EGE ÜNİVERSİTESİ

MÜHENDİSLİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ DEPARTMANI

MOBİL PROGRAMLAMA

2022–2023 GÜZ SEMESTER

PROJE-1 FİNAL RAPOR

SUNUM TARİHİ

24/01/2023

HAZIRLAYANLAR

05190000073, Sinem Akyüz

05200000070, Nilay Taşel

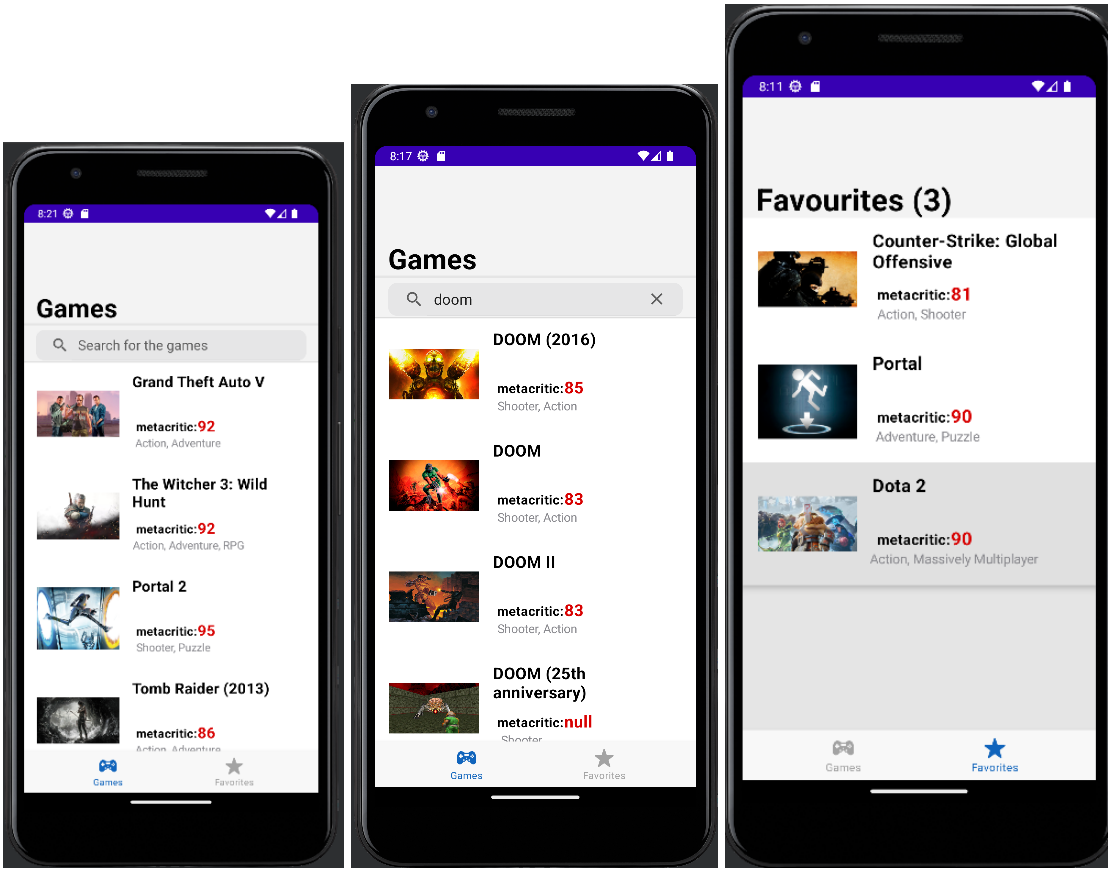
05200000045, Şevval Gönül

İçindekiler

1) Oyun Bilgileri Mobil Uygulama

Windows, Android Studio, Kotlin

1.a Kullanıcı Arayüzleri

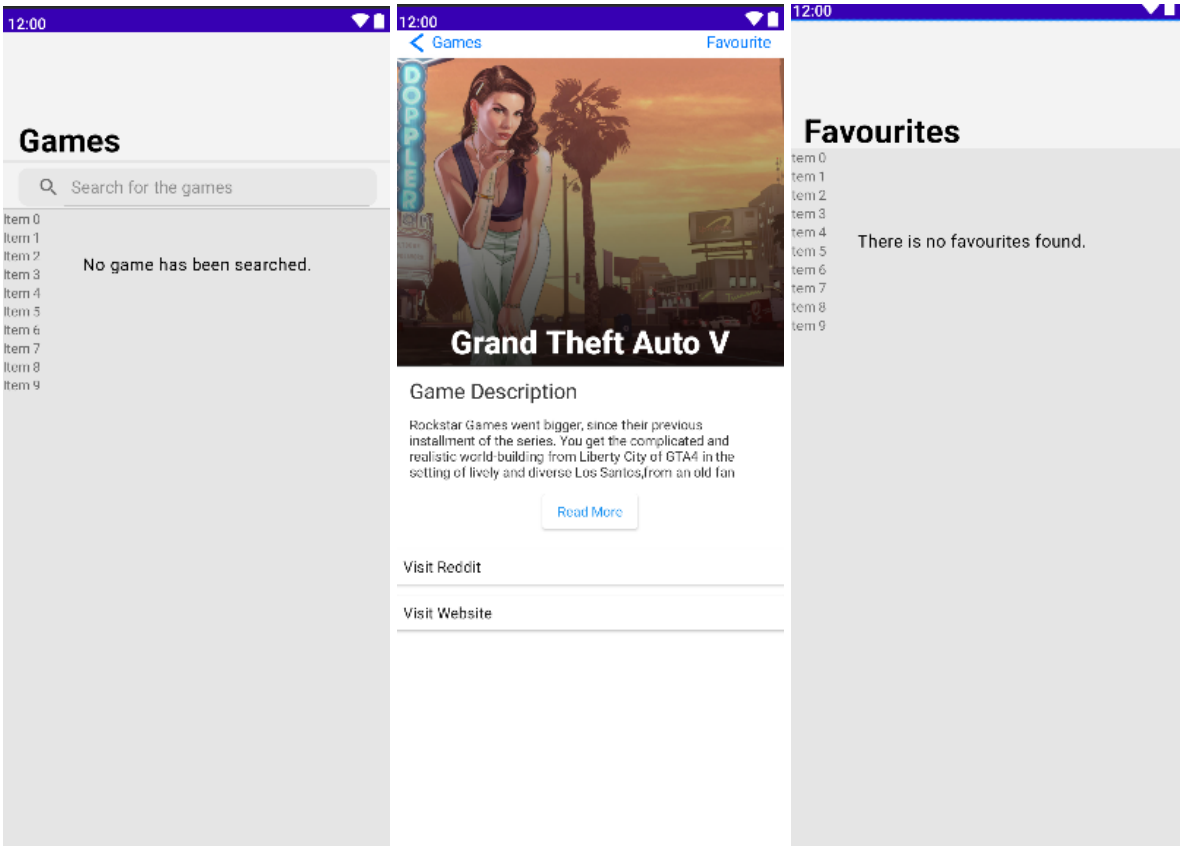


1.a.1 Başlangıç

İlk olarak favoriler, oyunlar, detaylar sayfalarının her birinin etkinliği arttırtmak adına fragment olması gerektiğine karar verdik. Geçişler için Navigation Component kullanmaya karar verdik. UI tasarımında Bottom Navigation View kullanıldığı için bu görünümü ekleyip bunu Navigation Component ile bağladık. Destinationlarımızı oluşturup her bir fragment için xml dosyalarımızı tasarlamaya başladık.

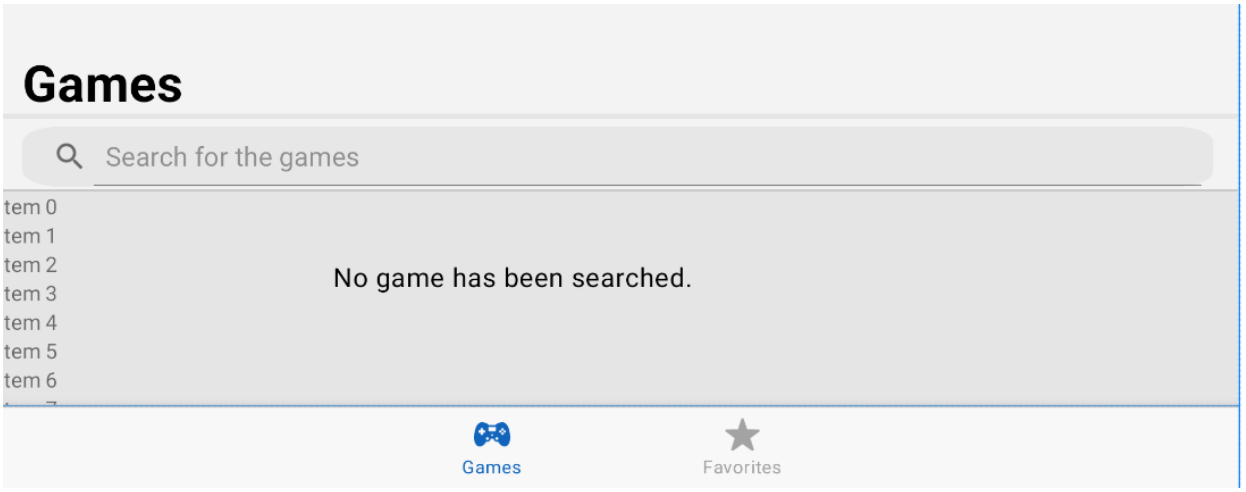
1.a.2 UI

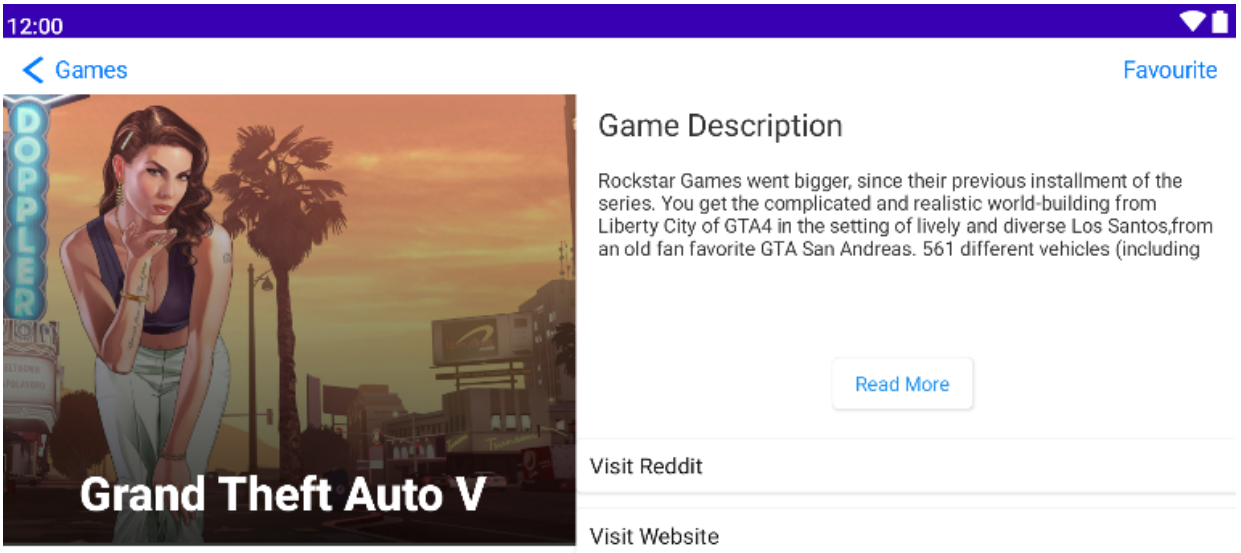
Bize verilen figma designı kullanarak fragmentlarımız ve template oyun itemimiz/cardımız için xml dosyaları oluşturduk. Her xml dosyamıza yerleştirilecek textview, imageview, buttons, view, searchview, recyclerview, scrollview elementleri belirledik. Xml dosyamızda belirlediğimiz elementleri oluşturduk. Constaintlerini ekledik ve UI üzerinde konumlandırdık. Details Fragmenta Image için gradienti üst üste gösterebilmek için bir view daha ekledik ve resimle yazının figmada istenen halde gözükmesini sağladık. Main xml'imizde de games fragmanı ve favoriler fragmanı arası geçişler için bottom navigation view'ın navigation controller'ına oluşturduğumuz navigation componentın controllerını ekledik.



1.a.3 Landscape

Kullandığımız kullanıcı arayüzlerinin yatay (lanscape) layout olarak da çalışması için landscape orientation ekledik. Bu lanscape orientionda main activitemiz, detail fragmentımız ve template game itemın Xml dosyalarında değişiklikler yaparak landscape modda da düzgün UI göstermesini sağladık.

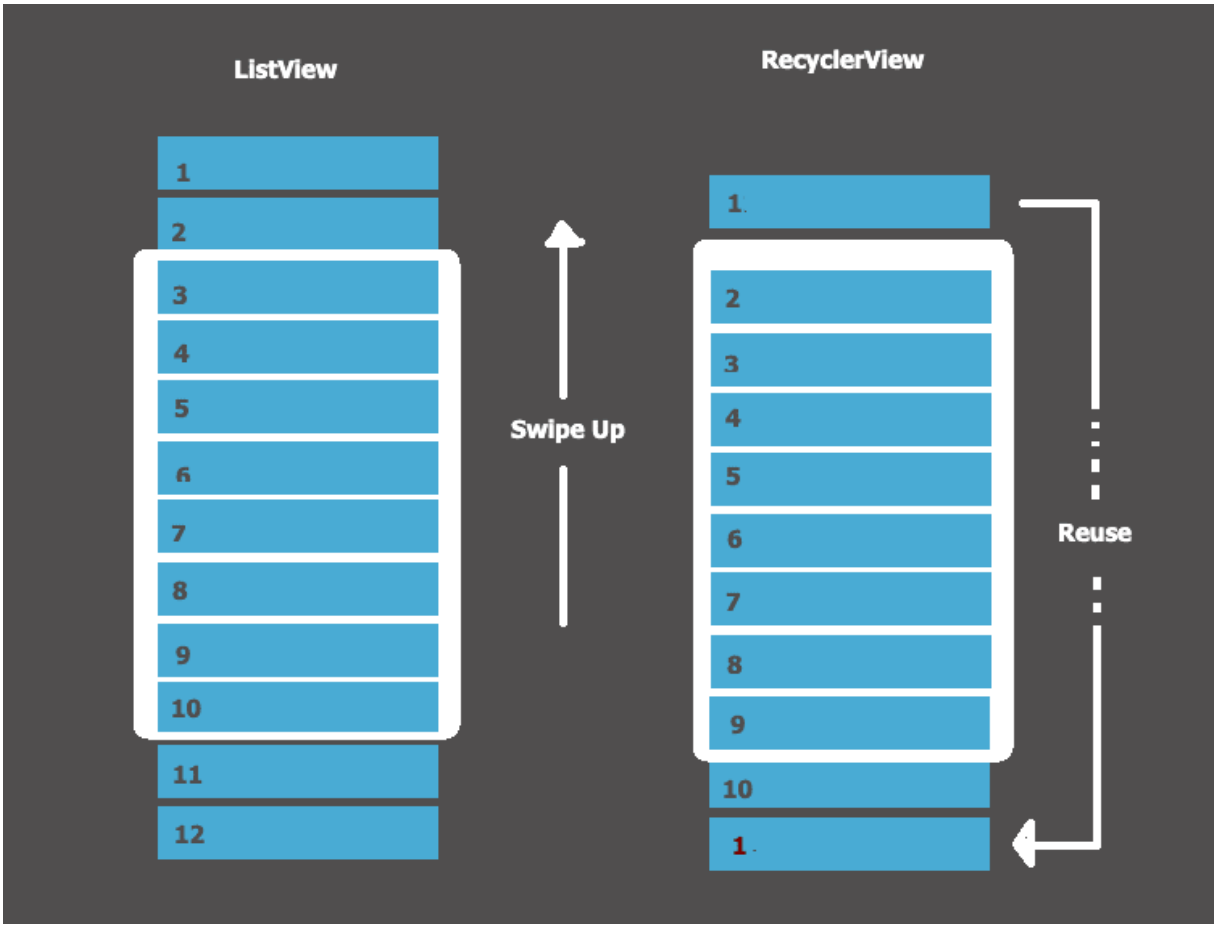




1.b Fragments

1.b.1 Recycler View

Büyük veri setlerini etkin bir biçimde gösterebilmesi, daha dinamik ve memory kullanımını aza indirmesi sebebiyle tüm oyunları listelemek için listview yerine recycler view kullanmaya karar verdik. Recyclerview'ımız template game card'ı geri dönüştürerek kullandı ve bu sayede performans ve responsiveness artışı sağlarken güç tüketimini azalttı.



1.c Program Dizayn

1.c.1 Fragmentlar Arası Geçişler ve Haberleşme

Görünümlerle daha kolay etkileşim kurabilmek için ViewBinding özelliğini ekledik ve çoğu yerde findViewById yapmak yerine kullandık.

BottomNavigationView'daki menu itemlarımıza Navigation Component'taki destinationlarımızla aynı id'yi vererek bağlama işlemi yaptık. MainActivity'de BottomNavigationView ve navigation controller'ımızı initialize edip bunları mergeledik.

1.c.2 Background Change in Item

Her bir game tıklandığında recycler viewdaki itemin backgroundu değiştirildi ve recyclerview ilerledikçe aynı kutucuğun başka oyun için backgroundunun değişmemesi sağlandı. Oyunlar arasında dolaşırken tıklanmış oyunların backgroundunun her seferinde değişmesi arraylistte onClick ile detailFragmente gidenler eklenerek sağlandı.

1.c.3 Bilgilerin Api Aracılığıyla Çekilmesi

Öncelikle response model oluşturduk burada apiden dönen cevabın modeli belirtildi.

```
data class GamesResponse(  // Model classı
    val count: Int,
    val next: String,
    val previous: String?,
    val results: List<ResultGame>
)
```

```
data class ResultGame (
    //@Json(name = "id")
    val id: Int,
    val name: String,
    val background_image: String,
    val metacritic: Int?,
    val genres : List<GenreNames>,
)
```

```
data class DetailResponse (
    val id: Int,
    val name: String,
    val description: String,
    val background_image : String,
    val website : String,
    val reddit_url: String
    )
```

API Service yaratmak için bir interface oluşturuldu(GameApiService) ve kullanacağımız metodlar bu interface'in içinde tanımlandı. Retrofit kullanarak games için getGames methodu, details sayfası için id yi parametre olarak kullanarak getGame methodu ve son olarak arama sonucu yine aranılan objeyi parametre olarak vererek search methodu kullanılarak veriler apiden alındı.

```
private const val API_KEY = "3be8af6ebf124ffe81d90f514e59856c"
private val BASE_URL = "https://api.rawg.io/api/"
interface GamesApiService {
    @GET("games")
    fun getGames(
        @Query("key") key: String = API_KEY,
        @Query("page") page: Int = 1,  // That will later be helpfull to paginat
        @Query("page_size") pageSize: Int = 10
    ) : Call<GamesResponse>

    @GET("games/{id}")
    fun getGamesDetail(
        @Path("id") id: Int,
        @Query("key") key: String = API_KEY,
    ) : Call<DetailResponse>

    @GET("games")
    fun search(
        @Query("search") searchedText: String,
        @Query("key") key: String = API_KEY
    ) : Call<GamesResponse>

    companion object {
        private var apiService: GamesApiService? = null
```

getGames(), getGamesDetail() ve search() istek metotları uzaktaki bir servise istek atacağı için cevabın gelmesi belirli bir süre alır. Dolayısıyla bu metotları asenkron olarak çalıştırmazsak Main(UI) threadi bloklayacağından dönüş tipleri Call ile wrap edildi ve enqueue() metodu ile istekler asenkron olarak çalışarak Main threadimizi bloklamamış oldu.

```
companion object {
    private var apiService: GamesApiService? = null
    fun getInstance() : GamesApiService {
        if(apiService == null) {
            val moshi = Moshi.Builder() Moshi.Builder
                .add(KotlinJsonAdapterFactory()) Moshi.Builder!
                .build()

            val retrofit = Retrofit.Builder() Retrofit.Builder
                .addConverterFactory(MoshiConverterFactory.create(moshi)) Retrofit.Builder
                .baseUrl(BASE_URL)
                .build()

            apiService = retrofit.create(GamesApiService::class.java)
        }
        return apiService!!
    }
}
```

Tüm uygulamamızda aynı retrofit objesi üzerinden istek gönderilebilmesi için Singleton yapısı kullandık. API'den gelen JSON cevabını Kotlin'deki karmaşık objelere dönüştürebilmek için Moshi kütüphanesini kullandık ve retrofit objemize MoshiConverterFactory olarak ekledik.

1.c.4 Pagination

Infinite scroll uygulamalar için iyi ve efektif bir çözüm olmadığı için listelemeyi optimize etmek için pagination kullandık. Pagination ile içeriği farklı sayfalara bölmüş gibi olduk.

```
abstract class PaginationScrollListener(private val layoutManager: LinearLayoutManager) :
    RecyclerView.OnScrollListener() {
    override fun onScrolled(recyclerView: RecyclerView, dx: Int, dy: Int) {
        super.onScrolled(recyclerView, dx, dy)
        val visibleItemCount = layoutManager.childCount
        val totalItemCount = layoutManager.itemCount
        val firstVisibleItemPosition = layoutManager.findFirstVisibleItemPosition()
        if (!isLoading() && !isLastPage()) {
            val visibleCount = visibleItemCount + firstVisibleItemPosition
            if ((visibleCount >= totalItemCount)
                && (firstVisibleItemPosition >= 0)) {
                loadMoreItems()
            }
        }
    }

    protected abstract fun loadMoreItems()
    abstract fun isLastPage(): Boolean
    abstract fun isLoading(): Boolean
}
```

Bu scroll listenerı GamesFragment'ta recyclerView'ımıza ekleyerek tüm oyunları bir kereden almamış olduk. Scroll edildikçe yeni istekler atarak(loadNextPage metodu aracılığıyla) pagination'ı sağlamış olduk.

```
private fun addScrollListener(){

    binding.recyclerView.addOnScrollListener(object: PaginationScrollListener(linearLayoutManager) {

        override fun loadMoreItems() {
            isLoading = true
            currentPage += 1
            try {
                loadNextPage()
            } catch (ex: Exception) {
                isLoading = false
            }
        }

        override fun isLastPage(): Boolean {
            return isLastPage
        }

        override fun isLoading(): Boolean {
            return isLoading
        }
    })
}
```

1.c.5 FavoriteModel/GameModel

Uygulamamızın her classı, functionu, fragmentinde singleton bir şekilde verileri alabilmek için iki adet model classı oluşturduk. Favorite Model classında favorilenmiş oyunlar ve idlerini sakladık, gerekli fonksiyonları yazdık ve bunlara model üzerinden ulaşılmasını sağladık. Game Model classında apiden alınmış oyunlar ve idlerini sakladık, gerekli fonksiyonları yazdık ve bunlara model üzerinden ulaşılmasını sağladık. Bu sayede daha önce apiden alınmış oyunların tekrar alınmasına gerek kalmadı.

```
class GameModel {
    winused *
    companion object {
        private val clikedItemList: ArrayList<Int> = arrayListOf<Int>()
        private var searchList: List<ResultGame> = emptyList()
        private var allGameList: ArrayList<ResultGame> = arrayListOf<ResultGame>()
        winused
        fun addGameList(results: ArrayList<ResultGame>) {
            allGameList.addAll(results)
        }
        winused
        fun setGameList(results: ArrayList<ResultGame>) {
            allGameList = results
        }
        winused
        fun getGameList() : ArrayList<ResultGame> {
            return allGameList
        }
```

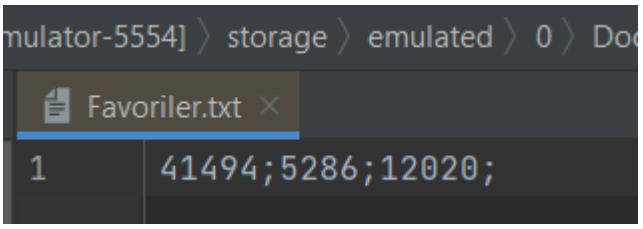
1.c.6 File Storage

Uygulamamızda File storage ile favorilenmiş oyunların idlerini saklayarak az yer kaplayan, efektif bir saklama yöntemi kullandık. Idler üzerinden oyunları bularak favoriler fragmentimizde görüntüledik.

```
class FavoriteModel {
    companion object {
        //private lateinit var gameList: List<ResultGame>
        var favList: ArrayList<ResultGame> = arrayListOf<ResultGame>()
        private var myFile : File? = null
        var favGameIdList : ArrayList<Int> = arrayListOf<Int>()

        fun getFavoritesFile() : File {
            var name = "/Documents/Favoriler.txt"
            println("File Adi = $name")

            if (myFile == null) {
                println("myFile == null")
                myFile = File( pathname: Environment.getExternalStorageDirectory().absolutePath + name)
                val myFilePath = myFile!!.absolutePath.toString()
                println("myFile.FullPath = $myFilePath")
                myFile!!.createNewFile()
            }
            return myFile!!
        }
```



2) Ekran görüntüleri/Geçişler