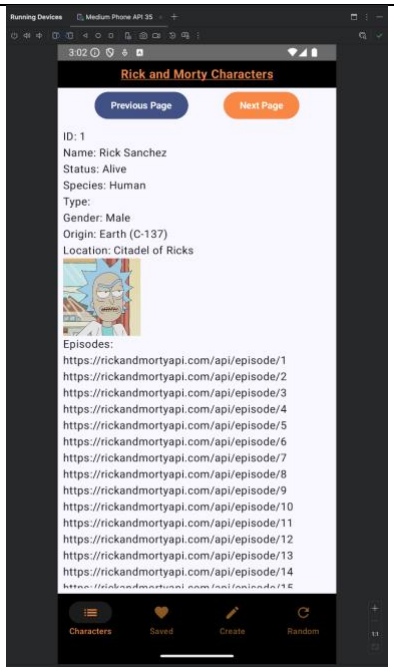
Eksamen PGR208 Rapport

1)

Funksjonalitet	**Beskrivelse**
CharacterItem	Viser detaljer om en karakterer fra API et.
UserCharacterItem	Viser detaljer om en brukeropprettet karakter hentet fra databasen.
Character	Dataklasse som representerer en karakter hentet fra API et.
CharacterList	Håndterer og strukturerer karakter data fra API et
UserCharacter	Dataklasse for opprettede karakterer som lagres i room databasen.
CharacterRepository	Utfører API kall til Rick and Morty karakterer.
CharacterService	Retrofit grensesnitt som strukturerer dataen ved Retrofit forespørsler for karakter.
RickAndMortyDatabase	Room database og som lagres til UserCharacter tabellen.
UserCharacterDao	DAO for kommunikasjon med tabellen room databasen.
UserCharacterRepository	For database relaterte brukeropprettelse av karakterer.
BottomNavigatonBar	Håndterer implementasjonen av navigasjonsbar i bunn av skjermen med faner.
ScreenNavigation	Håndterer navigasjon mellom skjermene i appen.
CharacterListViewModel	Håndterer data, API kall og paginering.
CreateCharacterViewModel	Lagrer brukeropprettede karakterer i databasen.
RandomCharacterViewModel	Håndterer API kall til å hente en tilfeldig karakter
SavedCharacterViewModel	Henter lagrede karakterer fra databasen
CharacterListScreen	Viser en liste over karakterer fra API et
CreateCharacterScreen	Lager nye karakterer via felter som lagres i databasen
RandomCharacterScreen	Henter og viser tilfeldig karakter fra API et.
SavedCharacterScreen	Viser en liste over lagrede karakterer fra databasen.

2)

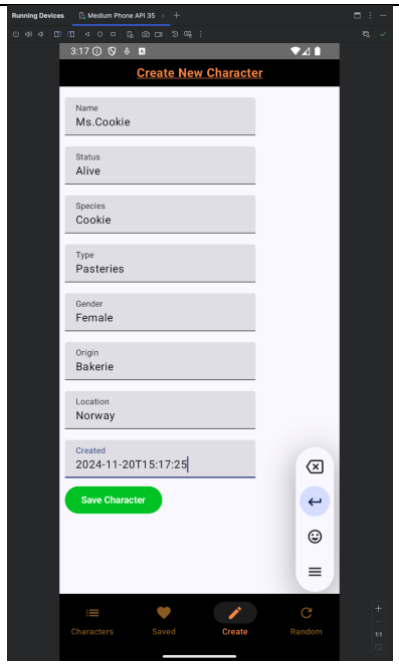
Skjerm 1) CharacterListScreen



Liste over karakterer hentet fra Rick and Morty API et

- Navigasjon mellom sider, «Previous Page» og «Next Page»
- Data fra API oppdateres dynamisk.

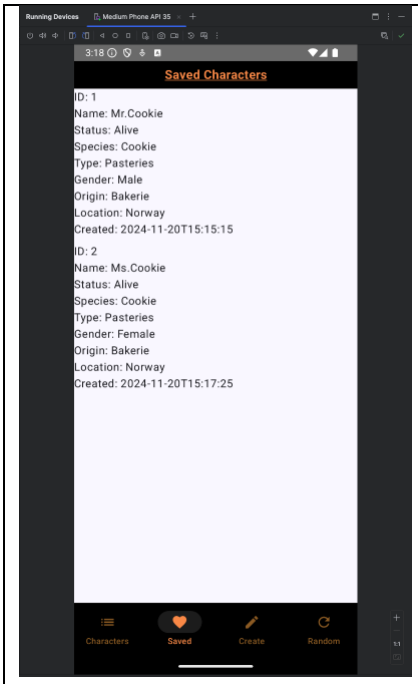
Skjerm 2) CreateCharacterScreen



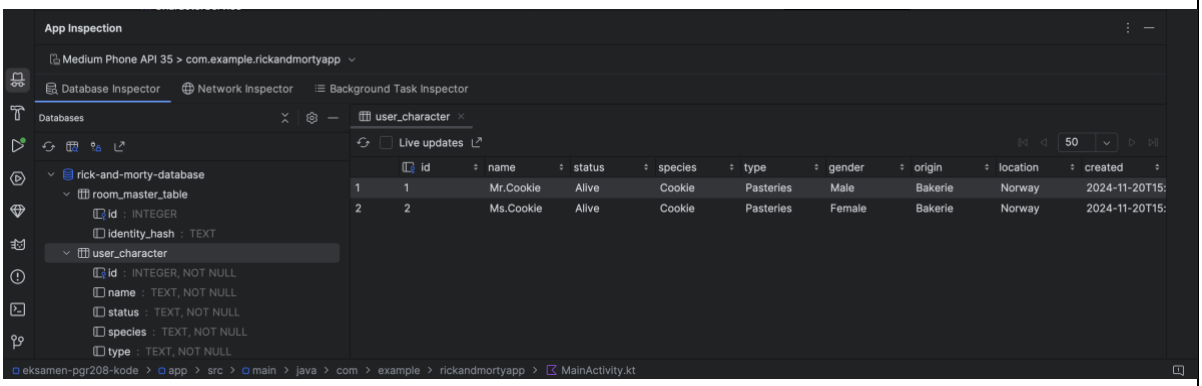
Opprettelse av ny karakter

- Lar brukeren opprette en egendefinert karakter
- Knappen validerer innholdet før karakteren lagres med isBlank.

Skjerm 3) SavedCharacterScreem

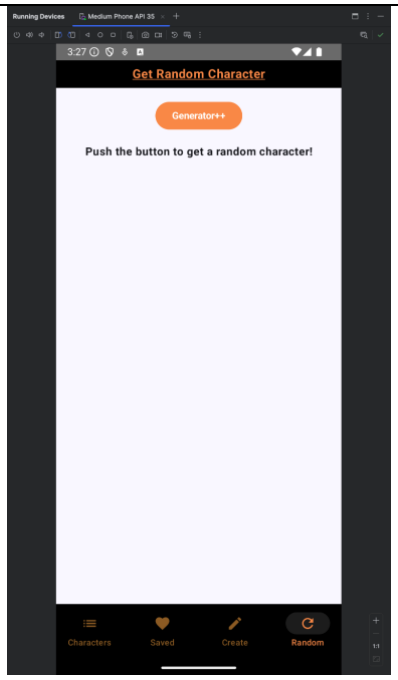


Når karakteren lagres, vises den i listen.

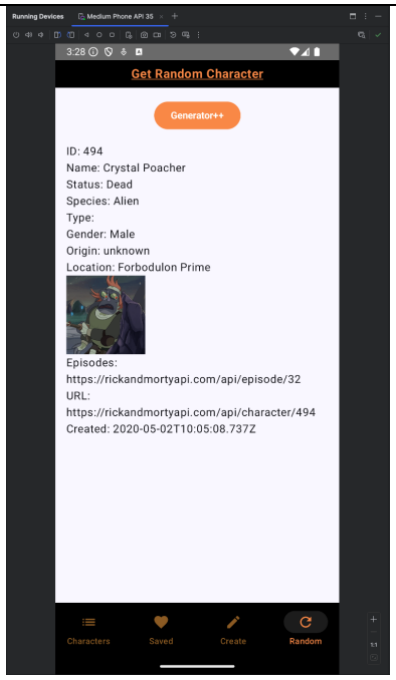


user_character tabellen.

Skjerm 4) RandomCharacterScreen



Viser melding til brukeren om å trykke på knappen



Tilfeldig karakter etter brukeren har trykket på knappen

3)

Hovedteknikker i appen:

- Button i RandomCharacterScreen til å hente en tilfeldig karakter:

```
Button(onClick = { viewModel.loadRandomCharacter() }, // <- Generer tilfeldig character
        colors = ButtonDefaults.buttonColors(
        containerColor = Color( red: 250,  green: 145,  blue: 70),
        contentColor = Color( red: 255,  green: 255,  blue: 255)),
) {
```

- TextField for brukerinput i CreateCharacterScreen:

```
// Input-felter for opprettelse av ny character
    TextField(
        value = characterName.value,
        onValueChange = { characterName.value = it },
        label = { Text( text: "Name") },
        modifier = Modifier.padding(vertical = 8.dp)
    )
```

- Text fra CharacterItem for visning av data i flere skjermer som SavedCharacterScreen:

```
Text(text = "ID: ${character.id}")
Text(text = "Name: ${character.name}")
Text(text = "Status: ${character.status}")
```

- Coil - AsyncImage for lasting og visning av bilder i CharacterItem:

```
// Bildet -> episoder av enkelte character
AsyncImage(
    model = character.image,
    contentDescription = "Image of: ${character.name}"
)
```

- Scaffold strukturerer med BottomNavigationBar og er sentral for layout:

```
Scaffold(
    modifier = Modifier.fillMaxSize(),
    bottomBar = {
        BottomNavigationBar( // <- Navigasjonbaren nederst på skjermen
            navController = navController,
            selectedTabIndex = selectedTabIndex
```

- NavHost for logikken mellom skjermene og routingen:

```
NavHost(
    navController = navController,
    startDestination = "character_list",
    Modifier.padding(innerPadding)
) {
    // Route navigering til de forskjellige skjermene
    composable( route: "character_list") {
        CharacterListScreen(characterListViewModel)
    }
    composable( route: "create_character") {
        CreateCharacterScreen(createCharacterViewModel)
    }
    composable( route: "saved_characters") {
        SavedCharacterScreen(savedCharacterViewModel)
    }
    composable( route: "random_character") {
        RandomCharacterScreen(randomCharacterViewModel)
    }
```

- LazyColumn for å vise liste med elementer, som karakterer hentet fra API:

```
    // Viser characters i en liste
LazyColumn {
    items(characters) { character ->
        UserCharacterItem(character)
        Spacer(modifier = Modifier.height(8.dp))
    }
```

- Retrofit for å hente data fra API. Paginering ved å bruke «page» parameteren.

```
// API Interface for alle characterdata fra Rick and Morty API
interface CharacterService {
    @GET("character/") // <- bestemmer siden som lastes
    suspend fun loadAllCharacters(
        @Query("page") page : Int) : Response<CharacterList>
```

- Room sikrer data er tilgjengelig selv uten nettverk. Benyttes for datalagring lokalt, og hente karakterer som bruker har opprettet i UserCharacterDao:

```
@Dao // <- DAO (Data Access Object) -> kommuniserer med databasen
interface UserCharacterDao {

        // Henter alle characters i databasen med SELECT * FROM
        @Insert
        suspend fun insertUserCharacter(character: UserCharacter): Long

        // Legger til ny character i databasen med "INSERT"
        @Query("SELECT * FROM user_character")
        suspend fun getAllUserCharacters(): List<UserCharacter>
```

4)

Under arbeidet med prosjektet har det vært viktig for meg å opprettholde en god struktur, både for oversikt og vedlikehold. Jeg har brukt MVVM-arkitektur som fundament, med applikasjonen delt inn i logiske pakker som components, dataclasses, retrofit, room, navigation, screens og viewmodels. Dette gjorde det enklere å forstå funksjonaliteten.

ViewModels:

Hver av de fire skjermene har sin egen ViewModel som gjør UI komponentene fokuserer kun på visning, mens ViewModels håndterer data og logikk. CharacterListViewModel henter data fra API et og administrerer paginering, mens CreateCharacterViewModel lagrer brukeropprettede karakterer i databasen. Dette holder koden ryddig og enkel å følge.

Kodekvalitet:

Variabler, funksjoner og klasser har tydelige navn som forklarer hva de gjør. For eksempel heter funksjonsnavnet loadCharacterFromApi() akkurat det den gjør. I tillegg har jeg unngått duplikasjoner av kode med felles logikk i repositories og ViewModels.

Testing, vedlikehold og feilhåndtering:

Mock previews ble brukt for å visualisere hvordan UI-en så ut uten å måtte kjøre emulatoren hver gang. Som gjorde det raskt å teste. Room databasen sikrer lagring av brukerens data lokalt, slik at appen fungerer uten internettforbindelse. Jeg har også brukt Log.e for å identifisere og rette feil under utviklingen.

Navigasjon:

Navigasjonen er satt opp med Compose Navigation der ScreenNavigation håndterer alle rutene mellom skjermene. BottomNavigationBar gir brukeren enkel måte å navigere mellom skjermene på.

Denne strukturen gjorde at prosjektet ble enkelt å lese, forstå og vedlikeholde.

Kilder:

https://rickandmortyapi.com/documentation