

# Web Programlama 2

## Lesson 12: CI ve Deployment

---

# Slaytlar Hakkında

---



BU SLAYTLAR OLDUKÇA YÜKSEK  
SEVİYEDE GENEL BİR BAKIŞ AÇISI  
SUNMAKTADIR



DETAYLAR GİT REPOZİTORY'SİNDEKİ  
YORUM SATIRLARINDA  
BULUNMAKTADIR

# Continuous Integration (CI)

---



# Code Evrimi

---

Her kod değiştirdiğinizde uygulamanızın doğru bir şekilde çalışıyor olmasını istersiniz

Olası Problemler:

- Kod compile edilemeyebilir
- Değişiklikler bazı işlevsellikleri bozabilir ve bazı testler hataya düşebilir

Peki ne zaman kontrol edilmeli? Her bir Git Push işleminde

Geliştiricilere her commit öncesinde “*mvn clean verify*” yazması istenebilir, ancak:

- Unutulabilir
- Test senaryolarının çalışması **saatlerce** sürebilir

# CI Sunucuları

---

Her bir Push sonucunda otomatik olarak Git'den pull yapan sunuculardır

Uygulamanızı build eder ve testleri çalıştırır

Eğer hataya düşerse kullanıcıları bilgilendirir (ör, email, Telegram vs.)

Build geçmişini takip edebilmenizi sağlar

Birleştirmeden önce Git PR (Pull Request) kontrolü yapar

*Jenkins* en fazla kullanılan CI sunucusudur ve kendi makinalarınıza kurabilirsiniz

Takım ile birlikte çalışırken aşırı derecede faydalıdır

- CI kullanmayan bir şirkette çalışmaya başlarsanız, **oradan kaçın!** 😊

# CI Sağlayıcılar

---

Açık kaynak projeler için bazı CI sağlayıcıları bulunmaktadır

- GitHub Actions, CircleCI, Travis (2020'ye kadar), vs.

## GitHub Actions

- Repository GUI'sinden aktifleştirmeniz gerekir
- `.github/workflows/ci.yml` gibi bir ayar dosyasına ihtiyaç bulunur

Veritabanı Bakımı



Şu ana kadar Hibernate ayarlarından «create-drop» tercih etmiştik böylelikle her bir çalıştırmada veritabanı şeması yeniden oluşturuluyordu



Tabii bu production ortamında çalışmadığımız içindi. Her uygulama başlamasında veritabanını silmeyi istemezsiniz!!!



Mümkün (ancak iyi değil) çözümlerden biri “*update*”’dir

Eğer veritabanı başta bulunmuyorsa oluşturur (entity’inize bağlı olarak) aksi takdirde mevcut olanı güncellemeye çalışır

# Database Migration



Eğer yeni bir  
sütun @Entity  
eklediyseniz ne  
olur?

Bazı @Entity  
sınıflarınızı  
değiştirirseniz  
ne olur (bir  
tabloyu ikiye  
ayırmak gibi)?

Yanlışlıkla veya  
bug sonucu  
@Entity sınıfınızı  
silerseniz ne  
olur?

*Eğer mevcut  
satır  
veritabanında  
bulunuyorsa ne  
olacak?*

“Update” ile  
ilgili  
problemler

# Çözüm

---

- Veritabanının değişimi bazı özel araçlarla ele alınmalıdır
- *@Entity* sınıfları yalnızca mevcut veritabanında ne bulunduğunu göstermelidir veritabanı şema oluşturma/güncelleme işlemlerini yapmamalıdır
  - İlk production yayınlamasından ayrı olarak
- “*create-drop*” ve “*update*” yerine “**validate**”
  - Eğer veritabanı şeması *@Entity* ile eşleşmezse exception fırlatır
- Araçlar: *Flyway* veya *Liquibase*

# Flyway

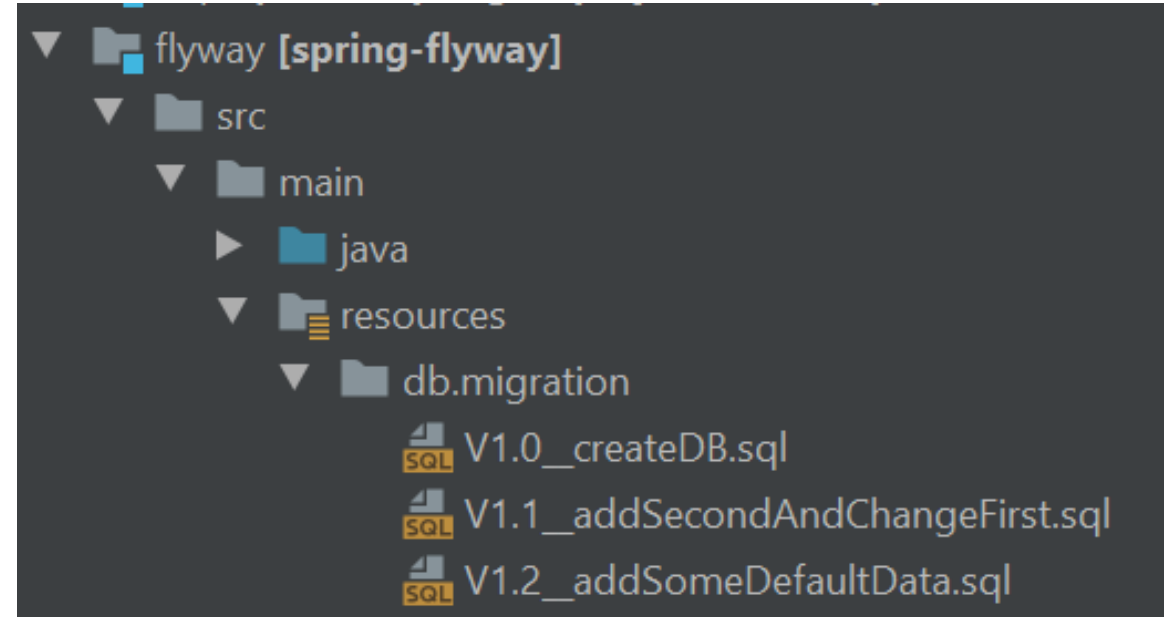
---

- Bütün operasyonlar SQL dosyalarındaki SQL komutları ile yürütülür
- Her bir migration'da dosyanın bir versiyon numarası vardır ve sıra ile artar
- *Flyway* daha önce bir migration gerçekleşti mi kontrol eder gerçekleşmediyse yalnızca bir sefer uygular
- Hangi migration'lar gerçekleşti takip etmek için kendi tablosunu oluşturur
- SpringBoot eğer classpath'de bulursa otomatik olarak FlyWay'i çalıştırır

# db/migration Klasörü

---

- Migration dosyaları  
*Vx.y\_someName.sql* yapısındadır
- Her bir çalıştırma sırayla gerçekleşir, ör  
*V1.0*, *V1.1*'den önce çalıştırılır
- Hatırlatma: IntelliJ /resource altındaki  
dosyaları package olarak gösterir
  - yani “*db.migration*” klasörü  
bulunmamaktadır kalsör  
“*db/migration*” şeklindedir



Loglama



# Log



Uygulama içerisinde neler gerçekleştiğini takip etmek önemlidir



Özellikle bir yerde bug bulunuyorsa oldukça önemlidir ve exception'ların stack-trace'lerini kaydetmek isteyebilirsiniz

Bu türde bir bug'ı debug etmek için gerekir



Logları kaydetmek ise biraz zordur bu yüzden yardımcı olacak kütüphaneler bulunmaktadır

I/O işlemleri yaptığı için Just-in-Time (JIT) compiler'ı etkiler bu yüzden çeşitli optimizasyonlar yapılmalıdır

# SLF4J ve Logback

SLF4J Java loglama için en popüler kütüphanedir

SLF4J üzerine kurulu farklı framework'ler de bulunmaktadır ve Logback en popüler kütüphanelerden biridir

Pek çok 3. parti uygulama SLF4J kütüphanesini kullanır

- Örneğin SLF4J, JPA gibiyse Logback de Hibernate gibidir

# Logger



Genellikle her bir sınıf için bir logger üretilir ve isim olarak da sınıfın kendisi verilir

using  
`LoggerFactory.getLogger(name)`  
ile oluşturulur ve final static bir değişken olarak saklanır



Ayarlamalar *logback.xml* dosyası içerisindedir



Test için önceliğe sahip olacak farklı *logback-test.xml* file dosyası bulunur



Pek çok olası ayarlamalar mevcuttur

ör, log ile ne yapılacak, yalnızca konsola mı yazılacak, dosyaya mı yazılmalı, uzak sunucuya mı gönderilmeli vs.



# Log İfadesi ve Seviyeler

- Farklı Metotlar: **log.debug(msg)** ve **log.error(msg)** gibi
- Loglama seviyesine bağlı olarak bazı mesajlar gösterilmez
  - Örnek: Sisteme DEBUG seviyesinde logları gösterme, WARN'ları konsola yaz ERROR'ları hem konsola hem dosyaya yaz diyebilirsiniz
- Seviyeler: **TRACE, DEBUG, INFO, WARN** ve **ERROR**
- Bunlar bir sıradadır: eğer bir seviyeyi aktif edersen üstteki seviyeler de aktif olur
  - ör, DEBUG aktif edilirse TRACE hariç hepsi aktifleşir

# Log Level Ayarları

Seviyeler arası ince ayar yapılabilir

Bütün uygulama için bir log-seviyesi bulunabilir, ör: WARN veya ERROR

Daha sonra bazı log seviyeleri override edilebilir

- Sınıflarınıza INFO seviyesi oluşturabilirsiniz ancak 3. parti uygulamalar eklenemez

Test/Debug sırasında bazı sınıflara sadece DEBUG seviyesi verebilirsiniz

# String Birleştirme

---

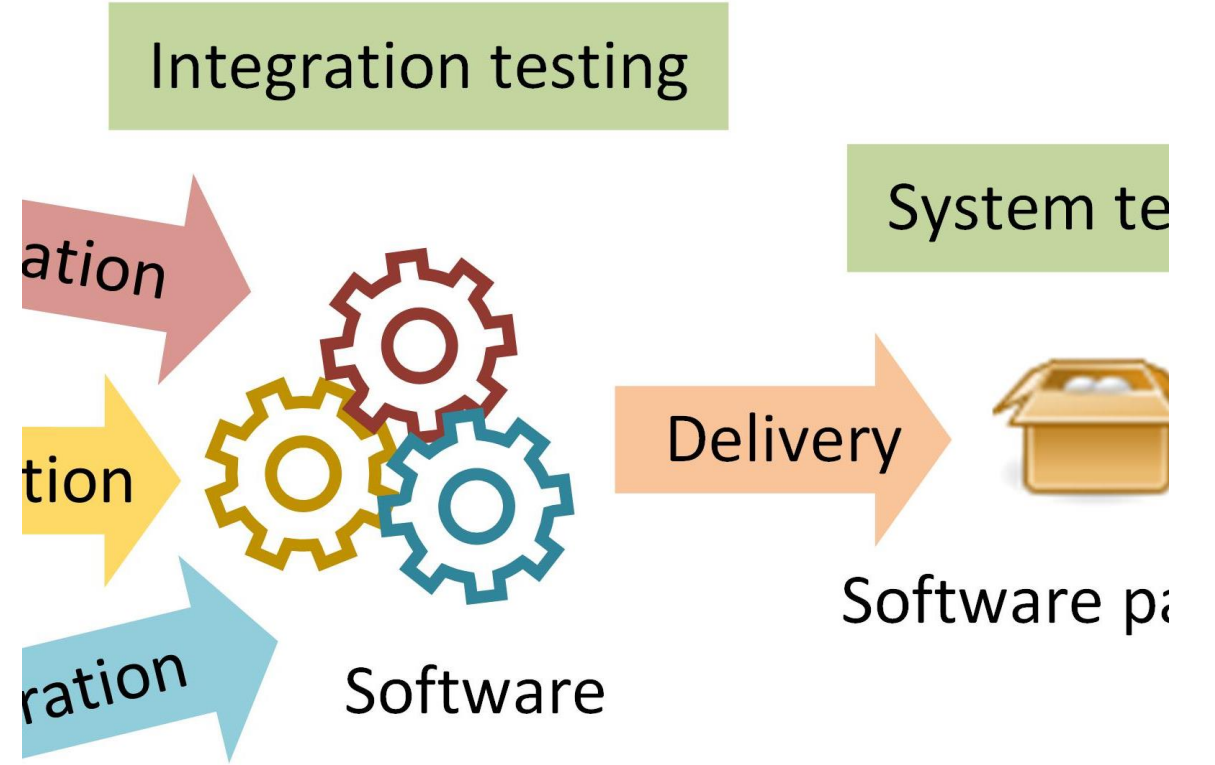
- `log.debug("" + x + "=" + y)`'i ele alalım
- Bu kötü bir yaklaşımdır: sıklıkla Debug seviyesinde log'lar yok sayılır ancak `"" + x + "=" + y` işlemi hesaplanır ve boş yere CPU harcar
- String birleştirme **maliyetlidir**: hatırlatma: String ifadeler immutable'dır ve her bir `+` operatörü uygulandığında tamamen yeni bir String ifade oluşturur
- Çözüm: `log.debug("{}={}", x, y)`
  - Log ifadeleri string enterpolasyona izin verir ve `{}` ifadeleri ile yapabilirsiniz
  - bir log yoksayılırsa (örneğin, WARN seviyesi), String, enterpolasyona gerek kalmadan atılır

# Cloud Deployment

A thick, hand-drawn style orange line that underlines the text "Cloud Deployment". It starts under the 'C' and ends under the 't', following the width of the text.

# Deployment

- Eğer uygulamanız hazırsa deploy etmeniz gerekir
- Ancak nereye?
- Kendi sunucunuza deploy edebilirsiniz ancak her şeyiyle kendiniz ilgilenmeniz gerekir
  - donanım (alım ve bakım), yedekleme, DNS vs...
- Pek çok firma bunu yapmaktadır ancak startup ve bireysel girişimler için zordur



# Cloud Deployment

Farklı firmalar belli bir ücret karşılığında cloud hizmeti sunmaktadır.

*Amazon Web Services (aws)* içlerinde en ünlülerinden biridir.

- Netflix AWS üzerinde çalışmaktadır

*Otomatik ölçekleme:* Eğer daha fazla yük oluşursa, daha fazla node kiralanabilir ve böylelikle yük ölçeklenebilir

# Cloud Tanimi



# Heroku

- Önemli cloud sağlayıcılarından biridir.
- Şu anda kullanımı kolay ve ücretsizdir.
  - Ücretsiz olması ilerleyen zaman diliminde değişebilir
- Java ve SpringBoot uygulamalarını destekler
- Komut satırından çalıştırılabilir JAR dosyasının deploy edilebildiği maven plugin'i bulunur
- *Otomatik olarak* Spring'i Heroku veritabanını kullanabilir şekilde ayarlayabilir



# Heroku Kullanımı

- İlk olarak [www.heroku.com](https://www.heroku.com) adresinden bir hesap oluşturulmalıdır
- Heroku komutlarını komut satırı aracılığı ile çalıştırmanızı sağlayan CLI'ı kurun.
- Web arayüzünden uygulamanın adını girerek yeni bir uygulama oluşturun.  
Örnek: bs436-wp
  - Bu isim tekil olmalıdır.

# Jar Deployment



*heroku-maven-plugin* ayarlamasını yapın



Maven çalıştırılmalıdır



**mvn clean package heroku:deploy -Dheroku.logProgress=true**



Uygulama <https://bs435-wp.herokuapp.com> adresi  
üzerinden erişilebilir olacaktır

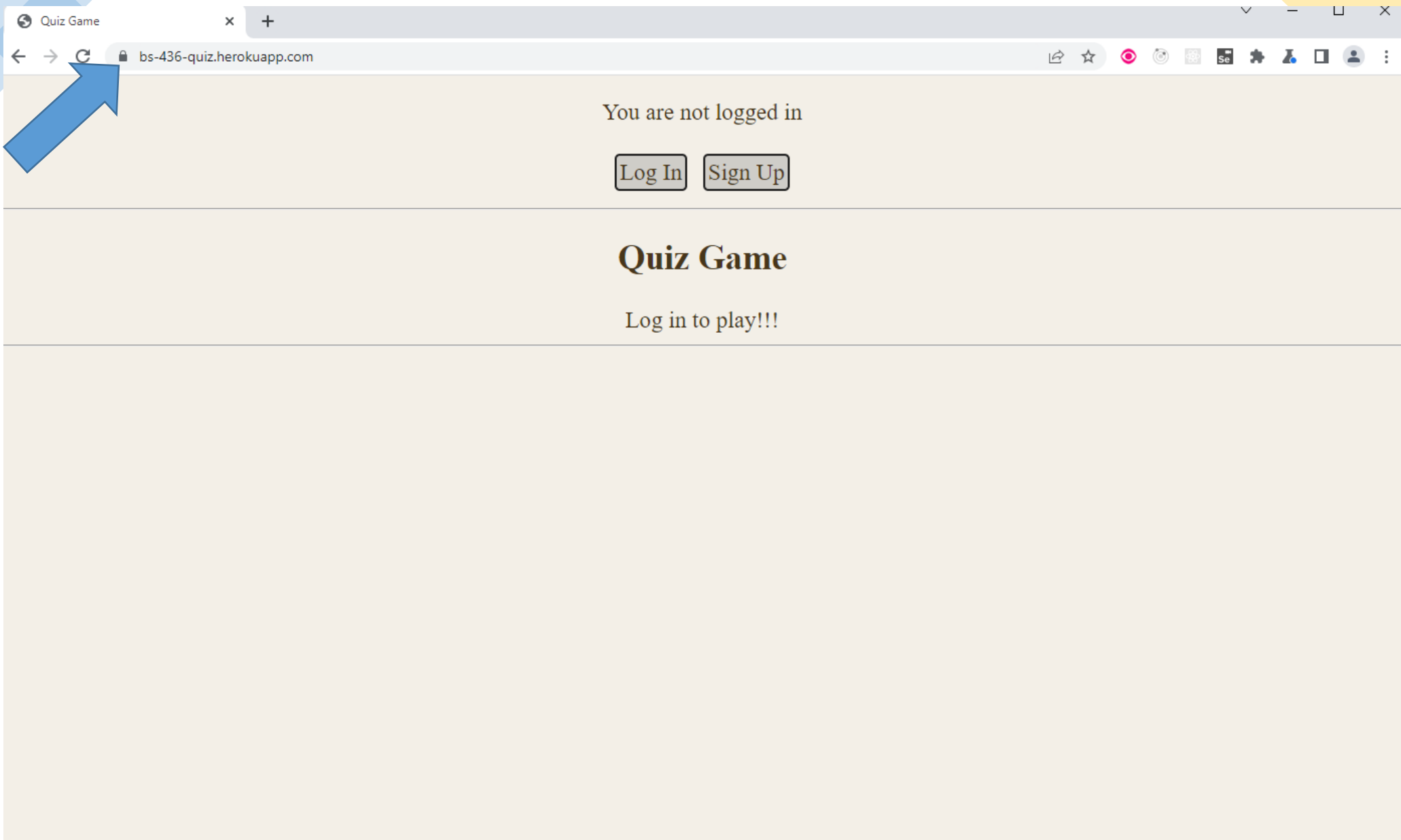
Not: https üzerinden



Ancak erişmeden önce ortam ayarlamalarını yapmak gerekir

# Komut Satırından (CLI)

- **heroku login**
  - Kimliklendirme gerektiren komutları çalıştırmak için giriş yapmayı sağlar
  - Not: Eğer windows kullanıyorsanız GitBash ile çalışmayabilir, dahili terminali kullanarak çalıştırabilirsiniz
- **heroku ps:scale web=1 --app bs-436-quiz**
  - Node kaynaklarını uygulamayı çalıştırmak için uygun hale getirir
  - Not: henüz JAR'ı en az bir sefer deploy etmediyseniz *"Scaling dynos... ! Couldn't find that process type"* hatası alabilirsiniz
- **heroku addons:create heroku-postgresql --app bs-436-quiz**
  - Postgres ekleme
- **heroku pg --app bs-436-quiz**
  - Postgres'in mevcut durumunu görüntüle
- Not: Bazı komutlar web arayüz ile de gerçekleştirilebilir





# Continuous Delivery (CD)

- Deployment kısmı maven'in buildine ait bir parça olarak da gerçekleştirilebilir
  - Git Push sonucunda CI sunucu (Travis veya Jenkins gibi) deployment için tetiklenebilir
  - Tabii yalnızca kod compile edilir ve bütün testler geçerse...
  - Ancak deployment için Git'de özel bir branch açmak isteyebilirsiniz
    - ör, geliştirme için "*development*" git branchi açılır ve geliştirme süreci burada yürürken burada sonuçlanan süreçler "deployment" branchi ile merge edilir
-

# Peki daha sonra?

- Şimdiye kadar öğrendiklerinizle eksiksiz bir enterprise web uygulaması gerçekleştirebilirsiniz
    - GUI, güvenlik, test, veritabanı, cloud deployment vs.
  - Ancak bu tür uygulamalar monolitik olarak adlandırılır ve büyük sistemlerde pek ölçeklenebilir değildir
  - Daha sonraki ders olan Web Programlama 2 dersinde:
    - Web Servislerin ayrıntılarına ineceğiz ve HTTP protokolünü ayrıntılı inceleyeceğiz
    - Mikroservisler gibi dağıtık sistemleri inceleyeceğiz
    - JS+Ajax+WebSoketler'in frontend ile entegrasyonunu inceleyeceğiz
-



# Git Repository Modülü

- *NOT: açıklamaların büyük bir çoğunluğu kod içerisinde yorum satırı olarak bulunmaktadır, burada slaytlarda bulunmamaktadır*
  - **intro/spring/flyway**
  - **intro/spring/logging**
  - **intro/spring/deployment**
  - Ders 11 alıştırması (dokümantasyona bakınız)
-