1

**TÜRK-ALMAN ÜNİVERSİTESİ**

**MÜHENDİSLİK FAKÜLTESİ**

**STAJ DEFTERİ**



**Öğrencinin**

**Adı Soyadı : ……………………………**

**Bölümü : Bilgisayar Mühendisliği**

**Sınıfı : ……………………………**

**Numarası : ……………………………**

**Staj Yaptığı Kurumun Adı : ……………………………**

**Staj Türü : Temel Staj ( ) Meslek Stajı (x)**

**Staj Başlangıç Tarihi : \_\_\_/\_\_\_/20\_\_**

**Staj Bitiş Tarihi : \_\_\_/\_\_\_/20\_\_**

**Staj Defteri Teslim Tarihi : \_\_\_/\_\_\_/20\_\_**

**Türk-Alman Üniversitesi Mühendislik Fakültesi – Şahinkaya Cad.No:86 34820 Beykoz/İstanbul**

**Tel: 0216 333 31 00 Faks: 0216 333 31 05**

|  |  |
| --- | --- |
| **Staj Yapılan Kuruma Ait Bilgiler** | |
| Adı | **……………………………** |
| Adres | **……………………………** |
| Telefon | **……………………………** |
| E-posta | **……………………………** |
| Web Sayfası | **……………………………** |
| **Staja Yönelik Bilgiler** | |
| Staj Yapılan Bölüm | **……………………………** |
| Stajın Konusu | **……………………………** |
| Stajın Süresi (Resmi Tatiller Hariç) | **……………………………** |
| **Kurumdaki Yetkili Kişiye Ait Bilgiler** | |
| Adı Soyadı | **……………………………** |
| Unvan | **……………………………** |
| Telefon | **……………………………** |
| E-posta | **……………………………** |
| **Bu Bölüm Staj Sorumlusu tarafından doldurulacaktır.** | |
| Adı Soyadı | **……………………………** |
| Unvan | **……………………………** |
| Stajyer Değerlendirme Formu | Teslim Edildi ( ) Teslim Edilmedi ( ) |
| Kabul durumu | Kabul ( ) Kısmen kabul ( ) Red ( ) |
| Kabul edilen staj günü sayısı |  |

|  |  |
| --- | --- |
| **Haftalık Staj Planı**  Her staj gününde gerçekleştirilen çalışmaları ilgili günün yanındaki kısımda kısaca belirtiniz. Resmi tatilleri ilgili günün yanındaki kısımda belirtiniz. Gerektiğinde yeni sayfa ekleyiniz. | |
|
|
|
| **1. Hafta** | **18/07/2022 - 24/07/2022** |
|
| 1. Gün | Web Servisleri ve web servislerinde kullanılan teknolojiler araştırıldı. |
|
|
| 2. Gün | ASP.NET, Spring Framework ve Spring Boot araştırıldı.  Spring Boot için gereken geliştirme ortamı kuruldu. |
|
|
| 3. Gün | Java kod örnekleri araştırıldı.  Java Lambda Expression, Anonymous ve Abstract Classları, Callback fonksiyonları araştırıldı. |
|
|
| 4. Gün | Spring Boot tutorialleri araştırıldı. |
|
|
| 5. Gün | Spring Boot tutorialleri araştırıldı. |
|
|
|  | |
| **2. Hafta** | **25/07/2022 - 31/07/2022** |
|
| 1. Gün | Spring Boot tutorialleri araştırıldı. |
|
|
| 2. Gün | Spring Boot tutorialleri araştırıldı. |
|
|
| 3. Gün | Spring Boot tutorialleri araştırıldı. |
|
|
| 4. Gün | Basit bir web servisi ve arayüz yazıldı, sonra servis deploy edildi. |
|
|
| 5. Gün | Bu hafta ve geçen hafta ne yapıldığına dair rapor yazılıp gönderildi. |
|
|
| **TASDİK EDEN YETKİLİNİN** | |
| **İmza-Mühür** | |
|  | |
|

|  |  |
| --- | --- |
| **Haftalık Staj Planı**  Her staj gününde gerçekleştirilen çalışmaları ilgili günün yanındaki kısımda kısaca belirtiniz. Resmi tatilleri ilgili günün yanındaki kısımda belirtiniz. Gerektiğinde yeni sayfa ekleyiniz. | |
|
|
|
| **3. Hafta** | **01/08/2022 - 07/08/2022** |
|
| 1. Gün | Web Servislerinde güvenlik konusunda araştırmalara başlandı.  Güvenlik yöntemleri araştırıldı. |
|
|
| 2. Gün | HTTP Auth. metotları ve JWT araştırıldı. |
|
|
| 3. Gün | OAuth 1.0 ve 2.0 araştırıldı. |
|
|
| 4. Gün | İstenilen güvenlik şekline yönelik araştırmalar yapıldı. |
|
|
| 5. Gün | API Keyleri araştırıldı.  Demo API Key kodu yazıldı. |
|
|
|  | |
| **4. Hafta** | **08/08/2022 - 14/08/2022** |
|
| 1. Gün | Güvenlik için gerekli kod yazıldı. |
|
|
| 2. Gün | Güvenlik kodu test edildi ve düzeltildi. |
|
|
| 3. Gün | API Key. properties dosyasından çekilecek şekilde ayarlandı. |
|
|
| 4. Gün | Yazılan kod test edildi ancak beklendiği gibi çalışmadı.  .properties'den çekme işlemi düzeltildi. |
|
|
| 5. Gün | Güvenlik, sadece belirli adresleri güvenlik altına alınacak şekilde değiştirildi. |
|
|
| **TASDİK EDEN YETKİLİNİN** | |
| **İmza-Mühür** | |
|  | |
|

|  |  |
| --- | --- |
| **Staj İçeriği**  Staj kapsamında gerçekleştirdiğiniz çalışmalar hakkında detaylı bilgi veriniz. Gerektiğinde yeni sayfa ekleyiniz. | |
|
|
| **Tarih:** | **18/07/2022 - 31/07/2022** |
|
| **Konu** | Web Servisleri üzerinde araştırmalar ve örnekler |
|
| İlk günler implementasyon görevleri alınmadığı için web servislerinin ne olduğu ve hangi teknolojilerin kullanıldığı araştırıldı.  Web servisleri bir uygulamanın web'i kullanma biçimi olarak tanımlanabilir. Web teknolojileri üzerinden API sağlayarak çalışırlar. Web sayfalarından farkı, web sayfaları bir insanın kullanması için grafik ara yüzüyle (GUI) tasarlanmıştır ancak web servisleri ise kullanıcıdan ziyade bir uygulamaya bilgi veya hizmet (servis) sağlamak için tasarlanmıştır.  Web servisleri ana olarak ikiye ayrılır, SOAP(Simple Object Access Protocol) ve RESTful(REpresentational State Transfer) web servisleri. SOAP spesifikasyonları sıkı tanımlanmış XML bazlı bir protokoldür. REST ise bir protokol değil mimaridir. REST'in belirttiği yönergeler şunlardır:   * Servisi üreten ve tüketenler olmalıdır. * Servis durum bilgisi tutmamalıdır. * Sonuçlar önbelleklenebilir olmalıdır. * Ara yüz tek tiptir ve kaynaklara erişimi sağlar. * Servis katmanlı yapıda olmalıdır.   İki tür de genelde yaygınlığı yüzünden HTTP üzerinden çalışırlar.  Eskiden web servislerinin tanımını (description) ve keşfedilmesini (discovery) sağlayan mekanizmalar da vardı. Spesifik olarak WSDL (Web Services Description Language) ve UDDI (Universal Description, Discovery, and Integration). WSDL web servisiyle iletişimin nasıl gerçekleşeceğini sağlarken, UDDI ise web servisinin keşfedilmesini sağlıyordu. Günümüzde ise bu ikisinin kullanımı nerdeyse kalmamıştır, çünkü daha kolay bir şekilde web servislerinin tanımı ve kullanımı dökümasyonlar ile, keşfi ise DNS üzerinden (web servisleri genelde web adresleri üzerinden erişildiği için) sağlanmaktadır.  Web servislerinde veri aktarımı için popüler olan iki yöntem vardır. XML (Extensible Markup Language) ve JSON (JavaScript Object Notation). Objeleri yeniden oluşturmak için gereken bilgileri belirli bir yapıda string olarak saklarlar ve alıcı taraf bunu tekrar yazılımın içinde kullanılmak üzere bir objeye dönüştürür. Web servisleri açısından iki yöntem serialization ve deserialization için kullanılmaktadır. Dosyaların (genelde büyük) aktarımı ise genelde başka bir formata çevrilmeden direkt olarak gerçekleşir.  Web servislerinin ne olduğu araştırıldıktan sonra web servislerinin nasıl yazılabileceği araştırıldı.  Google üzerinden "web servisi nasıl yazılır" diye aratıldığında yüksek bir oranda Spring ve Spring Boot frameworkleriyle karşılaşıldı. Spring bir Java frameworküdür yani Java ile yazılmıştır ve Spring uygulamaları da Java ile yazılır ancak uygulamaların yazılması için Java standart kütüphanesiyle birlikte Spring'in kendi fonksiyonlarının da kullanılması lazımdır. Örneğin bir Spring uygulamasının çalıştırılabilmesi için main fonksiyonunun içinde Spring'in kendi run fonksiyonu da çağrılmaktadır.    Spring Boot ise Spring'in konfigürasyonlarının otomatize edilerek kullanımının çok daha basitleştirilmiş halidir.  Bir diğer seçenek olarak ise ASP.NET ile karşılaşıldı ancak popülerliği sebebiyle Spring Boot kullanımının lehine karar verildi.  Java ile en son bir kaç yıl önce çalıştığımdan öncelikle hatırlatma amacıyla Java kod örneklerine bakıldı. Sonra Spring Boot'ta lambda ifadeleri, callback fonksiyonları, multithreading gibi konularla karşılaşınca bunlar da araştırıldı.  Daha sonra Spring Boot'ta web servisi tutoriallerine başlanıldı. Klasik olarak ilk "Hello World" örneği yazıldı.    Spring uygulamalarının yapıları MVC mimarisini kullanıyor ancak duruma göre "View" çıkartılabiliyor.  Spring Boot'a aşina olmak amacıyla örnek kodların yazımlarına devam edildi. Bir sonraki yazılan uygulama ise Spring Boot ile MongoDB kullanan bir örnekti. Body içinde gönderilen JSON'daki bilgilere göre bir "Person" objesi oluşturup MongoDB içinde saklıyordu ve daha sonra istek atarak geri getirebiliyordu.      Başka bir tutorial ise, bir Id ve isimden oluşan bir ürünü (Product) HTTP istekleri üzerinden manipüle etmek üzerineydi. Bu tutorialin öncekilere göre pek bir farkı görülemeyebilir ancak, Spring frameworkünde önemli olan birkaç yere değinilmişti.      Eski çıktılardaki gibi varsayılan bir cevap geri döndürmektense artık belirli bir cevap geri döndürülüyor ve ürünler JSON olarak gönderilip Spring tarafından otomatikman "Product" objesine çevriliyor.  Ayrıca spesifik ürünler web adresleri aracılığıyla manipüle edilebiliyor (.../products/**3**).    Bu özellikler yukarıda işaretlenen Annotationlar ve ResponseEntity objesi sayesinde gerçekleşiyor.  Bu örneğin devamı olarak da bir consumer (tüketici) web servisi yazıldı. Bu web servisi başka bir web servisiyle iletişim kurup aldığı cevabı geri döndürebiliyordu veya bir POST isteğini iletebiliyordu. Bu işlemler de RestTemplate objesiyle gerçekleşiyordu.    Son tutorial olarak spring üzerinden dosya yükleme ve indirme üzerine çalışıldı. Özet olarak, saf HTML'den oluşan bir ara yüz üzerinden seçilen doya POST isteği ile sunucuya yükleniyordu.    HTML ve Thymeleaf ile ilk defa burada karşılaşıldı. Sayfanın yapısı bir geribildirim mesajı, dosya seçme formu ve yüklenen dosyaları içeren listeden oluşuyordu.  Yukarıda işaretlenen yerler, geribildirim mesajı ve dosya listesi, Spring uygulaması tarafında dolduruluyor, sonra Thymeleaf aracılığıyla web tarayıcılarının anlayabileceği HTML şekline çevriliyor. Bu sayede Spring üzerinden gösterilen web sayfalarına dinamiklik katılıyor.        Sunucu tarafında Spring üzerinden nasıl web sayfası gösterilir, yüklenen dosyayı kabul etmek için gereken objeler ve parametreler ve herhangi bir dosyayı indirmek için gereken adımlar öğrenildi.  Ayrıca dosya yüklemek ve getirmek için gerekli olan servis sınıfı örnek dahilinde verilmemişti, tarafımca yazılmıştı. Sadece Spring değil Java açısından da yararlı bir örnekti.  Örneklerden sonra yetkili üstüm, aynı zamanda patron, tarafından iki sayıyı toplayan ve sonucunu geri döndüren bir web servisi yazmam, daha sonra deploy etmem ve bu servisi çağıran bir uygulama yazmam istendi.  İki sayıyı toplayan web servisini yazmak zor değildi. Toplama işlemini çağıran bir endpointten ve sayıları alan parametrelerden oluşuyordu. Local'de test edildikten sonra ücretsiz bir sağlayıcı olan Heroku'ya deploy edildi.      Daha sonra C# ile bu web servisini çağıran bir uygulamaya yazıldı.    En son olarak bu zamana kadar yapılanlar rapor edilerek yetkili üste gönderildi ve bir sonraki yapılacak olanlar hakkında konuşuldu. | |
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
| **TASDİK EDEN YETKİLİNİN** | |
| **İmza-Mühür** | |
|  | |
|

|  |  |
| --- | --- |
| **Staj İçeriği**  Staj kapsamında gerçekleştirdiğiniz çalışmalar hakkında detaylı bilgi veriniz. Gerektiğinde yeni sayfa ekleyiniz. | |
|
|
| **Tarih:** | **01/08/2022 - 14/08/2022** |
|
| **Konu** | Web servislerinde ve Spring'de güvenlik |
|
| Yetkili üstümle konuştuğumda, güvenlik konusunu araştırmamı ve yazdığım toplama servisini koruma altına almamı, izinsiz birinin erişememesini, istemişti ama kullanıcı adı ve şifre ile giriş tarzında değil.  Öncelikle web servislerinde güvenliğin nasıl sağlandığı araştırıldı. Bu kavramı araştırılırken erişimi kısıtlamaktan daha çok web servisinin saldırılardan korunmasına yönelik bilgiler bulundu. Saldırılardan korunmak da güvenlik kapsamındadır ancak ihtiyaç duyulan kısım burası değildi. Buna ek olarak bir kaç anahtar kavramlar da öğrenildi;   * Authentication (Kimlik Doğrulama): Bir kullanıcının iddia ettiği kişi olduğunun doğrulanması. * Authorization (Yetkilendirme): Kullanıcının yetki seviyesine göre belirli kaynaklara erişim izni verilmesi. * Confidentality (Gizlilik): Bilgilerin şifrelenerek saklı tutulması. * Integrity (Bütünlük): Çeşitli imzalama yöntemleriyle gönderilen bilginin aktarım sırasında değişime uğramadığından emin olunması.   Bu açıklamalardan yola çıkılarak şuana kadar integrity ve confidentality kavramlarıyla karşılaşıldı ve ana olarak authentication yöntemlerinin araştırılması gerekliydi.  Birden çok kaynağa göre 5 adet authentication yöntemi mevcuttur. Bunlar;   * Şifreyle doğrulama, * Birden çok doğrulama yönteminin birleştirilmesiyle oluşan çok katmanlı doğrulama (MFA), * Biyometrik doğrulama, * Sertifika ile doğrulama, * Tokenler ile doğrulama.   Daha sonra Spring'in hangi tür authentication yöntemlerini desteklediğini ve nasıl yapıldıklarına bakıldı (implementasyon açısından). Spring uygulamaları HTTP üzerinden sunulduğu için HTTP Authentication yöntemlerini destekliyor. Bu doğrulama yöntemleri basit olarak ikiye ayrılabilir;   * Basic (Temel) Authentication: Kullanıcı adı ve şifresini Base64 biçiminde şifreleyip HTTP Authentication headerine koyulmasıyla gerçekleşir. * Digest (Sindirme) Authentication: Kullanıcı adını ve şifresini bir hash fonksiyonundan geçirir, bu sayede basic authenticationdaki gibi kolayca geri çevrilemez. Ek olarak nonce de kullanır. Nonce sadece bir kere geçerli olan bir sayıdır.   HTTP authentication methodlarına ek olarak popüler olarak JWT (JSON Web Token) ile karşılaşıldı. JWT güvenli ve kompakt bir şekilde veri alışverişinin gerçekleşmesini hedefleyen standarttır. Üç kısımdan oluşur, header, payload ve signature. Header kısmında kullanılacak olan şifreleme algoritması, payload kısmında token'e ait bilgiler, signature kısmında ise dijital imza bulunur. Dijital imza header ve signature kısımlarından oluşturulduğu için token'in bütünlüğünü sağlamış olur.  JWT'nin çalışma biçimi ve örnek kodları kompleks gelmişti, ayrıca login sistemi istenmediğinden diğer doğrulama yöntemleri araştırılmaya başlandı.  OAuth üçüncü parti bir uygulamanın, kullanıcının izni üzerine kullanıcıya ait başka bir yerdeki kaynağa erişmesini sağlayan protokoldür. Örneğin telefondaki bir uygulama Google Drive'deki bir dosyaya erişmek istiyor ve bunun için de kullanıcının hesabına erişmesi gerekir. Hesap bilgilerini açık şekilde üçüncü bir partiyle paylaşmak güven ve güvenlik sorunu ortaya çıkarır. Bunun yerine uygulama kullanıcıdan bir token ister ve bu token de Google tarafından verilir. Bu sayede uygulama, hesap bilgilerine sahip olmadan dosyaya erişebilir.  OAuth 1.0 araştırılırken "API'yi koruma altına alır", "Kullanıcı adı ve şifreye gerek yoktur" gibi söylemlerle karşılaşınca istenilen güvenlik biçimine uyacağı düşünüldü ve OAuth 1.0'ın Spring'deki implementasyonları araştırıldı, ancak OAuth 1.0 eski ve kompleks olarak tanımlandığından Spring implementasyonu hiç bulunamadı. Bunun üzeirne OAuth 2.0 araştırıldı.  OAuth 2.0'ın da işlevi OAuth 1.0 ile aynıdır, aralarındaki fark ise implemetasyon farkındandır. OAuth 2.0 implementasyonun daha kolay ve daha hızlı çalıştığı kabul edilir.  OAuth 2.0'ın Springdeki kullanımına bakıldığında ise yukarıda yazıldığı hep login tarzı örnekler bulundu. İsteğe uygun olmayınca bu yöntemlerden de vazgeçildi.  Şuana kadar araştırılan yöntemlerden ziyade daha önce başka bir API kullanırken karşılaştığım API anahtarı aklıma gelmişti. API anahtarlarını araştırdığımda istenilen şekilde sadece bir anahtar/şifre ile doğrulamayı sağlayabiliyor olduğunu öğrendim.  İlk olarak API anahtarı sistemini sadece tek bir anahtar kabul edecek şekilde ve interceptorler ile denedim. Interceptorler, Spring'de bir isteğin işlenmesinden önce veya sonra çağrılan fonksiyonlardır. "x-api-key" headerinin içindeki değeri alıp önceden belirlenmiş anahtarla karşılaştırıyor duruma göre true ya da false geri döndürüyordu. preHandle() fonksiyonunda false geri döndürüldüğü zaman HTTP isteğinin işlenmesi durduruluyordu.    Daha sonra interceptorlerin güvenlik açısından kullanımının uygun olup olmadığına bakıldı. Spring'de güvenlik zincirleme filtrelerle gerçekleştirilir ve bu filtreler istek kabul edilmeden önce çalıştığı için interceptorler yerine Spring Security'nin kullanılması öneriliyordu.  Spring'de API anahtarı implementasyon örneklerini araştırdım. Bir kaç tane örnek bulabildim. Önce denediklerim versiyon farklılığı yüzünden çalışmadı. En son denediğim ise çalışıyor gibiydi ancak beklenmedik hatalar alınıyordu. Sebep olabilecekler hatalar ve benzer kod örnekleri araştırıldı ancak    Spring'de özel (custom) güvenlik filtreleri hakkında uzun ve karmaşık resmi dokümantasyonu haricinde fazla bir bilgi bulunamadı. Dokümantasyonda filtrelerin zincirleme olduğu ve farklı filtre sınıfları yer alıyordu ancak böyle bir yapı hakkında pek bir fikrim yoktu. Kodu debug modunda adım adım çalıştırdığımda zincir yapısı çalışırken hata oluştuğunu fark ettim. Fonksiyonu dikkatli incelediğimde parametre olarak bir FilterChain objesi aldığını gördüm. Bu objeyi kurcalarken doFilter() metoduna denk geldim. Fonksiyon kesilmeden önce chain.doFilter(request, response) kod parçasını ekledikten sonra hata giderildi.  Sonra kodun içine yerleşik olan API anahtarını örnekteki gibi. properties dosyasının içine taşıyıp Spring'in annotationları ile çekmeye çalıştım. Örnekte verilen şekilde denediğimde olmamıştı. Hatayı araştırdığımda yaygın olduğunu gördüm, birden fazla çözüm yazılmıştı. Çok fazla şekilde @Autowired annotation'u denendi ancak sadece bir tanesi çalıştı.    Endpointlerin güvenliğini test ederken de iki sorunla karşılaştım. İlki yanlış anahtar girildiğinde yanlış hata kodu ve hata sayfasını geri döndürmesiydi. Beklenilen cevap ise 401 kodu geri döndürmesiydi. Çalışma kayıtlarına bakınca hata verdiğini de gördüm. İnternette araştırma ve debug sonucunda, geçersiz anahtar girilmesi durumunda geri dönecek cevabı manüel olarak ayarlamam sonucunda oluştuğunu anladım. Bu kod parçalarını kaldırınca veya reset() fonksiyonunu çağırınca hata düzelmişti.    İkinci hata ise her endpointte anahtar kontrolünün çalışmasıydı. Hangi adreslerin koruma altına alınacağı configure() metodunun aldığı HttpSecurity objesinin üzerinden yapılıyordu. İstenilen endpointler ve istek türleri AntPathRequestMatcher objesi ile ekleniyordu. Diğer endpointler ise .anyRequest().permitAll() fonskiyonları ile güvenliğe dahil edilmeden kalıyordu.  Hatalar çözüldükten sonra beklenmedik bir durum daha vardı. Bu durum yanlış anahtar girildiğinde 401 yerine 403 kodu geri döndürülmesiyle oluşuyordu. Custom filtrenin pozisyonunu değiştirerek çözümü denendi ancak ya bir değişiklik olmuyor ya da çalışmıyordu. İnternette hatayı araştırdığımda da kendi durumumla ilgili sonuçlar bulamadım. Kodun doğru çalışmasına engel olmadığı için bu durumu es geçtim.  Son olarak kısa bir rapor hazırlayıp yetkili üstüme gönderdim. Sonrasında da telefon üzerinden bu güvenlik sisteminin, şu anlık sadece bir tane anahtar olması, ya da bir anahtarın açığa çıkması durumunda olabilecek durumlar vb. eksik tarafları hakkında konuştuk.  P.S.: Aslında Basic veya Digest yöntemleri de kullanılabilirdi ve kullanan sitelerde var. Kullanıcı adı ve şifreyi görünce sadece bir login şeklinde kullanılabileceğini düşünmüştüm. İster header'e koyulsun ister body'e, önemli olan sunucu tarafında nasıl işleneceği (Ancak framework kullanılması bir nevi kısıtlıyor) ve bağlantının güvenli olması. Örneğin Stripe API'sine erişim, API anahtarıyla çalışıyor ancak bu sistemin altında Basic yöntemi var, kullanıcı adı kısmını boş bırakarak gerçekleştiriyor. Yani pek fazla açıklaması ve örneği olmayan özel bir sınıf yazmaya gerek yoktu, hali hazırdaki bir Basic örneğini temel alarak kullanıcı adı kısmını boş bırakabilirdim. | |
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
| **TASDİK EDEN YETKİLİNİN** | |
| **İmza-Mühür** | |
|  | |
|