rast mobile sunum

1. Giriş

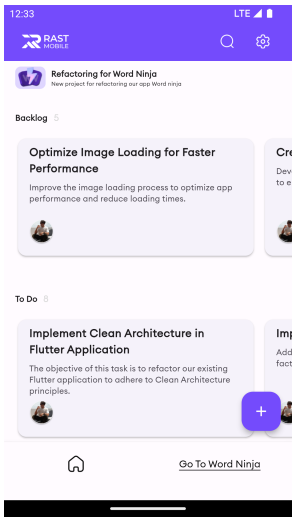
1.1 Proje Tanımı:

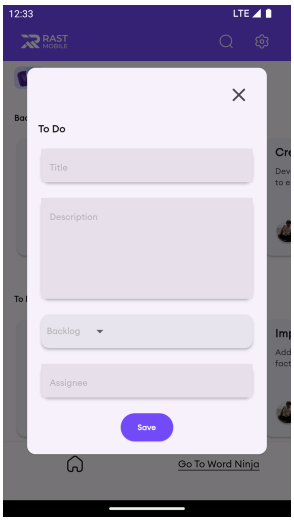
Bir kanban board hazırlanılması istendi. İçinde çeşitli etiketlerde listeler olan ve bu listelere kart şeklinde eleman eklenebilen bir yapı kuruldu.

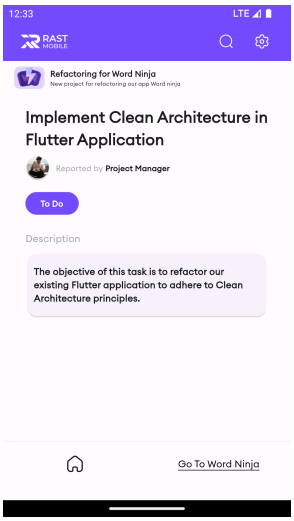
2. Uygulama Geliştirilirken İzlenilen Adımlar

2.1 Frontend:

Uygulamanın öncelikle home, pop_up, detail gibi ön yüz sayfaları hazırlanıldı. Api ve veritabanı bağlantılarını simüle edebilmek için default değerler atanılarak önyüz sayfaları test edildi. Önyüzde kullanılacak renkler, splash time ve sık kullanılan widgetlar ayrı klasörlere ayrılarak kod tekrarı minimuma indirildi.







2.2 Model:

arkaplan işlemleri yapılmadan önce kullanılacak veri türüne özgü bir model sınıfı oluşturuldu. Böylelikle veritabanı ve apiden gelen verileri o model sınıfı ile karşılayabileceğiz.

```dart

class Task {
  String title;
  String description;
  String status;
  String assignee;

  Task(
      {required this.title,
      required this.description,
      required this.status,
      required this.assignee});

  factory Task.fromJson(Map<String, dynamic> json) {
    return Task(
      title: json['title'] ?? '',
      description: json['description'] ?? '',
      status: json['status'] ?? '',
      assignee: json['assignee'] ?? '',
    );
  }
}

```

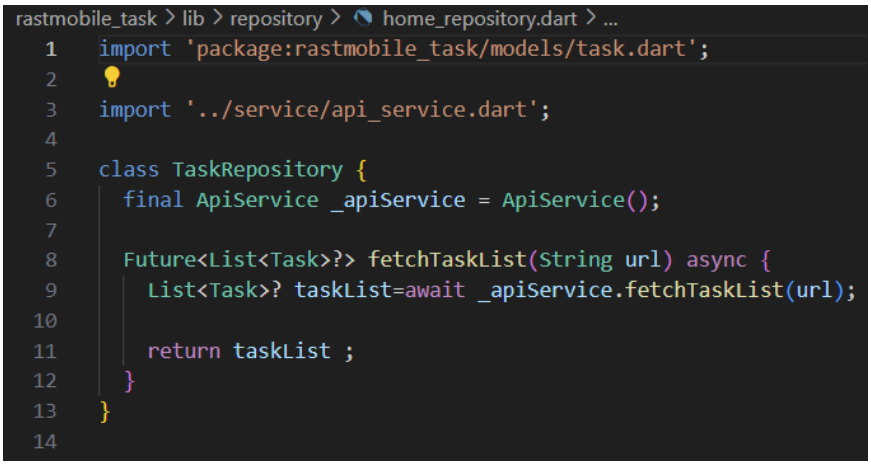
2.3 Api Service:

Bir rest api servisi simüle edebilmek adına github reposuna yüklenilen bir json dosyası üzerinden http sorguları yapıldı.

```json
"tasks": [
    {
        "title": "Implement Clean Architecture in Flutter Application",
        "description": "The objective of this task is to refactor our existing Flutter application to adhere to Clean Architecture principles.",
        "status": "To Do",
        "assignee": "semih"
    },
    {
        "title": "Update User Profile Page",
        "description": "Enhance the user profile page with additional information and customization options.",
        "status": "In Progress",
        "assignee": "melis"
    },
```

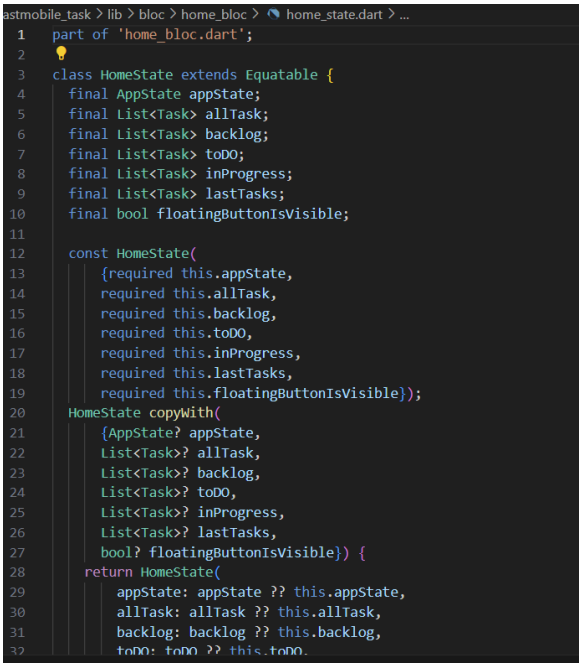
flutter http paketi kullanılarak repodaki json dosyasından veriler çekilmeye çalışılarak olası hata durumları ele alındı.

daha sonra bu service sınıfının state management yöntemiyle önyüze aktarılmasını ve service sınıfının kullanımını kolaylaştırmak için repository oluşturuldu.



2.4 State Management:

state management yöntemi olarak bloc kullanıldı. Bloc, verilerimizi tutması için hazırlanan modellerimizi önyüze aktarmak ve değişimleri canlı olarak göstermek için kullanılmıştır.



Bloc’un state dosyası değişkenleri tutmak için kullanılır. Ui da gösterilecek ve bağzı durumları ele almak için kullanılan değişkenler bu dosyada yer alır.

```dart
part of 'home_bloc.dart';

abstract class HomeEvent extends Equatable {}

class GetAllTasks extends HomeEvent {
  @override
  List<Object?> get props => [];
}

class SetTaskLists extends HomeEvent {
  @override
  List<Object?> get props => [];
}

class ChangeButtonVisibility extends HomeEvent {
  final bool visibility;
  ChangeButtonVisibility({required this.visibility});

  @override
  List<Object?> get props => [visibility];
}

class AddTask extends HomeEvent {
  final Task task;
  AddTask({required this.task});

  @override
  List<Object?> get props => [task];
}
```

Bloc'un event dosyası ele alınacak olayların taslağının hazırlandığı dosyadır. Örneğin task listesinin apiden çekilip statedeki listeye eklenmesi olayı bu dosyada belirtilir ve durum ele alınırken gerekli değişkenleri burda tanımlarız.

```dart
part 'home_event.dart';
part 'home_state.dart';

class HomeBloc extends Bloc<HomeEvent, HomeState> {
  HomeBloc()
      : super(const HomeState(
            appState: AppState.loading,
            allTask: [],
            backlog: [],
            toDO: [],
            inProgress: [],
            lastTasks: [],
            floatingButtonIsVisible: true)) {
    on<GetAllTasks>(_getAllTask);
    on<SetTaskLists>(_setTaskLists);
    on<ChangeButtonVisibility>(_changeButtonVisibility);
    on<AddTask>(_addTask);
    on<AddLastTask>(_addLastTask);
  }
```

Bloc sayfası eklenilen eventlerin içeriğinin yazıldığı yerdir. Eventte taslak hazırlanırken yapılan iş burada tanımlanır ve stateleri burdaki tanımların içinde emit yöntemi ile değiştiririz. Bu şekilde ui tarafında izlenilen stateler otomatik olarak güncellenir.

2.5 Local Database

Database tarafında hive veritabanı kullanılmıştır. Api tarafından çekilen veriler hive veritabanına kaydedilip daha sonrasında ise eklenilen veriler ve bloc tarafından okunan veriler tamamen veritabanına kaymıştır (Apiden veri çekilmesi yalnızca uygulamanın ilk açılışında yapılmıştır).

Hive veritabanına model kaydetmek için öncelikle modelimiz için bir adaptor oluşturduk.

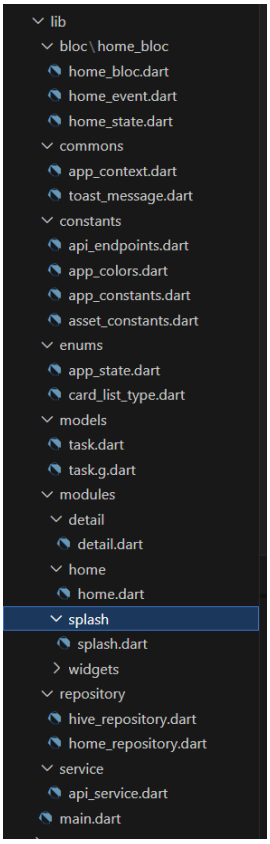
```
@HiveType(typeId: 1)
class Task {
  @HiveField(0)
  String title;

  @HiveField(1)
  String description;

  @HiveField(2)
  String status;

  @HiveField(3)
  String assignee;

  Task(
      {required this.title,
      required this.description,
      required this.status,
      required this.assignee});

  factory Task.fromJson(Map<String, dynamic> json) {
    return Task(
      title: json['title'] ?? '',
      description: json['description'] ?? '',
      status: json['status'] ?? '',
      assignee: json['assignee'] ?? '',
    );
  }
```

Belirtilen Hivetype ve Hivefieldlar kullanılarak otomatik olarak adaptor sınıfı generate edildi. Daha sonra veri tabanı işlemleri için bir repository oluşturuldu.

```dart
class HiveRepository {
  Future<bool> isBoxExist() async {
    bool isExist = await Hive.boxExists("tasklist");
    return isExist;
  }

  bool isBoxOpen() {
    bool isOpen = Hive.isBoxOpen("tasklist");
    return isOpen;
  }

  Future<void> openTaskBox() async {
    await Hive.openBox<Task>("tasklist");
  }

  Future<void> initHive() async {
    await Hive.initFlutter();
    Hive.registerAdapter(TaskAdapter());
  }

  Future<void> addTask(Task task) async {
    var box = Hive.box<Task>("tasklist");
    await box.add(task);
  }

  List<Task> getAllTasks() {
    var box = Hive.box<Task>("tasklist");
    return box.values.toList();
  }
```

3 Dosyalama

Proje geliştirilirken mümkün olduğunca kodların parçalanması ve ayrı dosyalarda ele alınmasına uğraşılmıştır. Böylelikle uygulamada yapılacak değişiklikler daha kolay bir şekilde yapılabilmektedir.


lib
bloc\home_bloc
home_bloc.dart
home_event.dart
home_state.dart
commons
app_context.dart
toast_message.dart
constants
api_endpoints.dart
app_colors.dart
app_constants.dart
asset_constants.dart
enums
app_state.dart
card_list_type.dart
models
task.dart
task.g.dart
modules
detail
detail.dart
home
home.dart
splash
splash.dart
widgets
repository
hive_repository.dart
home_repository.dart
service
api_service.dart
main.dart