

MenuAnNam - Mobile Application Source Code Documentation

Nguyen Thien Nguyen - 10422059

January 20, 2026

Contents

1	Overview	2
1.1	Project Description	2
1.2	Technology Stack	2
1.3	AWS Lambda Integration	2
2	Application Entry Point	3
2.1	MainActivity.kt	3
2.2	Utils.kt	3
3	Data Layer	4
3.1	Database	4
3.1.1	FlashCardDao.kt	4
3.1.2	MenuDatabase.kt	4
3.2	Entity	5
3.2.1	FlashCard.kt	5
3.3	Network	5
3.3.1	DataTypes.kt	5
3.3.2	NetworkService.kt	5
4	Presentation Layer	6
4.1	Navigation	6
4.1.1	Navigator.kt	6
4.1.2	Routes.kt	8
4.2	Components	8
4.2.1	TopBarComponent.kt	8
4.2.2	BottomBarComponent.kt	8
4.3	Screens	9
4.3.1	MenuScreen.kt	9
4.3.2	LoginScreen.kt	9
4.3.3	TokenScreen.kt	10
4.3.4	AddScreen.kt	11
4.3.5	FilterScreen.kt	12
4.3.6	SearchScreen.kt	13
4.3.7	EditScreen.kt	14
4.3.8	StudyScreen.kt	16
5	UI Theme	19
5.1	Color.kt	19
5.2	Theme.kt	19
5.3	Type.kt	19
6	Build Configuration	20
6.1	build.gradle.kts (App Module)	20
6.2	libs.versions.toml	21
7	Mobile Application Use Cases & Implementation	23
7.1	Use Cases Implementation Status	23
7.1.1	1. Request an Authentication Token	23
7.1.2	2. Save the Pair Email Address/Authentication Token in the App	23
7.1.3	3. Add a Flashcard	23
7.1.4	4. Search Flashcards	24
7.1.5	5. Edit a Flashcard	24
7.1.6	6. Study Flashcards	25

Chapter 1

Overview

1.1 Project Description

MenuAnNam is an Android mobile application built with Jetpack Compose for learning Vietnamese vocabulary through flashcards. The app integrates with AWS Lambda services for authentication and audio synthesis, uses Room database for local storage, and implements modern Android architecture patterns.

1.2 Technology Stack

- **UI Framework:** Jetpack Compose with Material 3
- **Language:** Kotlin
- **Database:** Room (SQLite)
- **Networking:** Retrofit 2 with OkHttp
- **Data Storage:** DataStore Preferences
- **Audio Playback:** ExoPlayer (Media3)
- **Navigation:** Type-safe Compose Navigation
- **Backend:** AWS Lambda Functions

1.3 AWS Lambda Integration

- **Token Generation:** <https://egsbwqh7kildllpkijk6nt4soq0wlgpe.lambda-url.ap-southeast-1.on.aws/>
- **Audio Synthesis:** <https://ityqvv3rx5vifjpyufgnpkv5te0ibr cx.lambda-url.ap-southeast-1.on.aws/>

Chapter 2

Application Entry Point

2.1 MainActivity.kt

```
1 package com.example.menuannam
2
3 // Main entry point: initializes database, network,
  navigation, and DataStore for app
4 import android.content.Context
5 import android.os.Bundle
6 import androidx.activity.ComponentActivity
7 import androidx.activity.compose.setContent
8 import androidx.activity.enableEdgeToEdge
9 import androidx.compose.runtime.rememberCoroutineScope
10 import androidx.datastore.preferences.core.
    stringPreferencesKey
11 import androidx.datastore.preferences.preferencesDataStore
12 import androidx.navigation.compose.rememberNavController
13 import com.example.menuannam.data.database.FlashCardDatabase
14 import com.example.menuannam.data.network.NetworkService
15 import com.example.menuannam.presentation.navigation.
    AppNavigation
16 import com.example.menuannam.ui.theme.MenuAnNamTheme
17 import okhttp3.OkHttpClient
18 import retrofit2.Retrofit
19 import retrofit2.converter.gson.GsonConverterFactory
20 import java.util.concurrent.TimeUnit
21
22 // DataStore for persistent user credentials (email/token)
23 val Context.dataStore by preferencesDataStore(name = "
    user_credentials")
24 val TOKEN = stringPreferencesKey("token")
25 val EMAIL = stringPreferencesKey("email")
26
27 class MainActivity : ComponentActivity() {
28     override fun onCreate(savedInstanceState: Bundle?) {
29         super.onCreate(savedInstanceState)
30
31         enableEdgeToEdge()
32         setContent {
33             MenuAnNamTheme {
34                 val navController = rememberNavController()
35                 val appContext = applicationContext
36                 val scope = rememberCoroutineScope()
37
38                 // Database singleton for flashcard CRUD
39                 val db = FlashCardDatabase.getDatabase(
40                     appContext)
41                 val flashCardDao = db.flashCardDao()
42
43                 // HTTP client with 30s timeouts to prevent
44                 // Lambda timeout issues
45                 val sharedOkHttpClient = OkHttpClient.
46                     Builder()
47                         .connectTimeout(30, TimeUnit.SECONDS)
48                         .readTimeout(30, TimeUnit.SECONDS)
49                         .build()
50
51                 // Retrofit for Lambda API calls (token
52                 // generation, audio synthesis)
53                 val retrofit = Retrofit.Builder()
54                     .baseUrl("https://placeholder.com") //
55                     // URL overridden per endpoint
56                     .client(sharedOkHttpClient)
57                     .addConverterFactory(
58                         GsonConverterFactory.create())
59                     .build()
60
61                 val networkService = retrofit.create(
62                     NetworkService::class.java)
63
64                 // Pass all services to navigation system
65                 AppNavigation(
66                     navController,
67                     flashCardDao,
68                     scope,
69                     networkService
70                 )
71             }
72         }
73     }
74 }
```

./app/src/main/java/com/example/menuannam/MainActivity.kt

2.2 Utils.kt

```
1 package com.example.menuannam
2
3 import android.content.Context
4 import java.io.File
5 import java.io.FileOutputStream
6 import java.security.MessageDigest
7
8 // Convert string to MD5 hash for unique audio filenames (e.
9 // g., "hello" → "5d41402abc...")
10 fun String.toMd5(): String {
11     val bytes = MessageDigest.getInstance("MD5").digest(this
12     .toByteArray())
13     return bytes.joinToString("") { "%02x".format(it) }
14 }
15
16 // Save Base64-decoded audio bytes to app's private storage
17 // as MP3
18 fun saveAudioToInternalStorage(context: Context, audioData:
19     ByteArray, filename: String): File {
20     val file = File(context.filesDir, filename) // Private
21     // app directory
22     FileOutputStream(file).use { fos ->
23         fos.write(audioData)
24     }
25     return file // Return File for ExoPlayer playback
26 }
27
28 // Check if audio file exists in cache before making API
29 // call
30 fun getCachedAudioFile(context: Context, word: String): File
31 ? {
32     val file = File(context.filesDir, "${word.toMd5()}.mp3")
33     return if (file.exists()) file else null
34 }
```

./app/src/main/java/com/example/menuannam/Utils.kt

Chapter 3

Data Layer

3.1 Database

3.1.1 FlashCardDao.kt

```
1 package com.example.menuannam.data.database
2
3 import androidx.room.Dao
4 import androidx.room.Delete
5 import androidx.room.Insert
6 import androidx.room.OnConflictStrategy
7 import androidx.room.Query
8 import androidx.room.RawQuery
9 import androidx.sqlite.db.SupportSQLiteQuery
10 import com.example.menuannam.data.entity.FlashCard
11
12 // DAO for FlashCard CRUD operations; all suspend functions
13 // run on background thread
14 @Dao
15 interface FlashCardDao {
16     // Raw SQL for database checkpoint operations
17     @RawQuery
18     fun checkpoint(supportSQLiteQuery: SupportSQLiteQuery):
19         Int
20
21     @Query("SELECT * FROM FlashCards")
22     suspend fun getAll(): List<FlashCard>
23
24     // Random shuffle for study sessions
25     @Query("SELECT * FROM FlashCards ORDER BY RANDOM() LIMIT
26         :size")
27     suspend fun getLesson(size: Int): List<FlashCard>
28
29     // Duplicate detection by matching both fields
30     @Query(
31         "SELECT * FROM FlashCards WHERE english_card LIKE :
32         english AND " +
33         "vietnamese_card LIKE :vietnamese LIMIT 1"
34     )
35     suspend fun findByCards(english: String, vietnamese:
36         String): FlashCard?
37
38     // IGNORE strategy: silently skip if duplicate exists (
39     // returns -1)
40     @Insert(onConflict = OnConflictStrategy.IGNORE)
41     suspend fun insert(flashCard: FlashCard): Long
42
43     @Insert(onConflict = OnConflictStrategy.IGNORE)
44     suspend fun insertAll(vararg flashCard: FlashCard)
45
46     // Update by old English/Vietnamese pair (search key) to
47     // new values
48     @Query(
49         "UPDATE FlashCards SET english_card = :englishNew "
50         +
51         " , vietnamese_card =:vietnameseNew " +
```

```
52         "WHERE english_card = :englishOld " +
53         "AND vietnamese_card = :vietnameseOld"
54     )
55     suspend fun updateFlashCard(
56         englishOld: String,
57         vietnameseOld: String,
58         englishNew: String,
59         vietnameseNew: String
60     )
61
62     @Query(
63         "DELETE FROM FlashCards WHERE english_card = :
64         english " +
65         "AND vietnamese_card =:vietnamese"
66     )
67     suspend fun deleteFlashCard(english: String, vietnamese:
68         String)
69
70     @Query("SELECT * FROM FlashCards WHERE uid = :id")
71     suspend fun getById(id: Int): FlashCard?
72
73     @Query("SELECT COUNT(*) FROM FlashCards")
74     suspend fun getCount(): Int
75
76     @Query("UPDATE FlashCards SET english_card = :english,
77         vietnamese_card = :vietnamese WHERE uid = :id")
78     suspend fun update(id: Int, english: String, vietnamese:
79         String)
80
81     @Delete
82     suspend fun delete(flashCard: FlashCard)
83
84     // CASE WHEN for exact vs partial match: exactEn=1 →
85     // LIKE :en, exactEn=0 → LIKE '%' || :en || '%'
86     @Query(
87         "SELECT * FROM FlashCards WHERE " +
88         "(CASE WHEN :exactEn THEN english_card LIKE
89         :en " +
90         "WHEN NOT :exactEn THEN english_card LIKE
91         '%' || :en || '%' END) " +
92         "AND " +
93         "(CASE WHEN :exactVn THEN vietnamese_card
94         LIKE :vn " +
95         "WHEN NOT :exactVn THEN vietnamese_card LIKE
96         '%' || :vn || '%' END)"
97     )
98     suspend fun getFilteredFlashCards(en: String, exactEn:
99         Int, vn: String, exactVn: Int): List<FlashCard>
100 }
```

./app/src/main/java/com/example/menuannam/data/database/FlashCardDao.kt

3.1.2 MenuDatabase.kt

```
1 package com.example.menuannam.data.database
2
3 import android.content.Context
4 import androidx.room.Database
5 import androidx.room.Room
6 import androidx.room.RoomDatabase
7 import com.example.menuannam.data.entity.FlashCard
8 import kotlin.jvm.Volatile
9
10 // Room database with singleton pattern for thread-safe
11 // FlashCard persistence
```

```
11 @Database(entities = [FlashCard::class], version = 1)
12 abstract class FlashCardDatabase : RoomDatabase() {
13     abstract fun flashCardDao(): FlashCardDao
14
15     companion object {
16         @Volatile // All threads see latest value
17         immediately
18         private var INSTANCE: FlashCardDatabase? = null
19
20         // Double-checked locking: fast path check, then
21         // synchronized creation to prevent race conditions
```

```

20     fun getDatabase(context: Context): FlashCardDatabase
21     {
22         return INSTANCE ?: synchronized(this) {
23             val instance = Room.databaseBuilder(
24                 context.applicationContext, // Use app
25                 context to prevent memory leaks
26                 FlashCardDatabase::class.java,
27                 "FlashCardDatabase"
28             ).build()

```

```

27         INSTANCE = instance
28     }
29     instance
30 }
31 }
32 }

```

./app/src/main/java/com/example/menuannam/data/database/M

3.2 Entity

3.2.1 FlashCard.kt

```

1 package com.example.menuannam.data.entity
2
3 import androidx.room.ColumnInfo
4 import androidx.room.Entity
5 import androidx.room.Index
6 import androidx.room.PrimaryKey
7 import kotlinx.serialization.Serializable
8
9 // FlashCard entity with unique index on (english_card,
10 // vietnamese_card) to prevent duplicates
11 @Entity(tableName = "FlashCards", indices = [Index(
12     value = ["english_card", "vietnamese_card"],
13     unique = true // Database-level constraint enforced by

```

```

13 Room
14 ))
15 @Serializable
16 data class FlashCard(
17     @PrimaryKey(autoGenerate = true) val uid: Int, // Auto-
18     increment from 1
19     @ColumnInfo(name = "english_card") val englishCard:
20     String?,
21     @ColumnInfo(name = "vietnamese_card") val vietnameseCard
22     : String?
23 )

```

./app/src/main/java/com/example/menuannam/data/entity/Flash

3.3 Network

3.3.1 DataTypes.kt

```

1 package com.example.menuannam.data.network
2
3 import kotlinx.serialization.Serializable
4
5 // Request payloads and responses for Lambda API calls
6
7 @Serializable
8 data class UserCredential(val email: String) // Token
9 // generation request
10
11 @Serializable
12 data class TokenResponse(
13     val code: Int, // HTTP status (200 = success)
14     val message: String // Token string or error message
15 )
16 @Serializable

```

```

17 data class AudioRequest(
18     val word: String, // Vietnamese word for audio synthesis
19     val email: String, // User tracking
20     val token: String // Auth token (expires after some time
21 )
22
23 @Serializable
24 data class AudioResponse(
25     val code: Int, // HTTP status (200 = success, 500 =
26     token invalid/expired)
27     val message: String // Base64-encoded MP3 or error
28     message
29 )

```

./app/src/main/java/com/example/menuannam/data/network/Da

3.3.2 NetworkService.kt

```

1 package com.example.menuannam.data.network
2
3 import retrofit2.http.Body
4 import retrofit2.http.PUT
5 import retrofit2.http.Url
6
7 // Retrofit interface for AWS Lambda endpoints; default URLs
8 // overridden per call
9 interface NetworkService {
10     // Token generation: sends email, returns token or error
11     @PUT
12     suspend fun generateToken(
13         @Url url: String = "https://
14         egsbwqh7kild1lpkijk6nt4soq0wlgpe.lambda-url.ap-
15         southeast-1.on.aws/",

```

```

13         @Body email: UserCredential
14     ): TokenResponse
15
16     // Audio synthesis: sends Vietnamese word + token,
17     // returns Base64 MP3 or 500 if token expired
18     @PUT
19     suspend fun generateAudio(
20         @Url url: String = "https://
21         ityqwv3rx5vifjpyufgnpkv5te0ibrcx.lambda-url.ap-
22         southeast-1.on.aws/",
23         @Body request: AudioRequest
24     ): AudioResponse
25 }

```

./app/src/main/java/com/example/menuannam/data/network/Ne

Chapter 4

Presentation Layer

4.1 Navigation

4.1.1 Navigator.kt

```
1 package com.example.menuannam.presentation.navigation
2
3 import androidx.compose.foundation.layout.padding
4 import androidx.compose.material3.Scaffold
5 import androidx.compose.runtime.*
6 import androidx.navigation.NavHostController
7 import androidx.navigation.compose.NavHost
8 import androidx.navigation.compose.composable
9 import androidx.navigation.compose.currentBackStackEntryAsState
10 import androidx.navigation.toRoute
11 import kotlinx.coroutines.CoroutineScope
12 import kotlinx.coroutines.launch
13 import com.example.menuannam.data.database.FlashCardDao
14 import com.example.menuannam.presentation.screens.MenuScreen
15 import com.example.menuannam.presentation.screens.AddScreen
16 import com.example.menuannam.presentation.screens.StudyScreen
17 import com.example.menuannam.presentation.screens.CardViewMode
18 import com.example.menuannam.presentation.screens.SearchScreen
19 import com.example.menuannam.presentation.screens.FilterScreen
20 import com.example.menuannam.presentation.screens.EditScreen
21 import com.example.menuannam.presentation.screens.LoginScreen
22 import com.example.menuannam.presentation.screens.TokenScreen
23 import com.example.menuannam.data.network.NetworkService
24 import com.example.menuannam.presentation.components.TopBarComponent
25 import com.example.menuannam.presentation.components.BottomBarComponent
26
27 // Central Navigation Hub: Controls all screen transitions
28 // and manages app state
29 // Uses Compose Navigation with type-safe routes for compile
30 // -time error checking
31 @Composable
32 fun AppNavigation(
33     navigation: NavHostController, // NavController from
34     MainActivity
35     flashCardDao: FlashCardDao, // Database access
36     coroutineScope: CoroutineScope, // For async operations
37     (launched from MainActivity)
38     networkService: NetworkService // Retrofit service for
39     API calls
40 ) {
41     // Shared message across all screens - updated by each
42     // screen, displayed in BottomBar
43     var message by remember { mutableStateOf("Welcome!") }
44     val changeMessage: (String) -> Unit = { message = it }
45
46     // Get current route name from back stack to determine
47     // what screen is currently displayed
48     val backStackEntry by navigation.currentBackStackEntryAsState()
49     val routeName = backStackEntry?.destination?.route
50
51     // Map route names to user-friendly titles displayed in
52     // TopBar
53     val title = when (routeName) {
54         HomeRoute::class.qualifiedName -> "Menu An Nam"
55         StudyCardsRoute::class.qualifiedName -> "Study Cards"
56         "
57         AddCardRoute::class.qualifiedName -> "Add Card"
58         SearchCardsRoute::class.qualifiedName -> "Search
59         Results"
60
61         FilterRoute::class.qualifiedName -> "Search Cards"
62         LoginRoute::class.qualifiedName -> "Login"
63         ShowCardRoute::class.qualifiedName -> "Flash Card"
64         EditCardRoute::class.qualifiedName -> "Edit Card"
65         else -> "Menu An Nam"
66     }
67
68     /**
69      * Show back button on all screens EXCEPT home
70      * Only home doesn't have a back button (it's the start
71      * destination)
72      */
73     val showBack = routeName != HomeRoute::class.qualifiedName
74     val navigateBack: () -> Unit = { navigation.navigateUp() }
75
76     // ===== SCAFFOLD LAYOUT =====
77     /**
78      * Scaffold provides standard app structure:
79      * - topBar: Title and back button
80      * - content: NavHost with screen composition
81      * - bottomBar: Status message display
82      */
83     Scaffold(
84         topBar = {
85             TopBarComponent(
86                 title = title,
87                 showBack = if (showBack) navigateBack else
88                 null
89             )
90         },
91         bottomBar = {
92             BottomBarComponent(message = message)
93         }
94     ) { innerPadding ->
95
96         // ===== NAVIGATION HOST =====
97         /**
98          * NavHost manages all composable screens
99          * startDestination = HomeRoute (app always starts
100          * at home)
101          * Padding prevents overlap with TopBar/BottomBar
102          */
103         NavHost(
104             navController = navigation,
105             startDestination = HomeRoute,
106             modifier = androidx.compose.ui.Modifier.padding(
107                 innerPadding)
108         ) {
109             //
110             // =====
111             // HOME ROUTE - Main menu
112             //
113             // =====
114             composable<HomeRoute> {
115                 MenuScreen(
116                     onStudy = {
117                         navigation.navigate(StudyCardsRoute)
118                     },
119                     onAdd = { navigation.navigate(
120                         AddCardRoute) },
121                     onSearch = { navigation.navigate(
122                         FilterRoute) },
123                     onLogin = { navigation.navigate(
124                         LoginRoute) },
125                 )
126             }
127         }
128     }
129 }
```

```

106         changeMessage = changeMessage
107     )
108 }
109 //
110 =====
111 // LOGIN ROUTE - Get email and request token
112 //
113 =====
114 composable<LoginRoute> {
115     LoginScreen(
116         changeMessage = changeMessage,
117         networkService = networkService,
118         navigateToToken = { enteredEmail ->
119             navigation.navigate(TokenRoute(
120                 enteredEmail))
121         }
122     )
123 }
124 //
125 =====
126 // STUDY CARDS ROUTE - Interactive flashcard
127 study session
128 //
129 =====
130 composable<StudyCardsRoute> {
131     StudyScreen(
132         changeMessage = changeMessage,
133         flashCardDao = flashCardDao,
134         networkService = networkService,
135         mode = CardViewMode.STUDY_SESSION,
136         coroutineScope = coroutineScope
137     )
138 }
139 //
140 =====
141 // ADD CARD ROUTE - Create new flashcard
142 //
143 =====
144 composable<AddCardRoute> {
145     AddScreen(
146         changeMessage = changeMessage,
147         insertFlashCard = { card ->
148             coroutineScope.launch {
149                 // Insert card (Room ignores if
150                 duplicate)
151                 flashCardDao.insertAll(card)
152                 changeMessage("Added: ${card.
153                     englishCard}")
154             }
155         }
156     )
157 }
158 //
159 =====
160 // SEARCH CARDS ROUTE - Display filtered search
161 results
162 //
163 =====
164 composable<SearchCardsRoute> { backStackEntry ->
165     val route = backStackEntry.toRoute<
166     SearchCardsRoute>()
167     SearchScreen(
168         changeMessage = changeMessage,
169         flashCardDao = flashCardDao,
170         onEdit = { cardId ->
171             navigation.navigate(EditCardRoute(
172                 cardId))
173         },
174         englishText = route.englishText,
175         exactEnglish = route.exactEnglish,
176         vietnameseText = route.vietnameseText,
177         exactVietnamese = route.exactVietnamese
178     )
179 }
180 //
181 =====
182 // FILTER ROUTE - Search input form
183 //
184 =====
185 composable<FilterRoute> {
186     FilterScreen(
187         changeMessage = changeMessage,
188         onSearch = { en, exactEn, vn, exactVn ->
189             navigation.navigate(
190                 SearchCardsRoute(
191                     englishText = en,
192                     exactEnglish = exactEn,
193                     vietnameseText = vn,
194                     exactVietnamese = exactVn
195                 )
196             )
197         }
198     )
199 }
200 //
201 =====
202 // EDIT CARD ROUTE - Edit existing flashcard
203 with audio management
204 //
205 =====
206 composable<EditCardRoute> { backStackEntry ->
207     val route = backStackEntry.toRoute<
208     EditCardRoute>()
209     EditScreen(
210         cardId = route.id,
211         flashCardDao = flashCardDao,
212         networkService = networkService,
213         changeMessage = changeMessage,
214         onCardUpdated = { navigation.navigateUp
215             () }
216     )
217 }
218 //
219 =====
220 // SHOW CARD ROUTE - View single card and delete
221 if desired
222 //
223 =====
224 /**
225  * Receives cardId from SearchScreen
226  * Displays full card details with delete and
227  * play audio options
228  */
229 composable<ShowCardRoute> { backStackEntry ->
230     val route = backStackEntry.toRoute<
231     ShowCardRoute>()
232     StudyScreen(
233         changeMessage = changeMessage,
234         flashCardDao = flashCardDao,
235         networkService = networkService,
236         mode = CardViewMode.SINGLE_CARD,
237         cardId = route.id,
238         onCardDeleted = { navigation.navigateUp
239             () }
240     )
241 }
242 //
243 =====
244 // TOKEN ROUTE - Store token from login
245 //
246 =====
247 /**
248  * Receives email parameter from LoginRoute
249  * User enters token string received via email
250  * Saves token to DataStore for later audio API
251  * calls
252  */
253 composable<TokenRoute> { backStackEntry ->
254     val route = backStackEntry.toRoute<
255     TokenRoute>()
256     TokenScreen(
257         email = route.email,
258         changeMessage = changeMessage,
259         navigateToHome = {
260             navigation.navigate(HomeRoute) {
261                 popUpTo(HomeRoute) { inclusive =
262                     true }
263             }
264         }
265     )
266 }
267 }
268 }
269 }
270 }
271 }
272 }
273 }
274 }
275 }
276 }
277 }
278 }
279 }
280 }
281 }
282 }
283 }
284 }
285 }
286 }
287 }
288 }
289 }
290 }
291 }
292 }
293 }
294 }
295 }
296 }
297 }
298 }
299 }
300 }
301 }
302 }
303 }
304 }
305 }
306 }
307 }
308 }
309 }
310 }
311 }
312 }
313 }
314 }
315 }
316 }
317 }
318 }
319 }
320 }
321 }
322 }
323 }
324 }
325 }
326 }
327 }
328 }
329 }
330 }
331 }
332 }
333 }
334 }
335 }
336 }
337 }
338 }
339 }
340 }
341 }
342 }
343 }
344 }
345 }
346 }
347 }
348 }
349 }
350 }
351 }
352 }
353 }
354 }
355 }
356 }
357 }
358 }
359 }
360 }
361 }
362 }
363 }
364 }
365 }
366 }
367 }
368 }
369 }
370 }
371 }
372 }
373 }
374 }
375 }
376 }
377 }
378 }
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
501 }
502 }
503 }
504 }
505 }
506 }
507 }
508 }
509 }
510 }
511 }
512 }
513 }
514 }
515 }
516 }
517 }
518 }
519 }
520 }
521 }
522 }
523 }
524 }
525 }
526 }
527 }
528 }
529 }
530 }
531 }
532 }
533 }
534 }
535 }
536 }
537 }
538 }
539 }
540 }
541 }
542 }
543 }
544 }
545 }
546 }
547 }
548 }
549 }
550 }
551 }
552 }
553 }
554 }
555 }
556 }
557 }
558 }
559 }
560 }
561 }
562 }
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
600 }
601 }
602 }
603 }
604 }
605 }
606 }
607 }
608 }
609 }
610 }
611 }
612 }
613 }
614 }
615 }
616 }
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 }
628 }
629 }
630 }
631 }
632 }
633 }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 }
642 }
643 }
644 }
645 }
646 }
647 }
648 }
649 }
650 }
651 }
652 }
653 }
654 }
655 }
656 }
657 }
658 }
659 }
660 }
661 }
662 }
663 }
664 }
665 }
666 }
667 }
668 }
669 }
670 }
671 }
672 }
673 }
674 }
675 }
676 }
677 }
678 }
679 }
680 }
681 }
682 }
683 }
684 }
685 }
686 }
687 }
688 }
689 }
690 }
691 }
692 }
693 }
694 }
695 }
696 }
697 }
698 }
699 }
700 }
701 }
702 }
703 }
704 }
705 }
706 }
707 }
708 }
709 }
710 }
711 }
712 }
713 }
714 }
715 }
716 }
717 }
718 }
719 }
720 }
721 }
722 }
723 }
724 }
725 }
726 }
727 }
728 }
729 }
730 }
731 }
732 }
733 }
734 }
735 }
736 }
737 }
738 }
739 }
740 }
741 }
742 }
743 }
744 }
745 }
746 }
747 }
748 }
749 }
750 }
751 }
752 }
753 }
754 }
755 }
756 }
757 }
758 }
759 }
760 }
761 }
762 }
763 }
764 }
765 }
766 }
767 }
768 }
769 }
770 }
771 }
772 }
773 }
774 }
775 }
776 }
777 }
778 }
779 }
780 }
781 }
782 }
783 }
784 }
785 }
786 }
787 }
788 }
789 }
790 }
791 }
792 }
793 }
794 }
795 }
796 }
797 }
798 }
799 }
800 }
801 }
802 }
803 }
804 }
805 }
806 }
807 }
808 }
809 }
810 }
811 }
812 }
813 }
814 }
815 }
816 }
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }
836 }
837 }
838 }
839 }
840 }
841 }
842 }
843 }
844 }
845 }
846 }
847 }
848 }
849 }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }
1000 }

```

./app/src/main/java/com/example/menuannam/presentation/navi

4.1.2 Routes.kt

```
1 package com.example.menuannam.presentation.navigation
2
3 import kotlinx.serialization.Serializable
4
5 // TYPE-SAFE NAVIGATION ROUTES: @Serializable objects for
6 // compile-time verification
7 // Kotlinx.serialization handles automatic type conversion
8 // for route parameters
9
10 @Serializable
11 object HomeRoute // Main menu: Study, Add Cards, Search,
12 // Login
13
14 @Serializable
15 object AddCardRoute // Create new flashcard with English/
16 // Vietnamese input
17
18 @Serializable
19 object StudyCardsRoute // Study 5 random cards with audio
20 // playback
21
22 @Serializable
23 data class SearchCardsRoute( // Display filtered search
24 // results with edit/delete options
25 val englishText: String = "", // English search term
26 val exactEnglish: Int = 0, // 1 for exact match, 0 for
27 // partial match
```

```
21 val vietnameseText: String = "", // Vietnamese search
22 // term
23 val exactVietnamese: Int = 0 // 1 for exact match, 0 for
24 // partial match
25 )
26 @Serializable
27 object LoginRoute // User enters email to get token for
28 // audio synthesis
29
30 @Serializable
31 data class ShowCardRoute(val id: Int) // View single
32 // flashcard with delete and audio playback
33
34 @Serializable
35 data class EditCardRoute(val id: Int) // Edit existing
36 // flashcard by ID
37
38 @Serializable
39 data class TokenRoute(val email: String) // Token input
40 // screen - email passed from LoginRoute for context
41
42 @Serializable
43 object FilterRoute // Filter search screen with English/
44 // Vietnamese terms and exact/partial match options
```

./app/src/main/java/com/example/menuannam/presentation/navi

4.2 Components

4.2.1 TopBarComponent.kt

```
1 package com.example.menuannam.presentation.components
2
3 import androidx.compose.ui.semantics.contentDescription
4 import androidx.compose.material3.ButtonDefaults
5 import androidx.compose.material3.CenterAlignedTopAppBar
6 import androidx.compose.material3.ExperimentalMaterial3Api
7 import androidx.compose.material3.MaterialTheme
8 import androidx.compose.material3.Text
9 import androidx.compose.material3.TextButton
10 import androidx.compose.material3.TopAppBarDefaults
11 import androidx.compose.runtime.Composable
12 import androidx.compose.ui.Modifier
13 import androidx.compose.ui.semantics.semantics
14
15 @OptIn(ExperimentalMaterial3Api::class)
16 @Composable
17 fun TopBarComponent(
18     title: String, // Screen name to display (e.g., "Study
19 // Cards", "Add Card")
20 showBack: (() -> Unit)? = null, // Optional callback to
21 // navigate up - if null, no back button shown
22 ) {
23     CenterAlignedTopAppBar(
24         title = { Text(title) },
25         navigationIcon = {
26             if (showBack != null) { // Conditional back
27                 button: shows on all screens except HomeRoute
28                 TextButton(
29                     onClick = showBack,
```

```
27 modifier = Modifier.semantics { //
28 // Enables testing framework to find elements
29 contentDescription = "navigateBack"
30 },
31 colors = ButtonDefaults.textButtonColors
32 (
33     contentColor = MaterialTheme.
34     colorScheme.onPrimary
35     )
36     ) {
37         Text("Back")
38     }
39 }
40 },
41 colors = TopAppBarDefaults.topAppBarColors(
42     containerColor = MaterialTheme.colorScheme.
43     primary,
44     titleContentColor = MaterialTheme.colorScheme.
45     onPrimary,
46     navigationIconContentColor = MaterialTheme.
47     colorScheme.onPrimary,
48     actionIconContentColor = MaterialTheme.
49     colorScheme.onPrimary
50     )
51 )
52 }
```

./app/src/main/java/com/example/menuannam/presentation/com

4.2.2 BottomBarComponent.kt

```
1 package com.example.menuannam.presentation.components
2
3 import androidx.compose.foundation.layout.fillMaxWidth
4 import androidx.compose.material3.BottomAppBar
5 import androidx.compose.material3.Text
6 import androidx.compose.runtime.Composable
7 import androidx.compose.ui.Modifier
8 import androidx.compose.ui.semantics.contentDescription
9 import androidx.compose.ui.semantics.semantics
10 import androidx.compose.ui.text.style.TextAlign
11
12 @Composable
13 fun BottomBarComponent(
14     message: String // Current status message (updated by
15 // Navigator's changeMessage callback)
```

```
15 ) {
16     BottomAppBar(){
17         Text( // Shows real-time feedback: "Card 1 of 5", "
18 // Logged out", "Error loading flashcards: ..."
19 modifier = Modifier
20 // .fillMaxWidth()
21 // .semantics { contentDescription = "Message"
22 },
23     textAlign = TextAlign.Center,
24     text = message
25     )
26 }
```

./app/src/main/java/com/example/menuannam/presentation/com

4.3 Screens

4.3.1 MenuScreen.kt

```
1 package com.example.menuannam.presentation.screens
2
3 import androidx.compose.foundation.layout.Arrangement
4 import androidx.compose.foundation.layout.Column
5 import androidx.compose.foundation.layout.fillMaxSize
6 import androidx.compose.foundation.layout.fillMaxWidth
7 import androidx.compose.foundation.layout.padding
8 import androidx.compose.material3.Button
9 import androidx.compose.material3.ExperimentalMaterial3Api
10 import androidx.compose.material3.Text
11 import androidx.compose.runtime.Composable
12 import androidx.compose.runtime.LaunchedEffect
13 import androidx.compose.runtime.getValue
14 import androidx.compose.runtime.mutableStateOf
15 import androidx.compose.runtime.remember
16 import androidx.compose.runtime.rememberCoroutineScope
17 import androidx.compose.runtime.setValue
18 import androidx.compose.ui.Alignment
19 import androidx.compose.ui.Modifier
20 import androidx.compose.ui.platform.LocalContext
21 import androidx.compose.ui.semantics.contentDescription
22 import androidx.compose.ui.semantics.semantics
23 import androidx.compose.ui.unit.dp
24 import androidx.datastore.preferences.core.edit
25 import com.example.menuannam.EMAIL
26 import com.example.menuannam.TOKEN
27 import com.example.menuannam.dataStore
28 import kotlinx.coroutines.flow.first
29 import kotlinx.coroutines.launch
30
31 @OptIn(ExperimentalMaterial3Api::class)
32 @Composable
33 fun MenuScreen(
34     onStudy: () -> Unit, // Navigate to StudyCardsRoute
35     onAdd: () -> Unit, // Navigate to AddCardRoute
36     onSearch: () -> Unit, // Navigate to SearchCardsRoute
37     onLogin: () -> Unit, // Navigate to LoginRoute
38     changeMessage: (String) -> Unit = {} // Update bottom
39 ) {
40     val context = LocalContext.current
41     val appContext = context.applicationContext
42     val scope = rememberCoroutineScope()
43
44     var email by remember { mutableStateOf("") }
45
46     // Load saved email from DataStore on screen load -
47     // shows user they are logged in if email stored
48     LaunchedEffect(Unit) {
49         val prefs = appContext.dataStore.data.first()
50         email = prefs[EMAIL] ?: ""
51         changeMessage("Email loaded: $email")
52     }
53
54     Column(
55         modifier = Modifier
```

```
56         .padding(16.dp),
57         verticalArrangement = Arrangement.spacedBy(16.dp,
58             Alignment.CenterVertically),
59         horizontalAlignment = Alignment.CenterHorizontally
60     ) {
61         Button(
62             onClick = onStudy,
63             modifier = Modifier.fillMaxWidth()
64             .semantics { contentDescription = "
65                 navigateToStudyCards" }
66             ) { Text("Study Cards") }
67
68         Button(
69             onClick = onAdd,
70             modifier = Modifier.fillMaxWidth()
71             .semantics { contentDescription = "
72                 navigateToAddCard" }
73             ) { Text("Add Cards") }
74
75         Button(
76             onClick = onSearch,
77             modifier = Modifier.fillMaxWidth()
78             .semantics { contentDescription = "
79                 navigateToSearchCards" }
80             ) { Text("Search Cards") }
81
82         Button(
83             onClick = onLogin,
84             modifier = Modifier.fillMaxWidth()
85             .semantics { contentDescription = "
86                 navigateToLoginScreen" }
87             ) { Text("Login") }
88
89         Button(
90             onClick = {
91                 scope.launch {
92                     appContext.dataStore.edit {
93                         it.remove(EMAIL)
94                         it.remove(TOKEN)
95                     }
96                     email = ""
97                     changeMessage("Logged out")
98                 }
99             },
100             modifier = Modifier.fillMaxWidth()
101             .semantics { contentDescription = "
102                 ExecuteLogout" }
103             ) { Text("Log out", modifier = Modifier.semantics {
104                 contentDescription = "Logout" })
105         }
106     }
```

./app/src/main/java/com/example/menuannam/presentation/screens

4.3.2 LoginScreen.kt

```
1 package com.example.menuannam.presentation.screens
2
3 import androidx.compose.foundation.layout.Arrangement
4 import androidx.compose.foundation.layout.Column
5 import androidx.compose.foundation.layout.fillMaxWidth
6 import androidx.compose.foundation.layout.padding
7 import androidx.compose.material3.Button
8 import androidx.compose.material3.OutlinedTextField
9 import androidx.compose.material3.Text
10 import androidx.compose.runtime.Composable
11 import androidx.compose.runtime.LaunchedEffect
12 import androidx.compose.runtime.getValue
13 import androidx.compose.runtime.mutableStateOf
14 import androidx.compose.runtime.rememberCoroutineScope
15 import androidx.compose.runtime.saveable.rememberSaveable
16 import androidx.compose.runtime.setValue
17 import androidx.compose.ui.Modifier
18 import androidx.compose.ui.semantics.contentDescription
19 import androidx.compose.ui.semantics.semantics
20 import androidx.compose.ui.unit.dp
21 import com.example.menuannam.data.network.UserCredential
22 import com.example.menuannam.data.network.NetworkService
23 import kotlinx.coroutines.Dispatchers
24 import kotlinx.coroutines.launch
25 import kotlinx.coroutines.withContext
26
27 @Composable
28 fun LoginScreen(
29     changeMessage: (String) -> Unit, // Updates status bar
```

```
30     with feedback
31     networkService: NetworkService, // Retrofit interface
32     for Lambda API
33     navigateToToken: (String) -> Unit // Callback to show
34     TokenScreen with email
35 ) {
36     var email by rememberSaveable { mutableStateOf("") }
37     val scope = rememberCoroutineScope()
38
39     LaunchedEffect(Unit) {
40         changeMessage("Please, introduce your email.")
41     }
42
43     Column(
44         modifier = Modifier.padding(16.dp),
45         verticalArrangement = Arrangement.spacedBy(16.dp)
46     ) {
47         OutlinedTextField(
48             value = email,
49             onChange = { email = it },
50             modifier = Modifier
51                 .fillMaxWidth()
52                 .semantics { contentDescription = "
53                     emailTextField" },
54             label = { Text("email") }
55         )
56
57         Button(
58             modifier = Modifier
```

```

55         .fillMaxWidth()
56         .semantics { contentDescription = "Enter" },
57         onClick = {
58             scope.launch {
59                 try { // Send email to AWS Lambda for
token generation
60                     val result = withContext(Dispatchers
.IO) {
61                         networkService.generateToken(
email = UserCredential(email))
62                     }
63                     if (result.code == 200) { //
Response code 200 means success
64                         changeMessage("Token sent to
email: ${result.message}")
65                         navigateToToken(email)
66                     } else {

```

```

67         changeMessage("Error: ${result.
message}")
68     }
69     } catch (e: Exception) {
70         changeMessage("There was an error in
the token request.")
71     }
72 }
73 }
74 }
75 ) {
76     Text("Enter")
77 }
78 }
79 }

```

./app/src/main/java/com/example/menuannam/presentation/screens

4.3.3 TokenScreen.kt

```

1 package com.example.menuannam.presentation.screens
2
3 import androidx.compose.foundation.layout.Arrangement
4 import androidx.compose.foundation.layout.Column
5 import androidx.compose.foundation.layout.fillMaxSize
6 import androidx.compose.foundation.layout.fillMaxWidth
7 import androidx.compose.foundation.layout.padding
8 import androidx.compose.material3.Button
9 import androidx.compose.material3.OutlinedTextField
10 import androidx.compose.material3.Text
11 import androidx.compose.runtime.Composable
12 import androidx.compose.runtime.LaunchedEffect
13 import androidx.compose.runtime.getValue
14 import androidx.compose.runtime.mutableStateOf
15 import androidx.compose.runtime.remember
16 import androidx.compose.runtime.rememberCoroutineScope
17 import androidx.compose.runtime.setValue
18 import androidx.compose.ui.Modifier
19 import androidx.compose.ui.platform.LocalContext
20 import androidx.compose.ui.semantics.contentDescription
21 import androidx.compose.ui.semantics.semantics
22 import androidx.compose.ui.unit.dp
23 import androidx.datastore.preferences.core.edit
24 import com.example.menuannam.EMAIL
25 import com.example.menuannam.TOKEN
26 import com.example.menuannam.dataStore
27 import kotlinx.coroutines.Dispatchers
28 import kotlinx.coroutines.launch
29 import kotlinx.coroutines.withContext
30
31 @Composable
32 fun TokenScreen(
33     email: String, // User email from LoginScreen, displayed
as context
34     changeMessage: (String) -> Unit, // Updates status bar
with feedback
35     navigateToHome: (String) -> Unit // Return to main menu
after save
36 ) {
37     val scope = rememberCoroutineScope()
38     val context = LocalContext.current
39     val appContext = context.applicationContext
40     var token by remember { mutableStateOf("") }
41
42     LaunchedEffect(Unit) {
43         changeMessage("Please, introduce your token.")
44     }
45
46     Column(

```

```

47         modifier = Modifier
48         .fillMaxSize()
49         .padding(16.dp),
50         verticalArrangement = Arrangement.spacedBy(8.dp)
51     ) {
52         OutlinedTextField(
53             value = token,
54             onChange = { token = it },
55             modifier = Modifier
56                 .fillMaxWidth()
57                 .semantics { contentDescription = "
tokenTextField" },
58             label = { Text("token") }
59         )
60         OutlinedTextField(
61             value = email,
62             onChange = {},
63             modifier = Modifier
64                 .fillMaxWidth()
65                 .semantics { contentDescription = "
emailTextField" },
66             label = { Text("email") },
67             readOnly = true
68         )
69         Button(
70             modifier = Modifier
71                 .fillMaxWidth()
72                 .semantics { contentDescription = "Enter" },
73             onClick = {
74                 scope.launch {
75                     withContext(Dispatchers.IO) { // Save
token and email to DataStore for future audio requests
76                         appContext.dataStore.edit {
77                             preferences[EMAIL] = email
78                             preferences[TOKEN] = token
79                         }
80                     }
81                     navigateToHome(token)
82                 }
83             }
84         ) {
85             Text("Enter")
86         }
87     }
88 }

```

./app/src/main/java/com/example/menuannam/presentation/screens

4.3.4 AddScreen.kt

```
1 package com.example.menuannam.presentation.screens
2
3 import androidx.compose.foundation.layout.*
4 import androidx.compose.material3.*
5 import androidx.compose.runtime.*
6 import androidx.compose.runtime.saveable.rememberSaveable
7 import androidx.compose.ui.Alignment
8 import androidx.compose.ui.Modifier
9 import androidx.compose.ui.semantics.contentDescription
10 import androidx.compose.ui.semantics.editableText
11 import androidx.compose.ui.semantics.semantics
12 import androidx.compose.ui.semantics.text
13 import androidx.compose.ui.text.AnnotatedString
14 import androidx.compose.ui.unit.dp
15 import com.example.menuannam.data.entity.FlashCard
16
17 @OptIn(ExperimentalMaterial3Api::class)
18 @Composable
19 fun AddScreen(
20     changeMessage: (String) -> Unit, // Updates status bar
21     with feedback
22     insertFlashCard: (FlashCard) -> Unit // Callback to save
23     card (receives FlashCard)
24 ) {
25     var english by rememberSaveable { mutableStateOf("") }
26     // TextFields persist state across recomposition
27     var vietnamese by rememberSaveable { mutableStateOf("") }
28
29     LaunchedEffect(Unit) {
30         changeMessage("Add a new flashcard")
31     }
32
33     Column(
34         modifier = Modifier
35             .fillMaxSize()
36             .padding(16.dp),
37         verticalArrangement = Arrangement.spacedBy(16.dp),
38         Alignment.CenterVertically,
39         horizontalAlignment = Alignment.CenterHorizontally
40     ) {
41         TextField(
42             value = english,
43             onValueChange = { english = it },
44             label = { Text("English") },
45             modifier = Modifier
46                 .fillMaxWidth()
47                 .semantics { contentDescription = "
48                 enTextField" }
49         )
50
51         TextField(
```

```
52             value = vietnamese,
53             onValueChange = { vietnamese = it },
54             label = { Text("Vietnamese") },
55             modifier = Modifier
56                 .fillMaxWidth()
57                 .semantics { contentDescription = "
58                 viTextField" }
59         )
60
61         Row(
62             modifier = Modifier.fillMaxWidth(),
63             horizontalArrangement = Arrangement.spacedBy(16.
64             dp, Alignment.CenterHorizontally),
65             verticalAlignment = Alignment.CenterVertically
66         ) {
67             Button(
68                 onClick = {
69                     try { // Room handles OnConflictStrategy
70                         .IGNORE (silently ignores duplicates)
71                         insertFlashCard(FlashCard(0, english
72                         , vietnamese))
73
74                         changeMessage("Flash card
75                         successfully added to your database.")
76                         english = ""
77                         vietnamese = ""
78                     } catch (e: Exception) { // Duplicate
79                         detection
80                             changeMessage("Flash card already
81                             exists in your database.")
82                     }
83                 },
84                 modifier = Modifier.semantics {
85                     contentDescription = "Add" }
86             ) {
87                 Text("Add")
88             }
89
90             Button(
91                 onClick = {
92                     english = ""
93                     vietnamese = ""
94                 },
95                 modifier = Modifier.semantics {
96                     contentDescription = "Clear" }
97             ) {
98                 Text("Clear")
99             }
100         }
101     }
102 }
```

./app/src/main/java/com/example/menuannam/presentation/scre

4.3.5 FilterScreen.kt

```

1 package com.example.menuannam.presentation.screens
2
3 import androidx.compose.foundation.layout.Arrangement
4 import androidx.compose.foundation.layout.Column
5 import androidx.compose.foundation.layout.Row
6 import androidx.compose.foundation.layout.Spacer
7 import androidx.compose.foundation.layout.fillMaxWidth
8 import androidx.compose.foundation.layout.padding
9 import androidx.compose.foundation.layout.size
10 import androidx.compose.material3.Button
11 import androidx.compose.material3.Checkbox
12 import androidx.compose.material3.OutlinedTextField
13 import androidx.compose.material3.Text
14 import androidx.compose.runtime.Composable
15 import androidx.compose.runtime.LaunchedEffect
16 import androidx.compose.runtime.getValue
17 import androidx.compose.runtime.mutableStateOf
18 import androidx.compose.runtime.remember
19 import androidx.compose.runtime.setValue
20 import androidx.compose.ui.Alignment
21 import androidx.compose.ui.Modifier
22 import androidx.compose.ui.semantics.contentDescription
23 import androidx.compose.ui.semantics.semantics
24 import androidx.compose.ui.unit.dp
25
26 @Composable
27 fun FilterScreen(
28     changeMessage: (String) -> Unit = {}, // Updates status
29     onSearch: (en: String, exactEn: Int, vn: String, exactVn
30         : Int) -> Unit // Navigate to SearchCardsRoute with
31         parameters
32 ) {
33     var englishText by remember { mutableStateOf("") } //
34     English search term
35     var vietnameseText by remember { mutableStateOf("") } //
36     Vietnamese search term
37     var exactEnglish by remember { mutableStateOf(false) }
38     // Exact match checkbox for English
39     var exactVietnamese by remember { mutableStateOf(false) }
40     // Exact match checkbox for Vietnamese
41
42     LaunchedEffect(Unit) {
43         changeMessage("Enter search criteria and click
44             Search")
45     }
46
47     Column(
48         modifier = Modifier
49             .fillMaxWidth()
50             .padding(16.dp),
51         verticalArrangement = Arrangement.spacedBy(16.dp),
52         horizontalAlignment = Alignment.CenterHorizontally
53     ) {
54         // English search field with checkbox
55         Row(
56             modifier = Modifier.fillMaxWidth(),
57             verticalAlignment = Alignment.CenterVertically,
58             horizontalArrangement = Arrangement.spacedBy(8.
59
60         dp) {
61             OutlinedTextField(
62                 value = englishText,
63                 onChange = { englishText = it },
64                 label = { Text("English") },
65                 modifier = Modifier
66                     .weight(1f)
67                     .semantics { contentDescription = "
68                     enSearchField" },
69                 singleLine = true
70             )
71             Checkbox(
72                 checked = exactEnglish,

```

```

64                 onChangeChange = { exactEnglish = it },
65                 modifier = Modifier.semantics {
66                     contentDescription = "enExactCheckbox" }
67             )
68         }
69         // Vietnamese search field with checkbox
70         Row(
71             modifier = Modifier.fillMaxWidth(),
72             verticalAlignment = Alignment.CenterVertically,
73             horizontalArrangement = Arrangement.spacedBy(8.
74
75         dp) {
76             OutlinedTextField(
77                 value = vietnameseText,
78                 onChange = { vietnameseText = it },
79                 label = { Text("Vietnamese") },
80                 modifier = Modifier
81                     .weight(1f)
82                     .semantics { contentDescription = "
83                     vnSearchField" },
84                 singleLine = true
85             )
86             Checkbox(
87                 checked = exactVietnamese,
88                 onChangeChange = { exactVietnamese = it },
89                 modifier = Modifier.semantics {
90                     contentDescription = "vnExactCheckbox" }
91             )
92         }
93         Spacer(modifier = Modifier.size(16.dp))
94
95         // Search button
96         Button(
97             onClick = {
98                 val en = englishText.trim().isEmpty { "" }
99                 val vn = vietnameseText.trim().isEmpty { ""
100
101                 val exactEn = if (exactEnglish) 1 else 0
102                 val exactVn = if (exactVietnamese) 1 else 0
103
104                 changeMessage("Searching...")
105                 onSearch(en, exactEn, vn, exactVn)
106             },
107             modifier = Modifier
108                 .fillMaxWidth()
109                 .semantics { contentDescription = "
110                 searchButton" }
111         ) {
112             Text("Search")
113         }
114
115         // Clear button
116         Button(
117             onClick = {
118                 englishText = ""
119                 vietnameseText = ""
120                 exactEnglish = false
121                 exactVietnamese = false
122                 changeMessage("Search criteria cleared")
123             },
124             modifier = Modifier
125                 .fillMaxWidth()
126                 .semantics { contentDescription = "
127                 clearButton" }
128         ) {
129             Text("Clear")
130         }
131     }
132 }

```

./app/src/main/java/com/example/menuannam/presentation/screens

4.3.6 SearchScreen.kt

```
1 package com.example.menuannam.presentation.screens
2
3 import androidx.compose.foundation.border
4 import androidx.compose.foundation.clickable
5 import androidx.compose.foundation.layout.Arrangement
6 import androidx.compose.foundation.layout.Column
7 import androidx.compose.foundation.layout.Row
8 import androidx.compose.foundation.layout.Spacer
9 import androidx.compose.foundation.layout.fillMaxWidth
10 import androidx.compose.foundation.layout.padding
11 import androidx.compose.foundation.layout.size
12 import androidx.compose.foundation.layout.width
13 import androidx.compose.foundation.lazy.LazyColumn
14 import androidx.compose.foundation.lazy.items
15 import androidx.compose.material3.Button
16 import androidx.compose.material3.ButtonDefaults
17 import androidx.compose.material3.MaterialTheme
18 import androidx.compose.material3.OutlinedTextField
19 import androidx.compose.material3.Text
20 import androidx.compose.runtime.Composable
21 import androidx.compose.runtime.LaunchedEffect
22 import androidx.compose.runtime.getValue
23 import androidx.compose.runtime.mutableStateOf
24 import androidx.compose.runtime.remember
25 import androidx.compose.runtime.rememberCoroutineScope
26 import androidx.compose.runtime.setValue
27 import androidx.compose.ui.Alignment
28 import androidx.compose.ui.Modifier
29 import androidx.compose.ui.graphics.Color
30 import androidx.compose.ui.semantics.contentDescription
31 import androidx.compose.ui.semantics.semantics
32 import androidx.compose.ui.unit.dp
33 import androidx.compose.ui.unit.sp
34 import kotlinx.coroutines.launch
35 import com.example.menuannam.data.database.FlashCardDao
36 import com.example.menuannam.data.entity.FlashCard
37
38 // FlashCardList component - LazyColumn displaying
39 // flashcards with edit/delete buttons
40 @Composable
41 fun FlashCardList(
42     flashCards: List<FlashCard>,
43     onEdit: (Int) -> Unit = {}, // Navigate to EditCardRoute
44     with card ID
45     onDelete: (FlashCard) -> Unit = {} // Delete card from
46     database
47 ) {
48     LazyColumn(
49         modifier = Modifier.padding(16.dp)
50     ) {
51         items(
52             items = flashCards,
53             key = { flashCard ->
54                 flashCard.uid
55             }
56         ) { flashCard ->
57             Row(
58                 modifier = Modifier
59                     .fillMaxWidth()
60                     .border(width = 1.dp, color = Color.
61                         LightGray)
62                     .padding(8.dp),
63                 verticalAlignment = Alignment.
64                     CenterVertically
65             ) {
66                 Column(
67                     modifier = Modifier
68                         .weight(1f)
69                         .padding(6.dp)
70                 ) {
71                     Text("${flashCard.englishCard ? : ""} = $
72                         {flashCard.vietnameseCard ? : ""}")
73                     Spacer(modifier = Modifier.width(8.dp))
74                     Row(
75                         horizontalArrangement = Arrangement.
76                             spacedBy(4.dp)
77                     ) {
78                         Button(
79                             onClick = { onEdit(flashCard.uid) },
80                             colors = ButtonDefaults.buttonColors
81                                 (containerColor = MaterialTheme.colorScheme.primary)
82                         ) {
83                             Text("Edit", fontSize = 12.sp)
84                         }
85                         Button(
86                             onClick = { onDelete(flashCard) },
87                             colors = ButtonDefaults.buttonColors
88                                 (containerColor = MaterialTheme.colorScheme.error)
89                         ) {
90                             Text("Delete", fontSize = 12.sp)
91                         }
92                     }
93                 }
94             }
95         }
96     }
97 }
```

```
79         onClick = { onDelete(flashCard) },
80         colors = ButtonDefaults.buttonColors
81         (containerColor = MaterialTheme.colorScheme.error)
82     ) {
83         Text("Delete", fontSize = 12.sp)
84     }
85 }
86
87 }
88
89 }
90
91 @Composable
92 fun SearchScreen(
93     changeMessage: (String) -> Unit = {},
94     flashCardDao: FlashCardDao,
95     onEdit: (Int) -> Unit = {},
96     englishText: String = "",
97     exactEnglish: Int = 0,
98     vietnameseText: String = "",
99     exactVietnamese: Int = 0
100 ) {
101     var filteredCards by remember { mutableStateOf<List<
102         FlashCard>>(emptyList()) }
103     var isLoading by remember { mutableStateOf(false) }
104     val coroutineScope = rememberCoroutineScope()
105
106     val performSearch = {
107         coroutineScope.launch {
108             try {
109                 isLoading = true
110                 filteredCards = flashCardDao.
111                     getFilteredFlashCards(englishText, exactEnglish,
112                         vietnameseText, exactVietnamese)
113                 changeMessage("Found ${filteredCards.size}
114                     cards")
115                 isLoading = false
116             } catch (e: Exception) {
117                 changeMessage("Error: ${e.message}")
118                 isLoading = false
119             }
120         }
121     }
122
123     LaunchedEffect(Unit) {
124         performSearch()
125     }
126
127     Column(
128         modifier = Modifier
129             .fillMaxWidth()
130             .padding(16.dp),
131         horizontalAlignment = Alignment.CenterHorizontally,
132         verticalArrangement = Arrangement.spacedBy(12.dp),
133     ) {
134         // Results display
135         if (isLoading) {
136             Text("Searching...")
137         } else if (filteredCards.isEmpty()) {
138             Text("No cards found. Try different search terms
139                 .")
140         } else {
141             FlashCardList(
142                 flashCards = filteredCards,
143                 onEdit = onEdit,
144                 onDelete = { cardToDelete ->
145                     coroutineScope.launch {
146                         try {
147                             flashCardDao.delete(cardToDelete)
148                             changeMessage("Card deleted
149                                 successfully!")
150                             performSearch()
151                         } catch (e: Exception) {
152                             changeMessage("Error deleting
153                                 card: ${e.message}")
154                         }
155                     }
156                 }
157             )
158         }
159     }
160 }
```

./app/src/main/java/com/example/menuannam/presentation/scre

4.3.7 EditScreen.kt

```
1 package com.example.menuannam.presentation.screens
2
3 import android.util.Base64
4 import androidx.compose.foundation.layout.Arrangement
5 import androidx.compose.foundation.layout.Column
6 import androidx.compose.foundation.layout.Spacer
7 import androidx.compose.foundation.layout.fillMaxWidth
8 import androidx.compose.foundation.layout.padding
9 import androidx.compose.foundation.layout.size
10 import androidx.compose.material3.Button
11 import androidx.compose.material3.ButtonDefaults
12 import androidx.compose.material3.MaterialTheme
13 import androidx.compose.material3.OutlinedTextField
14 import androidx.compose.material3.Text
15 import androidx.compose.runtime.Composable
16 import androidx.compose.runtime.LaunchedEffect
17 import androidx.compose.runtime.getValue
18 import androidx.compose.runtime.mutableStateOf
19 import androidx.compose.runtime.remember
20 import androidx.compose.runtime.rememberCoroutineScope
21 import androidx.compose.runtime.setValue
22 import androidx.compose.ui.Alignment
23 import androidx.compose.ui.Modifier
24 import androidx.compose.ui.platform.LocalContext
25 import androidx.compose.ui.semantics.contentDescription
26 import androidx.compose.ui.semantics.semantics
27 import androidx.compose.ui.unit.dp
28 import androidx.core.net.toUri
29 import androidx.media3.common.MediaItem
30 import androidx.media3.exoplayer.ExoPlayer
31 import com.example.menuannam.EMAIL
32 import com.example.menuannam.TOKEN
33 import com.example.menuannam.data.database.FlashCardDao
34 import com.example.menuannam.data.entity.FlashCard
35 import com.example.menuannam.data.network.AudioRequest
36 import com.example.menuannam.data.network.NetworkService
37 import com.example.menuannam.dataStore
38 import com.example.menuannam.saveAudioToInternalStorage
39 import com.example.menuannam.toMd5
40 import kotlinx.coroutines.flow.first
41 import kotlinx.coroutines.launch
42 import java.io.File
43
44 @Composable
45 fun EditScreen(
46     cardId: Int, // Flashcard ID to edit
47     flashCardDao: FlashCardDao, // Database access
48     networkService: NetworkService, // API for audio
49     generation
50     changeMessage: (String) -> Unit, // Status message
51     callback
52     onCardUpdated: () -> Unit // Navigate back after
53     successful update
54 ) {
55     val context = LocalContext.current
56     val appContext = context.applicationContext
57     val coroutineScope = rememberCoroutineScope()
58
59     var flashCard by remember { mutableStateOf<FlashCard?>(
60         null) }
61     var englishText by remember { mutableStateOf("") }
62     var vietnameseText by remember { mutableStateOf("") }
63     var audioExists by remember { mutableStateOf(false) }
64     var audioFilePath by remember { mutableStateOf("") }
65     var isLoading by remember { mutableStateOf(true) }
66
67     // Load flashcard and check audio existence on screen
68     load
69     LaunchedEffect(cardId) {
70         try {
71             val card = flashCardDao.getById(cardId)
72             if (card != null) {
73                 flashCard = card
74                 englishText = card.englishCard ?: ""
75                 vietnameseText = card.vietnameseCard ?: ""
76
77                 // Check if audio file exists for current
78                 Vietnamese word
79                 val vn = card.vietnameseCard ?: ""
80                 if (vn.isNotBlank()) {
81                     val fileName = "${vn.toMd5()}.mp3"
82                     val file = File(appContext.filesDir,
83                         fileName)
84                     audioExists = file.exists()
85                     audioFilePath = if (audioExists) file.
86                         absolutePath else ""
87                 }
88                 changeMessage("Please, edit the flashcard.")
89             } else {
90                 changeMessage("Flash card not found")
91             }
92         } catch (e: Exception) {
93             changeMessage("Error loading flash card: ${e.
94                 message}")
95         }
96     }
97
98     } finally {
99         isLoading = false
100     }
101 }
102
103 if (isLoading) {
104     Column(
105         modifier = Modifier
106             .fillMaxWidth()
107             .padding(16.dp),
108         horizontalAlignment = Alignment.
109             CenterHorizontally
110     ) {
111         Text("Loading...")
112     }
113 } else if (flashCard == null) {
114     Column(
115         modifier = Modifier
116             .fillMaxWidth()
117             .padding(16.dp),
118         horizontalAlignment = Alignment.
119             CenterHorizontally
120     ) {
121         Text(
122             text = "Flash card not found",
123             color = MaterialTheme.colorScheme.error
124         )
125     }
126 } else {
127     Column(
128         modifier = Modifier
129             .fillMaxWidth()
130             .padding(16.dp),
131         verticalArrangement = Arrangement.spacedBy(12.dp),
132         horizontalAlignment = Alignment.
133             CenterHorizontally
134     ) {
135         // English text field
136         OutlinedTextField(
137             value = englishText,
138             onValueChange = { englishText = it },
139             label = { Text("en") },
140             modifier = Modifier
141                 .fillMaxWidth()
142                 .semantics { contentDescription = "
143                     enTextField" }
144         )
145
146         // Vietnamese text field
147         OutlinedTextField(
148             value = vietnameseText,
149             onValueChange = { vietnameseText = it },
150             label = { Text("vn") },
151             modifier = Modifier
152                 .fillMaxWidth()
153                 .semantics { contentDescription = "
154                     vnTextField" }
155         )
156
157         // Audio file path display (if exists)
158         if (audioExists) {
159             OutlinedTextField(
160                 value = audioFilePath,
161                 onValueChange = {},
162                 label = { Text("audio") },
163                 readOnly = true,
164                 modifier = Modifier.fillMaxWidth(),
165                 singleLine = false,
166                 maxLines = 3
167             )
168         }
169
170         Spacer(modifier = Modifier.size(4.dp))
171
172         // Update flashcard button - saves changes to
173         database
174         Button(
175             onClick = {
176                 coroutineScope.launch {
177                     try {
178                         flashCardDao.update(
179                             id = cardId,
180                             english = englishText,
181                             vietnamese = vietnameseText
182                         )
183                         changeMessage("Flash card
184                             updated successfully")
185                         onCardUpdated()
186                     } catch (e: Exception) {
187                         changeMessage("Error updating
188                             flash card: ${e.message}")
189                     }
190                 }
191             }
192         ),
193     }
194 }
```

```

173         modifier = Modifier
174             .fillMaxWidth()
175             .semantics { contentDescription = "
updateButton" }
176     ) {
177         Text("Update flashcard")
178     }
179
180     // Conditional audio buttons based on file
181     existence
182     if (audioExists) { // If audio exists: "Clean
audio" and "Play audio"
183         // Clean audio button - deletes audio file
184         from storage
185         Button(
186             onClick = {
187                 coroutineScope.launch {
188                     try {
189                         val file = File(
190                             audioFilePath)
191                         if (file.exists() && file.
delete()) {
192                             audioExists = false
193                             audioFilePath = ""
194                             changeMessage("Audio
deleted successfully")
195                         } else {
196                             changeMessage("Failed to
delete audio")
197                         }
198                     } catch (e: Exception) {
199                         changeMessage("Error
deleting audio: ${e.message}")
200                     }
201                 },
202                 modifier = Modifier
203                     .fillMaxWidth()
204                     .semantics { contentDescription = "
cleanAudioButton" },
205                 colors = ButtonDefaults.buttonColors(
206                     containerColor = MaterialTheme.
colorScheme.error
207                 )
208             ) {
209                 Text("Clean audio")
210             }
211
212         // Play audio button
213         Button(
214             onClick = {
215                 coroutineScope.launch {
216                     try {
217                         changeMessage("Playing audio
...")
218                         val mediaItem = MediaItem.
fromUri(audioFilePath.toUri())
219                         val player = ExoPlayer.
Builder(appContext).build()
220                         player.addListener(object :
androidx.media3.common.Player.Listener {
221                             override fun
onPlaybackStateChanged(playbackState: Int) {
222                                 when (playbackState)
223                                 {
224                                     androidx.media3.
common.Player.STATE_BUFFERING ->
225                                     changeMessage("Buffering...")
226                                     androidx.media3.
common.Player.STATE_READY ->
227                                     changeMessage("Playing")
228                                     androidx.media3.
common.Player.STATE_ENDED -> {
229                                         player.
release()
230                                         changeMessage("Finished")
231                                     }
232                                     else -> {}
233                                 }
234                             })
235                             player.setMediaItem(
mediaItem)
236                             player.prepare()
237                             player.play()

```

```

236             } catch (e: Exception) {
237                 changeMessage("Error playing
audio: ${e.message}")
238             }
239         },
240         modifier = Modifier
241             .fillMaxWidth()
242             .semantics { contentDescription = "
playAudioButton" }
243     ) {
244         Text("Play audio")
245     }
246 } else { // If audio doesn't exist: "Generate
audio"
247     // Generate audio button - calls API to
synthesize audio
248     Button(
249         onClick = {
250             coroutineScope.launch {
251                 try { // Load token and email
from DataStore for audio API request
252                     val prefs = appContext.
dataStore.data.first()
253                     val email = prefs[EMAIL] ?:
""
254                     val token = prefs[TOKEN] ?:
""
255                     val word = vietnameseText.
trim()
256                     if (word.isBlank() || email.
isBlank() || token.isBlank()) { // Missing required
data
257                         changeMessage("Missing
data to request audio")
258                         return@launch
259                     }
260                     changeMessage("Generating
audio...")
261                     val resp = networkService.
generateAudio( // Send Vietnamese word + token to AWS
Lambda
262                     request = AudioRequest(
263                         word, email, token)
264                 )
265                 if (resp.code == 200) { //
Response code 200 = success
266                     val audioBytes = Base64.
decode(resp.message, Base64.DEFAULT) // Decode Base64-
encoded MP3
267                     val fileName = "${word.
toMd5()}.mp3"
268                     val file =
saveAudioToInternalStorage(appContext, audioBytes,
fileName)
269                     audioExists = true
270                     audioFilePath = file.
absolutePath
271                     changeMessage("Audio
generated successfully")
272                 } else { // 500 = token
invalid/expired
273                     changeMessage("Audio
generation failed (${resp.code}): ${resp.message}")
274                 } catch (e: Exception) {
275                     changeMessage("Error
generating audio: ${e.message}")
276                 }
277             },
278             modifier = Modifier
279                 .fillMaxWidth()
280                 .semantics { contentDescription = "
generateAudioButton" }
281         ) {
282             Text("Generate audio")
283         }
284     }
285 }

```

./app/src/main/java/com/example/menuannam/presentation/screen

4.3.8 StudyScreen.kt

```
1 package com.example.menuannam.presentation.screens
2
3 import androidx.compose.foundation.clickable
4 import androidx.compose.foundation.layout.Arrangement
5 import androidx.compose.foundation.layout.Box
6 import androidx.compose.foundation.layout.Column
7 import androidx.compose.foundation.layout.Row
8 import androidx.compose.foundation.layout.Spacer
9 import androidx.compose.foundation.layout.fillMaxSize
10 import androidx.compose.foundation.layout.fillMaxWidth
11 import androidx.compose.foundation.layout.padding
12 import androidx.compose.foundation.layout.size
13 import androidx.compose.material3.Button
14 import androidx.compose.material3.Card
15 import androidx.compose.material3.CardDefaults
16 import androidx.compose.material3.MaterialTheme
17 import androidx.compose.material3.Text
18 import androidx.compose.runtime.Composable
19 import androidx.compose.runtime.LaunchedEffect
20 import androidx.compose.runtime.getValue
21 import androidx.compose.runtime.mutableIntStateOf
22 import androidx.compose.runtime.mutableStateOf
23 import androidx.compose.runtime.remember
24 import androidx.compose.runtime.rememberCoroutineScope
25 import androidx.compose.runtime.setValue
26 import androidx.compose.ui.Alignment
27 import androidx.compose.ui.Modifier
28 import androidx.compose.ui.platform.LocalContext
29 import androidx.compose.ui.semantics.contentDescription
30 import androidx.compose.ui.semantics.semantics
31 import androidx.compose.ui.text.font.FontWeight
32 import androidx.compose.ui.text.style.TextAlign
33 import androidx.compose.ui.unit.dp
34 import androidx.compose.ui.unit.sp
35 import androidx.core.net.toUri
36 import android.util.Base64
37 import java.io.File
38 import androidx.media3.common.MediaItem
39 import androidx.media3.exoplayer.ExoPlayer
40 import kotlinx.coroutines.CoroutineScope
41 import kotlinx.coroutines.flow.first
42 import kotlinx.coroutines.launch
43 import com.example.menuannam.EMAIL
44 import com.example.menuannam.TOKEN
45 import com.example.menuannam.dataStore
46 import com.example.menuannam.toMd5
47 import com.example.menuannam.saveAudioToInternalStorage
48 import com.example.menuannam.getCachedAudioFile
49 import com.example.menuannam.data.database.FlashCardDao
50 import com.example.menuannam.data.entity.FlashCard
51 import com.example.menuannam.data.network.AudioRequest
52 import com.example.menuannam.data.network.NetworkService
53
54 // StudyScreen supports two modes: SINGLE_CARD (view one
55 // card) and STUDY_SESSION (interactive 3-card learning)
56 enum class CardViewMode {
57     SINGLE_CARD, // View one specific card with delete
58     // button (from search)
59     STUDY_SESSION // Interactive 3-card learning with flip
60     and audio
61 }
62
63 @Composable
64 fun StudyScreen(
65     changeMessage: (String) -> Unit = {}, // Updates status
66     bar with feedback
67     flashCardDao: FlashCardDao,
68     networkService: NetworkService,
69     mode: CardViewMode = CardViewMode.STUDY_SESSION,
70     cardId: Int = 0, // Card ID for SINGLE_CARD mode
71     onCardDeleted: () -> Unit = {}, // Navigate back after
72     deletion
73     coroutineScope: CoroutineScope? = null
74 ) {
75     val context = LocalContext.current
76     val appContext = context.applicationContext
77     val scope = coroutineScope ?: rememberCoroutineScope()
78
79     var flashCard by remember { mutableStateOf<FlashCard?>(
80         null) }
81     var lesson by remember { mutableStateOf<List<FlashCard
82         >>(emptyList()) }
83     var isLoading by remember { mutableStateOf(true) }
84     var errorMessage by remember { mutableStateOf<String?>(
85         null) }
86     var currentIndex by remember { mutableIntStateOf(0) }
87     var showVietnamese by remember { mutableStateOf(false) }
88
89     // Load data based on mode: SINGLE_CARD loads one card,
90     // STUDY_SESSION loads 5 random cards
91     LaunchedEffect(Unit) {
92         scope.launch {
93             try {
94                 isLoading = true
95                 when (mode) {
96                     CardViewMode.SINGLE_CARD -> {
97                         flashCard = flashCardDao.getById(
98                             cardId)
99                         if (flashCard == null) {
100                             errorMessage = "Flash card not
101                                 found"
102                             changeMessage("Flash card not
103                                 found")
104                         } else {
105                             changeMessage("Viewing flash
106                                 card details")
107                         }
108                     }
109                     CardViewMode.STUDY_SESSION -> {
110                         val allCards = flashCardDao.getAll()
111                         lesson = if (allCards.size >= 3) {
112                             allCards.shuffled().take(3)
113                         } else {
114                             allCards
115                         }
116                         if (lesson.isEmpty()) {
117                             changeMessage("Card ${
118                                 currentIndex + 1} of ${lesson.size}")
119                         } else {
120                             errorMessage = "No flashcards
121                                 found"
122                             changeMessage("No flashcards
123                                 found")
124                         }
125                     }
126                 }
127             } catch (e: Exception) {
128                 errorMessage = "Error loading data: ${e.
129                     message}"
130                 changeMessage("Error loading data")
131             } finally {
132                 isLoading = false
133             }
134         }
135     }
136
137     // Play audio utility
138     val playAudio: (String) -> Unit = { word ->
139         scope.launch {
140             try {
141                 if (word.isBlank()) {
142                     changeMessage("Missing data to request
143                         audio")
144                     return@launch
145                 }
146
147                 // Try cached audio first
148                 val cachedFile = getCachedAudioFile(
149                     appContext, word)
150                 if (cachedFile != null) {
151                     val mediaItem = MediaItem.fromUri(
152                         cachedFile.absolutePath.toUri())
153                     val player = ExoPlayer.Builder(
154                         appContext).build()
155                     player.addListener(object : androidx.
156                         media3.common.Player.Listener {
157                         override fun onPlaybackStateChanged(
158                             playbackState: Int) {
159                             when (playbackState) {
160                                 Player.STATE_BUFFERING -> changeMessage("Buffering...")
161                                 Player.STATE_READY -> changeMessage("Playing...")
162                                 Player.STATE_ENDED -> {
163                                     player.release()
164                                     changeMessage("Ready")
165                                 }
166                             }
167                         }
168                     })
169                     player.setMediaItem(mediaItem)
170                     player.prepare()
171                     player.play()
172                     return@launch
173                 }
174
175                 // If no cache, call API to generate audio
176                 val prefs = appContext.dataStore.data.first
177                 (
178                     val email = prefs[EMAIL] ?: ""
179                     val token = prefs[TOKEN] ?: ""
180                     if (email.isBlank() || token.isBlank()) { //
181                         Missing authentication data
182                         changeMessage("Missing data to request
183                             audio")
184                         return@launch
185                     }
186                 )
187             }
188         }
189     }
```

```

162         changeMessage("Generating audio...")
163         val resp = networkService.generateAudio( //
164             Send Vietnamese word + token to AWS Lambda
165             token)
166         request = AudioRequest(word, email,
167             token)
168         if (resp.code == 200) { // Response code 200
169             = success
170             val audioBytes = Base64.decode(resp.
171             message, Base64.DEFAULT) // Decode Base64-encoded MP3
172             val fileName = "${word.toMd5()}.mp3"
173             val file = saveAudioToInternalStorage(
174             appContext, audioBytes, fileName) // Save for future
175             use
176             val mediaItem = MediaItem.fromUri(file.
177             absolutePath.toUri())
178             val player = ExoPlayer.Builder(
179             appContext).build()
180             player.addListener(object : androidx.
181             media3.common.Player.Listener { // ExoPlayer state
182             callbacks
183                 override fun onPlaybackStateChanged(
184                 playbackState: Int) {
185                     when (playbackState) {
186                         androidx.media3.common.
187                         Player.STATE_BUFFERING -> changeMessage("Buffering...")
188                         androidx.media3.common.
189                         Player.STATE_READY -> changeMessage("Playing...")
190                         androidx.media3.common.
191                         Player.STATE_ENDED -> {
192                             player.release()
193                             changeMessage("Ready")
194                         }
195                         else -> {}
196                     }
197                 }
198             })
199             player.setMediaItem(mediaItem)
200             player.prepare()
201             player.play()
202             } else { // 500 = token invalid/expired
203                 changeMessage("Audio error (${resp.code
204                 }): ${resp.message}")
205             }
206             } catch (e: Exception) {
207                 changeMessage("Audio error: ${e.message}")
208             }
209         }
210     }
211     // UI Layout
212     Column(
213         modifier = Modifier
214             .fillMaxWidth()
215             .padding(16.dp),
216         horizontalAlignment = Alignment.CenterHorizontally,
217         verticalArrangement = Arrangement.spacedBy(16.dp)
218     ) {
219         if (isLoading) {
220             Text("Loading...")
221         } else if (errorMessage != null) {
222             Text(
223                 text = errorMessage!!,
224                 color = MaterialTheme.colorScheme.error
225             )
226         } else when (mode) {
227             CardViewMode.SINGLE_CARD -> {
228                 // Single Card View
229                 if (flashCard != null) {
230                     Card(
231                         modifier = Modifier.fillMaxWidth(),
232                         elevation = CardDefaults.
233                         cardElevation(defaultElevation = 4.dp)
234                     ) {
235                         Column(
236                             modifier = Modifier.padding(16.
237                             dp),
238                             horizontalAlignment = Alignment.
239                             CenterHorizontally,
240                             verticalArrangement =
241                             Arrangement.spacedBy(12.dp)
242                         ) {
243                             Text(
244                                 text = "Flash Card #${
245                                 flashCard!!.uid}",
246                                 style = MaterialTheme.
247                                 typography.headlineSmall,
248                                 fontWeight = FontWeight.Bold
249                             )
250                             Spacer(modifier = Modifier.size
251                             (8.dp))
252                             Text(
253                                 text = "English:",
254                                 style = MaterialTheme.
255                                 typography.labelLarge,

```

```

256             fontWeight = FontWeight.
257             Medium
258             )
259             Text(
260                 text = flashCard!!.
261                 style = MaterialTheme.
262                 typography.bodyLarge
263             )
264             Spacer(modifier = Modifier.size
265             (8.dp))
266             Text(
267                 text = "Vietnamese:",
268                 style = MaterialTheme.
269                 typography.labelLarge,
270                 fontWeight = FontWeight.
271                 Medium
272             )
273             Text(
274                 text = flashCard!!.
275                 style = MaterialTheme.
276                 typography.bodyLarge
277             )
278             }
279             Row(
280                 horizontalArrangement = Arrangement.
281                 Center,
282                 modifier = Modifier.fillMaxWidth()
283             ) {
284                 Button(
285                     onClick = {
286                         scope.launch {
287                             try {
288                                 flashCardDao.delete(
289                                     flashCard!!)
290                                 changeMessage("Card
291                                 deleted successfully!")
292                                 onCardDeleted()
293                                 } catch (e: Exception) {
294                                     changeMessage("Error
295                                     : ${e.message}")
296                                 }
297                             }
298                         ) {
299                             Text("Delete")
300                         }
301                         Spacer(modifier = Modifier.size(8.dp)
302                         ))
303                 Button(
304                     onClick = {
305                         playAudio(flashCard!!)
306                     }
307                 ) {
308                     Text("Play Audio")
309                 }
310             }
311         }
312     }
313     CardViewMode.STUDY_SESSION -> {
314         // Study Session View
315         if (lesson.isNotEmpty()) {
316             val currentCard = lesson[currentIndex]
317             Text(
318                 text = "Card ${currentIndex + 1} of
319                 ${lesson.size}",
320                 style = MaterialTheme.typography.
321                 titleMedium
322             )
323             Card(
324                 modifier = Modifier
325                     .fillMaxWidth()
326                     .weight(if)
327                     .clickable {
328                         showVietnamese = !
329                     },
330                 elevation = CardDefaults.
331                 cardElevation(defaultElevation = 8.dp)
332             ) {
333                 Box(
334                     modifier = Modifier.fillMaxSize
335                     (),
336                     contentAlignment = Alignment.
337                     Center
338                 ) {
339                     Text(
340                         text = if (showVietnamese) {
341                             currentCard.

```

```

315     vietnameseCard ?: ""
316     } else {
317         currentCard.englishCard
318     },
319     fontSize = 32.sp,
320     textAlign = TextAlign.Center
321     ),
322     ),
323     ),
324     ),
325     if (showVietnamese) {
326         Button(
327             modifier = Modifier
328                 .fillMaxWidth()
329                 .semantics {
330                     contentDescription = "PlayAudio" },
331             onClick = {
332                 playAudio(currentCard.
333                 vietnameseCard ?: "")
334             }
335         ),
336         Text("Play Audio")
337     }
338 }
339 }
340 }
341 }
342 }
343 }
344 }
345 }
346 }
347 }
348 }
349 }
350 }
351 }
352 }
353 }
354 }
355 }
356 }
357 }
358 }

```

```

337     if (showVietnamese) {
338         Button(
339             modifier = Modifier.fillMaxWidth
340             ),
341             onClick = {
342                 val nextIndex = (
343                     currentIndex + 1) % lesson.size
344                 if (nextIndex == 0) {
345                     lesson = lesson.shuffled
346                 }
347                 currentIndex = nextIndex
348                 showVietnamese = false
349                 changeMessage("Card ${
350                     currentIndex + 1} of ${lesson.size}")
351             }
352         ) {
353             Text("Next")
354         }
355     }
356 }
357 }
358 }

```

./app/src/main/java/com/example/menuanam/presentation/screen

Chapter 5

UI Theme

5.1 Color.kt

```
1 package com.example.menuannam.ui.theme
2
3 import androidx.compose.ui.graphics.Color
4
5 val Purple80 = Color(0xFFD0BCFF)
6 val PurpleGrey80 = Color(0xFFCCC2DC)
7 val Pink80 = Color(0xFFE8B8C8)
```

```
8
9 val Purple40 = Color(0xFF6650a4)
10 val PurpleGrey40 = Color(0xFF625b71)
11 val Pink40 = Color(0xFF7D5260)
```

./app/src/main/java/com/example/menuannam/ui/theme/Color.k

5.2 Theme.kt

```
1 package com.example.menuannam.ui.theme
2
3 import android.app.Activity
4 import android.os.Build
5 import androidx.compose.foundation.isSystemInDarkTheme
6 import androidx.compose.material3.MaterialTheme
7 import androidx.compose.material3.darkColorScheme
8 import androidx.compose.material3.dynamicDarkColorScheme
9 import androidx.compose.material3.dynamicLightColorScheme
10 import androidx.compose.material3.lightColorScheme
11 import androidx.compose.runtime.Composable
12 import androidx.compose.ui.platform.LocalContext
13
14 private val DarkColorScheme = darkColorScheme(
15     primary = Purple80,
16     secondary = PurpleGrey80,
17     tertiary = Pink80
18 )
19
20 private val LightColorScheme = lightColorScheme(
21     primary = Purple40,
22     secondary = PurpleGrey40,
23     tertiary = Pink40
24
25     /* Other default colors to override
26     background = Color(0xFFFFFBFE),
27     surface = Color(0xFFFFFBFE),
28     onPrimary = Color.White,
29     onSecondary = Color.White,
30     onTertiary = Color.White,
31     onBackground = Color(0xFF1C1B1F),
```

```
32     onSurface = Color(0xFF1C1B1F),
33     */
34 )
35
36 @Composable
37 fun MenuAnNamTheme(
38     darkTheme: Boolean = isSystemInDarkTheme(),
39     // Dynamic color is available on Android 12+
40     dynamicColor: Boolean = true,
41     content: @Composable () -> Unit
42 ) {
43     val colorScheme = when {
44         dynamicColor && Build.VERSION.SDK_INT >= Build.
45             VERSION_CODES.S -> {
46                 val context = LocalContext.current
47                 if (darkTheme) dynamicDarkColorScheme(context)
48                 else dynamicLightColorScheme(context)
49             }
50         darkTheme -> DarkColorScheme
51         else -> LightColorScheme
52     }
53
54     MaterialTheme(
55         colorScheme = colorScheme,
56         typography = Typography,
57         content = content
58 )
```

./app/src/main/java/com/example/menuannam/ui/theme/Theme.k

5.3 Type.kt

```
1 package com.example.menuannam.ui.theme
2
3 import androidx.compose.material3.Typography
4 import androidx.compose.ui.text.TextStyle
5 import androidx.compose.ui.text.font.FontFamily
6 import androidx.compose.ui.text.font.FontWeight
7 import androidx.compose.ui.unit.sp
8
9 // Set of Material typography styles to start with
10 val Typography = Typography(
11     bodyLarge = TextStyle(
12         fontFamily = FontFamily.Default,
13         fontWeight = FontWeight.Normal,
14         fontSize = 16.sp,
15         lineHeight = 24.sp,
16         letterSpacing = 0.5.sp
17     )
18
19     /* Other default text styles to override
```

```
19     titleLarge = TextStyle(
20         fontFamily = FontFamily.Default,
21         fontWeight = FontWeight.Normal,
22         fontSize = 22.sp,
23         lineHeight = 28.sp,
24         letterSpacing = 0.sp
25     ),
26     labelSmall = TextStyle(
27         fontFamily = FontFamily.Default,
28         fontWeight = FontWeight.Medium,
29         fontSize = 11.sp,
30         lineHeight = 16.sp,
31         letterSpacing = 0.5.sp
32     )
33     */
34 )
```

./app/src/main/java/com/example/menuannam/ui/theme/Type.k

Chapter 6

Build Configuration

6.1 build.gradle.kts (App Module)

```
1 plugins {
2     alias(libs.plugins.android.application)
3     alias(libs.plugins.kotlin.android)
4     alias(libs.plugins.kotlin.compose)
5     id("org.jetbrains.kotlin.plugin.serialization")
6     id("com.google.devtools.ksp")
7 }
8
9 android {
10     namespace = "com.example.menuannam"
11     compileSdk = 36
12
13     defaultConfig {
14         applicationId = "com.example.menuannam"
15         minSdk = 26 // originally 24
16         targetSdk = 36
17         versionCode = 27
18         versionName = "1.0"
19         //testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
20     }
21
22     // For Kotlin projects using KSP:
23     ksp {
24         arg("room.schemaLocation", "$projectDir/schemas")
25     }
26
27     buildTypes {
28         release {
29             isMinifyEnabled = false
30             isDebuggable = false
31             proguardFiles(
32                 getDefaultProguardFile("proguard-android-optimize.txt"),
33                 "proguard-rules.pro"
34             )
35         }
36     }
37
38     compileOptions {
39         sourceCompatibility = JavaVersion.VERSION_21
40         targetCompatibility = JavaVersion.VERSION_21
41     }
42
43     kotlin {
44         compilerOptions {
45             jvmTarget.set(org.jetbrains.kotlin.gradle.dsl.JvmTarget.JVM_21)
46         }
47     }
48
49     buildFeatures {
50         compose = true
51     }
52
53     androidResources {
54         generateLocaleConfig = true
55     }
56
57     testOptions {
58         unitTests {
59             isIncludeAndroidResources = true
60         }
61     }
62 }
63
64 dependencies {
65     implementation(libs.androidx.core.ktx)
66     implementation(libs.androidx.lifecycle.runtime.ktx)
67     implementation(libs.androidx.activity.compose)
68     implementation(platform(libs.androidx.compose.bom))
69     implementation(libs.androidx.material3)
70     implementation(libs.androidx.navigation.runtime.ktx)
71     implementation(libs.androidx.navigation.compose)
72     implementation(libs.androidx.navigation.testing)
```

```

73     implementation(libs.core.ktx)
74     implementation(libs.androidx.compose.ui.test.junit4)
75
76     implementation(libs.androidx.compose.ui)
77     implementation(libs.androidx.compose.foundation)
78     implementation(libs.androidx.compose.foundation.layout)
79     implementation(libs.androidx.media3.exoplayer)
80     //implementation(libs.androidx.compose.ui.test)
81     testImplementation(libs.junit)
82     // For local unit tests
83     //testImplementation(libs.androidx.core.testing)
84     testImplementation(libs.robolectric)
85     // Needed for createComposeRule(), but not for createAndroidComposeRule<YourActivity>():
86     debugImplementation(libs.androidx.compose.ui.test.manifest)
87
88     //androidTestImplementation(libs.androidx.junit)
89     //androidTestImplementation(libs.androidx.espresso.core)
90     androidTestImplementation(platform(libs.androidx.compose.bom))
91     // Test rules and transitive dependencies:
92     androidTestImplementation(libs.androidx.compose.ui.test.junit4)
93
94     /* Dependencies related to room */
95     // room-compiler: is for the code generation that happens
96     // during the build process to create the necessary database
97     // infrastructure based on your annotations.
98     // implementation(libs.androidx.room.compiler)
99     // room-runtime: is for the code that runs on
100    // your device to interact with the database.
101    implementation(libs.androidx.room.runtime)
102    // If this project uses any Kotlin source, use Kotlin Symbol Processing (KSP)
103    // See Add the KSP plugin to your project
104    ksp(libs.androidx.room.compiler)
105    // optional - Kotlin Extensions and Coroutines support for Room
106    implementation(libs.androidx.room.ktx)
107    // optional - Test helpers
108    testImplementation(libs.androidx.room.testing)
109
110    // serialization
111    implementation(libs.kotlinx.serialization.json)
112    implementation(libs.androidx.navigation.compose.v280alpha08)
113
114    // retrofit
115    implementation("com.squareup.retrofit2:retrofit:2.9.0")
116    implementation("com.squareup.retrofit2:converter-gson:2.9.0")
117
118    // datastore
119    implementation("androidx.datastore:datastore-preferences:1.2.0")
120    implementation("androidx.datastore:datastore-preferences-core:1.2.0")
121 }

```

./app/build.gradle.kts

6.2 libs.versions.toml

```

1 [versions]
2
3 agp = "8.12.0"
4
5 composeVersion = "1.9.4"
6
7 kotlin = "2.1.10"
8
9 coreKtx = "1.17.0"
10
11 junit = "4.13.2"
12
13 #junitVersion = "1.3.0"
14
15 #espressoCore = "3.7.0"
16
17 lifecycleRuntimeKtx = "2.9.4"
18
19 activityCompose = "1.11.0"
20
21 composeBom = "2025.10.01"
22
23 navigationCompose = "2.9.5"
24
25 material3 = "1.4.0"
26
27 navigationRuntimeKtx = "2.9.5"
28
29 navigationTesting = "2.9.5"
30
31 robolectric = "4.16"
32
33 roomVersion = "2.8.3"
34
35 coreKtxVersion = "1.7.0"
36
37 ui = "1.9.4"
38
39 composeFoundation = "1.9.4"
40
41 media3Version = "1.4.1"
42
43 kotlinxSerializationVersion = "1.7.3"

```

```

44
45
46
47
48 [libraries]
49
50 androidx-compose-ui-test-junit4 = { module = "androidx.compose.ui:ui-test-junit4", version.ref = "composeVersion" }
51
52 androidx-compose-ui-test-manifest = { module = "androidx.compose.ui:ui-test-manifest", version.ref = "composeVersion" }
53
54 androidx-core-ktx = { group = "androidx.core", name = "core-ktx", version.ref = "coreKtx" }
55
56 androidx-room-compiler = { module = "androidx.room:room-compiler", version.ref = "roomVersion" }
57
58 androidx-room-ktx = { module = "androidx.room:room-ktx", version.ref = "roomVersion" }
59
60 androidx-room-runtime = { module = "androidx.room:room-runtime", version.ref = "roomVersion" }
61
62 androidx-room-testing = { module = "androidx.room:room-testing", version.ref = "roomVersion" }
63
64 junit = { group = "junit", name = "junit", version.ref = "junit" }
65
66 #androidx-junit = { group = "androidx.test.ext", name = "junit", version.ref = "junitVersion" }
67
68 #androidx-espresso-core = { group = "androidx.test.espresso", name = "espresso-core", version.ref = "espressoCore" }
69
70 androidx-lifecycle-runtime-ktx = { group = "androidx.lifecycle", name = "lifecycle-runtime-ktx", version.ref = "
    lifecycleRuntimeKtx" }
71
72 androidx-activity-compose = { group = "androidx.activity", name = "activity-compose", version.ref = "activityCompose" }
73
74 androidx-compose-bom = { group = "androidx.compose", name = "compose-bom", version.ref = "composeBom" }
75
76 androidx-navigation-compose = { group = "androidx.navigation", name = "navigation-compose", version.ref = "navigationCompose"
    }
77
78 androidx-material3 = { group = "androidx.compose.material3", name = "material3", version.ref = "material3" }
79
80 androidx-navigation-runtime-ktx = { group = "androidx.navigation", name = "navigation-runtime-ktx", version.ref = "
    navigationRuntimeKtx" }
81
82 androidx-navigation-testing = { group = "androidx.navigation", name = "navigation-testing", version.ref = "navigationTesting"
    }
83
84 roboelectric = { module = "org.robolectric:roboelectric", version.ref = "roboelectric" }
85
86 core-ktx = { group = "androidx.test", name = "core-ktx", version.ref = "coreKtxVersion" }
87
88 androidx-compose-ui = { group = "androidx.compose.ui", name = "ui", version.ref = "ui" }
89
90 androidx-compose-foundation = { group = "androidx.compose.foundation", name = "foundation", version.ref = "composeFoundation"
    }
91
92 androidx-compose-foundation-layout = { group = "androidx.compose.foundation", name = "foundation-layout", version.ref = "
    composeFoundation" }
93
94 androidx-media3-exoplayer = { group = "androidx.media3", name = "media3-exoplayer", version.ref = "media3Version" }
95
96 kotlin-serialization-json = { group = "org.jetbrains.kotlin", name = "kotlin-serialization-json", version.ref = "
    kotlinSerializationVersion" }
97
98 androidx-navigation-compose-v280alpha08 = { group = "androidx.navigation", name = "navigation-compose", version = "2.8.0-
    alpha08" }
99
100
101
102 [plugins]
103
104 android-application = { id = "com.android.application", version.ref = "agp" }
105
106 kotlin-android = { id = "org.jetbrains.kotlin.android", version.ref = "kotlin" }
107
108 kotlin-compose = { id = "org.jetbrains.kotlin.plugin.compose", version.ref = "kotlin" }

```

./gradle/libs.versions.toml

Chapter 7

Mobile Application Use Cases & Implementation

7.1 Use Cases Implementation Status

7.1.1 1. Request an Authentication Token

✓ **Status:** Implemented

- **Requirement:** The user can request a token (for authentication) by providing his/her email address. The token will be sent to the user's provided email address.
- **Implementation:**
 - **Screen:** LoginScreen.kt
 - **Lambda API:** Token Generation Endpoint
 - **Flow:** User enters email → API call to Lambda → Token sent to email → User receives token
- **Code Location:** LoginScreen.kt; NetworkService.kt; DataTypes.kt

7.1.2 2. Save the Pair Email Address/Authentication Token in the App

✓ **Status:** Implemented

- **Requirement:** The user can save the authentication token and the corresponding email address in the app's preferences datastore.
- **Implementation:**
 - **Screen:** TokenScreen.kt
 - **Storage:** DataStore Preferences (EMAIL and TOKEN keys)
 - **Flow:** User enters token from email → Click save → Both email and token persisted in DataStore
- **Code Location:** TokenScreen.kt; MainActivity.kt (DataStore Setup)

7.1.3 3. Add a Flashcard

✓ **Status:** Implemented

- **Requirement:** The user can add a flashcard to the app's (Room) database by providing the corresponding English and Vietnamese words. The (Room) database should not store duplicated flashcards. If the user requests to add a flashcard that already exists in the (Room) database, the app will refuse to do so and will inform the user accordingly. After the user successfully adds a flashcard to the (Room) database, the user should be allowed to continue adding more flashcards until he/she decides not to do so (e.g., when the user clicks on a "back" button).
- **Implementation:**
 - **Screen:** AddScreen.kt
 - **Database:** FlashCard entity with unique index on (english_card, vietnamese_card)
 - **Strategy:** OnConflictStrategy.IGNORE prevents duplicates
 - **User Feedback:** App informs user if duplicate exists
 - **UI:** Clear button resets fields, Back button available for navigation
 - **Continue Adding:** User can add multiple flashcards without leaving the screen
- **Code Location:** AddScreen.kt; FlashCard.kt; FlashCardDao.kt (Insert Methods)

7.1.4 4. Search Flashcards

✓ **Status:** Implemented

- **Requirement:** The user can search the flashcards stored in the app's (Room) database using "filters". For each search, the user can specify one of the following filters:
 - a) The flashcard's English word should match exactly the English word provided by the user and the flashcard's Vietnamese word should match exactly the Vietnamese word provided by the user.
 - b) The flashcard's English word should match exactly the English word provided by the user and the flashcard's Vietnamese word should contain the Vietnamese word provided by the user.
 - c) The flashcard's Vietnamese word should match exactly the Vietnamese word provided by the user and the flashcard's English word should contain the English word provided by the user.
 - d) The flashcard's English word and the flashcard's Vietnamese word should contain, respectively, the English word and the Vietnamese word provided by the user.

When requested to do so (e.g., when the user clicks on a "search" button), the app will show in a table the flashcards that satisfy the "filter" specified by the user. Each row in the table will contain an "edit" button that allows the user to Edit the corresponding flashcard.

- **Implementation:**

- **Screens:** FilterScreen.kt (search form), SearchScreen.kt (results table)
- **Database Query:** CASE WHEN logic in FlashCardDao.getFilteredFlashCards()
- **UI:** Checkboxes for exact/partial match per field
- **Results Display:** LazyColumn table with Edit button per row

- **Code Location:** FilterScreen.kt; SearchScreen.kt; FlashCardDao.kt (Filter Query)

7.1.5 5. Edit a Flashcard

✓ **Status:** Implemented

- **Requirement:** The user can edit a flashcard. When editing a flashcard, the app will initially show the flashcard's English and Vietnamese words as they are stored in the app's (Room) database, and the name of the file in the app's internal storage containing the audio corresponding to the pronunciation of the flashcard's Vietnamese word (if exists). Afterwards:

- The user can update the flashcard's English and Vietnamese words in the (Room) database by providing the new English and Vietnamese words.
- The user can delete the audio file shown in the editing-screen.
- If no audio file is shown in the editing-screen, if the user is authenticated, he/she can request the audio file corresponding to the pronunciation of the Vietnamese word shown in the editing-screen, and save it in the internal storage.
- If an audio file is shown in the editing screen, the user can play it on his/her device.

- **Implementation:**

- **Screen:** EditScreen.kt
- **Initial Display:** Shows English/Vietnamese words and audio filename (if exists)
- **Update Feature:** Update button saves changes to Room database
- **Delete Audio:** Clean audio button removes file from internal storage
- **Generate Audio:** Generate audio button (shown if no audio exists and user authenticated)
- **Play Audio:** Play audio button (shown if audio file exists)
- **Audio Storage:** Internal app storage with MD5 hash filenames
- **Audio Player:** ExoPlayer (Media3)
- **Lambda API:** Audio Synthesis Endpoint

- **Code Location:** EditScreen.kt; Utils.kt; FlashCardDao.kt (Update and GetById Methods); NetworkService.kt; DataTypes.kt

7.1.6 6. Study Flashcards

✓ **Status:** Implemented

- **Requirement:** The user can request the app to create a "lesson" for him/her to study. A lesson is a group of 3 flashcards randomly chosen from the app's (Room) database. When studying a lesson, the app will show first the flashcard's English word. Then, if the user asks to see the flashcard's Vietnamese word (e.g., by clicking on the flashcard's English word), the flashcard's Vietnamese word will be shown. When the Vietnamese word is shown:
 - The user can move to study the "next" flashcard in the lesson (in a loop).
 - The user can play the audio file corresponding to the pronunciation of the flashcard's Vietnamese word, if exists in the app's internal storage.
 - If the audio file corresponding to the pronunciation of the flashcard's Vietnamese word does not exist in the app's internal storage, if the user is authenticated, he/she can request the audio, and save it in his/her app's internal storage.
- **Implementation:**
 - **Screen:** StudyScreen.kt
 - **Lesson Size:** 3 flashcards randomly chosen from Room database
 - **Mode:** STUDY_SESSION enum value
 - **Initial Display:** Shows English word first
 - **Reveal Mechanism:** Tap card to reveal Vietnamese word
 - **Navigation:** Next button moves to next flashcard (loops back to first)
 - **Play Audio:** Play audio button (if audio exists in internal storage)
 - **Generate Audio:** Generate audio button (if authenticated and audio doesn't exist)
 - **Audio Caching:** Checks local cache first, then API fallback
 - **Audio Helpers:** Utils.kt provides 'toMd5', 'getCachedAudioFile', and 'saveAudioToInternalStorage' used by StudyScreen playback/generation
 - **Audio Playback:** ExoPlayer with state callbacks
- **Code Location:** StudyScreen.kt; FlashCardDao.kt (GetAll Method); Utils.kt; NetworkService.kt; DataTypes.kt