


# Eksamen PGR208 Android Programming

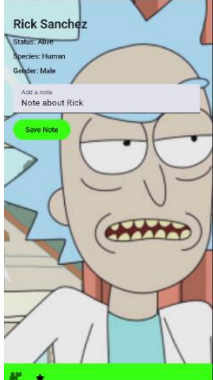
(1)

| Funksjonalitet              | Beskrivelse   |
|-----------------------------|---|
| <b>API</b>                  | Henter karakterer fra API-et til Rick and Morty og lagrer dem så i lokal database                                     |
| <b>Notat for karakterer</b> | Brukeren kan selv skrive og lagre notater til karakterer, som videre også lagres i databasen                          |
| <b>Room</b>                 | Lokal lagring av karakterdata med bruk av Room database   |
| <b>Navigasjon</b>           | Kan navigere seg mellom de forskjellige sidene med ikoner og tilbake knapper  |
| <b>Favorittkarakterer</b>   | Bruker kan ved klikk lagre karakterer fra api som favoritt karakter som videre blir lagret på en egen side for bruker |
| <b>Character creation</b>   | Bruker kan selv lage karakterer som blir lagret på en egen side slik at brukeren får god oversikt                     |

(2)

|   |   |
|---|---|
|  | <p>Hjemskjerm</p> <p>Første som vises når bruker går innpå appen. Skjermen viser en liste med karakterer hentet fra API. Bruker kan:</p> <ul style="list-style-type: none"><li>- Scrolle</li><li>- Trykke på stjerne for å velge favoritt karakter/karakterer</li><li>- Kan trykke innpå karakter kortet for å ha mulighet for å skrive notat og se flere detaljer</li></ul> <p>Skjermen har top og bottom navbar som gjør det lett å komme seg til andre sider; lage karakter, se lagde karakterer og en side som viser favoritt karakterer brukeren selv har laget.</p> |
|---|---|

|   |  |
|---|--|
|    | <p><b>Lage egen karakter</b></p> <p>Her kan bruker selv lage karakter, ved klikk «save character» blir karakteren lagret inne på «saved characters» siden med egne karakter kort og blir også lagret i databasen.</p> <p>Her er det også mulig å navigere seg til de forskjellige sidene</p> |
|   | <p><b>Lagde karakterer</b></p> <p>Her har bruker mulighet til å se deres skapte karakterer med detaljene dem har angitt</p>  |
|  | <p><b>Favoritt skjerm</b></p> <p>Egen side med favoriserte karakterer.</p> <p>Man kan også trykke seg mellom de forskjellige sidene</p>  |

|   |  |
|---|--|
|  | <h3>Karakterdetaljer og kommentarer</h3> <p>Ved at man trykker på en karakter på hjemskjerm kommer karakteren opp med flere detaljer, her er det også mulig å skrive et notat som videre lagres.</p> |
|---|--|

(3)

Jeg brukte Jetpack Compose for å lage responsive og fleksible brukergrensesnitt. Den deklaratve tilnærmingen gjør koden enklere å lese og tilpasse, og Column brukes for å strukturere elementer vertikalt med god layout. Dette gir en bedre brukeropplevelse og mer effektiv utvikling.

```
Column(
    modifier = Modifier
        .fillMaxSize()
        .padding(16.dp)
        .background(color = DeepCharcoal),
    verticalArrangement = Arrangement.spacedBy(16.dp)
) {
    if (characters.isEmpty()) {
        // Viser om ingen karakterer finnes
        Text(
            text: "No characters found",
            modifier = Modifier.align(Alignment.CenterHorizontally),
            color = Color.White
        )
    } else {
        // Grid for å vise karakterene
        LazyVerticalGrid(
            columns = GridCells.Fixed(count: 2),
            modifier = Modifier.fillMaxSize()
        ) {
            items(characters) { character ->
                Column(
                    modifier = Modifier

```

Jeg har valgt å bruke ViewModel fordi det effektivt håndterer tilstandsendringer som skjermrotasjon, uten å miste data. Dette sikrer at appen gir en sømløs brukeropplevelse og at informasjonen forblir intakt selv om skjermen roteres. Ved å bruke ViewModel kan jeg også holde UI-logikk adskilt fra datahåndtering, noe som gjør koden lettere å vedlikeholde og utvide i fremtiden.

```
fun SavedCharactersScreen(
    viewModel: SavedCharactersViewModel, // ViewModel for lagrede karakterer
    savedInstanceState: Bundle? = null // Bundle for lagrede karakterer

) {
    // Oppretter viewModels for karakterliste og lagrede karakterer
    val characterListViewModel = viewModel { CharacterListViewModel(CharacterRepository) }
    val savedCharactersViewModel = viewModel { SavedCharactersViewModel(CharacterRepository) }
```

Jeg brukte Navigation Component for å skape en oversiktlig å lett måte for bruker å kunne navigere seg mellom de forskjellige sidene i appen.

```
CharacterDetailsScreen(
    onBackButtonClick = { navController.popBackStack() }, // Tilbakeknapp
    viewModel = characterDetailsViewModel
)
}

// Skjerm for lagrede karakterer
composable(route: "saved_characters") {
    SavedCharactersScreen(navController = navController, viewModel = savedCharactersViewModel)
}

// Skjerm for favorittkarakterer
composable(route: "favorite_characters") {
    FavoriteCharactersScreen(navController = navController, characterRepository = CharacterRepository)
}
}
```

Room database effektivisert handling av data hvor karakterene blir lagret og håndtert slik at de fortsatt er tilgjengelige selvom appen lukkes.

```
// Karakterdata som lagres i databasen
@Entity(tableName = "character")
data class Character(
    @PrimaryKey val id: Int,
    val name: String,
    val status: String = "unknown",
    val species: String = "unknown",
    val gender: String = "unknown",
    val image: String = "",
    val isUserCreated: Boolean = false,

    var isFavorite: Boolean = false,
    var note: String? = null
)
```

State Management brukes sammen med compose for å oppdatere UI ved dataendringer eller brukerinteraksjoner. Som når bruker skriver inn egen data om karakterer dem skaper.

```
// Tekstfeltene for karakterinfo
var characterName by remember { mutableStateOf(value: "") }
var characterSpecies by remember { mutableStateOf(value: "") }
var characterStatus by remember { mutableStateOf(value: "") }
var characterGender by remember { mutableStateOf(value: "") }

// Layout for skjerm
Column(
    modifier = Modifier
        .fillMaxSize()
        .padding(16.dp),
    horizontalAlignment = Alignment.CenterHorizontally,
    verticalArrangement = Arrangement.spacedBy(8.dp)
) {
    // Tekstfeltet for input av bruker
    TextField(
        value = characterName,
        onValueChange = { characterName = it },
        label = { Text(text = "Character Name") }
    )
}
```

(4)

Jeg har strukturert prosjektet ved å dele det opp i mapper, funksjoner, variabler og klasser med navn som tydelig reflekterer hva de representerer og gjør. For eksempel inkluderer prosjektet klassen `SavedCharactersViewModel` for å håndtere lagrede karakterer, skjermkomponenten `CharacterCreationScreen` for å opprette nye karakterer, og mappen `data` for datarelaterte klasser.

For UI-et har jeg brukt deklorative metoder i Jetpack Compose, som `Column` og `Modifier`, for å organisere komponentene logisk. Dette sikrer en ryddig og responsiv layout, som er enkel å vedlikeholde og utvide.

Jeg har også sørget for jevn kommentering gjennom koden. Dette gjør det enklere for både meg selv og andre som jobber med prosjektet å forstå logikken, lokalisere problemer og utføre nødvendige rettelser.

Kilder:

Jeg har tatt i bruk tidligere kode fra foreleser og fra kode lært/vist i timene.

Videre har eg også tatt i bruk chat som en veileder med enkelte deler av koden, der eg har direkte tatt fra chat er kommentert klart i koden.

OpenAI. (2024). *ChatGPT (November 2024 version)*. Retrieved from <https://chat.openai.com>