DOSYA YÜKLEME VE YÖNETİMİ UYGULAMASI
TEKNİK RAPOR

ABDÜSSAMED KARA

Giriş

Projenin amacı, kullanıcıların sisteme giriş yaptıktan sonra dosya yükleyip bu dosyaları görüntüleyebilmesi ve isterse silebilmesi gibi temel işlevleri yerine getirebileceği bir web uygulaması geliştirmektir.

Uygulamanın backend kısmı için **Java+Spring Boot** kullanılmıştır. Güvenli bir kullanıcı girişi sağlamak amacıyla **Basit Session** ile kimlik doğrulama yapılacak. Kullanıcı arayüzü için ise Spring ile uyumlu olan **Thymeleaf** tercih edilecektir. Dosyalar dizinde saklanacak ve bu dosyalara sadece giriş yapan kullanıcılar erişebilecektir.

Kullanılan Yöntemler Ve Teknolojiler

Bu projede backend geliştirme için **Java+Spring Boot** kullanılmıştır. REST API mimarisi sayesinde kullanıcıdan gelen istekler (örneğin dosya yükleme, listeleme veya silme gibi) kolayca işlenebilmiştir.

Kullanıcı giriş ve oturum yönetimi için **basit session (oturum) yapısı** tercih edilmiştir. Kullanıcı sisteme giriş yaptığında, oturum bilgisi backend tarafında tutulur ve her işlemde bu oturum kontrol edilerek işlem yapılmasına izin verilir.

Kullanıcı arayüzü kısmında ise **Thymeleaf** şablon motoru kullanılmıştır. Thymeleaf sayesinde HTML sayfalarında backend'den gelen veriler doğrudan gösterilebilmiş ve sade, anlaşılır bir arayüz oluşturulmuştur.

Dosya yükleme işlemlerinde, yüklenen dosyalar dış bir sunucuya değil, projenin içinde oluşturulan **uploads adlı klasöre** kaydedilmektedir. Dosya silme ve listeleme işlemleri de bu klasör üzerinden yapılmaktadır.

Problemin Tanımı Ve Kullanılan Teknoloji Nedenleri

Bu projede, kullanıcıların dosya yükleme, listeleme ve silme işlemlerini gerçekleştirebileceği, basit ama işlevsel bir web uygulamasının geliştirilmesi amaçlanmıştır. Uygulama, gerçek hayattaki dosya yönetimi ihtiyaçlarına yönelik temel bir çözüm sunmayı hedeflemektedir.

Neden Java?

Bu projede dosya yükleme, listeleme ve silme gibi işlemleri kullanıcı oturumu üzerinden güvenli bir şekilde yönetmek gerekir. Java, sağlam oturum (session) yapısı bu ihtiyaçlara doğrudan cevap verir. Aynı zamanda Java'nın geniş kütüphane desteği, dosya sistemleri ve HTTP işlemleri gibi konularda projeye hız kattı

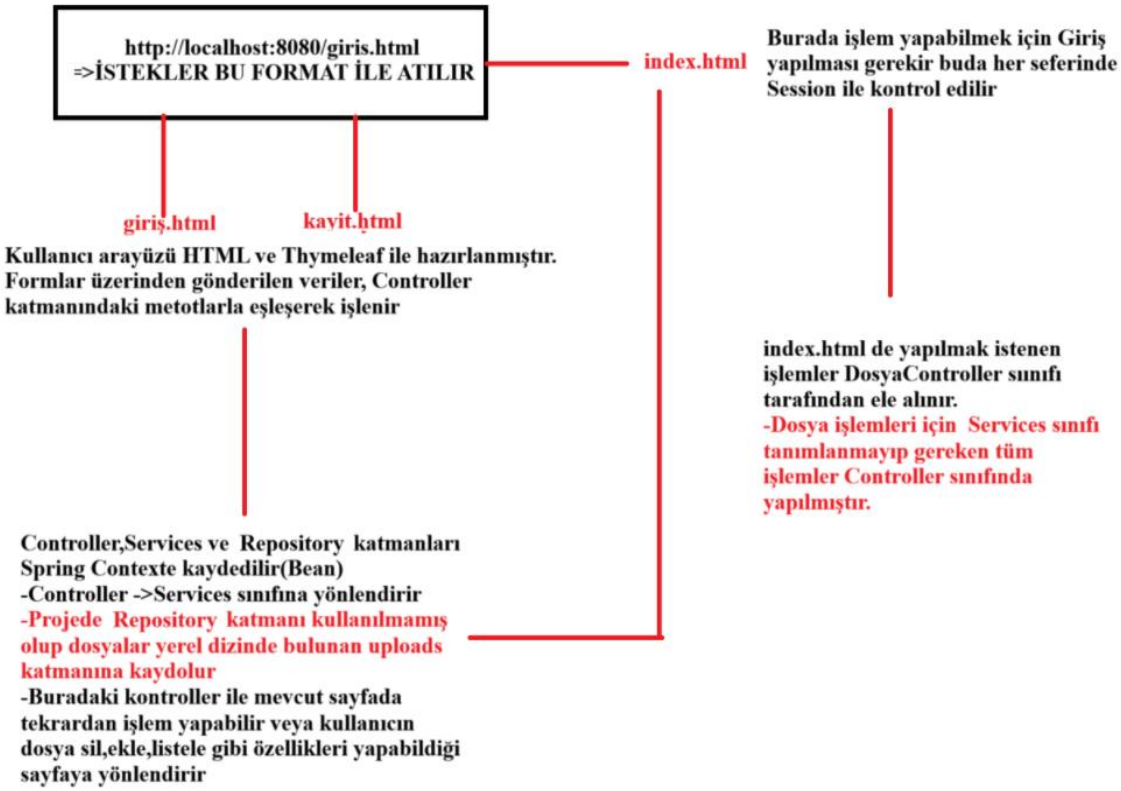
Neden Spring Boot?

Dosya işlemleriyle ilgili endpoint'lerin oluşturulması, oturum kontrolü ve hata yönetimi gibi işlemleri basit ve düzenli şekilde yazabilmek için Spring Boot tercih edildi. Bu projede @RestController, @PostMapping, @GetMapping gibi Spring'e özel notasyonlar sayesinde, backend işlemleri çok daha anlaşılır hale geldi. Katmanlı(MVC) mimari yapısı sayesinde düzgün bir Sistem Mimarisi hedeflendi. Ayrıca session giriş kontrolü güvenlik gereksinimlerini de çözmemizi sağladı.

Neden HTML + JavaScript + Thymeleaf?

Kullanıcı arayüzünün oluşturulmasında HTML ve JavaScript gibi temel web teknolojileri kullanılmıştır. Thymeleaf, Spring Boot ile doğrudan uyumlu çalışan bir şablon motorudur. JavaScript sayesinde mevcut web siteleri arasında yönledirme işlemleri yapıldı.

Uygulamanın Mimari Yapısı ve Uygulanışı



Uygulumanın Mimarı Katmanı;

-Controller Katmanı: Amacı atılan istekleri yakalayıp uygun yerlere yönlendirmek. DosyaController ve KimlikDenetimiController olmak üzere iki sınıftan oluşmaktadır.

-Services: Controllerde çeşitli ayıklama(yöneltme) işlemlerinden sonra durumlara göre Services katmanı Controller tarafından çağırılır. Olayın asıl arka planı bu kısımdır.

-Model: Kullanıcılara ait özellikler burada belirlenir **@AllArgsConstructor-@Getter-@Setter-@ToString** Anatasyonları ile kod tekrarının önüne geçilmiştir.

Bu üç katmanın haberleşmesi @Autowired ve Context sayesinde yapılır.

Controller

- **KimlikDenetimiController:**

POST isteğiyle atılan **/kayit** endpoint'i üzerinden çalışan **kayitOl()** metodu sayesinde, kullanıcının formda doldurduğu **kullaniciAdi** ve **sifre** bilgileri @**RequestParam** anotasyonu ile alınır. Bu bilgiler üzerinde önce boşluk kontrolü yapılır, ardından kullanıcı adının daha önce kullanılıp kullanılmadığı (benzersizlik) kontrol edilir. Dönüş olarak bir **HashMap** yapısı kullanılır ve örneğin **"mesaj", "Kayıt başarılı"** şeklinde bir mesaj gönderilir. **kayit.html** sayfasında bu mesajlar koşul ifadeleriyle değerlendirilerek, kullanıcı başarılı bir şekilde kayıt olmuşsa **giris.html** sayfasına yönlendirilir, aksi halde sayfa **tekrar** gösterilir.

POST isteğiyle atılan **/giris** endpoint'i üzerinden çalışan **girisYap()** metodu ile kullanıcıdan gelen **kullaniciAdi** ve **sifre** bilgileri yine @**RequestParam** ile alınır. Bu bilgiler boş değilse, doğrulama için Service sınıfına gönderilir. Girilen bilgiler doğruysa, kullanıcıya özel bir **session** oluşturulur. Geriye yine bir **HashMap** yapısı ile örneğin "mesaj", "Giriş başarılı" şeklinde mesaj döndürülür. **giris.html** sayfasında bu mesaj değerlendirilerek başarılı girişte **index.html** sayfasına yönlendirme yapılır, başarısız durumda **aynı sayfa yeniden yüklenir**.

POST isteğiyle atılan **/cikis** endpoint'i üzerinden çalışan **cikisYap()** metodu sayesinde, oturum açmış olan kullanıcının **session** bilgisi **sistemden silinir**. Böylece kullanıcı, tekrar giriş yapmadan herhangi bir işlem yapamaz hale getirilmiş olur.

- **Dosya Controller**

POST isteğiyle atılan **/dosyalar/yukle** endpoint'i üzerinden çalışan **dosyaYukle()** metodu sayesinde, kullanıcının yüklemek istediği dosya **sisteme** alınır. İlk olarak, oturum açmış bir kullanıcı olup olmadığı **HttpSession** üzerinden kontrol edilir. Eğer kullanıcı giriş **yapmamışsa, işlem reddedilir**. **Giriş yapılmışsa, dosyanın boş olup olmadığı ve uzantısının geçerli olup olmadığı (pdf, png, jpg, jpeg) denetlenir**. Dosya uygunsa, benzersiz bir ad verilerek **uploads** klasörüne kaydedilir ve kullanıcıya bilgilendirici bir yanıt dönülür.

GET isteğiyle atılan **/dosyalar/liste** endpoint'i üzerinden çalışan **dosyaListele()** metodu sayesinde, sisteme daha önce yüklenen **mevcut kullanıcıya tüm dosyalar listelenir**. Bu işlem sadece oturum açmış kullanıcılar tarafından yapılabilir. Kullanıcının oturumu kontrol edilir; eğer giriş **yapılmamışsa erişim reddedilir**. Giriş yapılmışsa, uploads klasöründeki tüm dosya adları okunarak kullanıcıya bir **liste** halinde döndürülür.

DELETE isteğiyle atılan **/dosyalar/sil/dosyaAdi** endpoint'i üzerinden çalışan **dosyaSil()** metodu sayesinde, belirtilen ada sahip **dosya sistemden silinir.** Öncelikle kullanıcının oturum açmış olup olmadığı kontrol edilir. Oturum yoksa işlem yapılmaz. Oturum varsa, belirtilen dosya sistemde aranır. Dosya mevcutsa silinir, değilse kullanıcıya "**dosya bulunamadı**" mesajı döner.

Services

Kullanıcılar bir **ArrayList içerisinde tutulmaktadır**. Bu liste, sisteme kayıt olan kullanıcıları geçici olarak bellekte saklar.

- **KullaniciServices**

Controller sınıfının yönlendirmesi ile çalışan **kayitOl()** metodu, kullanıcıdan gelen **kullaniciAdi** ve **sifre** bilgilerini alır. İlk olarak, ArrayList içerisinde bu kullanıcı adının daha önce **kayıtlı olup olmadığı kontrol edilir (benzersizlik kontrolü)**.
Eğer kullanıcı adı benzersizse (liste içinde yoksa), **yeni kullanıcı new Kullanici() ile oluşturularak listeye eklenir.** Dönüş tipi olarak **boolean** kullanılır.

Controller sınıfının yönlendirmesi ile çalışan **dogrula()** metodu sayesinde gelen bilgilerin **ArrayList'te kontrolü yapılır**. Dönüş olarak **boolean** türü kullanılır.

Sonuç Ve Değerlendirme

Bu projede, temel dosya yükleme, listeleme ve silme işlemlerini gerçekleştiren bir web uygulaması geliştirilmiştir. Uygulama Spring Boot ile yazılmış ve MVC mimarisine uygun şekilde yapılandırılmıştır. Görsel arayüz Thymeleaf destekli HTML sayfalarıyla oluşturulmuştur.
Kimlik doğrulama için basit bir session yapısı kullanılmıştır. Daha gelişmiş bir yapı için JWT tercih edilebilirdi. Ayrıca sunucu üzerinden yayınlama yapılmamıştır, proje yerel ortamda çalışacak şekilde bırakılmıştır.

Frontend kısmı basit tutulmuştur, görsel iyileştirmeler ve kullanıcı deneyimi açısından geliştirilebilir. Ancak genel olarak uygulama işlevini yerine getirmektedir ve temel seviyede amacına ulaşmıştır.