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://ityqwv3rx5vifjpyufgnpkv5te0ibrcx.lambda-url.ap-southeast-1.on.aws/`

Chapter 2

Application Entry Point

2.1 MainActivity.kt

```kotlin
package com.example.menuannam

// Main entry point: initializes database, network, navigation, and DataStore for app
import android.content.Context
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.activity.enableEdgeToEdge
import androidx.compose.runtime.rememberCoroutineScope
import androidx.datastore.preferences.core.stringPreferencesKey
import androidx.datastore.preferences.preferencesDataStore
import androidx.navigation.compose.rememberNavController
import com.example.menuannam.data.database.FlashCardDatabase
import com.example.menuannam.data.network.NetworkService
import com.example.menuannam.presentation.navigation.AppNavigation
import com.example.menuannam.ui.theme.MenuAnNamTheme
import okhttp3.OkHttpClient
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory
import java.util.concurrent.TimeUnit

// DataStore for persistent user credentials (email/token)
val Context.dataStore by preferencesDataStore(name = "user_credentials")
val TOKEN = stringPreferencesKey("token")
val EMAIL = stringPreferencesKey("email")

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        enableEdgeToEdge()
        setContent {
            MenuAnNamTheme {
                val navController = rememberNavController()
                val appContext = applicationContext
                val scope = rememberCoroutineScope()

                // Database singleton for flashcard CRUD operations
                val db = FlashCardDatabase.getDatabase(appContext)
                val flashCardDao = db.flashCardDao()

                // HTTP client with 30s timeouts to prevent Lambda timeout issues
                val sharedOkHttpClient = OkHttpClient.Builder()
                    .connectTimeout(30, TimeUnit.SECONDS)
                    .readTimeout(30, TimeUnit.SECONDS)
                    .build()

                // Retrofit for Lambda API calls (token generation, audio synthesis)
                val retrofit: Retrofit = Retrofit.Builder()
                    .baseUrl("https://placeholder.com") // URL overridden per endpoint
                    .client(sharedOkHttpClient)
                    .addConverterFactory(GsonConverterFactory.create())
                    .build()

                val networkService = retrofit.create(NetworkService::class.java)

                // Pass all services to navigation system
                AppNavigation(
                    navController,
                    flashCardDao,
                    scope,
                    networkService
                )
            }
        }
    }
}
```

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

2.2 Utils.kt

```kotlin
package com.example.menuannam

import android.content.Context
import java.io.File
import java.io.FileOutputStream
import java.security.MessageDigest

// Convert string to MD5 hash for unique audio filenames (e.g., "hello" → "5d41402abc...")
fun String.toMd5(): String {
    val bytes = MessageDigest.getInstance("MD5").digest(this.toByteArray())
    return bytes.joinToString("") { "%02x".format(it) }
}

// Save Base64-decoded audio bytes to app's private storage as MP3
fun saveAudioToInternalStorage(context: Context, audioData: ByteArray, filename: String): File {
    val file = File(context.filesDir, filename) // Private app directory
    FileOutputStream(file).use { fos ->
        fos.write(audioData)
    }
    return file // Return File for ExoPlayer playback
}

// Check if audio file exists in cache before making API call
fun getCachedAudioFile(context: Context, word: String): File? {
    val file = File(context.filesDir, "${word.toMd5()}.mp3")
    return if (file.exists()) file else null
}
```

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

Chapter 3

Data Layer

3.1 Database

3.1.1 FlashCardDao.kt

```kotlin
package com.example.menuannam.data.database

import androidx.room.Dao
import androidx.room.Delete
import androidx.room.Insert
import androidx.room.OnConflictStrategy
import androidx.room.Query
import androidx.room.RawQuery
import androidx.sqlite.db.SupportSQLiteQuery
import com.example.menuannam.data.entity.FlashCard

// DAO for FlashCard CRUD operations; all suspend functions run on background thread
@Dao
interface FlashCardDao {
    // Raw SQL for database checkpoint operations
    @RawQuery
    fun checkpoint(supportSQLiteQuery: SupportSQLiteQuery): Int

    @Query("SELECT * FROM FlashCards")
    suspend fun getAll(): List<FlashCard>

    // Random shuffle for study sessions
    @Query("SELECT * FROM FlashCards ORDER BY RANDOM() LIMIT :size")
    suspend fun getLesson(size: Int): List<FlashCard>

    // Duplicate detection by matching both fields
    @Query(
        "SELECT * FROM FlashCards WHERE english_card LIKE :english AND " +
                "vietnamese_card LIKE :vietnamese LIMIT 1"
    )
    suspend fun findByCards(english: String, vietnamese: String): FlashCard?

    // IGNORE strategy: silently skip if duplicate exists (returns -1)
    @Insert(onConflict = OnConflictStrategy.IGNORE)
    suspend fun insert(flashCard: FlashCard): Long

    @Insert(onConflict = OnConflictStrategy.IGNORE)
    suspend fun insertAll(vararg flashCard: FlashCard)

    // Update by old English/Vietnamese pair (search key) to new values
    @Query(
        "UPDATE FlashCards SET english_card = :englishNew " +
                ", vietnamese_card =:vietnameseNew " +
                "WHERE english_card = :englishOld " +
                "AND vietnamese_card = :vietnameseOld"
    )
    suspend fun updateFlashCard(
        englishOld: String,
        vietnameseOld: String,
        englishNew: String,
        vietnameseNew: String
    )

    @Query(
        "DELETE FROM FlashCards WHERE english_card = :english " +
                "AND vietnamese_card =:vietnamese"
    )
    suspend fun deleteFlashCard(english: String, vietnamese: String)

    @Query("SELECT * FROM FlashCards WHERE uid = :id")
    suspend fun getById(id: Int): FlashCard?

    @Query("SELECT COUNT(*) FROM FlashCards")
    suspend fun getCount(): Int

    @Query("UPDATE FlashCards SET english_card = :english, vietnamese_card = :vietnamese WHERE uid = :id")
    suspend fun update(id: Int, english: String, vietnamese: String)

    @Delete
    suspend fun delete(flashCard: FlashCard)

    // CASE WHEN for exact vs partial match: exactEn=1 → LIKE :en, exactEn=0 → LIKE '%' || :en || '%'
    @Query(
        "SELECT * FROM FlashCards WHERE " +
                "(CASE WHEN :exactEn THEN english_card LIKE :en " +
                "WHEN NOT :exactEn THEN english_card LIKE '%' || :en || '%' END) " +
                "AND " +
                "(CASE WHEN :exactVn THEN vietnamese_card LIKE :vn " +
                "WHEN NOT :exactVn THEN vietnamese_card LIKE '%' || :vn || '%' END)"
    )
    suspend fun getFilteredFlashCards(en: String, exactEn: Int, vn: String, exactVn: Int): List<FlashCard>
}
```

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

3.1.2 MenuDatabase.kt

```kotlin
package com.example.menuannam.data.database

import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase
import com.example.menuannam.data.entity.FlashCard
import kotlin.jvm.Volatile

// Room database with singleton pattern for thread-safe FlashCard persistence
@Database(entities = [FlashCard::class], version = 1)
abstract class FlashCardDatabase : RoomDatabase() {
    abstract fun flashCardDao(): FlashCardDao

    companion object {
        @Volatile // All threads see latest value immediately
        private var INSTANCE: FlashCardDatabase? = null

        // Double-checked locking: fast path check, then synchronized creation to prevent race conditions
```

```
        fun getDatabase(context: Context): FlashCardDatabase {
            return INSTANCE ?: synchronized(this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext, // Use app context to prevent memory leaks
                    FlashCardDatabase::class.java,
                    "FlashCardDatabase"
                ).build()
                INSTANCE = instance
                instance
            }
        }
    }
}
```

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

3.2 Entity

3.2.1 FlashCard.kt

```
package com.example.menuannam.data.entity

import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.Index
import androidx.room.PrimaryKey
import kotlinx.serialization.Serializable

// FlashCard entity with unique index on (english_card, vietnamese_card) to prevent duplicates
@Entity(tableName = "FlashCards", indices = [Index(
    value = ["english_card", "vietnamese_card"],
    unique = true // Database-level constraint enforced by Room
)])
@Serializable
data class FlashCard(
    @PrimaryKey(autoGenerate = true) val uid: Int, // Auto-increment from 1
    @ColumnInfo(name = "english_card") val englishCard: String?,
    @ColumnInfo(name = "vietnamese_card") val vietnameseCard: String?
)
```

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

3.3 Network

3.3.1 DataTypes.kt

```
package com.example.menuannam.data.network

import kotlinx.serialization.Serializable

// Request payloads and responses for Lambda API calls

@Serializable
data class UserCredential(val email: String) // Token generation request

@Serializable
data class TokenResponse(
    val code: Int, // HTTP status (200 = success)
    val message: String // Token string or error message
)

@Serializable
data class AudioRequest(
    val word: String, // Vietnamese word for audio synthesis
    val email: String, // User tracking
    val token: String // Auth token (expires after some time)
)

@Serializable
data class AudioResponse(
    val code: Int, // HTTP status (200 = success, 500 = token invalid/expired)
    val message: String // Base64-encoded MP3 or error message
)
```

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

3.3.2 NetworkService.kt

```
package com.example.menuannam.data.network

import retrofit2.http.Body
import retrofit2.http.PUT
import retrofit2.http.Url

// Retrofit interface for AWS Lambda endpoints; default URLs overridden per call
interface NetworkService {
    // Token generation: sends email, returns token or error
    @PUT
    suspend fun generateToken(
        @Url url: String = "https://egsbwqh7kildllpkijk6nt4soq0wlgpe.lambda-url.ap-southeast-1.on.aws/",
        @Body email: UserCredential
    ): TokenResponse

    // Audio synthesis: sends Vietnamese word + token, returns Base64 MP3 or 500 if token expired
    @PUT
    suspend fun generateAudio(
        @Url url: String = "https://ityqwv3rx5vifjpyufgnpkv5te0ibrcx.lambda-url.ap-southeast-1.on.aws/",
        @Body request: AudioRequest
    ): AudioResponse
}
```

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

Chapter 4

Presentation Layer

4.1 Navigation

4.1.1 Navigator.kt

```kotlin
package com.example.menuannam.presentation.navigation

import androidx.compose.foundation.layout.padding
import androidx.compose.material3.Scaffold
import androidx.compose.runtime.*
import androidx.navigation.NavHostController
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
import androidx.navigation.compose.currentBackStackEntryAsState
import androidx.navigation.toRoute
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.launch
import com.example.menuannam.data.database.FlashCardDao
import com.example.menuannam.presentation.screens.MenuScreen
import com.example.menuannam.presentation.screens.AddScreen
import com.example.menuannam.presentation.screens.StudyScreen
import com.example.menuannam.presentation.screens.CardViewMode
import com.example.menuannam.presentation.screens.SearchScreen
import com.example.menuannam.presentation.screens.FilterScreen
import com.example.menuannam.presentation.screens.EditScreen
import com.example.menuannam.presentation.screens.LoginScreen
import com.example.menuannam.presentation.screens.TokenScreen
import com.example.menuannam.data.network.NetworkService
import com.example.menuannam.presentation.components.TopBarComponent
import com.example.menuannam.presentation.components.BottomBarComponent

// Central Navigation Hub: Controls all screen transitions and manages app state
// Uses Compose Navigation with type-safe routes for compile-time error checking
@Composable
fun AppNavigation(
    navigation: NavHostController, // NavController from MainActivity
    flashCardDao: FlashCardDao, // Database access
    coroutineScope: CoroutineScope, // For async operations (launched from MainActivity)
    networkService: NetworkService // Retrofit service for API calls
) {
    // Shared message across all screens - updated by each screen, displayed in BottomBar
    var message by remember { mutableStateOf("Welcome!") }
    val changeMessage: (String) -> Unit = { message = it }

    // Get current route name from back stack to determine what screen is currently displayed
    val backStackEntry by navigation.currentBackStackEntryAsState()
    val routeName = backStackEntry?.destination?.route

    // Map route names to user-friendly titles displayed in TopBar
    val title = when (routeName) {
        HomeRoute::class.qualifiedName -> "Menu An Nam"
        StudyCardsRoute::class.qualifiedName -> "Study Cards"
        AddCardRoute::class.qualifiedName -> "Add Card"
        SearchCardsRoute::class.qualifiedName -> "Search Results"
        FilterRoute::class.qualifiedName -> "Search Cards"
        LoginRoute::class.qualifiedName -> "Login"
        ShowCardRoute::class.qualifiedName -> "Flash Card"
        EditCardRoute::class.qualifiedName -> "Edit Card"
        else -> "Menu An Nam"
    }

    /**
     * Show back button on all screens EXCEPT home
     * Only home doesn't have a back button (it's the start destination)
     */
    val showBack = routeName != HomeRoute::class.qualifiedName
    val navigateBack: () -> Unit = { navigation.navigateUp() }

    // ===== SCAFFOLD LAYOUT =====
    /**
     * Scaffold provides standard app structure:
     * - topBar: Title and back button
     * - content: NavHost with screen composition
     * - bottomBar: Status message display
     */
    Scaffold(
        topBar = {
            TopBarComponent(
                title = title,
                showBack = if (showBack) navigateBack else null
            )
        },
        bottomBar = {
            BottomBarComponent(message = message)
        }
    ) { innerPadding ->

        // ===== NAVIGATION HOST =====
        /**
         * NavHost manages all composable screens
         * startDestination = HomeRoute (app always starts at home)
         * Padding prevents overlap with TopBar/BottomBar
         */
        NavHost(
            navController = navigation,
            startDestination = HomeRoute,
            modifier = androidx.compose.ui.Modifier.padding(innerPadding)
        ) {

            //
            ============================================================

            // HOME ROUTE - Main menu
            //
            ============================================================

            composable<HomeRoute> {
                MenuScreen(
                    onStudy = {
                        navigation.navigate(StudyCardsRoute)
                    },
                    onAdd = { navigation.navigate(AddCardRoute) },
                    onSearch = { navigation.navigate(FilterRoute) },
                    onLogin = { navigation.navigate(LoginRoute) },
```

```
                changeMessage = changeMessage
            )
        }

        //
==========================================================

        // LOGIN ROUTE - Get email and request token
        //
==========================================================

        composable<LoginRoute> {
            LoginScreen(
                changeMessage = changeMessage,
                networkService = networkService,
                navigateToToken = { enteredEmail ->
                    navigation.navigate(TokenRoute(
enteredEmail))
                }
            )
        }

        //
==========================================================

        // STUDY CARDS ROUTE - Interactive flashcard
study session
        //
==========================================================

        composable<StudyCardsRoute> {
            StudyScreen(
                changeMessage = changeMessage,
                flashCardDao = flashCardDao,
                networkService = networkService,
                mode = CardViewMode.STUDY_SESSION,
                coroutineScope = coroutineScope
            )
        }

        //
==========================================================

        // ADD CARD ROUTE - Create new flashcard
        //
==========================================================

        composable<AddCardRoute> {
            AddScreen(
                changeMessage = changeMessage,
                insertFlashCard = { card ->
                    coroutineScope.launch {
                        // Insert card (Room ignores if
duplicate)
                        flashCardDao.insertAll(card)
                        changeMessage("Added: ${card.
englishCard}")
                    }
                }
            )
        }

        //
==========================================================

        // SEARCH CARDS ROUTE - Display filtered search
results
        //
==========================================================

        composable<SearchCardsRoute> { backStackEntry ->
            val route = backStackEntry.toRoute<
SearchCardsRoute>()
            SearchScreen(
                changeMessage = changeMessage,
                flashCardDao = flashCardDao,
                onEdit = { cardId ->
                    navigation.navigate(EditCardRoute(
cardId))
                },
                englishText = route.englishText,
                exactEnglish = route.exactEnglish,
                vietnameseText = route.vietnameseText,
                exactVietnamese = route.exactVietnamese
            )
        }

        //
==========================================================

        // FILTER ROUTE - Search input form
        //
==========================================================

        composable<FilterRoute> {
            FilterScreen(
                changeMessage = changeMessage,
                onSearch = { en, exactEn, vn, exactVn ->
                    navigation.navigate(
                        SearchCardsRoute(
                            englishText = en,
                            exactEnglish = exactEn,
                            vietnameseText = vn,
                            exactVietnamese = exactVn
                        )
                    )
                }
            )
        }

        //
==========================================================

        // EDIT CARD ROUTE - Edit existing flashcard
with audio management
        //
==========================================================

        composable<EditCardRoute> { backStackEntry ->
            val route = backStackEntry.toRoute<
EditCardRoute>()
            EditScreen(
                cardId = route.id,
                flashCardDao = flashCardDao,
                networkService = networkService,
                changeMessage = changeMessage,
                onCardUpdated = { navigation.navigateUp
() }
            )
        }

        //
==========================================================

        // SHOW CARD ROUTE - View single card and delete
 if desired
        //
==========================================================

        /**
         * Receives cardId from SearchScreen
         * Displays full card details with delete and
play audio options
         */
        composable<ShowCardRoute> { backStackEntry ->
            val route = backStackEntry.toRoute<
ShowCardRoute>()

            StudyScreen(
                changeMessage = changeMessage,
                flashCardDao = flashCardDao,
                networkService = networkService,
                mode = CardViewMode.SINGLE_CARD,
                cardId = route.id,
                onCardDeleted = { navigation.navigateUp
() }
            )
        }

        //
==========================================================

        // TOKEN ROUTE - Store token from login
        //
==========================================================

        /**
         * Receives email parameter from LoginRoute
         * User enters token string received via email
         * Saves token to DataStore for later audio API
calls
         */
        composable<TokenRoute> { backStackEntry ->
            val route = backStackEntry.toRoute<
TokenRoute>()

            TokenScreen(
                email = route.email,
                changeMessage = changeMessage,
                navigateToHome = {
                    navigation.navigate(HomeRoute) {
                        popUpTo(HomeRoute) { inclusive =
 true }
                    }
                }
            )
        }
    }
}
```

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

4.1.2 Routes.kt

```kotlin
package com.example.menuannam.presentation.navigation

import kotlinx.serialization.Serializable

// TYPE-SAFE NAVIGATION ROUTES: @Serializable objects for compile-time verification
// Kotlinx.serialization handles automatic type conversion for route parameters

@Serializable
object HomeRoute // Main menu: Study, Add Cards, Search, Login

@Serializable
object AddCardRoute // Create new flashcard with English/Vietnamese input

@Serializable
object StudyCardsRoute // Study 5 random cards with audio playback

@Serializable
data class SearchCardsRoute( // Display filtered search results with edit/delete options
    val englishText: String = "", // English search term
    val exactEnglish: Int = 0, // 1 for exact match, 0 for partial match
    val vietnameseText: String = "", // Vietnamese search term
    val exactVietnamese: Int = 0 // 1 for exact match, 0 for partial match
)

@Serializable
object LoginRoute // User enters email to get token for audio synthesis

@Serializable
data class ShowCardRoute(val id: Int) // View single flashcard with delete and audio playback

@Serializable
data class EditCardRoute(val id: Int) // Edit existing flashcard by ID

@Serializable
data class TokenRoute(val email: String) // Token input screen - email passed from LoginRoute for context

@Serializable
object FilterRoute // Filter search screen with English/Vietnamese terms and exact/partial match options
```

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

4.2 Components

4.2.1 TopBarComponent.kt

```kotlin
package com.example.menuannam.presentation.components

import androidx.compose.ui.semantics.contentDescription
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.CenterAlignedTopAppBar
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.material3.TextButton
import androidx.compose.material3.TopAppBarDefaults
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.semantics.semantics

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun TopBarComponent(
    title: String, // Screen name to display (e.g., "Study Cards", "Add Card")
    showBack: (() -> Unit)? = null, // Optional callback to navigate up - if null, no back button shown
) {
    CenterAlignedTopAppBar(
        title = { Text(title) },
        navigationIcon = {
            if (showBack != null) { // Conditional back button: shows on all screens except HomeRoute
                TextButton(
                    onClick = showBack,
                    modifier = Modifier.semantics { // Enables testing framework to find elements
                        contentDescription = "navigateBack"
                    },
                    colors = ButtonDefaults.textButtonColors(
                        contentColor = MaterialTheme.colorScheme.onPrimary
                    )
                ) {
                    Text("Back")
                }
            }
        },
        colors = TopAppBarDefaults.topAppBarColors(
            containerColor = MaterialTheme.colorScheme.primary,
            titleContentColor = MaterialTheme.colorScheme.onPrimary,
            navigationIconContentColor = MaterialTheme.colorScheme.onPrimary,
            actionIconContentColor = MaterialTheme.colorScheme.onPrimary
        )
    )
}
```

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

4.2.2 BottomBarComponent.kt

```kotlin
package com.example.menuannam.presentation.components

import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.material3.BottomAppBar
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.semantics.contentDescription
import androidx.compose.ui.semantics.semantics
import androidx.compose.ui.text.style.TextAlign

@Composable
fun BottomBarComponent(
    message: String // Current status message (updated by Navigator's changeMessage callback)
) {
    BottomAppBar(){
        Text( // Shows real-time feedback: "Card 1 of 5", "Logged out", "Error loading flashcards: ..."
            modifier = Modifier
                .fillMaxWidth()
                .semantics { contentDescription = "Message" },
            textAlign = TextAlign.Center,
            text = message
        )
    }
}
```

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

4.3 Screens

4.3.1 MenuScreen.kt

```kotlin
package com.example.menuannam.presentation.screens

import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.Button
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.rememberCoroutineScope
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.semantics.contentDescription
import androidx.compose.ui.semantics.semantics
import androidx.compose.ui.unit.dp
import androidx.datastore.preferences.core.edit
import com.example.menuannam.EMAIL
import com.example.menuannam.TOKEN
import com.example.menuannam.dataStore
import kotlinx.coroutines.flow.first
import kotlinx.coroutines.launch

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun MenuScreen(
    onStudy: () -> Unit, // Navigate to StudyCardsRoute
    onAdd: () -> Unit, // Navigate to AddCardRoute
    onSearch: () -> Unit, // Navigate to SearchCardsRoute
    onLogin: () -> Unit, // Navigate to LoginRoute
    changeMessage: (String) -> Unit = {} // Update bottom status bar
) {
    val context = LocalContext.current
    val appContext = context.applicationContext
    val scope = rememberCoroutineScope()

    var email by remember { mutableStateOf("") }

    // Load saved email from DataStore on screen load - shows user they are logged in if email stored
    LaunchedEffect(Unit) {
        val prefs = appContext.dataStore.data.first()
        email = prefs[EMAIL] ?: ""
        changeMessage("Email loaded: $email")
    }

    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp),
        verticalArrangement = Arrangement.spacedBy(16.dp, Alignment.CenterVertically),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {

        Button(
            onClick = onStudy,
            modifier = Modifier.fillMaxWidth()
                .semantics { contentDescription = "navigateToStudyCards" }
        ) { Text("Study Cards") }

        Button(
            onClick = onAdd,
            modifier = Modifier.fillMaxWidth()
                .semantics { contentDescription = "navigateToAddCard" }
        ) { Text("Add Cards") }

        Button(
            onClick = onSearch,
            modifier = Modifier.fillMaxWidth()
                .semantics { contentDescription = "navigateToSearchCards" }
        ) { Text("Search Cards") }

        Button(
            onClick = onLogin,
            modifier = Modifier.fillMaxWidth()
                .semantics { contentDescription = "navigateToLoginScreen" }
        ) { Text("Login") }

        Button(
            onClick = {
                scope.launch {
                    appContext.dataStore.edit {
                        it.remove(EMAIL)
                        it.remove(TOKEN)
                    }
                    email = ""
                    changeMessage("Logged out")
                }
            },
            modifier = Modifier.fillMaxWidth()
                .semantics { contentDescription = "ExecuteLogout" }
        ) {
            Text("Log out", modifier = Modifier.semantics { contentDescription = "Logout" })
        }
    }
}
```

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

4.3.2 LoginScreen.kt

```kotlin
package com.example.menuannam.presentation.screens

import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.Button
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.rememberCoroutineScope
import androidx.compose.runtime.saveable.rememberSaveable
import androidx.compose.runtime.setValue
import androidx.compose.ui.Modifier
import androidx.compose.ui.semantics.contentDescription
import androidx.compose.ui.semantics.semantics
import androidx.compose.ui.unit.dp
import com.example.menuannam.data.network.UserCredential
import com.example.menuannam.data.network.NetworkService
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext

@Composable
fun LoginScreen(
    changeMessage: (String) -> Unit, // Updates status bar with feedback
    networkService: NetworkService, // Retrofit interface for Lambda API
    navigateToToken: (String) -> Unit // Callback to show TokenScreen with email
) {
    var email by rememberSaveable { mutableStateOf("") }
    val scope = rememberCoroutineScope()

    LaunchedEffect(Unit) {
        changeMessage("Please, introduce your email.")
    }

    Column(
        modifier = Modifier.padding(16.dp),
        verticalArrangement = Arrangement.spacedBy(16.dp)
    ) {
        OutlinedTextField(
            value = email,
            onValueChange = { email = it },
            modifier = Modifier
                .fillMaxWidth()
                .semantics { contentDescription = "emailTextField" },
            label = { Text("email") }
        )

        Button(
            modifier = Modifier
```

```
                .fillMaxWidth()
                .semantics { contentDescription = "Enter" },
            onClick = {
                scope.launch {
                    try { // Send email to AWS Lambda for token generation
                        val result = withContext(Dispatchers.IO) {
                            networkService.generateToken(email = UserCredential(email))
                        }
                        if (result.code == 200) { // Response code 200 means success
                            changeMessage("Token sent to email: ${result.message}")
                            navigateToToken(email)
                        } else {
                            changeMessage("Error: ${result.message}")
                        }

                    } catch (e: Exception) {
                        changeMessage("There was an error in the token request.")
                    }
                }
            }
        ) {
            Text("Enter")
        }
    }
}
```

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

4.3.3 TokenScreen.kt

```
package com.example.menuannam.presentation.screens

import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.Button
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.rememberCoroutineScope
import androidx.compose.runtime.setValue
import androidx.compose.ui.Modifier
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.semantics.contentDescription
import androidx.compose.ui.semantics.semantics
import androidx.compose.ui.unit.dp
import androidx.datastore.preferences.core.edit
import com.example.menuannam.EMAIL
import com.example.menuannam.TOKEN
import com.example.menuannam.dataStore
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext

@Composable
fun TokenScreen(
    email: String, // User email from LoginScreen, displayed as context
    changeMessage: (String) -> Unit, // Updates status bar with feedback
    navigateToHome: (String) -> Unit // Return to main menu after save
) {
    val scope = rememberCoroutineScope()
    val context = LocalContext.current
    val appContext = context.applicationContext
    var token by remember { mutableStateOf("") }

    LaunchedEffect(Unit) {
        changeMessage("Please, introduce your token.")
    }

    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp),
        verticalArrangement = Arrangement.spacedBy(8.dp)
    ) {
        OutlinedTextField(
            value = token,
            onValueChange = { token = it },
            modifier = Modifier
                .fillMaxWidth()
                .semantics { contentDescription = "tokenTextField" },
            label = { Text("token") }
        )
        OutlinedTextField(
            value = email,
            onValueChange = {},
            modifier = Modifier
                .fillMaxWidth()
                .semantics { contentDescription = "emailTextField" },
            label = { Text("email") },
            readOnly = true
        )
        Button(
            modifier = Modifier
                .fillMaxWidth()
                .semantics { contentDescription = "Enter" },
            onClick = {
                scope.launch {
                    withContext(Dispatchers.IO) { // Save token and email to DataStore for future audio requests
                        appContext.dataStore.edit { preferences ->
                            preferences[EMAIL] = email
                            preferences[TOKEN] = token
                        }
                    }
                    navigateToHome(token)
                }
            }
        ) {
            Text("Enter")
        }
    }
}
```

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

4.3.4 AddScreen.kt

```kotlin
package com.example.menuannam.presentation.screens

import androidx.compose.foundation.layout.*
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.runtime.saveable.rememberSaveable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.semantics.contentDescription
import androidx.compose.ui.semantics.editableText
import androidx.compose.ui.semantics.semantics
import androidx.compose.ui.semantics.text
import androidx.compose.ui.text.AnnotatedString
import androidx.compose.ui.unit.dp
import com.example.menuannam.data.entity.FlashCard

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun AddScreen(
    changeMessage: (String) -> Unit, // Updates status bar with feedback
    insertFlashCard: (FlashCard) -> Unit // Callback to save card (receives FlashCard)
) {
    var english by rememberSaveable { mutableStateOf("") } // TextFields persist state across recomposition
    var vietnamese by rememberSaveable { mutableStateOf("") }

    LaunchedEffect(Unit) {
        changeMessage("Add a new flashcard")
    }

    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp),
        verticalArrangement = Arrangement.spacedBy(16.dp, Alignment.CenterVertically),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {

        TextField(
            value = english,
            onValueChange = { english = it },
            label = { Text("English") },
            modifier = Modifier
                .fillMaxWidth()
                .semantics { contentDescription = "enTextField" }
        )

        TextField(
            value = vietnamese,
            onValueChange = { vietnamese = it },
            label = { Text("Vietnamese") },
            modifier = Modifier
                .fillMaxWidth()
                .semantics { contentDescription = "viTextField" }
        )

        Row(
            modifier = Modifier.fillMaxWidth(),
            horizontalArrangement = Arrangement.spacedBy(16.dp, Alignment.CenterHorizontally),
            verticalAlignment = Alignment.CenterVertically
        ) {
            Button(
                onClick = {
                    try { // Room handles OnConflictStrategy.IGNORE (silently ignores duplicates)
                        insertFlashCard(FlashCard(0, english, vietnamese))
                        changeMessage("Flash card successfully added to your database.")
                        english = ""
                        vietnamese = ""
                    } catch (e: Exception) { // Duplicate detection
                        changeMessage("Flash card already exists in your database.")
                    }
                },
                modifier = Modifier.semantics { contentDescription = "Add" }
            ) {
                Text("Add")
            }

            Button(
                onClick = {
                    english = ""
                    vietnamese = ""
                },
                modifier = Modifier.semantics { contentDescription = "Clear" }
            ) {
                Text("Clear")
            }
        }
    }
}
```

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

4.3.5 FilterScreen.kt

```kotlin
package com.example.menuannam.presentation.screens

import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.material3.Button
import androidx.compose.material3.Checkbox
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.semantics.contentDescription
import androidx.compose.ui.semantics.semantics
import androidx.compose.ui.unit.dp

@Composable
fun FilterScreen(
    changeMessage: (String) -> Unit = {}, // Updates status bar with feedback
    onSearch: (en: String, exactEn: Int, vn: String, exactVn: Int) -> Unit // Navigate to SearchCardsRoute with parameters
) {
    var englishText by remember { mutableStateOf("") } // English search term
    var vietnameseText by remember { mutableStateOf("") } // Vietnamese search term
    var exactEnglish by remember { mutableStateOf(false) } // Exact match checkbox for English
    var exactVietnamese by remember { mutableStateOf(false) } // Exact match checkbox for Vietnamese

    LaunchedEffect(Unit) {
        changeMessage("Enter search criteria and click Search")
    }

    Column(
        modifier = Modifier
            .fillMaxWidth()
            .padding(16.dp),
        verticalArrangement = Arrangement.spacedBy(16.dp),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        // English search field with checkbox
        Row(
            modifier = Modifier.fillMaxWidth(),
            verticalAlignment = Alignment.CenterVertically,
            horizontalArrangement = Arrangement.spacedBy(8.dp)
        ) {
            OutlinedTextField(
                value = englishText,
                onValueChange = { englishText = it },
                label = { Text("English") },
                modifier = Modifier
                    .weight(1f)
                    .semantics { contentDescription = "enSearchField" },
                singleLine = true
            )
            Checkbox(
                checked = exactEnglish,
                onCheckedChange = { exactEnglish = it },
                modifier = Modifier.semantics { contentDescription = "enExactCheckbox" }
            )
        }

        // Vietnamese search field with checkbox
        Row(
            modifier = Modifier.fillMaxWidth(),
            verticalAlignment = Alignment.CenterVertically,
            horizontalArrangement = Arrangement.spacedBy(8.dp)
        ) {
            OutlinedTextField(
                value = vietnameseText,
                onValueChange = { vietnameseText = it },
                label = { Text("Vietnamese") },
                modifier = Modifier
                    .weight(1f)
                    .semantics { contentDescription = "vnSearchField" },
                singleLine = true
            )
            Checkbox(
                checked = exactVietnamese,
                onCheckedChange = { exactVietnamese = it },
                modifier = Modifier.semantics { contentDescription = "vnExactCheckbox" }
            )
        }

        Spacer(modifier = Modifier.size(16.dp))

        // Search button
        Button(
            onClick = {
                val en = englishText.trim().ifEmpty { "" }
                val vn = vietnameseText.trim().ifEmpty { "" }
                val exactEn = if (exactEnglish) 1 else 0
                val exactVn = if (exactVietnamese) 1 else 0

                changeMessage("Searching...")
                onSearch(en, exactEn, vn, exactVn)
            },
            modifier = Modifier
                .fillMaxWidth()
                .semantics { contentDescription = "searchButton" }
        ) {
            Text("Search")
        }

        // Clear button
        Button(
            onClick = {
                englishText = ""
                vietnameseText = ""
                exactEnglish = false
                exactVietnamese = false
                changeMessage("Search criteria cleared")
            },
            modifier = Modifier
                .fillMaxWidth()
                .semantics { contentDescription = "clearButton" }
        ) {
            Text("Clear")
        }
    }
}
```

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

4.3.6 SearchScreen.kt

```kotlin
package com.example.menuannam.presentation.screens

import androidx.compose.foundation.border
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.layout.width
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.rememberCoroutineScope
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.semantics.contentDescription
import androidx.compose.ui.semantics.semantics
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import kotlinx.coroutines.launch
import com.example.menuannam.data.database.FlashCardDao
import com.example.menuannam.data.entity.FlashCard

// FlashCardList component - LazyColumn displaying flashcards with edit/delete buttons
@Composable
fun FlashCardList(
    flashCards: List<FlashCard>,
    onEdit: (Int) -> Unit = {}, // Navigate to EditCardRoute with card ID
    onDelete: (FlashCard) -> Unit = {} // Delete card from database
) {
    LazyColumn(
        modifier = Modifier.padding(16.dp)
    ) {
        items(
            items = flashCards,
            key = { flashCard ->
                flashCard.uid
            }
        ) { flashCard ->
            Row(
                modifier = Modifier
                    .fillMaxWidth()
                    .border(width = 1.dp, color = Color.LightGray)
                    .padding(8.dp),
                verticalAlignment = Alignment.CenterVertically
            ) {
                Column(
                    modifier = Modifier
                        .weight(1f)
                        .padding(6.dp)
                ) {
                    Text("${flashCard.englishCard ?: ""} = ${flashCard.vietnameseCard ?: ""}")
                }
                Spacer(modifier = Modifier.width(8.dp))
                Row(
                    horizontalArrangement = Arrangement.spacedBy(4.dp)
                ) {
                    Button(
                        onClick = { onEdit(flashCard.uid) },
                        colors = ButtonDefaults.buttonColors(containerColor = MaterialTheme.colorScheme.primary)
                    ) {
                        Text("Edit", fontSize = 12.sp)
                    }
                    Button(
                        onClick = { onDelete(flashCard) },
                        colors = ButtonDefaults.buttonColors(containerColor = MaterialTheme.colorScheme.error)
                    ) {
                        Text("Delete", fontSize = 12.sp)
                    }
                }
            }
        }
    }
}


@Composable
fun SearchScreen(
    changeMessage: (String) -> Unit = {},
    flashCardDao: FlashCardDao,
    onEdit: (Int) -> Unit = {},
    englishText: String = "",
    exactEnglish: Int = 0,
    vietnameseText: String = "",
    exactVietnamese: Int = 0
) {
    var filteredCards by remember { mutableStateOf<List<FlashCard>>(emptyList()) }
    var isLoading by remember { mutableStateOf(false) }
    val coroutineScope = rememberCoroutineScope()

    val performSearch = {
        coroutineScope.launch {
            try {
                isLoading = true
                filteredCards = flashCardDao.getFilteredFlashCards(englishText, exactEnglish, vietnameseText, exactVietnamese)
                changeMessage("Found ${filteredCards.size} cards")
                isLoading = false
            } catch (e: Exception) {
                changeMessage("Error: ${e.message}")
                isLoading = false
            }
        }
    }

    LaunchedEffect(Unit) {
        performSearch()
    }

    Column(
        modifier = Modifier
            .fillMaxWidth()
            .padding(16.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.spacedBy(12.dp),
    ) {
        // Results display
        if (isLoading) {
            Text("Searching...")
        } else if (filteredCards.isEmpty()) {
            Text("No cards found. Try different search terms.")
        } else {
            FlashCardList(
                flashCards = filteredCards,
                onEdit = onEdit,
                onDelete = { cardToDelete ->
                    coroutineScope.launch {
                        try {
                            flashCardDao.delete(cardToDelete)
                            changeMessage("Card deleted successfully!")
                            performSearch()
                        } catch (e: Exception) {
                            changeMessage("Error deleting card: ${e.message}")
                        }
                    }
                }
            )
        }
    }
}
```

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

4.3.7 EditScreen.kt

```kotlin
package com.example.menuannam.presentation.screens

import android.util.Base64
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.rememberCoroutineScope
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.semantics.contentDescription
import androidx.compose.ui.semantics.semantics
import androidx.compose.ui.unit.dp
import androidx.core.net.toUri
import androidx.media3.common.MediaItem
import androidx.media3.exoplayer.ExoPlayer
import com.example.menuannam.EMAIL
import com.example.menuannam.TOKEN
import com.example.menuannam.data.database.FlashCardDao
import com.example.menuannam.data.entity.FlashCard
import com.example.menuannam.data.network.AudioRequest
import com.example.menuannam.data.network.NetworkService
import com.example.menuannam.dataStore
import com.example.menuannam.saveAudioToInternalStorage
import com.example.menuannam.toMd5
import kotlinx.coroutines.flow.first
import kotlinx.coroutines.launch
import java.io.File

@Composable
fun EditScreen(
    cardId: Int, // Flashcard ID to edit
    flashCardDao: FlashCardDao, // Database access
    networkService: NetworkService, // API for audio generation
    changeMessage: (String) -> Unit, // Status message callback
    onCardUpdated: () -> Unit // Navigate back after successful update
) {
    val context = LocalContext.current
    val appContext = context.applicationContext
    val coroutineScope = rememberCoroutineScope()

    var flashCard by remember { mutableStateOf<FlashCard?>(null) }
    var englishText by remember { mutableStateOf("") }
    var vietnameseText by remember { mutableStateOf("") }
    var audioExists by remember { mutableStateOf(false) }
    var audioFilePath by remember { mutableStateOf("") }
    var isLoading by remember { mutableStateOf(true) }

    // Load flashcard and check audio existence on screen load
    LaunchedEffect(cardId) {
        try {
            val card = flashCardDao.getById(cardId)
            if (card != null) {
                flashCard = card
                englishText = card.englishCard ?: ""
                vietnameseText = card.vietnameseCard ?: ""

                // Check if audio file exists for current Vietnamese word
                val vn = card.vietnameseCard ?: ""
                if (vn.isNotBlank()) {
                    val fileName = "${vn.toMd5()}.mp3"
                    val file = File(appContext.filesDir, fileName)
                    audioExists = file.exists()
                    audioFilePath = if (audioExists) file.absolutePath else ""
                }

                changeMessage("Please, edit the flashcard.")
            } else {
                changeMessage("Flash card not found")
            }
        } catch (e: Exception) {
            changeMessage("Error loading flash card: ${e.message}")
        } finally {
            isLoading = false
        }
    }

    if (isLoading) {
        Column(
            modifier = Modifier
                .fillMaxWidth()
                .padding(16.dp),
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            Text("Loading...")
        }
    } else if (flashCard == null) {
        Column(
            modifier = Modifier
                .fillMaxWidth()
                .padding(16.dp),
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            Text(
                text = "Flash card not found",
                color = MaterialTheme.colorScheme.error
            )
        }
    } else {
        Column(
            modifier = Modifier
                .fillMaxWidth()
                .padding(16.dp),
            verticalArrangement = Arrangement.spacedBy(12.dp),
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            // English text field
            OutlinedTextField(
                value = englishText,
                onValueChange = { englishText = it },
                label = { Text("en") },
                modifier = Modifier
                    .fillMaxWidth()
                    .semantics { contentDescription = "enTextField" }
            )

            // Vietnamese text field
            OutlinedTextField(
                value = vietnameseText,
                onValueChange = { vietnameseText = it },
                label = { Text("vn") },
                modifier = Modifier
                    .fillMaxWidth()
                    .semantics { contentDescription = "vnTextField" }
            )

            // Audio file path display (if exists)
            if (audioExists) {
                OutlinedTextField(
                    value = audioFilePath,
                    onValueChange = {},
                    label = { Text("audio") },
                    readOnly = true,
                    modifier = Modifier.fillMaxWidth(),
                    singleLine = false,
                    maxLines = 3
                )
            }

            Spacer(modifier = Modifier.size(4.dp))

            // Update flashcard button - saves changes to database
            Button(
                onClick = {
                    coroutineScope.launch {
                        try {
                            flashCardDao.update(
                                id = cardId,
                                english = englishText,
                                vietnamese = vietnameseText
                            )
                            changeMessage("Flash card updated successfully")
                            onCardUpdated()
                        } catch (e: Exception) {
                            changeMessage("Error updating flash card: ${e.message}")
                        }
                    }
                },
```

```
            modifier = Modifier
                .fillMaxWidth()
                .semantics { contentDescription = "updateButton" }
        ) {
            Text("Update flashcard")
        }

        // Conditional audio buttons based on file existence
        if (audioExists) { // If audio exists: "Clean audio" and "Play audio"
            // Clean audio button - deletes audio file from storage
            Button(
                onClick = {
                    coroutineScope.launch {
                        try {
                            val file = File(audioFilePath)
                            if (file.exists() && file.delete()) {
                                audioExists = false
                                audioFilePath = ""
                                changeMessage("Audio deleted successfully")
                            } else {
                                changeMessage("Failed to delete audio")
                            }
                        } catch (e: Exception) {
                            changeMessage("Error deleting audio: ${e.message}")
                        }
                    }
                },
                modifier = Modifier
                    .fillMaxWidth()
                    .semantics { contentDescription = "cleanAudioButton" },
                colors = ButtonDefaults.buttonColors(
                    containerColor = MaterialTheme.colorScheme.error
                )
            ) {
                Text("Clean audio")
            }

            // Play audio button
            Button(
                onClick = {
                    coroutineScope.launch {
                        try {
                            changeMessage("Playing audio...")
                            val mediaItem = MediaItem.fromUri(audioFilePath.toUri())
                            val player = ExoPlayer.Builder(appContext).build()
                            player.addListener(object : androidx.media3.common.Player.Listener {
                                override fun onPlaybackStateChanged(playbackState: Int) {
                                    when (playbackState) {
                                        androidx.media3.common.Player.STATE_BUFFERING ->
                                            changeMessage("Buffering...")
                                        androidx.media3.common.Player.STATE_READY ->
                                            changeMessage("Playing")
                                        androidx.media3.common.Player.STATE_ENDED -> {
                                            player.release()
                                            changeMessage("Finished")
                                        }
                                        else -> {}
                                    }
                                }
                            })
                            player.setMediaItem(mediaItem)
                            player.prepare()
                            player.play()
                        } catch (e: Exception) {
                            changeMessage("Error playing audio: ${e.message}")
                        }
                    }
                },
                modifier = Modifier
                    .fillMaxWidth()
                    .semantics { contentDescription = "playAudioButton" }
            ) {
                Text("Play audio")
            }
        } else { // If audio doesn't exist: "Generate audio"
            // Generate audio button - calls API to synthesize audio
            Button(
                onClick = {
                    coroutineScope.launch {
                        try { // Load token and email from DataStore for audio API request
                            val prefs = appContext.dataStore.data.first()
                            val email = prefs[EMAIL] ?: ""
                            val token = prefs[TOKEN] ?: ""
                            val word = vietnameseText.trim()

                            if (word.isBlank() || email.isBlank() || token.isBlank()) { // Missing required data
                                changeMessage("Missing data to request audio")
                                return@launch
                            }

                            changeMessage("Generating audio...")
                            val resp = networkService.generateAudio( // Send Vietnamese word + token to AWS Lambda
                                request = AudioRequest(word, email, token)
                            )

                            if (resp.code == 200) { // Response code 200 = success
                                val audioBytes = Base64.decode(resp.message, Base64.DEFAULT) // Decode Base64-encoded MP3
                                val fileName = "${word.toMd5()}.mp3"
                                val file = saveAudioToInternalStorage(appContext, audioBytes, fileName)
                                audioExists = true
                                audioFilePath = file.absolutePath
                                changeMessage("Audio generated successfully")
                            } else { // 500 = token invalid/expired
                                changeMessage("Audio generation failed (${resp.code}): ${resp.message}")
                            }
                        } catch (e: Exception) {
                            changeMessage("Error generating audio: ${e.message}")
                        }
                    }
                },
                modifier = Modifier
                    .fillMaxWidth()
                    .semantics { contentDescription = "generateAudioButton" }
            ) {
                Text("Generate audio")
            }
        }
    }
}
```

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

4.3.8 StudyScreen.kt

```kotlin
package com.example.menuannam.presentation.screens

import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.material3.Button
import androidx.compose.material3.Card
import androidx.compose.material3.CardDefaults
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableIntStateOf
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.rememberCoroutineScope
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.semantics.contentDescription
import androidx.compose.ui.semantics.semantics
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.core.net.toUri
import android.util.Base64
import java.io.File
import androidx.media3.common.MediaItem
import androidx.media3.exoplayer.ExoPlayer
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.flow.first
import kotlinx.coroutines.launch
import com.example.menuannam.EMAIL
import com.example.menuannam.TOKEN
import com.example.menuannam.dataStore
import com.example.menuannam.toMd5
import com.example.menuannam.saveAudioToInternalStorage
import com.example.menuannam.getCachedAudioFile
import com.example.menuannam.data.database.FlashCardDao
import com.example.menuannam.data.entity.FlashCard
import com.example.menuannam.data.network.AudioRequest
import com.example.menuannam.data.network.NetworkService

// StudyScreen supports two modes: SINGLE_CARD (view one card) and STUDY_SESSION (interactive 3-card learning)
enum class CardViewMode {
    SINGLE_CARD, // View one specific card with delete button (from search)
    STUDY_SESSION // Interactive 3-card learning with flip and audio
}

@Composable
fun StudyScreen(
    changeMessage: (String) -> Unit = {}, // Updates status bar with feedback
    flashCardDao: FlashCardDao,
    networkService: NetworkService,
    mode: CardViewMode = CardViewMode.STUDY_SESSION,
    cardId: Int = 0, // Card ID for SINGLE_CARD mode
    onCardDeleted: () -> Unit = {}, // Navigate back after deletion
    coroutineScope: CoroutineScope? = null
) {
    val context = LocalContext.current
    val appContext = context.applicationContext
    val scope = coroutineScope ?: rememberCoroutineScope()

    var flashCard by remember { mutableStateOf<FlashCard?>(null) }
    var lesson by remember { mutableStateOf<List<FlashCard>>(emptyList()) }
    var isLoading by remember { mutableStateOf(true) }
    var errorMessage by remember { mutableStateOf<String?>(null) }
    var currentIndex by remember { mutableIntStateOf(0) }
    var showVietnamese by remember { mutableStateOf(false) }

    // Load data based on mode: SINGLE_CARD loads one card, STUDY_SESSION loads 5 random cards
    LaunchedEffect(Unit) {
        scope.launch {
            try {
                isLoading = true
                when (mode) {
                    CardViewMode.SINGLE_CARD -> {
                        flashCard = flashCardDao.getById(cardId)
                        if (flashCard == null) {
                            errorMessage = "Flash card not found"
                            changeMessage("Flash card not found")
                        } else {
                            changeMessage("Viewing flash card details")
                        }
                    }
                    CardViewMode.STUDY_SESSION -> {
                        val allCards = flashCardDao.getAll()
                        lesson = if (allCards.size >= 3) {
                            allCards.shuffled().take(3)
                        } else {
                            allCards
                        }
                        if (lesson.isNotEmpty()) {
                            changeMessage("Card ${currentIndex + 1} of ${lesson.size}")
                        } else {
                            errorMessage = "No flashcards found"
                            changeMessage("No flashcards found")
                        }
                    }
                }
            } catch (e: Exception) {
                errorMessage = "Error loading data: ${e.message}"
                changeMessage("Error loading data")
            } finally {
                isLoading = false
            }
        }
    }

    // Play audio utility
    val playAudio: (String) -> Unit = { word ->
        scope.launch {
            try {
                if (word.isBlank()) {
                    changeMessage("Missing data to request audio")
                    return@launch
                }

                // Try cached audio first
                val cachedFile = getCachedAudioFile(appContext, word)
                if (cachedFile != null) {
                    val mediaItem = MediaItem.fromUri(cachedFile.absolutePath.toUri())
                    val player = ExoPlayer.Builder(appContext).build()
                    player.addListener(object : androidx.media3.common.Player.Listener {
                        override fun onPlaybackStateChanged(playbackState: Int) {
                            when (playbackState) {
                                androidx.media3.common.Player.STATE_BUFFERING -> changeMessage("Buffering...")
                                androidx.media3.common.Player.STATE_READY -> changeMessage("Playing...")
                                androidx.media3.common.Player.STATE_ENDED -> {
                                    player.release()
                                    changeMessage("Ready")
                                }
                                else -> {}
                            }
                        }
                    })
                    player.setMediaItem(mediaItem)
                    player.prepare()
                    player.play()
                    return@launch
                }

                // If no cache, call API to generate audio
                val prefs = appContext.dataStore.data.first()
                val email = prefs[EMAIL] ?: ""
                val token = prefs[TOKEN] ?: ""
                if (email.isBlank() || token.isBlank()) { // Missing authentication data
                    changeMessage("Missing data to request audio")
                    return@launch
                }
```

```
                changeMessage("Generating audio...")
                val resp = networkService.generateAudio( // Send Vietnamese word + token to AWS Lambda
                    request = AudioRequest(word, email, token)
                )

                if (resp.code == 200) { // Response code 200 = success
                    val audioBytes = Base64.decode(resp.message, Base64.DEFAULT) // Decode Base64-encoded MP3
                    val fileName = "${word.toMd5()}.mp3"
                    val file = saveAudioToInternalStorage(appContext, audioBytes, fileName) // Save for future use
                    val mediaItem = MediaItem.fromUri(file.absolutePath.toUri())
                    val player = ExoPlayer.Builder(appContext).build()
                    player.addListener(object : androidx.media3.common.Player.Listener { // ExoPlayer state callbacks
                        override fun onPlaybackStateChanged(playbackState: Int) {
                            when (playbackState) {
                                androidx.media3.common.Player.STATE_BUFFERING -> changeMessage("Buffering...")
                                androidx.media3.common.Player.STATE_READY -> changeMessage("Playing...")
                                androidx.media3.common.Player.STATE_ENDED -> {
                                    player.release()
                                    changeMessage("Ready")
                                }
                                else -> {}
                            }
                        }
                    })
                    player.setMediaItem(mediaItem)
                    player.prepare()
                    player.play()
                } else { // 500 = token invalid/expired
                    changeMessage("Audio error (${resp.code}): ${resp.message}")
                }
            } catch (e: Exception) {
                changeMessage("Audio error: ${e.message}")
            }
        }
    }

    // UI Layout
    Column(
        modifier = Modifier
            .fillMaxWidth()
            .padding(16.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.spacedBy(16.dp)
    ) {
        if (isLoading) {
            Text("Loading...")
        } else if (errorMessage != null) {
            Text(
                text = errorMessage!!,
                color = MaterialTheme.colorScheme.error
            )
        } else when (mode) {
            CardViewMode.SINGLE_CARD -> {
                // Single Card View
                if (flashCard != null) {
                    Card(
                        modifier = Modifier.fillMaxWidth(),
                        elevation = CardDefaults.cardElevation(defaultElevation = 4.dp)
                    ) {
                        Column(
                            modifier = Modifier.padding(16.dp),
                            horizontalAlignment = Alignment.CenterHorizontally,
                            verticalArrangement = Arrangement.spacedBy(12.dp)
                        ) {
                            Text(
                                text = "Flash Card #${flashCard!!.uid}",
                                style = MaterialTheme.typography.headlineSmall,
                                fontWeight = FontWeight.Bold
                            )

                            Spacer(modifier = Modifier.size(8.dp))

                            Text(
                                text = "English:",
                                style = MaterialTheme.typography.labelLarge,
                                fontWeight = FontWeight.Medium
                            )
                            Text(
                                text = flashCard!!.englishCard ?: "N/A",
                                style = MaterialTheme.typography.bodyLarge
                            )

                            Spacer(modifier = Modifier.size(8.dp))

                            Text(
                                text = "Vietnamese:",
                                style = MaterialTheme.typography.labelLarge,
                                fontWeight = FontWeight.Medium
                            )
                            Text(
                                text = flashCard!!.vietnameseCard ?: "N/A",
                                style = MaterialTheme.typography.bodyLarge
                            )
                        }
                    }

                    Row(
                        horizontalArrangement = Arrangement.Center,
                        modifier = Modifier.fillMaxWidth()
                    ) {
                        Button(
                            onClick = {
                                scope.launch {
                                    try {
                                        flashCardDao.delete(flashCard!!)
                                        changeMessage("Card deleted successfully!")
                                        onCardDeleted()
                                    } catch (e: Exception) {
                                        changeMessage("Error: ${e.message}")
                                    }
                                }
                            }
                        ) {
                            Text("Delete")
                        }
                        Spacer(modifier = Modifier.size(8.dp))
                        Button(
                            onClick = {
                                playAudio(flashCard!!.vietnameseCard ?: "")
                            }
                        ) {
                            Text("Play Audio")
                        }
                    }
                }
            }

            CardViewMode.STUDY_SESSION -> {
                // Study Session View
                if (lesson.isNotEmpty()) {
                    val currentCard = lesson[currentIndex]

                    Text(
                        text = "Card ${currentIndex + 1} of ${lesson.size}",
                        style = MaterialTheme.typography.titleMedium
                    )

                    Card(
                        modifier = Modifier
                            .fillMaxWidth()
                            .weight(1f)
                            .clickable {
                                showVietnamese = !showVietnamese
                            },
                        elevation = CardDefaults.cardElevation(defaultElevation = 8.dp)
                    ) {
                        Box(
                            modifier = Modifier.fillMaxSize(),
                            contentAlignment = Alignment.Center
                        ) {
                            Text(
                                text = if (showVietnamese) {
                                    currentCard.
```

```
    vietnameseCard ?: ""
                                } else {
                                    currentCard.englishCard
    ?: ""
                                },
                                fontSize = 32.sp,
                                textAlign = TextAlign.Center
    ,
                                modifier = Modifier.padding
    (32.dp)
                            )
                        }
                    }

                    if (showVietnamese) {
                        Button(
                            modifier = Modifier
                                .fillMaxWidth()
                                .semantics {
    contentDescription = "PlayAudio" },
                            onClick = {
                                playAudio(currentCard.
    vietnameseCard ?: "")
                            }
                        ) {
                            Text("Play Audio")
                        }
                    }

                    if (showVietnamese) {
                        Button(
                            modifier = Modifier.fillMaxWidth
    (),
                            onClick = {
                                val nextIndex = (
    currentIndex + 1) % lesson.size
                                if (nextIndex == 0) {
                                    lesson = lesson.shuffled
    ()
                                }
                                currentIndex = nextIndex
                                showVietnamese = false
                                changeMessage("Card ${
    currentIndex + 1} of ${lesson.size}")
                            }
                        ) {
                            Text("Next")
                        }
                    }
                }
            }
        }
    }
}
```

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

Chapter 5

UI Theme

5.1 Color.kt

```kotlin
package com.example.menuannam.ui.theme

import androidx.compose.ui.graphics.Color

val Purple80 = Color(0xFFD0BCFF)
val PurpleGrey80 = Color(0xFFCCC2DC)
val Pink80 = Color(0xFFEFB8C8)

val Purple40 = Color(0xFF6650a4)
val PurpleGrey40 = Color(0xFF625b71)
val Pink40 = Color(0xFF7D5260)
```

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

5.2 Theme.kt

```kotlin
package com.example.menuannam.ui.theme

import android.app.Activity
import android.os.Build
import androidx.compose.foundation.isSystemInDarkTheme
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.darkColorScheme
import androidx.compose.material3.dynamicDarkColorScheme
import androidx.compose.material3.dynamicLightColorScheme
import androidx.compose.material3.lightColorScheme
import androidx.compose.runtime.Composable
import androidx.compose.ui.platform.LocalContext

private val DarkColorScheme = darkColorScheme(
    primary = Purple80,
    secondary = PurpleGrey80,
    tertiary = Pink80
)

private val LightColorScheme = lightColorScheme(
    primary = Purple40,
    secondary = PurpleGrey40,
    tertiary = Pink40

    /* Other default colors to override
    background = Color(0xFFFFFBFE),
    surface = Color(0xFFFFFBFE),
    onPrimary = Color.White,
    onSecondary = Color.White,
    onTertiary = Color.White,
    onBackground = Color(0xFF1C1B1F),
    onSurface = Color(0xFF1C1B1F),
    */
)

@Composable
fun MenuAnNamTheme(
    darkTheme: Boolean = isSystemInDarkTheme(),
    // Dynamic color is available on Android 12+
    dynamicColor: Boolean = true,
    content: @Composable () -> Unit
) {
    val colorScheme = when {
        dynamicColor && Build.VERSION.SDK_INT >= Build.VERSION_CODES.S -> {
            val context = LocalContext.current
            if (darkTheme) dynamicDarkColorScheme(context) else dynamicLightColorScheme(context)
        }

        darkTheme -> DarkColorScheme
        else -> LightColorScheme
    }

    MaterialTheme(
        colorScheme = colorScheme,
        typography = Typography,
        content = content
    )
}
```

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

5.3 Type.kt

```kotlin
package com.example.menuannam.ui.theme

import androidx.compose.material3.Typography
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.sp

// Set of Material typography styles to start with
val Typography = Typography(
    bodyLarge = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Normal,
        fontSize = 16.sp,
        lineHeight = 24.sp,
        letterSpacing = 0.5.sp
    )
    /* Other default text styles to override
    titleLarge = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Normal,
        fontSize = 22.sp,
        lineHeight = 28.sp,
        letterSpacing = 0.sp
    ),
    labelSmall = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Medium,
        fontSize = 11.sp,
        lineHeight = 16.sp,
        letterSpacing = 0.5.sp
    )
    */
)
```

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

Chapter 6

Build Configuration

6.1 build.gradle.kts (App Module)

```
plugins {
    alias(libs.plugins.android.application)
    alias(libs.plugins.kotlin.android)
    alias(libs.plugins.kotlin.compose)
    id("org.jetbrains.kotlin.plugin.serialization")
    id("com.google.devtools.ksp")
}

android {
    namespace = "com.example.menuannam"
    compileSdk = 36

    defaultConfig {
        applicationId = "com.example.menuannam"
        minSdk = 26  // originally 24
        targetSdk = 36
        versionCode = 27
        versionName = "1.0"
        //testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
    }

    // For Kotlin projects using KSP:
    ksp {
        arg("room.schemaLocation", "$projectDir/schemas")
    }

    buildTypes {
        release {
            isMinifyEnabled = false
            isDebuggable = false
            proguardFiles(
                getDefaultProguardFile("proguard-android-optimize.txt"),
                "proguard-rules.pro"
            )
        }
    }

    compileOptions {
        sourceCompatibility = JavaVersion.VERSION_21
        targetCompatibility = JavaVersion.VERSION_21
    }

    kotlin {
        compilerOptions {
            jvmTarget.set(org.jetbrains.kotlin.gradle.dsl.JvmTarget.JVM_21)
        }
    }

    buildFeatures {
        compose = true
    }

    androidResources {
        generateLocaleConfig = true
    }

    testOptions {
        unitTests {
            isIncludeAndroidResources = true
        }
    }
}

dependencies {
    implementation(libs.androidx.core.ktx)
    implementation(libs.androidx.lifecycle.runtime.ktx)
    implementation(libs.androidx.activity.compose)
    implementation(platform(libs.androidx.compose.bom))
    implementation(libs.androidx.material3)
    implementation(libs.androidx.navigation.runtime.ktx)
    implementation(libs.androidx.navigation.compose)
    implementation(libs.androidx.navigation.testing)
```

```
    implementation(libs.core.ktx)
    implementation(libs.androidx.compose.ui.test.junit4)

    implementation(libs.androidx.compose.ui)
    implementation(libs.androidx.compose.foundation)
    implementation(libs.androidx.compose.foundation.layout)
    implementation(libs.androidx.media3.exoplayer)
    //implementation(libs.androidx.compose.ui.test)
    testImplementation(libs.junit)
    // For local unit tests
    //testImplementation(libs.androidx.core.testing)
    testImplementation(libs.robolectric)
    // Needed for createComposeRule(), but not for createAndroidComposeRule<YourActivity>():
    debugImplementation(libs.androidx.compose.ui.test.manifest)

    //androidTestImplementation(libs.androidx.junit)
    //androidTestImplementation(libs.androidx.espresso.core)
    androidTestImplementation(platform(libs.androidx.compose.bom))
    // Test rules and transitive dependencies:
    androidTestImplementation(libs.androidx.compose.ui.test.junit4)

    /* Dependencies related to room */
    // room-compiler: is for the code generation that happens
    // during the build process to create the necessary database
    // infrastructure based on your annotations.
    // implementation(libs.androidx.room.compiler)
    // room-runtime: is for the code that runs on
    // your device to interact with the database.
    implementation(libs.androidx.room.runtime)
    // If this project uses any Kotlin source, use Kotlin Symbol Processing (KSP)
    // See Add the KSP plugin to your project
    ksp(libs.androidx.room.compiler)
    // optional - Kotlin Extensions and Coroutines support for Room
    implementation(libs.androidx.room.ktx)
    // optional - Test helpers
    testImplementation(libs.androidx.room.testing)

    // serialization
    implementation(libs.kotlinx.serialization.json)
    implementation(libs.androidx.navigation.compose.v280alpha08)

    // retrofit
    implementation("com.squareup.retrofit2:retrofit:2.9.0")
    implementation("com.squareup.retrofit2:converter-gson:2.9.0")

    // datastore
    implementation("androidx.datastore:datastore-preferences:1.2.0")
    implementation("androidx.datastore:datastore-preferences-core:1.2.0")
}
```

./app/build.gradle.kts

6.2 libs.versions.toml

```
[versions]

agp = "8.12.0"

composeVersion = "1.9.4"

kotlin = "2.1.10"

coreKtx = "1.17.0"

junit = "4.13.2"

#junitVersion = "1.3.0"

#espressoCore = "3.7.0"

lifecycleRuntimeKtx = "2.9.4"

activityCompose = "1.11.0"

composeBom = "2025.10.01"

navigationCompose = "2.9.5"

material3 = "1.4.0"

navigationRuntimeKtx = "2.9.5"

navigationTesting = "2.9.5"

robolectric = "4.16"

roomVersion = "2.8.3"

coreKtxVersion = "1.7.0"

ui = "1.9.4"

composeFoundation = "1.9.4"

media3Version = "1.4.1"

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 robolectric = { module = "org.robolectric:robolectric", version.ref = "robolectric" }
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 kotlinx-serialization-json = { group = "org.jetbrains.kotlinx", name = "kotlinx-serialization-json", version.ref = "
      kotlinxSerializationVersion" }
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