Git İş Akışı Kılavuzu

Bu kılavuz, 4 katmanlı projemizde ekip üyeleri (A, B, C, D) arasında sorunsuz bir iş birliği sağlamak için Git iş akışını açıklamaktadır. Çakışmaları önlemek ve kod entegrasyonunu doğru yapmak için bu adımları dikkatlice takip edin.

Proje Yapısı ve Atanan Görevler

Projemiz 4 katmandan oluşuyor ve bu hafta 4 ekip üyesine şu şekilde görevler atanmıştır:

Uygulama (Application) Katmanı

- **Kişi A**: `features/Auth` klasöründe çalışacak (command, queries, handler, DTO).
- **Kişi B**: `features/User` klasöründe çalışacak (command, queries, handler, DTO).

Altyapı (Infrastructure) Katmanı

- **Kişi C**: `data` klasöründe çalışacak.
- **Kişi D**: `services` klasöründe çalışacak.

Her kişi, kendisine atanan klasörde çalışmalı ve yalnızca kendi alanına ait değişiklikleri stage edip commit etmelidir.

Git İş Akışı

Çakışmaları önlemek ve iş birliğini sağlamak için şu sıralı Git iş akışını izleyin:

1. **Çalışma sırasını belirleyin**:

- Uygulama Katmanı için: A ve B, kimin önce başlayacağına karar verir (örneğin, A başlar, sonra B).

- Altyapı Katmanı için: C ve D, kimin önce başlayacağına karar verir (örneğin, C başlar, sonra D).

1. **Adımlar**:

- **İlk kişi (örneğin, A)**:
    1. Kendi kodlarını yazar ve tamamlar.
    2. Değişiklikleri yalnızca kendi çalıştığı klasör için stage eder. **Önemli**: `git add .` kullanmayın! Örneğin, A için:

```
 bash
git add features/Auth/*
```

    3. Değişiklikleri commit eder:

```
 bash
git commit -m "Auth klasöründe command ve queries eklendi"
```

    4. Emin olmak istiyorsanız, git status komutunu çalıştırarak hangi dosyaların stage edildiğini kontrol edebilirsiniz.
    5. Kodları Git'e push eder:

```
 bash
git push origin main
```

- **İkinci kişi (örneğin, B)**:
    1. İlk kişinin push ettiği kodları çeker:

```
 bash
git pull origin main
```

2. Kendi kodlarını yazar ve tamamlar.
3. Değişiklikleri yalnızca kendi çalıştığı klasör için stage eder. Örneğin, B için:

```
 bash
git add features/User/*
```

4. Değişiklikleri commit eder:

```
 bash
git commit -m "User klasöründe command ve queries eklendi"
```

5. Emin olmak istiyorsanız, git status komutunu çalıştırarak hangi dosyaların stage edildiğini kontrol edebilirsiniz.
6. Kodları Git'e push eder:

```
 bash
git push origin main
```

- **Üçüncü ve dördüncü kişiler (C ve D)**:
  - Aynı mantığı izler. Örneğin, C önce çalışıyorsa:

1. Önceki kodları çeker:

```
 bash
  git pull origin main
```

2. Kendi kodlarını `data` klasöründe yazar.
3. Değişiklikleri stage eder:

```
bash
 git add data/*
```

4. Commit eder:

```
bash
 git commit -m "Data klasöründe değişiklikler yapıldı"
```

5. Emin olmak istiyorsanız, git status komutunu çalıştırarak hangi dosyaların stage edildiğini kontrol edebilirsiniz.
6. Push eder:

```
bash
 git push origin main
```

- D, aynı adımları `services` klasörü için takip eder:

```
bash
  git pull origin main
  git add services/*
  git commit -m "Services klasöründe değişiklikler yapıldı"
  git status
  git push origin main
```

Önemli Notlar

- **Sakın `git add .` kullanmayın!** Yalnızca kendi çalıştığınız klasördeki dosyaları stage edin.
- Her zaman anlamlı commit mesajları yazın (örneğin, "Auth klasörüne DTO eklendi").
- Kodları push etmeden önce `git pull` ile en güncel değişiklikleri aldığınızdan emin olun.

- Çakışma (conflict) durumunda, ekip ile iletişime geçin ve çakışmayı çözün.

Örnek: Infrastructure Katmanında `features/Auth` için Git Komutları

Diyelim ki **Kişi A**, `Infrastructure` katmanında `features/Auth` klasöründe çalışıyor. Komutlar şu şekilde olacaktır:

```
 bash
git add features/Auth/*
git commit -m "Auth klasörüne command ve handler eklendi"
git push origin main
```

Proje Dizini

Proje ve Git dosyalarınız şu dizinde yer alıyor:

```
C:\Users\furka\OneDrive\Masaüstü\Proje
```

Bu adımları takip ederek projemizi düzenli ve çakışmasız bir şekilde tamamlayabiliriz. Herkese iyi çalışmalar!