ChatGPT

DnD Mobil Uygulaması – MVP Özellikleri ve Mimari

1. MVP Özellik Listesi

- **Karakter Oluşturma ve Yönetimi:** Kullanıcılar kendi DnD karakterlerini oluşturup düzenleyebilmelidir. Karakter adı, sınıfı, ırkı, yetenek puanları ve diğer istatistikleri girilebilecek bir form sunulur [1].
- **Karakter Kaydetme ve Senkronizasyon:** Oluşturulan karakter verisi, örneğin Firebase veya yerel veritabanı (Hive/SQFLite) üzerinden saklanır ve gerektiğinde geri yüklenir. Veri bulutta saklanarak cihazlar arası senkronizasyon sağlanabilir [1].
- **D&D Referans Verileri:** Uygulama, D&D 5. Baskı mekanikleri için temel içerikler sunar. Örneğin sınıflar, ırklar, büyüler, canavarlar, ekipman vb. listelenir (D&D API veya Open5e API kullanılarak) [2] [3].
- **Arama ve Filtreleme:** Kullanıcılar büyü, canavar veya sınıf arayabilir. Örneğin Open5e API'nin `?search=` parametresi ile metin arama yapılabilir [4].
- **Basit Kullanıcı Arayüzü:** Temel bir ana menü, karakter listesi sayfası, karakter oluşturma formu ve referans veri ekranları bulunur. Navigasyon ve form doğrulama da dahil edilir.

2. Uygulama Mimarisi (Flutter + API)

- **Katmanlı Mimari (MVVM/Repository):** Uygulama MVVM veya katmanlı yapıyla düzenlenebilir [5]. UI (View) katmanı widget'lar içerir; iş mantığı ViewModel/Provider ile yönetilir. Veri katmanında ise Repository/Service sınıfları API çağrıları yapar [5] [6]. Örneğin `MonsterRepository` sınıfı içinden DnD API'ya istek atılır ve JSON parse edilir.
- **State Management:** `Provider`, `Riverpod` veya `BLoC` gibi bir yöntemle ViewModel'ler (veya ChangeNotifier sınıfları) kullanılabilir. Ekranlar (Views) bu ViewModel'lerdeki verilere dinler konumda abone olur.
- **Ağ İletişimi:** Flutter'da HTTP istekleri için `http` veya `dio` paketi kullanılır. Bu servis sınıfları `ApiUtil` benzeri yardımcılarla yapılandırılabilir [6]. Örneğin:

```
Future<Map<String, dynamic>> fetchMonsterDetail(String index) async {
  final response = await http.get(Uri.parse('https://www.dnd5eapi.co/api/
monsters/$index'));
  if (response.statusCode == 200) {
    return json.decode(response.body);
  } else {
    throw Exception('Veri alınamadı');
  }
}
```

Bu şekilde `/monsters/{id}` endpoint'ine GET isteği atılır ve sonuç JSON'a dönüştürülür [7]. Hata yönetimi için yanıt kodları kontrol edilir.

- **Yerel Saklama:** MVP'de basitçe karakter verileri ve temel önbellek Hive/SQFLite ile tutulabilir. Repository katmanı gerekli gördüğü zaman hem API'dan hem de yerelden veri okuyup birleştirebilir.

3. Hangi Veriler Hangi API'den Çekilebilir?

MVP'de D&D içerikleri için genel SRD tabanlı açık API'ler kullanılır:

- **Sınıflar (Classes):** D&D 5e API üzerindeki `GET /api/classes` uç noktası [3].
- **Büyüler (Spells):** D&D 5e API ile `GET /api/spells` [8] (Ayrıca Open5e da kullanılabilir).
- **Irklar (Races):** Resmi D&D API'da sınırlıdır; Open5e üzerinden `GET /api/races` kullanılabilir.
- **Arka Planlar (Backgrounds):** Open5e API (`GET /api/backgrounds`) üzerinden çekilebilir.
- **Ekipman (Equipment):** D&D 5e API'da `GET /api/equipment` uç noktası [9].
- **Canavarlar (Monsters):** D&D 5e API'daki `GET /api/monsters` uç noktası [10].
- **Yetenekler (Skills), Özellikler (Feats) vb.:** Open5e API'den (ör. `GET /api/skills`, `GET /api/feats`).
- **Karakter Verileri:** Uygulamaya özel veritabanında (Firebase veya cihazdaki yerel DB) saklanır; üçüncü taraf API'sı yok.

4. API Çağrıları ve Örnek GET İstekleri

Flutter'da API çağrısı genellikle `http.get(Uri.parse(url))` şeklinde yapılır. Örnekler:

- **Tüm Sınıfları Almak:**

```
final response = await http.get(Uri.parse('https://www.dnd5eapi.co/api/classes'));
if (response.statusCode == 200) {
  var data = json.decode(response.body);
  var classes = data['results']; // sınıf listesini içerir
}
```

Bu istek `GET /classes` ile tüm sınıf listesini döndürür [3].

- **Filtreli Arama (Spells):** Örneğin, içinde "fireball" geçen büyüleri aramak için Open5e kullanımı:

```
final response = await http.get(Uri.parse('https://api.open5e.com/spells/?search=fireball'));
if (response.statusCode == 200) {
  var results = json.decode(response.body)['results'];
}
```

Bu şekilde `?search=` parametresi ile arama yapılır [4].

- **Parametre ile GET (Monsters):** Open5e API'de zorluk derecesine göre filtreleme:

```
final response = await http.get(Uri.parse('https://api.open5e.com/
monsters/?cr=3'));
```

Örneğin, `cr=3` parametresi Challenge Rating 3 olan canavarları getirir [11].

- **Tek Bir Kaydı Getirmek:** Belirli bir öğe için örnek:

```
final response = await http.get(Uri.parse('https://www.dnd5eapi.co/api/
monsters/young-red-dragon'));
```

Burada URL sonuna canavarın `index` değeri eklenerek detay alınır [7].

Her istekte `statusCode` kontrol edilmeli ve hata durumunda kullanıcıya hata mesajı gösterilmelidir [7].

5. Çevrimdışı (Offline) Destek Önerileri

- **Yerel Önbellekleme:** Uygulama, internet bağlantısı yokken de temel verileri göstermelidir. Bu amaçla sunucudan çekilen veriler Hive/SQFLite gibi bir yerel veritabanında saklanabilir. Daha sonra isteklerde sorun yaşanırsa veriler buradan okunur [12]. Bu yöntem, yerel veritabanını "tek doğru kaynak" haline getirir ve kullanıcıya kesintisiz deneyim sunar [12].
- **Senkronizasyon:** Kullanıcı offline iken oluşturduğu veya güncellediği karakterleri localde "pending" olarak işaretleyip saklamak, bağlantı sağlandığında sunucuya göndermek iyi bir yaklaşımdır [12] [13]. Örneğin Firebase Realtime Database kullanılırsa, Firebase otomatik olarak offline önbellekleme yapar ve bağlantı geri geldiğinde verileri senkronize eder [13] [14].
- **Bağlantı Durumu Kontrolü:** `connectivity_plus` paketi gibi araçlarla ağ durumu izlenebilir. İnternet kesildiğinde kullanıcı bilgilendirilebilir veya arka planda veri senkronizasyonu duraklatılabilir.

6. Veri Modelleme (Flutter'da Sınıflar/Modeller)

Her API kaynağı için Dart model sınıfları oluşturulur. Bu sınıflar JSON'u alıp tip güvenli Dart nesnelerine dönüştürür (genellikle `fromJson` ve `toJson` metotlarıyla). Örneğin bir sınıf modeli:

```
class CharacterClass {
  final String index;
  final String name;
  CharacterClass({required this.index, required this.name});

  factory CharacterClass.fromJson(Map<String, dynamic> json) {
    return CharacterClass(
      index: json['index'] as String,
      name: json['name'] as String,
    );
```

```
  }
}
```

`fromJson` yapıcısı JSON haritasından alanları atar [15]. Benzer şekilde `Spell`, `Monster`, `Race` gibi modeller de oluşturulur. Büyük projelerde `json_serializable` veya `freezed` gibi paketlerle otomatik model üreteçleri kullanılabilir, ancak temel olarak `fromJson` yapıcıları önerilir [15]. Modeller, API'dan gelen iç içe JSON nesnelerini de kendi modeline dönüştürmelidir.

7. Kullanıcı Deneyimi: Karakter Oluşturma ve Kaydetme Akışı

Karakter oluşturma süreci adım adım basitçe tasarlanmalıdır. Örneğin: 1. **Temel Bilgiler:** Kullanıcı karakter adı, ırk ve sınıf seçimini yapar [1]. (Sınıf/ırk listeleri API'den çekilip sunulur.)
2. **Yetenek Puanları:** Karakterin Strength, Dexterity vb. yetenek puanları belirlenir (zar atma veya point-buy yöntemi).
3. **Diğer Özellikler:** Arka plan (background), beceriler (skills), ekipman, diller vb. seçilir.
4. **Büyüler/Yetenekler:** Sınıfa bağlı olarak başlangıç büyüsü veya özel yetenekler atanır.
5. **Özet ve Onay:** Tüm bilgiler gözden geçirilir, kullanıcı formu onaylar.
6. **Kaydetme:** Karakter verisi kaydedilir ve karakter listesine dönülür. Karakter verisi Firebase veya local DB'ye yazılır [1]. Başarıyla kaydedilince kullanıcıya geri bildirim verilir.

Bu akışta her adım bir ekranda toplanabilir (stepper, sayfalı form vb.). Veriler her adımdan sonra geçici olarak tutulup nihai onayda toplu kaydedilebilir.

8. Geliştirme Sıralaması Önerisi

Bir MVP geliştirme sürecinde adımlar mantıklı bir sırayla ilerlemelidir. Önerilen geliştirme aşamaları:

1. **Proje Kurulumu:** Flutter projesi oluşturulması, gerekli paketlerin (`http`, `provider/riverpod`, `hive/sqflite` vb.) eklenmesi.
2. **Veri Katmanı:** API istemci sınıflarının yazılması ve model sınıflarının oluşturulması [15]. (Örn. `DndApiService` içinde endpoint'ler tanımlanır.)
3. **Temel UI:** Ana ekran, menü ve temel navigasyon altyapısı (Drawer/Tab/Navigator) oluşturulur.
4. **Referans Veri Görüntüleme:** Sınıf/ırk/büyü listelerini gösteren basit listeler geliştirilir. Bu listeler API'dan veri çekerek (`FutureBuilder` veya benzeri) doldurulur [16].
5. **Karakter Oluşturma Formu:** Adım adım karakter oluşturma ekranı geliştirilir (Form widget'ları, validasyon). Kullanıcı girdiği veriler ViewModel'da toplanır.
6. **Kaydetme ve Senkronizasyon:** Karakter oluşturma tamamlandığında veri kaydetme işlevi eklenir. Örneğin Firebase entegrasyonu yapılır veya Hive/SQFLite ile yerel saklama ayarlanır [1].
7. **Arama/Filtreleme Özellikleri:** Uygulamada büyü ve canavar arama, sonuçları sıralama ve filtreleme eklenir. Open5e API'nin `?search` veya `?limit` parametreleri kullanılır [11] [4].
8. **Çevrimdışı Destek:** İlk offline önbellekleme stratejisi uygulanır (veriler lokalde saklanır, bağlantı yoksa buradan yüklenir) [12].
9. **Test ve İyileştirme:** Tüm fonksiyonlar manuel olarak test edilir, hata düzeltilir. UI/UX düzenlemeleri ve performans optimizasyonu yapılır.

Her aşamada yapılacak işleri sprint'lere veya görev kartlarına bölmek, MVP sürecini yönetilebilir kılar. İlk aşamalar basit istekler ve listelerden oluşurken, son aşamalarda ileri özellikler (arama, offline, tasarım geliştirme) eklenir. Bu sırayla ilerlemek, projeyi erken test etmeyi ve en önemli özellikleri önce tamamlamayı sağlar.

Kaynaklar: DnD 5e API dokümanları ve Flutter geliştirme rehberleri kullanılarak derlenmiştir [2] [17] [12].

---

[1] Building a DnD Character Info App with Flutterflow: My Development Journey First Week | by Ömer Er | Medium
https://medium.com/@omerfaruker79/building-a-dnd-character-app-with-flutterflow-my-development-journey-cddc05adc273

[2] [3] [8] [9] [10] Dungeons and Dragons API - PublicAPI
https://publicapi.dev/dungeons-and-dragons-api

[4] [11] Api Docs - Open5e
https://open5e.com/api-docs

[5] Architecture guide | Flutter
https://docs.flutter.dev/app-architecture/guide

[6] Robust API structure for Flutter. Hello, Flutter birds! When your app... | by Sumeet Rukeja | Flutter Community | Medium
https://medium.com/flutter-community/robust-api-structure-for-flutter-7251c0b3180f

[7] [16] [17] D&D API in Flutter: HTTP & JSON Guide | Medium
https://maxim-gorin.medium.com/building-a-d-d-api-explorer-a-flutter-guide-to-http-and-json-735466af713d

[12] Build Flutter Apps to Work Offline: Best Practices & Insights | by Nayan Babariya | Medium
https://medium.com/@nayanbabariya/build-flutter-apps-to-work-offline-725b1e7653da

[13] [14] ⚡ Flutter ile Firebase Realtime Database: Çevrimdışı (Offline) Özellikler | by Emin alan | Jun, 2025 | Medium
https://eminalan.medium.com/flutter-ile-firebase-realtime-database-%C3%A7evrimd%C4%B1%C5%9F%C4%B1-offline-%C3%B6zellikler-26c835577223

[15] JSON | Flutter
https://docs.flutter.dev/data-and-backend/serialization/json