GYM COACHES FLUTTER GELİŞTİRME REHBERİ

1. GİRİŞ VE GENEL KURALLAR

Proje Yapısı

Bu projede **MVVM (Model-View-ViewModel)** mimarisi kullanılmaktadır. Flutter ile geliştirme yaparken:

- **View**: Sadece UI mantığı (sizin sorumluluğunuz)
- **ViewModel**: State yönetimi ve business logic (mentor sorumluluğu)
- **Model**: Veri modelleri (mentor sorumluluğu)

Temel Prensipler

- Clean Architecture prensiplerini takip edin
- SOLID prensiplerini uygulayın
- DRY (Don't Repeat Yourself) kuralına uyun
- Kod okunabilirliğini öncelik yapın

---

2. UI GELİŞTİRME KURALLARI

Çalışabileceğiniz Klasörler

- `lib/presentation/common/widgets/`: Ortak UI bileşenleri
- `lib/presentation/common/theme/`: Tema ayarları
- `lib/presentation/features/*/views/`: Ekran tasarımları
- `lib/presentation/features/*/widgets/`: Özelliğe özgü widget'lar
- `lib/routes.dart`: Routing yapılandırması

Dokunmamanız Gereken Klasörler

- `lib/core/`: Temel altyapı kodları
- `lib/data/`: Veri modelleri ve repository'ler
- `lib/domain/`: Domain katmanı
- `lib/presentation/*/viewmodels/`: ViewModel'lar (mentor tarafından sağlanacak)

UI Geliştirme Kuralları

- Responsive tasarım prensiplerini uygulayın
- MediaQuery kullanarak farklı ekran boyutlarını destekleyin
- Tema renklerini direkt kullanmak yerine ThemeData'dan alın
- Sabit değerler için ölçüleri `lib/core/constants/` altından kullanın
- Widget'ları mümkün olduğunca küçük ve tekrar kullanılabilir yapın

Ortak Widget Kullanımı

- Benzer UI bileşenlerini `common/widgets` altında oluşturun

- Widget parametrelerini required/optional olarak doğru işaretleyin
- Widget dokümantasyonu ekleyin (`///` ile yorum satırları)
- Komplex widget'ları stateless ama callback fonksiyonlarıyla yapın

---

3. ROUTING (GoRouter ile)

Router Yapılandırması

- Yeni ekranları `routes.dart` dosyasındaki router yapılandırmasına ekleyin
- Route tanımları için GoRoute yapısını kullanın
- Parametreler ve nested routes için GoRouter özelliklerini kullanın
- Route isimlerine AppRoutes sınıfından erişin

Örnek Routing Kodu

```
// Yeni route ekleme örneği
GoRoute(
  path: '/profile',
  name: 'profile',
  builder: (context, state) => const ProfileScreen(),
),

// Alt route ekleme örneği
GoRoute(
  path: '/students',
  name: 'studentList',
  builder: (context, state) => const StudentListScreen(),
  routes: [
    GoRoute(
      path: ':id/edit',
      name: 'editStudent',
      builder: (context, state) {
        final studentId = state.params['id']!;
        return EditStudentScreen(studentId: studentId);
      },
    ),
  ],
),

// Ekran geçişi örnekleri
// İsimle geçiş
context.goNamed(AppRoutes.profile);

// Parametreli geçiş
context.goNamed(
  AppRoutes.studentDetail,
  params: {'id': student.id},
);

// Query parametreleri ile geçiş
```

```
context.goNamed(
  AppRoutes.studentList,
  queryParams: {'filter': 'active'},
);

// Geri dönme
context.pop();
```

4. MVVM MİMARİSİNE UYUM

View (Sizin Sorumluluğunuz)

- Sadece UI mantığı içermeli
- ViewModel'dan değerleri ConsumerWidget ile almalı
- Business logic içermemeli, bunun için ViewModel kullanılmalı
- UI event'lerini ViewModel'a iletmeli

ViewModel (Mentor'un Sorumluluğu)

- State yönetimi için Riverpod kullanılır
- UI durumunu ve kullanıcı etkileşimlerini yönetir
- Repository'lerden veri alır, işler ve View'a sunar

Örnek View Yapısı

```
class LoginScreen extends ConsumerWidget {
  const LoginScreen({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context, WidgetRef ref) {
    // ViewModel'dan state'i okuma
    final loginState = ref.watch(loginControllerProvider);

    return Scaffold(
      // UI implementasyonu
      body: Column(
        children: [
          // UI elementleri
          ElevatedButton(
            // Event'i ViewModel'a iletme
            onPressed: () =>
ref.read(loginControllerProvider.notifier).login(email, password),
            child: const Text('Login'),
          ),
        ],
      ),
    );
  }
}
```

5. KODU DÜZENLEME VE KALİTE

Kod Formatı

- Dart formatı kullanın: `flutter format .`
- Linter kurallarına uyun: `flutter analyze`

Naming Convention

- Sınıflar: PascalCase (LoginScreen)
- Metotlar/değişkenler: camelCase (getUserData)
- Widget özellikleri: camelCase (backgroundColor)
- Dosya isimleri: snake_case (login_screen.dart)

Yorum Satırları

- Kompleks logic için yorum satırları ekleyin
- Widget'lar için dokümantasyon yazın (/// ile)
- TODO'ları `// TODO: açıklama` formatında yazın

6. İLETİŞİM VE GERİ BİLDİRİM

Sorular ve Engeller

- Mimari sorular için direkt mentor'a sorun
- UI tasarımı için gerekirse mockup talep edin
- Blocker'ları hemen bildirin, bekletmeyin

Geri Bildirim Süreci

- PR'ler için geri bildirimleri dikkate alın
- Değişiklik talepleri için yeni commit'ler ekleyin
- Anlamadığınız geri bildirimler için açıklama isteyin

7. GÖREVLER VE İLERLEME

GitHub Issues

- Size atanan görevleri GitHub Issues'dan takip edin
- Bir göreve başlamadan önce ilgili issue'ya yorum yapın
- Görevi tamamladığınızda PR'da issue numarasını belirtin (#123 gibi)
- PR merge edildiğinde issue otomatik kapanacaktır

Zaman Yönetimi

- Her görev için tahmini süre belirleyin
- Bir görev beklenenden uzun sürerse mentorünüze bildirin
- Paralel çalışabileceğiniz görevleri belirleyin

8. GITHUB KULLANIM REHBERİ

1. GitHub Temel Kavramlar

Repository (Repo)

Projenizin tüm dosyalarını, tarihçesini ve konfigürasyonlarını içeren depo.

Branch

Projenin ana kodundan ayrı olarak geliştirme yapabileceğiniz dal.

- **main**: Ana branch, kararlı sürüm
- **develop**: Geliştirme branch'i (burayı kullanacağız)
- **feature/xxx**: Özellik geliştirme branch'leri

Commit

Dosyalarda yapılan değişikliklerin kaydı.

Pull Request (PR)

Bir branch'teki değişikliklerin başka bir branch'e birleştirilmesi talebi.

Issue

Projede yapılacak görev, düzeltilecek hata veya geliştirilecek özellik.

2. Temel Git Komutları

Repo Klonlama

```
git clone https://github.com/DenizDogan21/gym_coaches.git
cd gym_coaches
```

Branch İşlemleri

```
# Mevcut branch'i görüntüleme
git branch

# Yeni branch oluşturma
git checkout -b feature/ui-login

# Branch değiştirme
git checkout develop

# Remote'dan branch güncelleme
git pull origin develop
```

Değişiklik Yapma ve Commit

```
# Değişiklikleri görmek için
git status

# Değişiklikleri staging'e ekleme
git add .  # Tüm değişiklikleri ekler
git add file.dart  # Belirli bir dosyayı ekler

# Commit oluşturma
git commit -m "Add login screen UI"

# Remote'a push etme
git push origin feature/ui-login
```

Diğer Branch'lerden Değişiklikleri Alma

```
# Önce current branch'inizi commit edin
git add .
git commit -m "WIP: Current changes"

# Develop branch'inden güncellemeleri alın
git checkout develop
git pull origin develop

# Tekrar kendi branch'inize dönün
git checkout feature/ui-login

# Develop'daki değişiklikleri kendi branch'inize alın
git merge develop

# Conflict varsa çözün ve commit edin
```

3. GitHub Web Arayüzü Kullanımı

Pull Request (PR) Oluşturma

1. GitHub'da repo sayfasına gidin
2. "Pull requests" sekmesine tıklayın
3. "New pull request" butonuna tıklayın
4. Base: develop, Compare: feature/ui-login seçin
5. "Create pull request" butonuna tıklayın
6. Başlık ve açıklama girin
7. "Create pull request" butonuna tıklayın

Issues İnceleme ve Oluşturma

1. GitHub'da repo sayfasına gidin
2. "Issues" sekmesine tıklayın
3. Mevcut issue'ları inceleyebilirsiniz
4. "New issue" butonuyla yeni issue oluşturabilirsiniz
5. Size atanan issue'ları "Assignee: @your-username" filtresiyle bulabilirsiniz

Code Review Yapma/Alma

1. PR sayfasında "Files changed" sekmesine tıklayın
2. Kod satırları üzerine tıklayarak yorum yapabilirsiniz
3. Genel yorumlar için "Review changes" butonunu kullanın
4. "Comment", "Approve" veya "Request changes" seçeneklerinden birini seçin

4. GitHub'da İş Akışı (Gym Coaches Projesi)

Yeni Bir Göreve Başlama

1. GitHub Issues'da size atanan görevi bulun
2. Issue detaylarını okuyun
3. Develop branch'ini güncelleyin:

```
git checkout develop
git pull origin develop
```

4. Yeni bir feature branch'i oluşturun:

```
git checkout -b feature/ui-[görev-adı]
```

5. Görev üzerinde çalışın ve düzenli commit'ler yapın
6. Tamamladığınızda PR açın ve mentorünüzü reviewer olarak ekleyin

PR Geri Bildirimleri ile Çalışma

1. Mentorünüzün PR'da yaptığı yorumları okuyun
2. Gerekli değişiklikleri yapın
3. Yeni commit'ler ekleyin
4. Push yapın - PR otomatik olarak güncellenir
5. Yorumları "Resolve conversation" ile çözüldü olarak işaretleyin

PR Merge Edildikten Sonra

1. Develop branch'ini güncelleyin:

```
git checkout develop
git pull origin develop
```

2. Eski feature branch'ini silebilirsiniz (isteğe bağlı):

```
git branch -d feature/ui-login
```

3. Yeni göreve başlayın

5. Yararlı İpuçları

Commit Mesajları

- İyi bir commit mesajı: "Add login form validation"
- Kötü bir commit mesajı: "changes", "fix", "update"

Commit Sıklığı

- Mantıksal birimler halinde commit edin
- Çok büyük veya çok küçük commit'lerden kaçının

Git Geçmişini Temiz Tutma

- PR açmadan önce değişikliklerinizi gözden geçirin
- Düzenli commit mesajları kullanın
- Gereksiz dosyaları (.DS_Store, .idea/, vb.) .gitignore'a ekleyin

Güvenlik

- API anahtarları, şifreler veya gizli bilgileri asla commit etmeyin
- Şüphelendiğiniz durumda mentorünüze danışın