Lista 1: Nawigacja w Jetpack Compose

Tematyka: Compose Navigation, NavHost, NavController, Nested Graphs, Drawer, Bottom Navigation, Back Stack.

Pytania Dodatkowe: Activity, Intent, Context.

1. Co to jest **Activity** i jaka jest jej rola w aplikacji opartej w całości o Jetpack Compose?
2. Czym jest **Context** i do czego jest potrzebny w aplikacji na Androida?
3. Do czego służy mechanizm **Intent** w systemie Android?
4. Jaka jest rola komponentu **NavHost** w nawigacji Compose?
5. Do czego służy **NavController**?
6. Co to jest **route** (ścieżka) w kontekście nawigacji?
7. Jak definiujemy ekran (cel nawigacji) wewnątrz **NavHost**?
8. Co określa parametr **startDestination** w **NavHost**?
9. Co to jest i w jakim celu stosujemy nawigacje zagnieżdżone (**nested navigation**)?
10. Co robi funkcja **navController.popBackStack()**?
11. Co to jest "***back stack***" (stos nawigacji)?
12. Jak przekazać prosty **argument** (np. ID produktu) **do innego ekranu** w Compose Navigation?
13. Czym jest **Deep Link** i jak **Intent Filter** w manifeście może być z nim powiązany?
14. Jak centralizacja ścieżek nawigacji (np. w obiekcie **object**) pomaga w utrzymaniu kodu?
15. Opisz cykl życia **Activity** (najważniejsze stany).
16. Co to jest **jawny (explicit) Intent**? Podaj przykład użycia.
17. Co to jest **niejawny (implicit) Intent**? Podaj przykład użycia.
18. Jak stan **NavController** (**rememberNavController()**) przeżywa rekompozycję?
19. Do czego służy funkcja **setContent** w **ComponentActivity**?
20. Jak otworzyć i zamknąć szufladę nawigacyjną (**Drawer**) programowo?
21. Czym jest **Application Context** oraz **Activity Context**?
22. Czy jeden **NavController** może zarządzać wieloma **NavHost**?

Lista 2: Wprowadzenie do Korutyn i Współbieżności

Tematyka: Blokowanie wątku UI, suspend, CoroutineScope, launch, async, await, Deferred, współbieżność.

Pytania Dodatkowe: Cykl życia Activity, Intent, Context.

1. Dlaczego wykonywanie **długich operacji** na **głównym wątku** UI jest problemem?
2. Co to jest **korutyna** i czym różni się od **tradycyjnego wątku**?
3. Czym jest **funkcja suspend**?
4. Jakiej reguły musimy przestrzegać przy **wywoływaniu funkcji suspend**?
5. Jaka jest fundamentalna różnica między **Thread.sleep() a delay()**?
6. Co to jest **CoroutineScope** i jaka jest jego rola?
7. Co to jest "***ustrukturyzowana współbieżność***" (Structured Concurrency)?
8. Co się dzieje z **korutynami**, gdy ich nadrzędny **CoroutineScope** jest **anulowany**?
9. Do czego służy konstruktor korutyn **launch**?
10. Czy **launch** zwraca wynik operacji? Jeśli nie, co zwraca?
11. Do czego służy konstruktor korutyn **async**?
12. Jaka jest główna różnica między **launch a async**?
13. Co to jest obiekt **Deferred**?
14. Co robi funkcja .**await()** wywołana na obiekcie **Deferred**?
15. Wyjaśnij, dlaczego **wykonanie trzech zapytań** (1.5s, 3s, 2s) **współbieżnie** trwa ~3s, a nie 6.5s.
16. Co to znaczy, że operacja jest "***asynchroniczna***"?
17. Co to znaczy, że operacje są "***współbieżne***"?
18. Co to jest **Activity** i jakie są jej główne stany cyklu życia?
19. Jak cykl życia Activity jest powiązany z **lifecycleScope**?
20. Czy **Intent** jest operacją **synchroniczną** czy **asynchroniczną**?
21. Jak można sprawdzić, czy korutyna jest aktywna?
22. Co robi **runBlocking** i dlaczego nie powinniśmy go używać w kodzie produkcyjnym na Androidzie?

23. Czy można uruchomić korutynę bez **CoroutineScope**?
24. Jak obsłużyć wyjątek (błąd) wewnątrz korutyny uruchomionej przez launch?
25. Czym różni się **współbieżność** od **równoległości**?
26. Jak **rememberCoroutineScope()** jest powiązany z **cyklem życia komponentu Composable**?

Lista 3: Reaktywna Architektura MVVM

Tematyka: MVVM, StateFlow, SharedFlow, "zimne" vs "gorące" strumienie, stateIn, sharedIn, withContext, combine.

Pytania Dodatkowe: Architektura Androida, Activity, Intent, Context.

1. Jaki problem rozwiązuje **architektura aplikacji**, np. **MVVM**?
2. Czym jest **viewModelScope** i jak jest powiązany z cyklem życia **ViewModelu**?
3. Opisz role poszczególnych komponentów w architekturze **MVVM: Model, View, ViewModel**.
4. Czym jest **Context**? Czy można go bezpiecznie przechowywać w **ViewModelu**?
5. Jaka jest główna odpowiedzialność **ViewModelu**?
6. Dlaczego **ViewModel** przeżywa **zmiany konfiguracji** (np. obrót ekranu)?
7. Jaka jest różnica między "***zimnym***" a "***gorącym***" strumieniem danych?
8. Którym typem strumienia jest **Flow**, a którym **StateFlow**?
9. Wymień kluczowe cechy **StateFlow**.
10. Do czego w architekturze **MVVM** najczęściej używamy **StateFlow**?
11. Do czego służy funkcja **.collectAsState()** (lub **collectAsStateWithLifecycle**) w **Compose**?
12. Jaki problem rozwiązuje **SharedFlow**?
13. Dlaczego nie powinniśmy używać **StateFlow** do wysyłania jednorazowych zdarzeń (np. nawigacji)?
14. Opisz wzorzec z **_privateMutableStateFlow** i **publicStateFlow**. Jaki jest jego cel?
15. Do czego służy funkcja **.asStateFlow()**?
16. Do czego służy operator **combine** na strumieniach **Flow**?
17. Do czego służy funkcja **withContext**? Podaj przykład użycia.

18. Czym się różni **Dispatchers.IO** od **Dispatchers.Default**?
19. Jaki jest cel operatora **stateIn**?
20. Co oznacza parametr **started = SharingStarted.WhileSubscribed(5000)**?
21. Jaki jest cel operatora **sharedIn**?
22. Czym różni się **withContext** od **flowOn**?
23. Co to jest **Activity** i jak komunikuje się z **ViewModel**?
24. Czym jest **Context** i dlaczego jest potrzebny do dostępu do zasobów systemowych?
25. Jak za pomocą **Intent** można przekazać dane między dwiema **Activity**?
26. Wyjaśnij, co to jest "***jednokierunkowy przepływ danych***" (**UDF**).
27. Czym się różni **collectAsState** od **collectAsStateWithLifecycle**?
28. Jaką rolę pełni **viewModelScope** w **ViewModelu**?

Lista 4: Lokalne i Zdalne Źródła Danych

Tematyka: Room, DataStore, Retrofit, Entity, DAO, Database, @Query, @GET.

Pytania Dodatkowe: Podstawy SQL, Activity, Intent, Context.

1. Czym **DataStore** różni się od **SharedPreferences**?
2. Dlaczego odczyt danych z **DataStore** zwraca **Flow**? Jaka jest tego zaleta?
3. Co to jest **relacyjna baza danych**?
4. Czym się różni **SQL** od **SQLite**?
5. Wymień i opisz krótko 4 podstawowe operacje **CRUD** w języku **SQL**.
6. Wymień 3 główne komponenty (adnotacje) biblioteki **Room**.
7. Jaka jest rola adnotacji **@Entity**?
8. Do czego służy **@PrimaryKey**?
9. Jaka jest rola interfejsu z adnotacją **@Dao**?
10. Jaka jest zaleta zwracania **Flow** z funkcji w **DAO**?
11. Do czego służy biblioteka **Retrofit**?
12. Co to jest **API REST**?

13. Co to jest **JSON**?
14. Jaka jest rola interfejsu **ApiService** w **Retrofit**?
15. Do czego służy adnotacja **@GET**?
16. Jak za pomocą adnotacji **@Query** przekazać parametr do zapytania?
17. Co to jest **DTO** (***Data Transfer Object***) i dlaczego go używamy?
18. Co to jest **Activity** i jakie ma znaczenie dla **cyklu życia bazy danych**?
19. Czym jest **Context** i dlaczego jest potrzebny do stworzenia instancji bazy **Room** lub **DataStore**?
20. Jak za pomocą **Intent** można otworzyć stronę internetową w przeglądarce?
21. Co oznacza **OnConflictStrategy.REPLACE** w adnotacji **@Insert**?
22. Co robi adnotacja **@Upsert** w Room?
23. Do czego służy konwerter (np. **GsonConverterFactory**) w Retrofit?
24. Co to jest **baseUrl** w **Retrofit**?
25. Jak wygląda typowy schemat przepływu danych w architekturze z **Room** i **Retrofit**?

Lista 5: Czysta Architektura i Wstrzykiwanie Zależności

Tematyka: Warstwa domeny, Use Case, Hilt, @Inject, @Module, @Provides, @HiltViewModel.

Pytania Dodatkowe: Activity, Intent, Context w kontekście DI.

1. Jaki problem rozwiązuje **Wstrzykiwanie Zależności** (Dependency Injection - DI)?
2. Czym się różni **Hilt** od **Daggera**?
3. Jak **Hilt** (lub inne DI) upraszcza pracę z **MVVM**?
4. Jaka jest rola adnotacji **@HiltAndroidApp** i gdzie ją umieszczamy?
5. Jaka jest rola adnotacji **@AndroidEntryPoint**?
6. Jak wstrzykujemy zależności do **ViewModelu** za pomocą **Hilt**? Jakich dwóch adnotacji użyjemy?
7. Czym różni się **@Inject** constructor od metody z adnotacją **@Provides**? Kiedy którego używamy?
8. Co to jest moduł **Hilt** (**@Module**)?

9. Do czego służy adnotacja **@InstallIn**?
10. Co oznacza **@InstallIn(SingletonComponent::class)**?
11. Dlaczego **Retrofit** i **Room Database** zazwyczaj dostarczamy jako **singletony**?
12. Jakiej funkcji używamy w **Composable**, aby uzyskać **ViewModel** zarządzany przez **Hilt**?
13. Dlaczego w architekturze aplikacji wprowadzamy dodatkową **warstwę domeny** (domain layer)?
14. Co to jest **Use Case** (lub **Interactor**)?
15. Jaka jest główna zasada przy projektowaniu **Use Case'ów**?
16. W jaki sposób **ViewModel** staje się "*chudszy*" po wprowadzeniu **Use Case'ów**?
17. Co robi operator **fun invoke** w klasie **Use Case**?
18. Czy **Use Case** może zależeć od **ViewModelu**? Uzasadnij odpowiedź.
19. Czy **Use Case** może zależeć od frameworka **Androida** (np. od **Context**)? Dlaczego?
20. Narysuj i opisz schemat **Czystej Architektury** (View -> ViewModel -> Use Case -> Repository).
21. Jakie są zalety posiadania **warstwy domeny**?
22. Co to jest **Activity** i jak **Hilt** zarządza zależnościami w jej cyklu życia?
23. Czym jest **Context** i jak **Hilt** go dostarcza (jakiej adnotacji używamy)?
24. Co to jest **Intent**?
25. Jak **Hilt** dostarcza **SavedStateHandle** do **ViewModelu**?
26. Wyjaśnij, jak **Hilt** buduje graf zależności na przykładzie: **ViewModel** potrzebuje **Repository**, a **Repository** potrzebuje **ApiService**.
27. Jaka jest rola adnotacji **@Singleton**?
28. Czy **Repository** zawsze musi być **singletonem**? Uzasadnij.
29. Opisz, jak **Hilt** rozwiązuje problem tworzenia **ViewModelu z parametrami**.