerin güncellenmesi ile sağlar.

2.2. MySQL Veritabanı

MySQL, altı milyondan fazla sistemde yüklü bulunan çoklu iş parçacıklı (İng. İngilizce: multi-threaded), çok kullanıcı (İng. İngilizce: multi-user), hızlı ve sağlam (İng. İngilizce: robust) bir veri tabanı yönetim sistemidir.

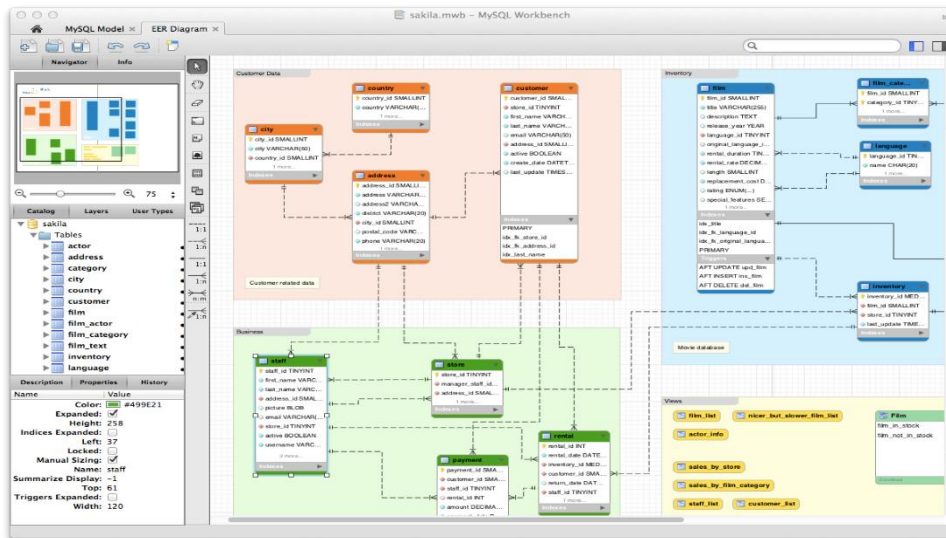
UNIX, OS/2 ve Windows platformları için ücretsiz dağıtılmakla birlikte ticari lisans kullanmak isteyenler için de ücretli bir lisans seçeneği de mevcuttur. Linux altında daha hızlı bir performans sergilemektedir. Kaynak kodu açık olan MySQL'in pek çok platform için çalıştırılabilir ikilik kod halindeki indirilebilir sürümleri de mevcuttur. Ayrıca ODBC sürücülerini de bulunduğu için birçok geliştirme platformunda rahatlıkla kullanılabilir.

Geliştiricileri, 500'den fazlası 7.000.000 kayıt içeren 10.000 tablodan oluşan kendi veritabanlarını (100 gigabyte civarında veri) MySQL'de tuttuklarını söylüyorlar.

Web sunucularında en çok kullanılan veri tabanı olup ASP, PHP gibi birçok Web programlama dili ile kullanılabilir.

MySQL aşağıdaki veritabanı nesnelerini desteklemektedir.

- Tables (tablolar)
- Views (görüntü(leme)ler)
- Procedures (prosedürler)
- Triggers (tetikleyiciler)
- Cursors (imleçler)



Şekil 7 - MySQL Workbench Örneği

Ayrıca sahip olduğu MySQL Workbench çalışma ortamı ile kolaylıkla veritabanı yönetimi yapmamızı sağlar.

2.3. Bootstrap

Bootstrap'i incelemeden önce, web uygulamalarının ön uç iskeletlerinin ne olduğuna yakından bakmakta fayda var. Bu iskeletler, bir uygulamanın kullanıcıların etkileşime geçtiği arayüzüdür. Yapıyı HTML kodları meydana getirir, basamaklı stil şablonları görsel formatı oluşturur ve JavaScript kodları slaytlar, açılan menüler gibi dinamik elementleri mümkün kılar.

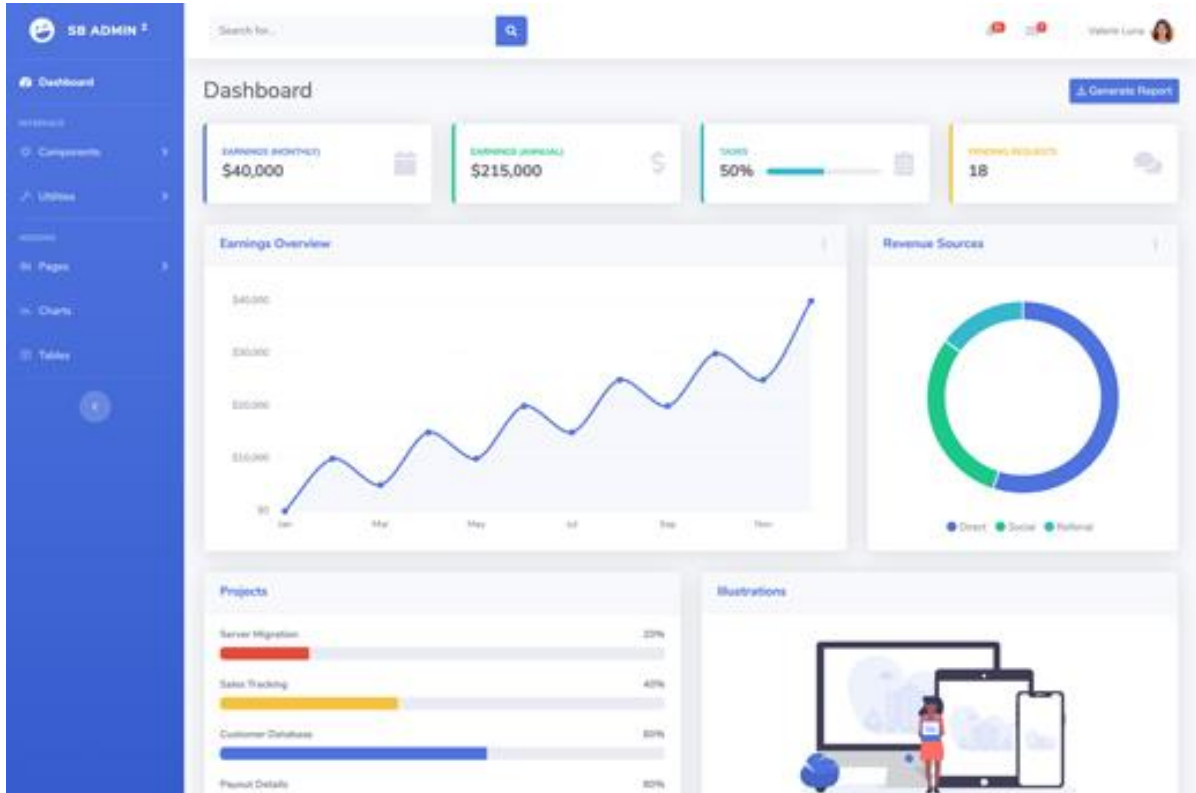
İskeletler, web sitelerinin üzerlerine inşa edildikleri temellerdir. Bir ön uç iskeleti ve araç takımı olan Bootstrap, Twitter mühendisleri Mark Otto ve Jacob Thornton tarafından geliştirilmiştir. Temel amacı web uygulamalarının hızlı, kullanışlı ve daha duyarlı hale getirilmesini kolaylaştırmaktır. Bootstrap günümüzde duyarlı ve mobil öncelikli web siteleri geliştirmek için en popüler CSS iskeleti olma unvanına sahiptir.

Bootstrap'in avantajları şunlardır :

- **Daha hızlı geliştirme:** Bootstrap'in en büyük avantajlarından bir tanesi web geliştirme projelerini hızlandırması ve bunu yaparken kaliteden ödün verilmesini gerektirmemesidir. Bootstrap olmadan duyarlı web sitesi ve uygulama geliştirmek çok uzun sürebilirken, bu araç takımı gereken süreyi bir hayli kısaltır.
- **Kolay kullanılabilir:** Giriş seviyesi HTML ve CSS bilgisine sahip olan herkes Bootstrap kullanarak web siteleri, uygulamalar ve temalar geliştirebilir. Java ile Veteriner Kliniği Otomasyonu Doğacan Özdemir 18 □ **Duyarlı tasarım:** Yukarıda bahsettiğimiz duyarlı tasarımları kolaylaştırması Bootstrap'in alametifarikalarından bir tanesi. Bootstrap ile geliştirilen duyarlı web siteleri tüm ekran boyutlarına göre otomatik adapte olur.
- **Tasarım devamlılığı:** Bootstrap ile tasarlanan bir web sayfası Firefox, Chrome, Edge, Opera ve Safari gibi tüm modern web tarayıcılarında aynı şekilde görünür.
- **Açık kaynaklı:** Bootstrap'in en iyi özelliği açık kaynaklı ve tamamen ücretsiz olmasıdır. Birçok geliştirici bu platform üzerinde geliştirdiği kaynakları ücretli ve ücretsiz olarak

diğerlerinin kullanıma sunar. Bu kaynaklar web siteleri ve uygulamaları geliştirmek için kullanılabilir.

- **Özelleştirme:** Bootstrap, web sitenizde ve uygulamanızda olduğu gibi kullanabileceğiniz ön yüklü bileşenlerle gelir. Navigasyon barları, açılır kapanır menüler, butonlar ve dahası gibi tasarım öğelerinden istediğinizi seçip kullanabilirsiniz. Bootstrap'in web sitesinden araç takımını indirmeden önce bazı özellikleri seçmeniz ve yalnızca istediklerinizi indirip kullanmaya başlamanız da mümkündür.
- **Dokümantasyon:** Bootstrap'i keşfetmek istiyorsanız, web sitesinde bulunan ve her kod parçası için tanımlayıcı ve açıklayıcı içerikler sunan dokümanlarından faydalanabilirsiniz. Bu açıklamalara eklenen kod örnekleri, yeni başlayan geliştiricilerin işini önemli ölçüde kolaylaştırır.
- **Temalar:** Bootstrap temelli geliştirilen web temaları yeni bir pazar haline dönüştü. Geliştiriciler ve tasarımcılar tarafından tasarlanan Bootstrap tabanlı şablonları alıp doğrudan kendi projelerinizde kullanabilir ya da kendi projelerinizi satışa çıkarabilirsiniz.



Şekil 8 - Bootstrap ile Oluşturulmuş Sayfa Örneği

2.4. Thymeleaf

Thymeleaf; açık kaynak kodlu ve kendini kanıtlamış, eklentiler (dialects) ile özelleştirilebilen, Spring Boot ve diğer Spring Framework projeleriyle tam uyumlu bir şablon motorudur.

Bunların yanı sıra, benim tercih etmemdeki en etkili sebep; XML ad uzayları ile direkt HTML etiketine uygulanması, dolayısıyla da sunum tarafındaki gereksiz kalabalıktan arındırması oldu.

2.5. Apache Tomcat Server

Apache Tomcat veya Tomcat Java tabanlı web uygulamalarını yayınlamak için kullanılan web sunucusudur.

Java, Java EE veya Java Teknolojileri içerisinde Java Servlet, JavaServer Pages, Java Expression Language, Java WebSocket gibi çeşitli teknolojiler yer alır.

Bu teknolojiler JCP (Java Community Process) olarak adlandırılan ve genellikle çeşitli firmalardaki geliştiriciler tarafından standart olarak belirlenir.

Standartlar belirlendikten sonra bu teknolojilerin kullanılabilmesi için bu standartların kodlara dökülmesi-implement gerekir.

Apache Tomcat bu standartları uygulayan ve içerisinde web sunucusu yer alan bir Java uygulamasıdır.

2.6. Spring Data JPA

Spring Data JPA, Java Persistence API (JPA) üzerine kurulmuş bir çerçeve olup, veri erişim katmanını kolaylaştırır. CRUD operasyonlarını, method adlarına dayalı sorguları ve pagination ile sorting işlemlerini basitçe gerçekleştirmenizi sağlar.

Ayrıca, JPQL ve native SQL sorgularını destekler ve Spring'in transaction yönetimi ile entegre çalışır. Spring Data JPA, daha az kodla daha okunabilir ve bakım kolaylığı yüksek veri erişim katmanları oluşturmanıza olanak tanır. Auditing (denetim) yetenekleri sayesinde veritabanı değişikliklerini izlemek ve kaydetmek de mümkündür.

2.7.Lombok

Lombok, Java projesi geliştirirken IDE'ye entegre edilebilen bir anotasyon ile kod üretme (code generation) kütüphanesidir. Lombok ile daha temiz ve daha az kod yazmış oluruz.

Java'da proje geliştirirken yaygın olarak yapmamız gereken bazı işlemler bulunmakta. Bunlar projemizin iş tarafına gerçek bir değer getirmez iken kodumuzun çok fazla ayrıntı barındırmasını zorunlu hale getiriyor. Bu durumlarda Lombok ile çözümü sağlayabiliriz.

2.8. Maven

Maven genellikle Java platformunda yer alan komutların derlenmesi sırasında kullanılan otomasyon ve inşa aracıdır.

Java programlama dili ile uygulama geliştirirken çeşitli kütüphaneler kullanamak isteyebiliriz. Örneğin; Java ile PDF dosyası oluşturmak için Apache PDFBox, iText, JPOD gibi çeşitli kütüphaneleri kullanabiliriz.

Her kütüphane için gerekli olan JAR dosyalarını indirmek ve projeye uygun olarak yerleştirmek (classpath) gerekir. Ancak sadece kütüphanelerin indirilmesi ve projeye dahil edilmesi yetmeyecektir. Ayrıca her yeni güncelleme sonrası güncel dosyaların takip edilmesi gerekecektir.

Maven proje dosyasına eklenen bağımlılıklar ile kolay bir şekilde indirmeyi ve proje yerleştirmeyi sağlar. Kullanılan kütüphaneler proje dosyasında yer aldığından taşınabilirlik sağlanmış olur.

Sunmuş olduğu dizin yapısı sayesinde diğer geliştiricilerin projeyi takibini kolaylaştır.

3. UYGULAMA DETAYLARI

3.1. Dosya Yolları

```

lab-master-v1.0
├── src
│   ├── main
│   │   ├── java
│   │   │   ├── com
│   │   │   │   ├── doa
│   │   │   │   │   ├── labmasterv10
│   │   │   │   │   │   ├── Config
│   │   │   │   │   │   │   ├── ResourceNotFoundException.java
│   │   │   │   │   │   │   ├── SecurityConfig.java
│   │   │   │   │   │   ├── Controller
│   │   │   │   │   │   │   ├── HomeController.java
│   │   │   │   │   │   │   ├── LaborantController.java
│   │   │   │   │   │   │   ├── ReportController.java
│   │   │   │   │   │   ├── Entities
│   │   │   │   │   │   │   ├── Laborant.java
│   │   │   │   │   │   │   ├── Report.java
│   │   │   │   │   │   ├── Repository
│   │   │   │   │   │   │   ├── LaborantRepository.java
│   │   │   │   │   │   │   ├── ReportRepository.java
│   │   │   │   │   │   ├── Service
│   │   │   │   │   │   │   ├── LaborantService.java
│   │   │   │   │   │   │   ├── ReportService.java
│   │   │   │   │   │   └── Application.java
│   │   │   │   ├── resources
│   │   │   │   │   ├── application.properties
│   │   │   │   │   ├── static
│   │   │   │   │   │   ├── images
│   │   │   │   │   ├── templates
│   │   │   │   │   │   ├── createLaborant.html
│   │   │   │   │   │   ├── createReport.html
│   │   │   │   │   │   ├── editLaborant.html
│   │   │   │   │   │   ├── editReport.html
│   │   │   │   │   │   ├── index.html
│   │   │   │   │   │   ├── laborantDetails.html
│   │   │   │   │   │   ├── laborants.html
│   │   │   │   │   │   ├── reportDetails.html
│   │   │   │   │   │   ├── reports.html
│   │   │   │   │   ├── uploads
│   │   │   │   │   │   └── **Hasta dosyalarının yüklendiği ve tutulduğu dizin**
│   │   └── test
│   │       ├── java
│   │       │   ├── com
│   │       │   │   ├── example
│   │       │   │   │   ├── demo
│   │       │   │   │   │   ├── ApplicationTests.java
│   │   ├── mvnw
│   │   ├── mvnw.cmd
│   │   ├── pom.xml
│   │   ├── Dockerfile
│   │   └── README.md

```

3.2. Proje Mimarisi

3.2.1. Config Paketi

Bu kısımda *ResourceNotFoundException.java* sınıfı ile rapor güncelleme ve silme işlemleri yaparken ilgili ID'ye sahip rapor getirilemezse oluşacak hata kontrolünü sağladık. Bu kısım RuntimeException sınıfını kullanarak derleme zamanının dışında kontrol edilen bir hata kontrol yönetimi sağlamaktadır.

```
public class ResourceNotFoundException extends RuntimeException { 6 usages  ⬆ Doğacan Özdemir
    public ResourceNotFoundException(String message){ 3 usages  ⬆ Doğacan Özdemir
        super(message);
    }
}
```

Şekil 9- ResourceNotFoundException.java

3.2.2. Repository Paketi

Bu paket içerisinde *LaborantRepository.java* ve *ReportRepository.java* dosyaları bulunmaktadır. İlgili dosyalar jakarta.util kütüphanesinden Repository metodunu kullanarak Controller katmanında bir repo oluşturulmasını sağlar.

```
public interface LaborantRepository extends JpaRepository<Laborant, Long> { 2 usages  ⬆ Doğacan Özdemir
    Optional<Laborant> findByHospitalID(Long hospitalID); 1 usage  ⬆ Doğacan Özdemir
}
```

Şekil 10- LaborantRepository.java

3.2.3. Entities Paketi

Bu paket içerisinde *Laborant.java* ve *Report.java* dosyaları bulunmaktadır. İlgili dosyalar gerekli nesnelerin oluşumunu ve get() , set() edilmesini sağlar. Ayrıca veritabanının ilgili kısmında tabloların ve gereksinimlerin oluşturulmasını sağlar.

- **@Entity** : Bu annotation, bir Java sınıfının JPA (Java Persistence API) varlığı olduğunu belirtir. Bu sınıfı veritabanı tablosuyla ilişkilendirir ve veritabanında bir varlık olarak saklanmasını sağlar.
- **@Data** : Lombok tarafından sağlanır ve bir sınıfa Getter, Setter ve diğer genel yöntemleri otomatik olarak ekler. Bu, kod tekrarını azaltır ve sınıfın daha kısa ve okunabilir olmasını sağlar.

```

@Entity 24 usages  ⤴ Doğan Özdemir
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Report {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String patientFirstName;
    private String patientLastName;

    @NotBlank(message = "T.C. Kimlik Numarası boş bırakılamaz.")
    @Pattern(regexp = "\\d{11}", message = "T.C. Kimlik Numarası sadece rakamlardan ve 11 karakterden oluşmalıdır.")
    @Column(unique = true)
    private String patientTC;

    private String diagnoseTitle;
    private String diagnoseDetails;
    private LocalDate reportDate;

    @ManyToOne
    @JoinColumn(name = "laborant_id", referencedColumnName = "id")
    private Laborant laborant;

    private String reportImagePath;
}

```

Şekil 11- Report.java

- **@AllArgsConstructor** : Lombok tarafından sağlanır ve bir sınıf için tüm alanları içeren bir constructor oluşturur. Bu, nesne oluştururken uzun constructor tanımlamalarından kaçınmak için kullanılır ve sınıfın tüm alanlarına değer ataması yapar.
- **@NoArgsConstructor** : Lombok tarafından sağlanır ve parametresiz bir constructor ekler. Bu, özellikle Hibernate gibi ORM kütüphaneleriyle uyumlu olması için gereklidir ve sınıfın nesne oluşturulmasını sağlar.
- **@Id** : JPA varlığının birincil anahtar (Primary Key) alanını belirtir. Bu annotation, bir sınıfın veya birincil anahtar alanının üzerine yerleştirilir ve JPA, bu alanı veritabanında birincil anahtar olarak kullanır.
- **@NotBlank** : İlgili alanın boş olmamasını belirtir. Yani, bu alanın değeri null, boş bir String veya boş bir dizi olamaz. Eğer bu koşul sağlanmazsa, belirtilen mesajı döndürür.
- **@Pattern** : Alanın değerinin belirtilen regex desenine (şablon) uygun olması gerektiğini belirtir. Bu durumda, alanın sadece 11

rakamdan oluşması gerektiğini belirtir. Eğer bu koşul sağlanmazsa, belirtilen mesajı döndürür.

- **@Column** : İlgili alanın veritabanında benzersiz olması gerektiğini belirtir. Yani, aynı değere sahip iki kayıt olamaz. Eğer bu koşul sağlanmazsa, ilgili veritabanı sorgusu bir hata üretecektir.
- **@ManyToOne** : İlişkisel bir alanın çoktan bir ilişki olduğunu belirtir. Yani, bir Laborant nesnesi birçok bu alan ile ilişkilendirilmiş olabilir.
- **@JoinColumn** : Bir ilişkisel alanın hangi sütunla ilişkilendirileceğini belirtir. Bu durumda, "laborant_id" sütunu ile "id" sütunu arasında bir ilişki belirlenmiştir. Bu, veritabanında dış anahtar (foreign key) ilişkisini temsil eder.

```
@Column(unique = true)
@Min(value = 1000000, message = "Hastane ID 7 karakter olmalı")
@Max(value = 9999999, message = "Hastane ID 7 karakter olmalı")
private Long hospitalID;
```

Şekil 12 - Laborant.java sınıfının bir kısmı

- **@Min** : İlgili alanın değerinin belirtilen minimum değere (1000000) eşit veya büyük olması gerektiğini belirtir. Bu durumda, Hastane ID'nin 7 karakter olması gerektiği belirtilmiştir. Eğer bu koşul sağlanmazsa, belirtilen mesajı döndürür.
- **@Max** : İlgili alanın değerinin belirtilen maksimum değere (9999999) eşit veya küçük olması gerektiğini belirtir. Bu durumda, Hastane ID'nin 7 karakter olması gerektiği belirtilmiştir. Eğer bu koşul sağlanmazsa, belirtilen mesajı döndürür. Bu mesajın gönderilmesiyle ilgili durumlar Controller kısmında tanımlanmıştır. Raporun ileri kısımlarında inceleyeceğiz.

3.2.4. Service Paketi

Bu paket içerisinde *LaborantService.java* ve *ReportService.java* dosyaları bulunmaktadır. İlgili dosyalar gerekli iş parçalarının oluşturulmuş olduğu yapılar olup ilgili işleri yapmakla görevlidir.

```
@Service 4 usages  ⤴ Doğacan Özdemir *
public class LaborantService {

    @Autowired
    private LaborantRepository laborantRepository;

    public List<Laborant> getAllLaborants() { 5 usages  ⤴ Doğacan Özdemir
        return laborantRepository.findAll();
    }

    public Optional<Laborant> getLaborantById(Long id) { 2 usages  ⤴ Doğacan Özdemir
        return laborantRepository.findById(id);
    }

    public Laborant saveLaborant(Laborant laborant) { 1 usage  ⤴ Doğacan Özdemir
        validateHospitalID(laborant.getHospitalID(), laborant.getId());
        return laborantRepository.save(laborant);
    }

    public void deleteLaborant(Long id) { 1 usage  ⤴ Doğacan Özdemir
        laborantRepository.deleteById(id);
    }
}
```

Şekil 13 - LaborantService.java #1

- **getAllLaborants()**: Tüm laborantları veritabanından alır ve bir liste olarak döndürür. Bu metod, laborantların listesini alma işlevini yerine getirir.
- **getLaborantById(Long id)**: Belirli bir laborantı veritabanından kimliği (id) kullanarak alır ve Optional bir nesne olarak döndürür. Optional olarak döndürmesinin sebebi null referansları değerlendirmektir. Bu metod, veritabanından belirli bir laborantı alma işlevini yerine getirir.

- **saveLaborant(Laborant laborant):** Yeni bir laborantı veritabanına kaydeder veya mevcut bir laborantı günceller. Kaydedilecek laborant nesnesini parametre olarak alır, önce bu laborantın hastane kimliğinin doğruluğunu kontrol eder ve sonra veritabanına kaydeder. Son olarak, kaydedilen veya güncellenen laborant nesnesini döndürür.
- **deleteLaborant(Long id):** Belirli bir laborantı veritabanından siler. Bu metod, veritabanından belirli bir laborantı silme işlevini yerine getirir.

```
public Laborant updateLaborant(Laborant laborant) { 1 usage  ⬆ Doğacan Özdemir *
    validateHospitalID(laborant.getHospitalID(), laborant.getId());
    Optional<Laborant> existingLaborantOpt = laborantRepository.findById(laborant.getId());
    if (existingLaborantOpt.isPresent()) {
        Laborant existingLaborant = existingLaborantOpt.get();

        existingLaborant.setHospitalID(laborant.getHospitalID());
        existingLaborant.setLaborantfirstName(laborant.getLaborantfirstName());
        existingLaborant.setLaborantlastName(laborant.getLaborantlastName());

        existingLaborant.getRaporlar().clear();
        for (Report newReport : laborant.getRaporlar()) {
            newReport.setLaborant(existingLaborant);
            existingLaborant.getRaporlar().add(newReport);
        }

        return laborantRepository.save(existingLaborant);
    } else {
        throw new IllegalArgumentException("Bu ID'ye sahip bir laborant bulunamadı : " + laborant.getId());
    }
}
```

Şekil 14 - LaborantService.java #2

- **updateLaborant(Laborant laborant) :** Bu metod, laborantların güncellenmesini sağlar. Öncelikle, laborantın hastane kimliğinin geçerli olduğunu doğrular. Daha sonra, güncellenecek laborantın kimliğini kullanarak veritabanından ilgili laborantı arar. Eğer laborant bulunursa, bu bilgi bir Optional nesnesi içinde döner. Eğer laborant mevcut değilse, bir IllegalArgumentException fırlatılır ve işlem sonlandırılır.

Laborant bulunduğunda, mevcut laborantın bilgileri güncellenir. Bu güncelleme işlemi, laborantın hastane kimliği, adı, soyadı ve raporları gibi alanları içerir. Yenilenmiş laborant bilgileri veritabanına kaydedilir ve işlem başarılı olursa güncellenmiş laborant nesnesi döndürülür. Ancak, laborant

bulunamadığı durumda bir hata mesajı ile birlikte işlem sonlandırılır. Bu, kullanıcıların mevcut olmayan bir laborantı güncellemeye çalıştığında uygun bir geri bildirim almasını sağlar.

```
private void validateHospitalID(Long hospitalID, Long laborantId) { 2 usages  ⚙ Doğacan Özdemir
    Optional<Laborant> Laborant = laborantRepository.findByHospitalID(hospitalID);
    if (Laborant.isPresent() && !Laborant.get().getId().equals(laborantId)) {
        throw new IllegalArgumentException(hospitalID + " numarası ile kayıtlı başka bir laborant mevcut.");
    }
}
```

Şekil 15 - LaborantService.java #3

- **validateHospitalID(Long hospitalID, Long laborantId)** : Bu metod, verilen hastane kimliği (hospitalID) ile veritabanında başka bir laborantın zaten kayıtlı olup olmadığını kontrol eder.

Öncelikle, veritabanından belirtilen hastane kimliği ile eşleşen laborantı arar. Eğer böyle bir laborant varsa, bu bilgi bir Optional nesnesi içinde döner. Ardından, eğer bu laborantın varlığı doğrulandıysa ve bu laborantın kimliği (id) verilen laborantın kimliği ile farklı ise, bir hata fırlatılır. Bu durum, aynı hastane kimliği ile birden fazla laborantın kayıtlı olamayacağını kontrol eder. Eğer laborant zaten kayıtlıysa ve kimliği farklı ise, bir IllegalArgumentException fırlatılır ve işlem sonlandırılır.

Bu, yeni bir laborant eklenirken veya mevcut bir laborantın bilgileri güncellenirken aynı hastane kimliği ile kaydedilen laborantları engeller. Bu da veritabanındaki verilerin bütünlüğünü korur.

3.2.5. Controller Paketi

Bu paket altında *LaborantController.java*, *ReportController.java* ve *HomeController.java* dosyaları bulunmaktadır. İlgili dosyalar gerekli Mapping işlemleri GET ve POST işlemlerini sağlamaktadır. *HomeController.java* dosyası altında Ana Sayfa'nın Mapping işlemleri tanımlanmıştır.

```
@Controller  ⚡ Doğacan Özdemir
@RequestMapping(⊕"/laborants")
public class LaborantController {
    @Autowired
    private LaborantService laborantService;

    @GetMapping(⊕"/")  ⚡ Doğacan Özdemir
    public ModelAndView getAllLaborants() {
        List<Laborant> laborants = laborantService.getAllLaborants();
        ModelAndView mav = new ModelAndView( viewName: "laborants");
        mav.addObject( attributeName: "laborants", laborants);
        return mav;
    }

    @GetMapping(⊕"/{id}")  ⚡ Doğacan Özdemir
    public ModelAndView getLaborantById(@PathVariable Long id) {
        ModelAndView mav = new ModelAndView( viewName: "laborantDetails");
        laborantService.getLaborantById(id).ifPresent(mav::addObject);
        return mav;
    }

    @GetMapping(⊕"/createLaborant")  ⚡ Doğacan Özdemir
    public ModelAndView newLaborantForm() {
        ModelAndView mav = new ModelAndView( viewName: "createLaborant");
        mav.addObject( attributeName: "laborant", new Laborant());
        return mav;
    }
}
```

Şekil 16 - LaborantController.java #1

- **getAllLaborants()**: Tüm laborantları alır ve bunları bir ModelAndView içinde "laborants" view'ine ekler.
- **getLaborantById(@PathVariable Long id)**: Belirli bir laborantın detaylarını alır ve bunları "laborantDetails" view'ine ekler. Eğer laborant bulunamazsa, boş bir ModelAndView döndürür.

- **newLaborantForm()**: Yeni bir laborant oluşturma formunu gösterir. Boş bir Laborant nesnesi oluşturur ve bunu "createLaborant" view'ine ekler.

```
@PostMapping("/{saveLaborant}")  ⚡ Doğacan Özdemir
public ModelAndView saveLaborant(@Valid @ModelAttribute Laborant laborant, BindingResult result) {
    ModelAndView mav = new ModelAndView( viewName: "createLaborant");
    if (result.hasErrors()) {
        mav.addObject( attributeName: "laborant", laborant);
        return mav;
    }
    try {
        laborantService.saveLaborant(laborant);
        mav.setViewName("redirect:/laborants");
    } catch (IllegalArgumentException e) {
        mav.addObject( attributeName: "errorMessage", e.getMessage());
        mav.addObject( attributeName: "laborant", laborant);
    }
    return mav;
}

@GetMapping("/{editLaborant}/{id}")  ⚡ Doğacan Özdemir
public ModelAndView editLaborantForm(@PathVariable Long id) {
    ModelAndView mav = new ModelAndView( viewName: "editLaborant");
    laborantService.getLaborantById(id).ifPresent(laborant -> mav.addObject( attributeName: "laborant", laborant));
    return mav;
}
```

Şekil 17 - LaborantController.java #2

- **saveLaborant(@Valid @ModelAttribute Laborant laborant, BindingResult result)**: Yeni bir laborantı veya mevcut bir laborantı kaydeder. Gelen laborant nesnesini doğrular (@Valid) ve herhangi bir doğrulama hatası varsa, bu hataları modelde saklar ve "createLaborant" view'ini tekrar gösterir. Eğer bir hata yoksa, laborantı kaydeder ve "laborants" sayfasına yönlendirir. Kaydetme işlemi sırasında bir hata olursa, hatayı modelde saklar ve "createLaborant" view'ini tekrar gösterir.
- **editLaborantForm(@PathVariable Long id)**: Bir laborantın düzenleme formunu gösterir. Verilen laborant kimliğine (id)

```

@PostMapping("/updateLaborant")
public ModelAndView updateLaborant(@Valid @ModelAttribute Laborant laborant, BindingResult result) {
    ModelAndView mav = new ModelAndView("editLaborant");
    if (result.hasErrors()) {
        mav.addObject("laborant", laborant);
        return mav;
    }
    try {
        laborantService.updateLaborant(laborant);
        mav.setViewName("redirect:/laborants");
    } catch (IllegalArgumentException e) {
        mav.addObject("errorMessage", e.getMessage());
        mav.addObject("laborant", laborant);
    }
    return mav;
}

@GetMapping("/delete/{id}")
public String deleteLaborant(@PathVariable Long id) {
    laborantService.deleteLaborant(id);
    return "redirect:/laborants";
}

```

Şekil 18 - LaborantController.java #3

göre ilgili laborantı alır ve "editLaborant" view'ine ekler. Bu, mevcut laborant bilgilerini düzenlemek için kullanılır.

- **updateLaborant(@Valid @ModelAttribute Laborant laborant, BindingResult result):** Mevcut bir laborantın bilgilerini günceller. Gelen laborant nesnesini doğrular (@Valid) ve doğrulama hatası varsa, bu hataları modelde saklar ve "editLaborant" view'ini tekrar gösterir. Eğer bir hata yoksa, laborant bilgilerini günceller ve "laborants" sayfasına yönlendirir. Güncelleme işlemi sırasında bir hata olursa, hatayı modelde saklar ve "editLaborant" view'ini tekrar gösterir.
- **deleteLaborant(@PathVariable Long id):** Belirli bir laborantı veritabanından siler. Verilen laborant kimliğine (id) göre ilgili laborantı siler ve "laborants" sayfasına yönlendirir. Bu, laborantın veritabanından kalıcı olarak kaldırılmasını sağlar.

3.4.6. Resources Paketi

Bu paket içerisinde temel görünüm dosyalarımız bulunmaktadır. Login sayfası Spring Security'nin hazır paketi olan bir sayfa şeklinde dönmektedir.

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  
  <a class="navbar-brand" th:href="@{/home}">lab-master PRO</a>
  <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav"
    aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
  </button>
  <div class="collapse navbar-collapse" id="navbarNav">
    <ul class="nav nav-pills nav-fill">
      <li class="nav-item active">
        <a class="nav-link" th:href="@{/home}">Ana Sayfa <span class="sr-only">(current)</span></a>
      </li>
      <li class="nav-item dropdown">
        <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button"
          aria-haspopup="true" aria-expanded="false">
          Raporlar
        </a>
        <div class="dropdown-menu" aria-labelledby="navbarDropdown">
          <a class="dropdown-item" th:href="@{/reports}">Tüm Raporlar</a>
          <a class="dropdown-item" th:href="@{/reports/createReport}">Rapor Ekle</a>
        </div>
      </li>
      <li class="nav-item dropdown">
        <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown2" role="button"
          aria-haspopup="true" aria-expanded="false">
          Laborantlar
        </a>
        <div class="dropdown-menu" aria-labelledby="navbarDropdown2">
          <a class="dropdown-item" th:href="@{/laborants}">Tüm Laborantlar</a>
          <a class="dropdown-item" th:href="@{/laborants/createLaborant}">Laborant Ekle</a>
        </div>
      </li>
    </ul>
  </div>
</nav>
```

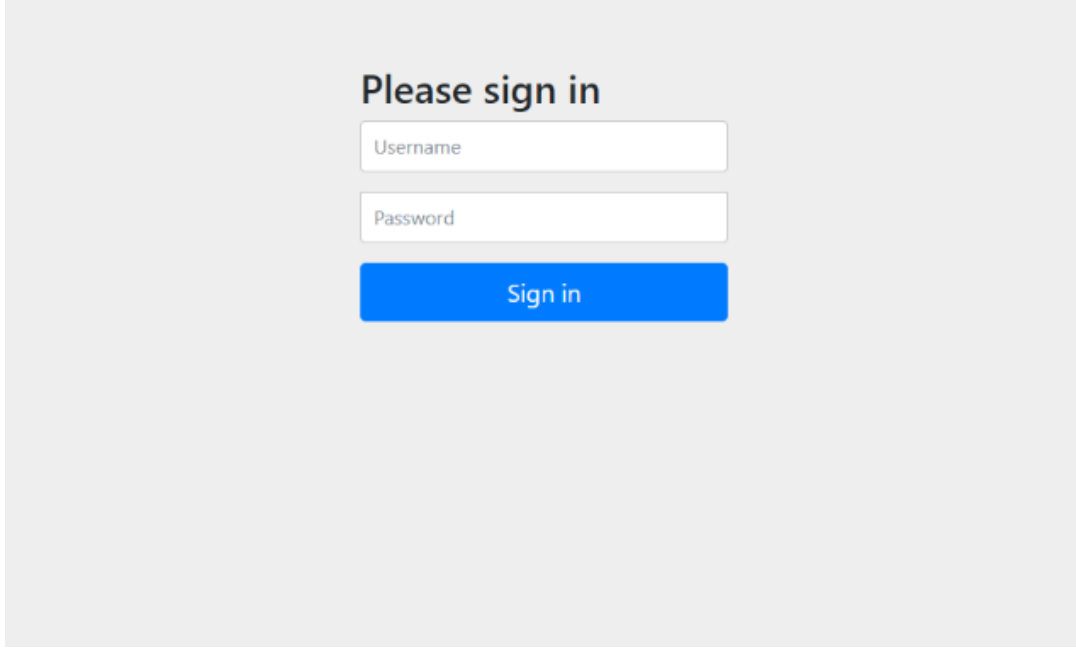
Şekil 19- Navigasyon Barı kodları

Şekil – 19’da bulunan kod dizisi tüm HTML sayfalarında bulunmaktadır. Navigasyon Bar tanımının yapıldığı kod dizisidir. Tasarım için Bootstrap kullanılmıştır. Bağlantılar için Thymeleaf kullanılmıştır. Diğer sayfalar da aynı şekilde oluşturulmuştur. İlgili kodların devamı kaynak dosya içerisinde ve github.com/dev-dogacanOzdemir adresinde mevcuttur.

3.3. Tomcat Server Üzerinde Çalışır Hali

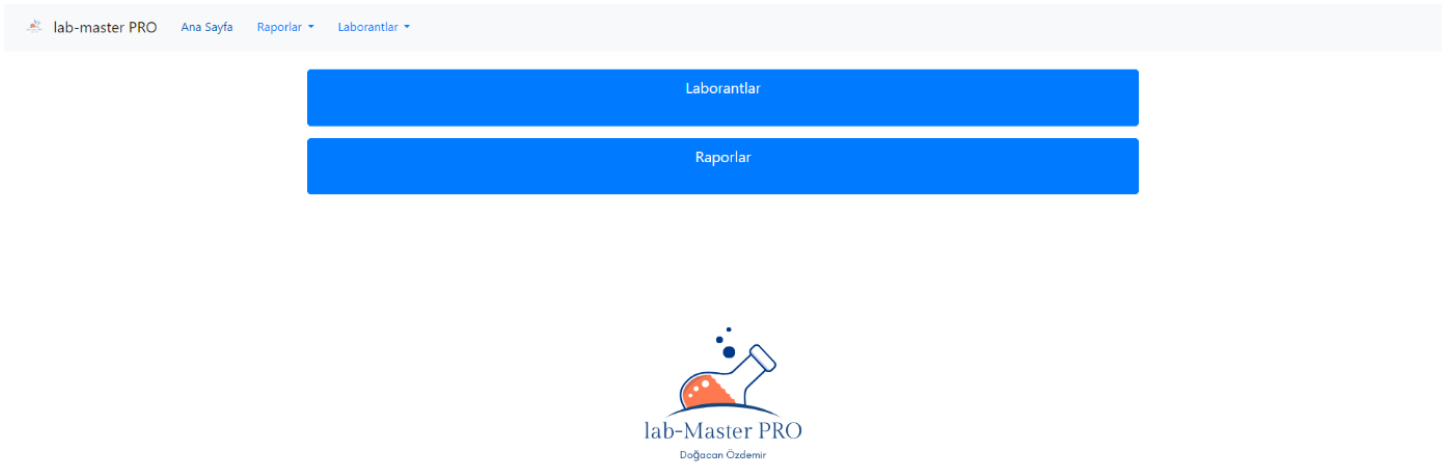
3.3.1. Login Sayfası

Bu sayfa Spring Security ile gelen varsayılan bir sayfadır. Tasarımını değiştirmek için Config içerisinde bir dosya tanımlanıp ilgili değişiklikler ile düzenlenebilir.

A screenshot of a login page with a light gray background. At the top center, the text "Please sign in" is displayed in a bold, dark font. Below this text are two input fields: the first is labeled "Username" and the second is labeled "Password". Both fields are white with a light gray border. Below the password field is a blue button with the text "Sign in" in white.

Şekil 20 - Login

3.3.2. Ana Sayfa



Şekil 21 - index.html

3.3.3. Laborant Listeleme Ekranı

Laborantlar

Show entries

Search:

ID	İsim	Soyisim	Hastane ID	İşlemler
1	Doğacan	Özdemir	1111111	İncele Düzenle Sil

Showing 1 to 1 of 1 entries

[Previous](#)
[1](#)
[Next](#)

[Laborant Oluştur](#)

Şekil 22 - laborants.html

3.3.4. Laborant Detayları Ekranı

Laborant Detayları

ID:	1
İsim:	Doğacan
Soyisim:	Özdemir
Hastane ID:	1111111

Raporlar

Rapor ID	Hasta T.C. Kimlik No	Rapor Tarihi
2	11122233444	2024-06-04

[Geri Dön](#)

Şekil 23 - laborantDetails.html

Bu kısımda arka-uç geliştirmesinde tanımladığımız şekilde her laboranta ait olan n adet rapor gösteriliyor.

3.3.5. Laborant Ekleme/Düzenleme Ekranı

Şekil-24'teki ekran görüntüsü laborant düzenleme sayfasından alınmış durumda ancak laborant oluşturma kısmında da aynı ekran kullanıcıya gösteriliyor.

Laborant Düzenle

İsim

Doğacan

Soyisim

Özdemir

Hastane ID

1111111

Kaydet

İptal

Şekil 24 - editLaborant.html

3.3.6. Rapor Ekleme Ekranı

Şekil-25'teki ekran görüntüsü rapor ekleme sayfasından alınmış durumda ancak rapor düzenleme kısmında da aynı ekran kullanıcıya gösteriliyor.

Yeni Rapor Ekle

Hasta Adı

Hasta Soyadı

Hasta T.C. Kimlik No

Teşhis Başlığı

Teşhis Detayları

Rapor Tarihi

gg.aa.yyyy

Laborant

Doğacan Özdemir

Rapor Görseli

Dosya Seç

Dosya seçilmedi

Kaydet

İptal

Şekil 25 - createReport.html

Ayrıca istenilen yapı da sağlanarak her rapora bir adet rapor görseli eklenebiliyor. İlgili rapor görseli daha sonrasında rapor detaylarından indirilerek inceleme yapılması sağlanıyor.

3.3.7. Rapor Detayları Ekranı

Rapor Detayları

ID:	2	Tanı	Mide
T.C. Kimlik Numarası:	11122233444	Başlığı:	
İsim:	İlhan	Tanı Detayı:	Mide ağrısı
Soyisim:	Özdemir	Rapor Tarihi:	2024-06-04
Laborant ID:	1111111	Ek Dosya:	Rapor Görselini İndir

[Geri Dön](#)

Şekil 26 - reportDetails.html

3.3.8. Rapor Listeleme Ekranı

Raporlar

Show10entries

Search:

T.C. Kimlik Numarası	İsim	Soyisim	Rapor Tarihi	Laborant Numarası	İşlemler
11122233444	İlhan	Özdemir	2024-06-04	1111111	<div><div>İncele</div><div>Düzenle</div><div>Sil</div></div>

Showing 1 to 1 of 1 entries

Previous

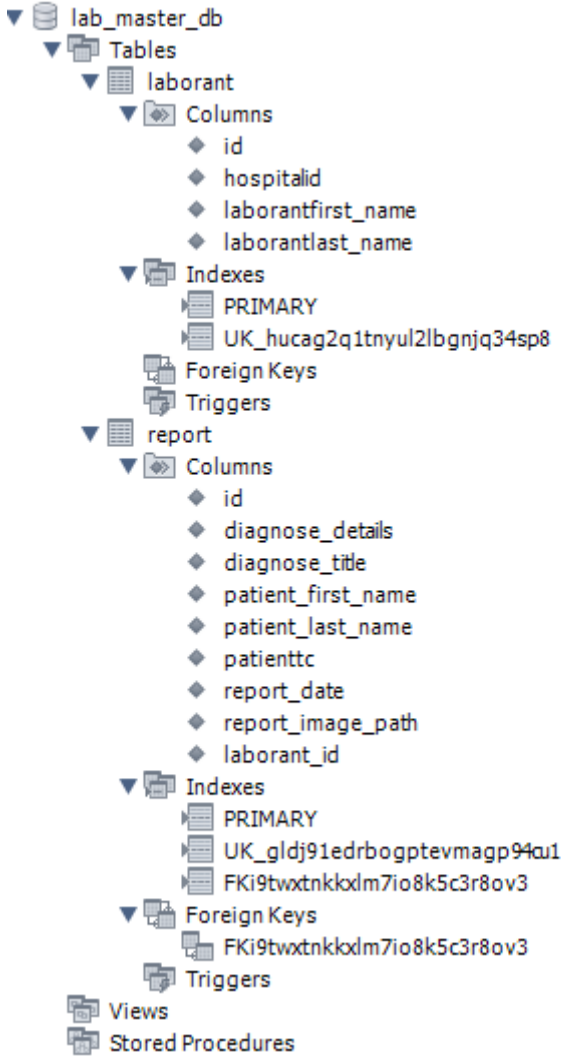
1

Next

Rapor Oluştur

Şekil 27- reports.html

3.4. Veritabanı İncelemesi (MySQL)



Şekil 28 - MySQL Proje Yapısı

Uygulamanın oluşturulmuş ve kullanılır haldeki veritabanı Şekil-28'deki gibi görünmektedir. Bazı Foreign Key'leri anotasyonlar aracılığıyla oluşturduğumuz için ismi ile değil de ilgili kütüphanenin tanımladığı isimle veritabanına kayıt olmaktadır.

SONUÇLAR VE ÖNERİLER

Bu projede belirli teknik seçimler ve kabuller yaptım. Bu seçimler, uygulamanın ihtiyaçlarını karşılamak, performansı optimize etmek ve kodun bakımını kolaylaştırmak amacıyla gerçekleştirilmiştir.

Uygulamayı, katmanlı mimari prensiplerine uygun olarak tasarladım. Bu mimari seçim, uygulamanın farklı bileşenlerini (Controller, Service, Repository) birbirinden ayırarak daha modüler, test edilebilir ve bakımı kolay bir yapı sağladı.

Projeyi yönetmek için Maven kullanmayı tercih ettim. Maven teknolojisine daha önceden aşina olmam ve bu yapı üzerinde rahat çalışabilmem bu tercihi yapmamda etkili oldu.

Projenin veritabanı için MySQL'i tercih ettim. Bu seçimin gerekçesi, MySQL'in lisansının ücretsiz olması ve MySQL Workbench uygulamasının arayüzüne hakim olmamdı.

Lombok kütüphanesini kullanarak Getter ve Setter işlemlerini tek bir anotasyon ile tanımlayarak kod kalabalığının önüne geçtim. Bu teknoloji, kodun daha temiz ve okunabilir olmasını sağladı.

Thymeleaf ve Bootstrap kullanarak kullanıcı dostu bir ön yüz tasarladım. Postman gibi araçlarla manuel istekler göndermek yerine, kullanıcıların etkileşimde bulunabileceği bir arayüz sağlamak daha kullanışlı oldu.

Hata yönetimi için iki yöntem kullandım. İlk yöntemde, özel hata mesajları üretecek şekilde bir sınıf tanımlayarak mesajın kullanıcıya iletilmesini sağladım. İkinci yöntemde ise, Jakarta kütüphanesinin validation metotlarını kullanarak doğrulamaların otomatik olarak gerçekleştirilmesini sağladım. Örneğin, T.C. Kimlik Numarası'nın 11 haneli olması gibi kuralları belirlemek için ek kod yazmama gerek kalmadı.

Daha önce yaptığım geliřtirmelerde dosya ykleme gibi bir yapı zerinde alıřmamıřtım. Bu projede MultipartFile ktphanesini kullanarak bu iřlemleri saėladım. Ayrıca Spring MVC ile uyumlu řekilde alıřması ve dosyaların izinleri ile kaydedilmesi sayesinde performans kaybı yařanmasının nne getim. Bu dosya iřlemlerini Spring Boot'ta tanımlı olan Resource ve ResponseEntity yapılarını kullanarak indirme iřlemini de saėladım.

Spring Data JPA, MySQL sorguları yazmak yerine hazır metotlar kullanmamı ve bunların arka planda gerekleřmesini saėlayarak iřimi ok kolaylařtırdı.

Ayrıca Spring Security'nin kendi ierisinde barındırdıė authenticator sistemi kusursuz bir řekilde alıřmaktadır ve geliřtirilmeye aık olması, bu zelliėinin daha birok projede kullanılabilir hale gelmesini saėlıyor.

Gelecekte yapılabilecek geliřtirmeler arasında, mikroservis mimarisine geiř, daha kapsamlı kullanıcı ynetimi ve rol tabanlı eriřim kontrol ile gerek zamanlı veri iřleme zelliklerinin eklenmesi yer alabilir. Bu tr geliřtirmeler, uygulamanın leklenebilirliėini ve kullanım alanını daha da geniřletebilir.

Spring Boot'un saėladıėı avantajları yeterince kullandıėım bir proje oldu. Sonu olarak, geliřtirdiėim proje ok byk aplı olmasa da ėrendiėim bilgiler gelecek iin faydalı oldu. Java ile geliřtirme yapan geliřtiricilerin bu teknolojileri iyice kavraması ve ėrenmesi gerektiėini dřnyorum.

KAYNAKÇA

- [1] https://tr.wikipedia.org/wiki/Programlama_dili
- [2] <https://aws.amazon.com/tr/what-is/java/>
- [3] <https://www.medium.com>
- [4] alicanakkus.github.io
- [5] www.opendart.com
- [6] www.yusufsezer.com.tr
- [7] tr.wikipedia.org
- [8] farukgenc.com