```
PROJECT FILE LAYOUT
===================
? .
    ? AndroidManifest.xml
? .idea
    ? misc.xml
    ? modules.xml
    ? workspace.xml
    ? inspectionProfiles
        ? profiles_settings.xml
? java
    ? com
        ? example
            ? holodex
                ? MyApp.kt
                ? auth
                    ? AuthRepository.kt
                    ? AuthViewModel.kt
                    ? LoginScreen.kt
                    ? TokenManager.kt
                ? background
                    ? FavoriteChannelSynchronizer.kt
                    ? HistorySynchronizer.kt
                    ? ISynchronizer.kt
                    ? LikesSynchronizer.kt
                    ? M4AExportWorker.kt
                    ? MetadataUpdateWorker.kt
                    ? MetadataWriter.kt
                    ? PlaylistSynchronizer.kt
                    ? StarredPlaylistSynchronizer.kt
                    ? SyncCoordinator.kt
                    ? SyncLogger.kt
                    ? SyncWorker.kt
                    ? hannelMigrationWorker.kt
                ? data
                    ? AppPreferences.kt
                    ? api
                        ? AuthenticatedMusicdexApiService.kt
                        ? HolodexApiService.kt
                        ? MusicdexApiService.kt
                        ? PlaylistDto.kt
                        ? PlaylistRequestDtos.kt
                    ? cache
                        ? BrowseListCache.kt
                        ? CacheKey.kt
                        ? CachePolicyAndException.kt
                        ? FetcherResult.kt
                        ? SearchListCache.kt
                    ? db
                        ? AppDatabase.kt
                        ? BrowsePageDao.kt
                        ? CachedPageEntities.kt
                        ? Converters.kt
                        ? DiscoveryDao.kt
                        ? Enums.kt
```

```
            ? HolodexSongListConverter.kt
            ? LocalEntities.kt
            ? ParentVideoMetadataDao.kt
            ? PlaylistDao.kt
            ? SearchPageDao.kt
            ? StarredPlaylistDao.kt
            ? StarredPlaylistEntity.kt
            ? SyncMetadataDao.kt
            ? UnifiedDao.kt
            ? UnifiedItemProjection.kt
            ? UnifiedMetadataEntity.kt
            ? UserInteractionEntity.kt
            ? VideoDao.kt
            ? entities.kt
            ? mappers
                ? SyncMappers.kt
        ? download
            ? DownloadCompletionObserver.kt
            ? DownloadExceptions.kt
            ? LegacyDownloadScanner.kt
        ? model
            ? AudioStreamDetails.kt
            ? ChannelSearchResult.kt
            ? HolodexSong.kt
            ? HolodexVideoItem.kt
            ? PaginatedVideosResponse.kt
            ? VideoSearchRequest.kt
            ? discovery
                ? ChannelDetails.kt
                ? DiscoveryResponse.kt
                ? FullPlaylist.kt
                ? MusicdexSong.kt
                ? PlaylistStub.kt
        ? repository
            ? DownloadRepository.kt
            ? HolodexRepository.kt
            ? SearchHistoryRepository.kt
            ? SyncRepository.kt
            ? UnifiedVideoRepository.kt
            ? UserPreferencesRepository.kt
            ? YouTubeStreamRepository.kt
        ? source
            ? CacheSchemeDataSourceFactory.kt
    ? di
        ? AppModule.kt
        ? AuthModule.kt
        ? CacheModule.kt
        ? DatabaseModule.kt
        ? DispatchersModule.kt
        ? NetworkModule.kt
        ? PlaybackModule.kt
        ? Qualifiers.kt
        ? RepositoryModule.kt
        ? SyncModule.kt
        ? UseCaseModule.kt
```

```
? export
? extractor
    ? DownloaderImpl.kt
? playback
    ? data
        ? mapper
            ? MediaItemMapper.kt
        ? model
            ? PlaybackDao.kt
            ? PlaybackEntities.kt
        ? persistence
        ? preload
            ? PreloadConfiguration.kt
            ? PreloadStatusController.kt
        ? queue
            ? PlaybackQueueState.kt
            ? QueueAction.kt
            ? ShuffleOrderProvider.kt
        ? repository
            ? HolodexStreamResolverRepositoryImpl.kt
        ? source
            ? HolodexResolvingDataSource.kt
            ? StreamResolutionCoordinator.kt
    ? domain
        ? model
            ? DomainPlaybackProgress.kt
            ? DomainPlaybackState.kt
            ? DomainRepeatMode.kt
            ? DomainShuffleMode.kt
            ? PersistedPlaybackData.kt
            ? PlaybackItem.kt
            ? PlaybackQueue.kt
            ? StreamDetails.kt
        ? repository
            ? PlaybackStateRepository.kt
            ? StreamResolverRepository.kt
        ? usecase
            ? AddItemToQueueUseCase.kt
            ? AddItemsToQueueUseCase.kt
            ? AddOrFetchAndAddUseCase.kt
            ? LoadPlaybackStateUseCase.kt
            ? ResolveStreamUrlUseCase.kt
            ? SavePlaybackStateUseCase.kt
    ? player
        ? Media3PlayerController.kt
        ? MediaControllerManager.kt
        ? PlaybackController.kt
    ? util
        ? PlaybackUtil.kt
        ? PlayerStateMapper.kt
? service
    ? HolodexDownloadService.kt
    ? MediaPlaybackService.kt
? ui
    ? MainActivity.kt
```

```
    ? MainScreenScaffold.kt
    ? composables
        ? ApiKeyInputScreen.kt
        ? CarouselShelf.kt
        ? ChannelCard.kt
        ? CustomPagedUnifiedList.kt
        ? FullPlayerScreen.kt
        ? HeroCard.kt
        ? HeroCarousel.kt
        ? ItemOptionsMenu.kt
        ? MainScreenLayout.kt
        ? Media3PlayerControls.kt
        ? MiniPlayerWithProgressBar.kt
        ? PlayerBackground.kt
        ? PlaylistArtwork.kt
        ? PlaylistCard.kt
        ? PlaylistManagementDialogs.kt
        ? StateDisplayComposables.kt
        ? UnifiedGridItem.kt
        ? UnifiedListItem.kt
        ? sheets
            ? BrowseFiltersSheet.kt
    ? dialogs
        ? AddExternalChannelDialog.kt
        ? CreatePlaylistDialog.kt
        ? SelectPlaylistDialog.kt
    ? screens
        ? ChannelDetailsScreen.kt
        ? DiscoveryScreen.kt
        ? DownloadsScreen.kt
        ? EditablePlaylistHeader.kt
        ? FavoritesScreen.kt
        ? ForYouScreen.kt
        ? FullListViewScreen.kt
        ? HistoryScreen.kt
        ? HomeScreen.kt
        ? LibraryScreen.kt
        ? PlaylistDetailsScreen.kt
        ? PlaylistsScreen.kt
        ? SettingsScreen.kt
        ? VideoDetailsScreen.kt
        ? navigation
            ? AppDestinations.kt
            ? BottomNavItem.kt
            ? HolodexNavHost.kt
    ? theme
        ? Color.kt
        ? Shape.kt
        ? Theme.kt
        ? Type.kt
? util
    ? ArtworkResolver.kt
    ? ComposableUtils.kt
    ? Extract_util.kt
    ? ImageUtils.kt
```

```
                ? PaletteExtractor.kt
                ? PlaylistFormatter.kt
                ? VideoFilteringUtil.kt
            ? viewmodel
                ? AddChannelViewModel.kt
                ? ChannelDetailsViewModel.kt
                ? DiscoveryViewModel.kt
                ? DownloadsViewModel.kt
                ? FavoritesViewModel.kt
                ? FullListViewModel.kt
                ? FullPlayerViewModel.kt
                ? HistoryViewModel.kt
                ? PlaybackUiStateSelectors.kt
                ? PlaybackViewModel.kt
                ? PlaylistDetailsViewModel.kt
                ? PlaylistManagementViewModel.kt
                ? SettingsViewModel.kt
                ? SharedViewModelTypes.kt
                ? UnifiedDisplayItem.kt
                ? VideoDetailsViewModel.kt
                ? VideoListViewModel.kt
                ? autoplay
                    ? AutoplayItemProvider.kt
                    ? ContinuationManager.kt
                ? mappers
                    ? UnifiedDisplayItemMapper.kt
                ? state
                    ? BrowseFilterState.kt
                    ? UiState.kt
? res
    ? drawable
        ? ic_default_album_art_placeholder.xml
        ? ic_error_image.xml
        ? ic_launcher_background.xml
        ? ic_launcher_foreground.xml
        ? ic_like_empty.xml
        ? ic_notification_small.xml
        ? ic_pause.xml
        ? ic_placeholder_image.xml
        ? ic_play_arrow.xml
        ? ic_repeat_off_24.xml
        ? ic_repeat_on_24.xml
        ? ic_repeat_one_24.xml
        ? ic_shuffle_off_24.xml
        ? ic_shuffle_on_24.xml
        ? ic_skip_next.xml
        ? ic_skip_previous.xml
        ? ic_stat_music_note.xml
        ? ic_twitter.xml
        ? ic_youtube.xml
        ? twitter.xml
        ? youtube.xml
    ? layout
    ? mipmap-anydpi-v26
        ? ic_launcher.xml
```

```
    ? mipmap-hdpi
    ? mipmap-mdpi
    ? mipmap-xhdpi
    ? mipmap-xxhdpi
    ? mipmap-xxxhdpi
    ? values
        ? Theme.xml
        ? attrs.xml
        ? colors.xml
        ? strings.xml
    ? values-night
        ? themes.xml
    ? xml
        ? backup_rules.xml
        ? data_extraction_rules.xml
? sqldelight
    ? com
        ? example
            ? holodex
                ? db

==================
CODE CONTENT

// File: AndroidManifest.xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.holodex">

    <!-- Permissions -->
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
    <uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>
    <uses-permission android:name="android.permission.FOREGROUND_SERVICE_MEDIA_PLAYBACK" />
    <uses-permission android:name="android.permission.FOREGROUND_SERVICE_DATA_SYNC" />

    <!-- START OF FIX: Add permissions for reading media -->
    <!-- For Android 13 (API 33) and above -->
    <uses-permission android:name="android.permission.READ_MEDIA_AUDIO" />

    <!-- For Android 12 (API 32) and below.
         The maxSdkVersion ensures this is only requested on older devices
         where it's necessary for MediaStore to scan all audio files. -->
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"
        android:maxSdkVersion="32" />
    <!-- END OF FIX -->

    <!-- We keep the old write permission for legacy devices where it might be needed -->
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
        android:maxSdkVersion="28" />


    <application
```

```
    android:name=".MyApp"
    android:allowBackup="true"
    android:dataExtractionRules="@xml/data_extraction_rules"
    android:fullBackupContent="@xml/backup_rules"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/Theme.Holodex"
    android:usesCleartextTraffic="true"
    tools:targetApi="34"
    android:enableOnBackInvokedCallback="true">

    <!-- (The rest of the file is unchanged) -->

    <provider
        android:name="androidx.startup.InitializationProvider"
        android:authorities="${applicationId}.androidx-startup"
        android:exported="false"
        tools:node="merge">

        <meta-data
            android:name="androidx.work.WorkManagerInitializer"
            android:value="androidx.startup"
            tools:node="remove" />
    </provider>
    <activity
        android:name=".ui.MainActivity"
        android:exported="true"
        android:windowSoftInputMode="adjustResize"
        android:launchMode="singleTop">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <service
        android:name=".service.MediaPlaybackService"
        android:foregroundServiceType="mediaPlayback"
        android:exported="true">
        <intent-filter>
            <action android:name="androidx.media3.session.MediaSessionService"/>
        </intent-filter>
    </service>

    <service
        android:name=".service.HolodexDownloadService"
        android:exported="false"
        android:foregroundServiceType="dataSync">
        <intent-filter>
            <action android:name="androidx.media3.exoplayer.download.DownloadService"/>
        </intent-filter>
    </service>

</application>
```

```
</manifest>

// File: .idea\misc.xml
<?xml version="1.0" encoding="UTF-8"?>
<project version="4">
  <component name="ProjectRootManager" version="2" project-jdk-name="Python 3.11" project-jdk-
</project>

// File: .idea\modules.xml
<?xml version="1.0" encoding="UTF-8"?>
<project version="4">
  <component name="ProjectModuleManager">
    <modules>
      <module fileurl="file://$PROJECT_DIR$/.idea/main.iml" filepath="$PROJECT_DIR$/.idea/main
    </modules>
  </component>
</project>

// File: .idea\workspace.xml
<?xml version="1.0" encoding="UTF-8"?>
<project version="4">
  <component name="ChangeListManager">
    <list default="true" id="071e5753-0ef0-489e-9c83-db4f6a13ebe4" name="Changes" comment="" /
    <option name="SHOW_DIALOG" value="false" />
    <option name="HIGHLIGHT_CONFLICTS" value="true" />
    <option name="HIGHLIGHT_NON_ACTIVE_CHANGELIST" value="false" />
    <option name="LAST_RESOLUTION" value="IGNORE" />
  </component>
  <component name="ProjectColorInfo"><![CDATA[{
  "associatedIndex": 8
}]]></component>
  <component name="ProjectId" id="2y3tTufEtGAMlv9Qc9BPajVvJjG" />
  <component name="ProjectViewState">
    <option name="hideEmptyMiddlePackages" value="true" />
    <option name="showLibraryContents" value="true" />
  </component>
  <component name="PropertiesComponent"><![CDATA[{
  "keyToString": {
    "ModuleVcsDetector.initialDetectionPerformed": "true",
    "RunOnceActivity.ShowReadmeOnStart": "true",
    "ignore.virus.scanning.warn.message": "true"
  }
}]]></component>
  <component name="SharedIndexes">
    <attachedChunks>
      <set>
        <option value="bundled-python-sdk-348a24fa61fa-5312c7369657-com.jetbrains.pycharm.comm
      </set>
    </attachedChunks>
  </component>
  <component name="TaskManager">
    <task active="true" id="Default" summary="Default task">
      <changelist id="071e5753-0ef0-489e-9c83-db4f6a13ebe4" name="Changes" comment="" />
      <created>1749075097916</created>
```

```
      <option name="number" value="Default" />
      <option name="presentableId" value="Default" />
      <updated>1749075097916</updated>
    </task>
    <servers />
  </component>
</project>

// File: .idea\inspectionProfiles\profiles_settings.xml
<component name="InspectionProjectProfileManager">
  <settings>
    <option name="USE_PROJECT_PROFILE" value="false" />
    <version value="1.0" />
  </settings>
</component>

// File: java\com\example\holodex\MyApp.kt
package com.example.holodex

import android.app.Application
import android.app.NotificationChannel
import android.app.NotificationManager
import android.content.Intent
import android.util.Log
import androidx.hilt.work.HiltWorkerFactory
import androidx.lifecycle.DefaultLifecycleObserver
import androidx.lifecycle.LifecycleOwner
import androidx.lifecycle.ProcessLifecycleOwner
import androidx.media3.common.util.UnstableApi
import androidx.work.Configuration
import coil.ImageLoader
import coil.ImageLoaderFactory
import coil.annotation.ExperimentalCoilApi
import com.example.holodex.data.download.DownloadCompletionObserver
import com.example.holodex.data.repository.DownloadRepository
import com.example.holodex.data.repository.HolodexRepository
import com.example.holodex.di.ApplicationScope
import com.example.holodex.extractor.DownloaderImpl
import dagger.hilt.android.HiltAndroidApp
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.delay
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext
import okhttp3.OkHttpClient
import org.schabi.newpipe.extractor.NewPipe
import org.schabi.newpipe.extractor.localization.ContentCountry
import org.schabi.newpipe.extractor.localization.Localization
import timber.log.Timber
import java.io.File
import java.util.Locale
import java.util.concurrent.TimeUnit
import javax.inject.Inject
import kotlin.system.exitProcess
```

```
@UnstableApi
@HiltAndroidApp
class MyApp : Application(), ImageLoaderFactory, DefaultLifecycleObserver, Configuration.Provi

    @Inject
    lateinit var workerFactory: HiltWorkerFactory

    @Inject
    lateinit var holodexRepository: HolodexRepository

    @Inject
    lateinit var downloadRepository: DownloadRepository

    @Inject
    lateinit var downloadCompletionObserver: DownloadCompletionObserver

    @Inject
    lateinit var imageLoader: ImageLoader

    @Inject
    @ApplicationScope
    lateinit var appScope: CoroutineScope

    override val workManagerConfiguration: Configuration
        get() = Configuration.Builder()
            .setWorkerFactory(workerFactory)
            .setMinimumLoggingLevel(if (BuildConfig.DEBUG) Log.DEBUG else Log.INFO)
            .build()

    override fun onCreate() {
        super<Application>.onCreate()
        ProcessLifecycleOwner.get().lifecycle.addObserver(this)

        if (BuildConfig.DEBUG) {
            Timber.plant(Timber.DebugTree())
        }

        createNotificationChannels()
        downloadCompletionObserver.initialize()

        appScope.launch {
            holodexRepository.cleanupExpiredCacheEntries()
        }

        val downloaderOkHttpClient = OkHttpClient.Builder()
            .connectTimeout(60, TimeUnit.SECONDS)
            .readTimeout(60, TimeUnit.SECONDS)
            .build()

        NewPipe.init(
            DownloaderImpl(downloaderOkHttpClient),
            Localization.fromLocale(Locale.JAPAN),
            ContentCountry(Locale.JAPAN.country)
        )
    }
```

```
@UnstableApi
override fun onStart(owner: LifecycleOwner) {
    Timber.i("MyApp entering foreground. Triggering reconciliations.")
    appScope.launch {
        downloadRepository.reconcileAllDownloads()
        downloadRepository.rescanStorageForDownloads()
    }
}

private fun createNotificationChannels() {
    val notificationManager = getSystemService(NOTIFICATION_SERVICE) as NotificationManage

    val downloadChannel = NotificationChannel(
        "download_channel",
        getString(R.string.download_notification_channel_name),
        NotificationManager.IMPORTANCE_LOW
    ).apply {
        description = getString(R.string.download_notification_channel_description)
    }

    val playbackChannel = NotificationChannel(
        "holodex_playback_channel_v3",
        getString(R.string.playback_notification_channel_name),
        NotificationManager.IMPORTANCE_LOW
    ).apply {
        description = getString(R.string.playback_notification_channel_description)
    }

    notificationManager.createNotificationChannel(downloadChannel)
    notificationManager.createNotificationChannel(playbackChannel)
}

override fun newImageLoader(): ImageLoader {
    return imageLoader // Use the Hilt-provided singleton
}

@OptIn(ExperimentalCoilApi::class)
@UnstableApi
fun clearAllAppCachesOnDemand(callback: (Boolean) -> Unit) {
    appScope.launch {
        var allSuccess = true
        try {
            withContext(Dispatchers.IO) {
                val mediaCacheDir = File(applicationContext.cacheDir, "exoplayer_media_cac
                val downloadCacheDir = File(applicationContext.getExternalFilesDir(null),
                mediaCacheDir.deleteRecursively()
                downloadCacheDir.deleteRecursively()
            }
        } catch (e: Exception) {
            Timber.e(e, "Error deleting ExoPlayer caches.")
            allSuccess = false
        }

        try {
```

```
                imageLoader.diskCache?.clear()
                imageLoader.memoryCache?.clear()
            } catch (e: Exception) {
                Timber.e(e, "Error clearing Coil cache.")
                allSuccess = false
            }

            try {
                holodexRepository.clearAllCachedData()
            } catch (e: Exception) {
                Timber.e(e, "Error clearing repository data.")
                allSuccess = false
            }

            callback(allSuccess)

            withContext(Dispatchers.Main) {
                delay(500)
                val packageManager = applicationContext.packageManager
                val intent = packageManager.getLaunchIntentForPackage(applicationContext.packa
                val componentName = intent!!.component
                val mainIntent = Intent.makeRestartActivityTask(componentName)
                applicationContext.startActivity(mainIntent)
                exitProcess(0)
            }
        }
    }
}

// File: java\com\example\holodex\auth\AuthRepository.kt
// File: java/com/example/holodex/auth/AuthRepository.kt
// (Create this new file)

package com.example.holodex.auth

import com.example.holodex.data.api.HolodexApiService
import com.example.holodex.data.api.LoginRequest
import net.openid.appauth.AuthorizationService
import net.openid.appauth.TokenRequest
import net.openid.appauth.TokenResponse
import kotlin.coroutines.suspendCoroutine
import kotlin.coroutines.resume
import kotlin.coroutines.resumeWithException

/**
 * Orchestrates the entire authentication flow, from exchanging the Discord
 * auth code to logging into the Holodex backend.
 */
class AuthRepository(
    private val holodexApiService: HolodexApiService,
    private val authService: AuthorizationService
) {

    /**
     * Exchanges a one-time authorization code from Discord for an access token.
```

```
     * This is a suspending function that wraps the AppAuth callback-based API.
     */
    suspend fun exchangeDiscordCodeForToken(tokenRequest: TokenRequest): TokenResponse {
        return suspendCoroutine { continuation ->
            authService.performTokenRequest(tokenRequest) { response, ex ->
                if (response != null) {
                    continuation.resume(response)
                } else {
                    continuation.resumeWithException(ex ?: IllegalStateException("Token exchan
                }
            }
        }
    }

    /**
     * Uses the Discord access token to log into the Holodex backend and get a JWT.
     */
    suspend fun loginToHolodex(discordAccessToken: String): String {
        val request = LoginRequest(service = "discord", token = discordAccessToken)
        val response = holodexApiService.login(request)

        if (response.isSuccessful && response.body() != null) {
            return response.body()!!.jwt
        } else {
            throw Exception("Holodex login failed: ${response.errorBody()?.string()}")
        }
    }
}

// File: java\com\example\holodex\auth\AuthViewModel.kt
// File: java/com/example/holodex/auth/AuthViewModel.kt
// (Create this new file)

package com.example.holodex.auth

import android.content.Context
import android.content.Intent
import android.util.Base64
import androidx.core.net.toUri
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.example.holodex.BuildConfig
import dagger.hilt.android.lifecycle.HiltViewModel
import dagger.hilt.android.qualifiers.ApplicationContext
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.launch
import net.openid.appauth.AuthorizationRequest
import net.openid.appauth.AuthorizationService
import net.openid.appauth.AuthorizationServiceConfiguration
import net.openid.appauth.ResponseTypeValues
import org.json.JSONObject
import timber.log.Timber
import java.nio.charset.StandardCharsets
```

```
import javax.inject.Inject

// Represents the different states of the authentication flow for the UI to observe.
sealed class AuthState {
    object LoggedOut : AuthState()
    object InProgress : AuthState()
    object LoggedIn : AuthState()
    data class Error(val message: String) : AuthState()
}
private fun getUserIdFromJwt(jwt: String): String? {
    return try {
        val parts = jwt.split(".")
        if (parts.size < 2) return null
        val payload = parts[1]
        val decodedBytes = Base64.decode(payload, Base64.URL_SAFE)
        val decodedString = String(decodedBytes, StandardCharsets.UTF_8)
        val json = JSONObject(decodedString)
        json.optInt("i", -1).takeIf { it != -1 }?.toString()
    } catch (e: Exception) {
        Timber.e(e, "Failed to decode JWT payload")
        null
    }
}
@HiltViewModel
class AuthViewModel @Inject constructor(
    @ApplicationContext private val appContext: Context,
    private val authRepository: AuthRepository,
    private val tokenManager: TokenManager
) : ViewModel() {

    private val _authState = MutableStateFlow<AuthState>(AuthState.LoggedOut)
    val authState: StateFlow<AuthState> = _authState.asStateFlow()

    private val authService = AuthorizationService(appContext)

    // Define the endpoints for Discord's OAuth2 service
    private val serviceConfig = AuthorizationServiceConfiguration(
        "https://discord.com/api/oauth2/authorize".toUri(),
        "https://discord.com/api/oauth2/token".toUri()
    )

    init {
        // On ViewModel creation, check if we are already logged in
        if (tokenManager.getJwt() != null) {
            _authState.value = AuthState.LoggedIn
        }
    }

    /**
     * Creates an intent to launch the Discord login flow in a Custom Tab.
     * This is called by the UI when the user taps the "Login" button.
     */
    fun getAuthorizationRequestIntent(): Intent {
        val authRequest = AuthorizationRequest.Builder(
            serviceConfig,
```

```
        BuildConfig.DISCORD_CLIENT_ID,
        ResponseTypeValues.CODE,
        BuildConfig.DISCORD_REDIRECT_URI.toUri()
    )
        .setScope("identify email")
        // AppAuth automatically generates and includes the PKCE parameters
        .build()

    Timber.d("Created authorization request for Discord.")
    return authService.getAuthorizationRequestIntent(authRequest)
}

/**
 * Handles the redirect intent received from the Custom Tab after the user
 * authorizes the app on Discord.
 */
fun onAuthorizationResponse(intent: Intent) {
    _authState.value = AuthState.InProgress

    val resp = net.openid.appauth.AuthorizationResponse.fromIntent(intent)
    val ex = net.openid.appauth.AuthorizationException.fromIntent(intent)

    if (resp == null) {
        val errorMessage = "Authorization failed: ${ex?.errorDescription ?: "Unknown error
        Timber.e(ex, errorMessage)
        _authState.value = AuthState.Error(errorMessage)
        return
    }

    // The authorization was successful, now exchange the code for a token.
    // AppAuth automatically includes the PKCE code_verifier it generated earlier.
    viewModelScope.launch {
        try {
            Timber.d("Exchanging authorization code for Discord token...")
            val tokenResponse = authRepository.exchangeDiscordCodeForToken(resp.createToke

            val discordAccessToken = tokenResponse.accessToken
            if (discordAccessToken == null) {
                throw IllegalStateException("Discord access token was null")
            }

            Timber.d("Successfully received Discord access token. Now logging into Holodex
            val holodexJwt = authRepository.loginToHolodex(discordAccessToken)
            tokenManager.saveJwt(holodexJwt)
            val userId = getUserIdFromJwt(holodexJwt)
            if (userId != null) {
                tokenManager.saveUserId(userId)
                Timber.i("Successfully logged in, saved Holodex JWT, and extracted User ID
            } else {
                Timber.w("Successfully logged in but could not extract User ID from JWT.")
            }
            _authState.value = AuthState.LoggedIn
            Timber.i("Successfully logged in and saved Holodex JWT.")

        } catch (e: Exception) {
```

```
                val errorMessage = "Full login flow failed: ${e.message}"
                Timber.e(e, errorMessage)
                _authState.value = AuthState.Error(errorMessage)
            }
        }
    }

    fun logout() {
        tokenManager.clearJwt()
        _authState.value = AuthState.LoggedOut
        Timber.i("User logged out and JWT cleared.")
    }

    override fun onCleared() {
        super.onCleared()
        authService.dispose()
    }
}

// File: java\com\example\holodex\auth\LoginScreen.kt
// File: java/com/example/holodex/auth/LoginScreen.kt
// (Create this new file)

package com.example.holodex.auth

import android.widget.Toast
import androidx.activity.compose.rememberLauncherForActivityResult
import androidx.activity.result.contract.ActivityResultContracts
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.Button
import androidx.compose.material3.CircularProgressIndicator
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.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.lifecycle.compose.collectAsStateWithLifecycle
import com.example.holodex.R

@Composable
fun LoginScreen(
    authViewModel: AuthViewModel = hiltViewModel(),
```

```
    onLoginSuccess: () -> Unit
) {
    val authState by authViewModel.authState.collectAsStateWithLifecycle()
    val context = LocalContext.current

    // This launcher will start the Custom Tab for Discord login.
    val authLauncher = rememberLauncherForActivityResult(
        contract = ActivityResultContracts.StartActivityForResult()
    ) { result ->
        // After the user returns from the Custom Tab, the result intent is passed here.
        result.data?.let { intent ->
            authViewModel.onAuthorizationResponse(intent)
        }
    }

    // Observe the auth state to react to changes.
    LaunchedEffect(authState) {
        when (val state = authState) {
            is AuthState.LoggedIn -> {
                Toast.makeText(context, "Login Successful!", Toast.LENGTH_SHORT).show()
                onLoginSuccess()
            }
            is AuthState.Error -> {
                Toast.makeText(context, "Login Failed: ${state.message}", Toast.LENGTH_LONG).s
            }
            else -> {
                // InProgress or LoggedOut, no side-effect needed here.
            }
        }
    }

    Box(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp),
        contentAlignment = Alignment.Center
    ) {
        when (authState) {
            is AuthState.InProgress -> {
                CircularProgressIndicator()
            }
            else -> {
                Column(
                    horizontalAlignment = Alignment.CenterHorizontally,
                    verticalArrangement = Arrangement.Center
                ) {
                    Text(
                        text = "Login Required",
                        style = MaterialTheme.typography.headlineSmall,
                    )
                    Spacer(modifier = Modifier.height(8.dp))
                    Text(
                        text = "Please log in with Discord to enable synchronization and other
                        style = MaterialTheme.typography.bodyMedium,
                        textAlign = TextAlign.Center
```

```
                    )
                    Spacer(modifier = Modifier.height(24.dp))
                    Button(
                        onClick = {
                            // Launch the authorization intent created by the ViewModel
                            authLauncher.launch(authViewModel.getAuthorizationRequestIntent())
                        }
                    ) {
                        Text(stringResource(R.string.login_with_discord)) // <-- Add this stri
                    }
                }
            }
        }
    }
}

// File: java\com\example\holodex\auth\TokenManager.kt
// File: java/com/example/holodex/auth/TokenManager.kt

package com.example.holodex.auth

import android.content.Context
import androidx.core.content.edit
import androidx.security.crypto.EncryptedSharedPreferences
import androidx.security.crypto.MasterKeys

class TokenManager(context: Context) {

    companion object {
        private const val PREF_FILE_NAME = "auth_token_prefs"
        private const val KEY_HOLODEX_JWT = "holodex_jwt"
        // --- ADDITION: Key for storing the User ID ---
        private const val KEY_USER_ID = "user_id"
    }

    private val masterKeyAlias = MasterKeys.getOrCreate(MasterKeys.AES256_GCM_SPEC)

    private val sharedPreferences = EncryptedSharedPreferences.create(
        PREF_FILE_NAME,
        masterKeyAlias,
        context,
        EncryptedSharedPreferences.PrefKeyEncryptionScheme.AES256_SIV,
        EncryptedSharedPreferences.PrefValueEncryptionScheme.AES256_GCM
    )

    fun saveJwt(token: String) {
        sharedPreferences.edit {
            putString(KEY_HOLODEX_JWT, token)
        }
    }

    fun getJwt(): String? {
        return sharedPreferences.getString(KEY_HOLODEX_JWT, null)
    }
```

```
    // --- ADDITION: New methods for User ID ---
    fun saveUserId(id: String) {
        sharedPreferences.edit {
            putString(KEY_USER_ID, id)
        }
    }

    fun getUserId(): String? {
        return sharedPreferences.getString(KEY_USER_ID, null)
    }
    // --- END OF ADDITION ---

    fun clearJwt() {
        sharedPreferences.edit {
            remove(KEY_HOLODEX_JWT)
            // --- ADDITION: Clear the User ID on logout ---
            remove(KEY_USER_ID)
        }
    }
}

// File: java\com\example\holodex\background\FavoriteChannelSynchronizer.kt
package com.example.holodex.background

import com.example.holodex.data.api.AuthenticatedMusicdexApiService
import com.example.holodex.data.api.HolodexApiService
import com.example.holodex.data.api.PatchOperation
import com.example.holodex.data.db.UnifiedMetadataEntity
import com.example.holodex.data.repository.SyncRepository
import javax.inject.Inject

class FavoriteChannelSynchronizer @Inject constructor(
    private val syncRepository: SyncRepository,
    private val authApiService: AuthenticatedMusicdexApiService,
    private val holodexApiService: HolodexApiService,
    private val logger: SyncLogger
) : ISynchronizer {

    override val name: String = "FAV_CHANNELS"
    private val TYPE = "FAV_CHANNEL"

    override suspend fun synchronize(): Boolean {
        logger.startSection(name)
        try {
            // ================================================================================
            // PHASE 1: UPSTREAM (Local -> Server)
            // ================================================================================

            val pendingDeletes = syncRepository.getPendingDeleteItems(TYPE)
            val dirtyItems = syncRepository.getDirtyItems(TYPE)

            // --- FIX START: Filter External Channels ---
            val validDirtyItems = mutableListOf<com.example.holodex.data.db.UserInteractionEnt

            for (item in dirtyItems) {
```

```
        val meta = syncRepository.getMetadata(item.itemId)
        // Check if it's explicitly marked External OR if it's a manually added channe
        // Note: "External" is the artistName/org we used in AddChannelViewModel
        if (meta?.org != "External" && meta?.artistName != "External") {
            validDirtyItems.add(item)
        } else {
            // It's external. Mark as SYNCED locally so we stop trying to push it.
            logger.info("  Skipping external channel sync for: ${meta?.title} (${item.
            syncRepository.markAsSynced(item.itemId, TYPE, "local_only")
        }
    }
    // --- FIX END ---

    if (pendingDeletes.isNotEmpty() || validDirtyItems.isNotEmpty()) {
        logger.info("Phase 1: Sending PATCH with ${pendingDeletes.size} removals, ${va

        val patchOps = mutableListOf<PatchOperation>()
        pendingDeletes.forEach { patchOps.add(PatchOperation("remove", it.itemId)) }
        validDirtyItems.forEach { patchOps.add(PatchOperation("add", it.itemId)) } //

        val response = authApiService.patchFavoriteChannels(patchOps)

        if (response.isSuccessful) {
            if (pendingDeletes.isNotEmpty()) syncRepository.confirmBatchDeletion(pendi
            if (validDirtyItems.isNotEmpty()) syncRepository.markBatchSynced(validDirt
            logger.info("  -> Upstream PATCH successful.")
        } else {
            logger.warning("  -> Upstream PATCH failed: ${response.code()}")
            logger.endSection(name, false)
            return false
        }
    } else {
        logger.info("Phase 1: No local changes to push.")
    }

    // ================================================================================
    // PHASE 2: DOWNSTREAM (Server -> Local)
    // ================================================================================
    logger.info("Phase 2: Fetching remote channels and reconciling...")

    val remoteRes = holodexApiService.getFavoriteChannels()
    if (!remoteRes.isSuccessful) throw Exception("Failed to get remote channels: ${rem

    val remoteChannels = remoteRes.body() ?: emptyList()
    val localSynced = syncRepository.getSyncedItems(TYPE)

    val remoteIdMap = remoteChannels.associateBy { it.id }
    val localIdMap = localSynced.associateBy { it.itemId }

    // 1. Insert New from Server
    val newFromServer = remoteChannels.filter { !localIdMap.containsKey(it.id) }
    if (newFromServer.isNotEmpty()) logger.info("  -> Found ${newFromServer.size} new

    for (remote in newFromServer) {
        // *** THE FIX: Insert into the Unified Tables ***
```

```
                val meta = UnifiedMetadataEntity(
                    id = remote.id,
                    title = remote.name ?: remote.english_name ?: "Unknown Channel",
                    artistName = remote.org ?: "",
                    type = "CHANNEL",
                    specificArtUrl = remote.photo,
                    uploaderAvatarUrl = remote.photo,
                    duration = 0,
                    channelId = remote.id,
                    description = null,
                    org = remote.org
                )

                // Server ID for a channel is its own ID
                syncRepository.insertRemoteItem(remote.id, TYPE, remote.id, meta)
                logger.logItemAction(LogAction.DOWNSTREAM_INSERT_LOCAL, remote.name, null, rem
            }

            // 2. Delete Removed on Server
            val deletedOnServer = localSynced.filter { !remoteIdMap.containsKey(it.itemId) }
            if (deletedOnServer.isNotEmpty()) logger.info("  -> Found ${deletedOnServer.size}

            for (local in deletedOnServer) {
                syncRepository.removeRemoteItem(local.itemId, TYPE)
                logger.logItemAction(LogAction.DOWNSTREAM_DELETE_LOCAL, local.itemId, null, lo
            }

            logger.endSection(name, true)
            return true
        } catch (e: Exception) {
            logger.error(e, "Channel Sync Failed")
            logger.endSection(name, false)
            return false
        }
    }
}

// File: java\com\example\holodex\background\hannelMigrationWorker.kt
package com.example.holodex.background

import android.content.Context
import androidx.hilt.work.HiltWorker
import androidx.work.CoroutineWorker
import androidx.work.WorkerParameters
import com.example.holodex.data.api.HolodexApiService
import com.example.holodex.data.db.UnifiedDao
import dagger.assisted.Assisted
import dagger.assisted.AssistedInject
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.flow.first
import kotlinx.coroutines.withContext
import timber.log.Timber

@HiltWorker
class ChannelRepairWorker @AssistedInject constructor(
```

```
    @Assisted context: Context,
    @Assisted workerParams: WorkerParameters,
    private val unifiedDao: UnifiedDao,
    private val holodexApiService: HolodexApiService
) : CoroutineWorker(context, workerParams) {

    override suspend fun doWork(): Result = withContext(Dispatchers.IO) {
        Timber.e("=== STARTING CHANNEL REPAIR ===")

        val channels = unifiedDao.getFavoriteChannels().first()

        for (channel in channels) {
            val meta = channel.metadata
            val id = meta.id

            // Skip LOLUET if we know it's external
            if (id == "UC1CGaG114jfEiiWzZ2SkyhQ") {
                Timber.e("Skipping known external: ${meta.title}")
                continue
            }

            try {
                val response = holodexApiService.getChannelDetails(id)
                if (response.isSuccessful && response.body() != null) {
                    val remote = response.body()!!
                    val org = remote.org ?: "Independents" // Default if null

                    Timber.e("REPAIRING: ${meta.title} ($id). Old Org: '${meta.org}'. New Org:

                    val newMeta = meta.copy(
                        org = org,
                        artistName = org, // Sync artist name
                        type = "CHANNEL" // Ensure type is correct
                    )
                    unifiedDao.updateMetadataRaw(newMeta)
                } else {
                    Timber.e("Failed to fetch details for ${meta.title}: ${response.code()}")
                }
            } catch (e: Exception) {
                Timber.e(e, "Error repairing ${meta.title}")
            }
            Thread.sleep(200) // Throttle
        }

        Timber.e("=== REPAIR COMPLETE ===")
        return@withContext Result.success()
    }
}

// File: java\com\example\holodex\background\HistorySynchronizer.kt
// File: java/com/example/holodex/background/HistorySynchronizer.kt
package com.example.holodex.background

import androidx.room.withTransaction
import com.example.holodex.auth.TokenManager
```

```
import com.example.holodex.data.db.AppDatabase
import com.example.holodex.data.db.SyncMetadataDao
import com.example.holodex.data.db.SyncMetadataEntity
import com.example.holodex.data.db.UnifiedDao
import com.example.holodex.data.db.UnifiedMetadataEntity
import com.example.holodex.data.db.UserInteractionEntity
import com.example.holodex.data.repository.HolodexRepository
import java.time.Instant
import javax.inject.Inject

class HistorySynchronizer @Inject constructor(
    private val repository: HolodexRepository,
    private val unifiedDao: UnifiedDao,
    private val database: AppDatabase, // For transaction support
    private val syncMetadataDao: SyncMetadataDao,
    private val tokenManager: TokenManager,
    private val logger: SyncLogger
) : ISynchronizer {
    override val name: String = "HISTORY"
    private val METADATA_KEY = "history_last_sync_timestamp"

    override suspend fun synchronize(): Boolean {
        logger.startSection(name)
        val userId = tokenManager.getUserId()
        if (userId.isNullOrBlank()) {
            logger.warning("User ID not found, skipping history sync.")
            logger.endSection(name, success = true)
            return true
        }

        try {
            logger.info("Phase 1: Fetching remote history playlist...")
            // Holodex stores history as a special playlist
            val historyPlaylistId = ":history[user_id=$userId]"
            val remoteResult = repository.getFullPlaylistContent(historyPlaylistId)

            if (remoteResult.isFailure) {
                throw remoteResult.exceptionOrNull() ?: Exception("Failed to fetch remote hist
            }

            val remotePlaylist = remoteResult.getOrThrow()
            val remoteTimestampStr = remotePlaylist.updatedAt

            // If remote has no timestamp, we force sync anyway if content exists
            val remoteTimestamp = if (!remoteTimestampStr.isNullOrBlank()) {
                Instant.parse(remoteTimestampStr).toEpochMilli()
            } else {
                System.currentTimeMillis()
            }

            val localTimestamp = syncMetadataDao.getLastSyncTimestamp(METADATA_KEY) ?: 0L
            logger.info("  -> Remote TS: $remoteTimestamp | Local TS: $localTimestamp")

            // Matching existing logic: If server is newer OR has content, we overwrite local
            if (remoteTimestamp > localTimestamp || (remotePlaylist.content?.isNotEmpty() == t
```

```
        logger.info("Phase 2: Updating local history cache from server.")

        val baseTimestamp = System.currentTimeMillis()
        val remoteSongs = remotePlaylist.content ?: emptyList()

        database.withTransaction {
            // 1. Clear existing local history to ensure exact match with server order
            // This matches the "current system" logic of wiping the old table.
            unifiedDao.deleteAllInteractionsByType("HISTORY")

            // 2. Insert new items
            remoteSongs.forEachIndexed { index, song ->
                if (!song.channelId.isNullOrBlank()) {
                    val itemId = "${song.videoId}_${song.start}"
                    val playedAt = baseTimestamp - index // Preserve server order usin

                    // A. Upsert Metadata (Safe insert)
                    val metadata = UnifiedMetadataEntity(
                        id = itemId,
                        title = song.name,
                        artistName = song.channel.name,
                        type = "SEGMENT",
                        specificArtUrl = song.artUrl,
                        uploaderAvatarUrl = song.channel.photoUrl,
                        duration = (song.end - song.start).toLong(),
                        channelId = song.channelId,
                        parentVideoId = song.videoId,
                        startSeconds = song.start.toLong(),
                        endSeconds = song.end.toLong(),
                        lastUpdatedAt = System.currentTimeMillis()
                    )
                    unifiedDao.upsertMetadata(metadata)

                    // B. Insert Interaction
                    val interaction = UserInteractionEntity(
                        itemId = itemId,
                        interactionType = "HISTORY",
                        timestamp = playedAt,
                        syncStatus = "SYNCED",
                        serverId = song.id // Server UUID for the history item if avai
                    )
                    unifiedDao.upsertInteraction(interaction)
                }
            }

            // 3. Update Sync Timestamp
            syncMetadataDao.setLastSyncTimestamp(
                SyncMetadataEntity(
                    dataType = METADATA_KEY,
                    lastSyncTimestamp = remoteTimestamp
                )
            )
        }
        logger.info("  -> Successfully synced ${remoteSongs.size} history items.")
    } else {
```

```
                logger.info("Phase 2: Local history is up-to-date.")
            }

            logger.endSection(name, success = true)
            return true

        } catch (e: Exception) {
            logger.error(e, "History sync failed.")
            logger.endSection(name, success = false)
            return false
        }
    }
}

// File: java\com\example\holodex\background\ISynchronizer.kt
// File: java/com/example/holodex/background/ISynchronizer.kt (NEW FILE)
package com.example.holodex.background

/**
 * Defines the contract for a class that can synchronize a specific type of data
 * between the local database and a remote server.
 */
interface ISynchronizer {
    /**
     * The unique name of this synchronizer, used for logging.
     */
    val name: String

    /**
     * Executes the full synchronization logic for this data type.
     * @return `true` if the synchronization was successful, `false` otherwise.
     */
    suspend fun synchronize(): Boolean
}

// File: java\com\example\holodex\background\LikesSynchronizer.kt
package com.example.holodex.background

import com.example.holodex.data.api.AuthenticatedMusicdexApiService
import com.example.holodex.data.api.LikeRequest
import com.example.holodex.data.db.UnifiedMetadataEntity
import com.example.holodex.data.repository.HolodexRepository
import com.example.holodex.data.repository.SyncRepository
import javax.inject.Inject

class LikesSynchronizer @Inject constructor(
    private val syncRepository: SyncRepository,
    private val holodexRepository: HolodexRepository, // Needed for Orphan lookup (fetchVideoA
    private val apiService: AuthenticatedMusicdexApiService,
    private val logger: SyncLogger
) : ISynchronizer {

    override val name: String = "LIKES"
    private val TYPE = "LIKE"
```

```
// Timeout period after which we trust the server's state over a local PENDING_DELETE.
private val PENDING_DELETE_TIMEOUT_MS = 35 * 60 * 1000L // 35 minutes

override suspend fun synchronize(): Boolean {
    logger.startSection(name)
    try {
        // ================================================================================
        // PHASE 0: PRE-SYNC REPAIR
        // Fix items marked DIRTY but missing a serverId (happens if added offline/error)
        // ================================================================================
        logger.info("Phase 0: Checking for orphaned local likes to repair...")
        val dirtyItems = syncRepository.getDirtyItems(TYPE)
        val orphanedLikes = dirtyItems.filter { it.serverId == null }

        if (orphanedLikes.isNotEmpty()) {
            logger.info("  -> Found ${orphanedLikes.size} orphaned items. Attempting to re
            for (orphan in orphanedLikes) {
                // Parse the composite ID (videoId_start)
                val videoId = orphan.itemId.substringBeforeLast('_')
                val startTime = orphan.itemId.substringAfterLast('_').toIntOrNull()

                if (startTime != null) {
                    // Song Segment: Fetch remote data to find the real Server UUID
                    val result = holodexRepository.fetchVideoAndFindSong(videoId, startTim
                    val song = result?.second

                    if (song?.id != null) {
                        // Found it! Update local DB with the Server ID so Phase 1 can syn
                        // We use 'upsertInteraction' via Repository to update just the se
                        val repaired = orphan.copy(serverId = song.id)
                        syncRepository.updateServerId(repaired.itemId, TYPE, song.id)
                        // Note: We keep it DIRTY so Phase 1 picks it up to send to server

                        logger.logItemAction(LogAction.RECONCILE_SKIP, "Song_${orphan.item
                    } else {
                        logger.warning("  -> FAILED repair for '${orphan.itemId}'. Could n
                    }
                } else {
                    // Full Video: Videos don't use UUIDs for likes, they use the VideoID
                    // If serverId is null, set it to videoId.
                    syncRepository.updateServerId(orphan.itemId, TYPE, orphan.itemId)
                    logger.logItemAction(LogAction.RECONCILE_SKIP, "Video_${orphan.itemId}
                }
            }
        }
        logger.info("Phase 0 complete.")

        // ================================================================================
        // PHASE 1: UPSTREAM (Local -> Server)
        // ================================================================================
        logger.info("Phase 1: Pushing local changes to server...")

        // 1. Handle Deletions (PENDING_DELETE)
        val pendingDeletes = syncRepository.getPendingDeleteItems(TYPE)
        for (item in pendingDeletes) {
```

```
        if (item.serverId == null) {
            // Local-only item, just delete
            syncRepository.confirmDeletion(item.itemId, TYPE)
            continue
        }

        val response = apiService.deleteLike(LikeRequest(song_id = item.serverId))
        if (response.isSuccessful || response.code() == 404) {
            syncRepository.confirmDeletion(item.itemId, TYPE)
            logger.logItemAction(LogAction.UPSTREAM_DELETE_SUCCESS, item.itemId, null,
        } else {
            logger.logItemAction(LogAction.UPSTREAM_DELETE_FAILED, item.itemId, null,
        }
    }

    // 2. Handle Additions (DIRTY)
    // Re-fetch dirty items because Phase 0 might have fixed some
    val readyToUpload = syncRepository.getDirtyItems(TYPE).filter { it.serverId != nul

    for (item in readyToUpload) {
        val response = apiService.addLike(LikeRequest(song_id = item.serverId!!))
        if (response.isSuccessful) {
            syncRepository.markAsSynced(item.itemId, TYPE, item.serverId!!)
            logger.logItemAction(LogAction.UPSTREAM_UPSERT_SUCCESS, item.itemId, null,
        } else {
            logger.logItemAction(LogAction.UPSTREAM_UPSERT_FAILED, item.itemId, null,
        }
    }
    logger.info("Phase 1 complete.")

    // ================================================================================
    // PHASE 2: DOWNSTREAM (Server -> Local)
    // ================================================================================
    logger.info("Phase 2: Fetching states and reconciling...")

    // 1. Fetch Remote
    val allRemoteLikes = mutableListOf<com.example.holodex.data.api.LikedSongApiDto>()
    var page = 1
    while (true) {
        val res = apiService.getLikes(page = page, paginated = true)
        if (!res.isSuccessful) throw Exception("Failed to fetch likes page $page")
        val body = res.body() ?: break
        allRemoteLikes.addAll(body.content)
        if (page >= body.page_count) break
        page++
    }

    // 2. Fetch Local SYNCED items (Ignore Dirty/Pending)
    val localSynced = syncRepository.getSyncedItems(TYPE)

    val remoteIdMap = allRemoteLikes.associateBy { it.id } // Key = Server UUID
    val localServerIdMap = localSynced.associateBy { it.serverId } // Key = Server UUI

    // --- RULE 1: Item on Server, Not Local -> INSERT ---
    val newFromServer = allRemoteLikes.filter { !localServerIdMap.containsKey(it.id) }
```

```
        if (newFromServer.isNotEmpty()) logger.info("  Found ${newFromServer.size} new lik

        for (remote in newFromServer) {
            // Construct Metadata (We must have this to insert interaction)
            val meta = UnifiedMetadataEntity(
                id = remote.video_id, // We use videoId (or composite) as the Metadata Key
                title = remote.name,
                artistName = remote.original_artist ?: "Unknown",
                type = "SEGMENT", // API likes are usually segments
                specificArtUrl = remote.art,
                uploaderAvatarUrl = remote.channel?.photo,
                duration = (remote.end - remote.start).toLong(),
                channelId = remote.channel_id,
                description = null,
                startSeconds = remote.start.toLong(),
                endSeconds = remote.end.toLong(),
                parentVideoId = remote.video_id,
                lastUpdatedAt = System.currentTimeMillis()
            )

            // The Interaction ID must match how we generate IDs locally (composite)
            val localItemId = "${remote.video_id}_${remote.start}"

            syncRepository.insertRemoteItem(localItemId, TYPE, remote.id, meta)
            logger.logItemAction(LogAction.DOWNSTREAM_INSERT_LOCAL, remote.name, null, rem
        }

        // --- RULE 2: Item Synced Locally, Not on Server -> DELETE ---
        // Only check items that claim to be SYNCED. If Dirty/Pending, ignore.
        val deletedOnServer = localSynced.filter { it.serverId != null && !remoteIdMap.con

        if (deletedOnServer.isNotEmpty()) logger.info("  Found ${deletedOnServer.size} lik

        for (local in deletedOnServer) {
            syncRepository.removeRemoteItem(local.itemId, TYPE)
            logger.logItemAction(LogAction.DOWNSTREAM_DELETE_LOCAL, local.itemId, null, lo
        }

        // --- RULE 3: Timeout Logic (Zombie Prevention) ---
        // If an item is PENDING_DELETE for too long, and it still exists on server,
        // assume the delete failed silently or was reverted by another device.
        val pendingToCheck = syncRepository.getPendingDeleteItems(TYPE)
        val now = System.currentTimeMillis()

        for (pending in pendingToCheck) {
            if (pending.serverId != null && remoteIdMap.containsKey(pending.serverId)) {
                // It's pending delete locally, but server still has it.
                if (now - pending.timestamp > PENDING_DELETE_TIMEOUT_MS) {
                    // Revert to SYNCED
                    syncRepository.markAsSynced(pending.itemId, TYPE, pending.serverId!!)
                    logger.logItemAction(LogAction.RECONCILE_SKIP, pending.itemId, null, p
                }
            }
        }
```

```
            logger.endSection(name, true)
            return true
        } catch (e: Exception) {
            logger.error(e, "Likes Sync Failed")
            logger.endSection(name, false)
            return false
        }
    }
}

// File: java\com\example\holodex\background\M4AExportWorker.kt
// File: java\com\example\holodex\background\M4AExportWorker.kt
package com.example.holodex.background

import android.content.ContentValues
import android.content.Context
import android.media.MediaMetadataRetriever
import android.net.Uri
import android.os.Build
import android.os.Environment
import android.provider.MediaStore
import androidx.annotation.OptIn
import androidx.core.net.toUri
import androidx.hilt.work.HiltWorker
import androidx.media3.common.MediaItem
import androidx.media3.common.MimeTypes
import androidx.media3.common.util.Clock
import androidx.media3.common.util.UnstableApi
import androidx.media3.datasource.DefaultHttpDataSource
import androidx.media3.datasource.cache.CacheDataSource
import androidx.media3.datasource.cache.SimpleCache
import androidx.media3.exoplayer.source.DefaultMediaSourceFactory
import androidx.media3.transformer.Composition
import androidx.media3.transformer.DefaultDecoderFactory
import androidx.media3.transformer.ExoPlayerAssetLoader
import androidx.media3.transformer.ExportException
import androidx.media3.transformer.ExportResult
import androidx.media3.transformer.InAppMp4Muxer
import androidx.media3.transformer.Transformer
import androidx.work.CoroutineWorker
import androidx.work.WorkerParameters
import com.example.holodex.data.db.DownloadStatus
import com.example.holodex.data.db.UnifiedDao
import com.example.holodex.di.DownloadCache
import dagger.assisted.Assisted
import dagger.assisted.AssistedInject
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.suspendCancellableCoroutine
import kotlinx.coroutines.withContext
import timber.log.Timber
import java.io.File
import java.io.IOException
import java.net.URLDecoder
import kotlin.coroutines.resume
import kotlin.coroutines.resumeWithException
```

```
@OptIn(UnstableApi::class)
@HiltWorker
class M4AExportWorker @AssistedInject constructor(
    @Assisted private val context: Context,
    @Assisted workerParams: WorkerParameters,
    @DownloadCache private val downloadCache: SimpleCache,
    private val unifiedDao: UnifiedDao,
    private val metadataWriter: MetadataWriter // <-- INJECT the MetadataWriter
) : CoroutineWorker(context, workerParams) {

    companion object {
        // Keys for WorkManager Data
        const val KEY_ITEM_ID = "ITEM_ID"
        const val KEY_ORIGINAL_URI = "ORIGINAL_URI"
        const val KEY_SONG_TITLE = "SONG_TITLE"
        const val KEY_ARTIST_NAME = "ARTIST_NAME"
        const val KEY_ALBUM_NAME = "ALBUM_NAME"
        const val KEY_ARTWORK_URI = "ARTWORK_URI"
        const val KEY_CLIP_START_MS = "CLIP_START_MS"
        const val KEY_CLIP_END_MS = "CLIP_END_MS"
        const val KEY_TRACK_NUMBER = "TRACK_NUMBER"
        private const val TAG = "M4AExportWorker"
    }

    override suspend fun doWork(): Result {
        val itemId = inputData.getString(KEY_ITEM_ID) ?: return Result.failure()
        val originalUriString = inputData.getString(KEY_ORIGINAL_URI) ?: return Result.failure
        val songTitle = inputData.getString(KEY_SONG_TITLE) ?: "Unknown Title"
        val artistName = inputData.getString(KEY_ARTIST_NAME) ?: "Unknown Artist"
        val albumName = inputData.getString(KEY_ALBUM_NAME) ?: "Unknown Album"
        val artworkUri = inputData.getString(KEY_ARTWORK_URI)
        val clipStartMs = inputData.getLong(KEY_CLIP_START_MS, -1)
        val clipEndMs = inputData.getLong(KEY_CLIP_END_MS, -1)
        val trackNumber = inputData.getInt(KEY_TRACK_NUMBER, -1)

        if (clipStartMs == -1L || clipEndMs == -1L) {
            Timber.e("$TAG: Invalid clip times provided.")
            unifiedDao.updateDownloadStatus(itemId, DownloadStatus.FAILED.name)
            return Result.failure()
        }

        val decodedTitle = try {
            URLDecoder.decode(songTitle, "UTF-8")
        } catch (_: Exception) {
            songTitle
        }
        val sanitizedDisplayTitle = decodedTitle.replace(Regex("[\\\\/:*?\"<>|]"), "_").take(1
        val finalFileName = "$sanitizedDisplayTitle.m4a"

        val tempOutputFile = File(context.cacheDir, "transformer_output_${itemId}.m4a")

        Timber.d("$TAG: Starting export for item: $itemId, filename: $finalFileName")

        try {
```

```
        // Step 1: Use Transformer to create a clean, clipped M4A file
        val mediaItem = MediaItem.Builder()
            .setUri(originalUriString.toUri())
            .setClippingConfiguration(
                MediaItem.ClippingConfiguration.Builder()
                    .setStartPositionMs(clipStartMs)
                    .setEndPositionMs(clipEndMs)
                    .build()
            )
            .build()

        val exportResult = withContext(Dispatchers.Main) {
            val transformer = buildTransformerOnMainThread()
            transformMediaItem(transformer, mediaItem, tempOutputFile.absolutePath)
        }

        if (exportResult.exportException != null) throw exportResult.exportException!!
        Timber.d("$TAG: Transformer successfully created temp file: ${tempOutputFile.absol

        // Step 2: Write metadata to the temporary file using the injected MetadataWriter
        metadataWriter.writeMetadata(
            context = context,
            targetFile = tempOutputFile,
            itemId = itemId,
            songTitle = songTitle,
            artistName = artistName,
            albumTitle = albumName,
            artworkUrl = artworkUri,
            trackNumber = trackNumber
        )
        Timber.d("$TAG: MetadataWriter successfully wrote tags to temp file.")


        // Step 3: Verify and export the now-tagged file
        verifyMetadataInFile(
            tempOutputFile,
            songTitle,
            artistName,
            albumName,
            artworkUri != null
        )
        val finalUri = exportToMediaStore(tempOutputFile, finalFileName)
            ?: throw IOException("Failed to export temp file to MediaStore.")

        // Step 4: Finalize and clean up
        unifiedDao.completeDownload(itemId, finalUri.toString())
        downloadCache.removeResource(itemId)
        tempOutputFile.delete()

        Timber.i("$TAG: Successfully exported item $itemId to $finalUri")
        return Result.success()

    } catch (e: Exception) {
        Timber.e(e, "$TAG: Export failed for item $itemId.")
        unifiedDao.updateDownloadStatus(itemId, DownloadStatus.FAILED.name)
```

```
        tempOutputFile.delete()
        return Result.failure()
    }
}

private fun buildTransformerOnMainThread(): Transformer {
    val upstreamDataSourceFactory = DefaultHttpDataSource.Factory()
    val cacheDataSourceFactory = CacheDataSource.Factory()
        .setCache(downloadCache)
        .setUpstreamDataSourceFactory(upstreamDataSourceFactory)
    val mediaSourceFactory =
        DefaultMediaSourceFactory(context).setDataSourceFactory(cacheDataSourceFactory)
    val assetLoaderFactory = ExoPlayerAssetLoader.Factory(
        context,
        DefaultDecoderFactory.Builder(context).build(),
        Clock.DEFAULT,
        mediaSourceFactory
    )

    return Transformer.Builder(context)
        .setAssetLoaderFactory(assetLoaderFactory)
        .setAudioMimeType(MimeTypes.AUDIO_AAC)
        .experimentalSetMp4EditListTrimEnabled(true)
        .setMuxerFactory(InAppMp4Muxer.Factory())
        .build()
}

private suspend fun transformMediaItem(
    transformer: Transformer,
    mediaItem: MediaItem,
    outputPath: String
): ExportResult {
    return suspendCancellableCoroutine { continuation ->
        val listener = object : Transformer.Listener {
            override fun onCompleted(composition: Composition, exportResult: ExportResult)
                if (continuation.isActive) continuation.resume(exportResult)
            }

            override fun onError(
                composition: Composition,
                exportResult: ExportResult,
                exportException: ExportException
            ) {
                if (continuation.isActive) continuation.resumeWithException(exportExceptio
            }
        }
        transformer.addListener(listener)
        continuation.invokeOnCancellation {
            transformer.removeListener(listener)
            transformer.cancel()
        }
        transformer.start(mediaItem, outputPath)
    }
}
```

```
private fun exportToMediaStore(sourceFile: File, finalFileName: String): Uri? {
    val resolver = context.contentResolver
    val contentValues = ContentValues().apply {
        put(MediaStore.MediaColumns.DISPLAY_NAME, finalFileName)
        put(MediaStore.MediaColumns.MIME_TYPE, "audio/mp4")
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {
            put(
                MediaStore.MediaColumns.RELATIVE_PATH,
                Environment.DIRECTORY_MUSIC + File.separator + "HolodexMusic"
            )
            put(MediaStore.Audio.Media.IS_PENDING, 1)
        } else {
            @Suppress("DEPRECATION")
            val musicDir =
                Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_MUSIC)
            val appMusicDir = File(musicDir, "HolodexMusic")
            if (!appMusicDir.exists() && !appMusicDir.mkdirs()) {
                return null
            }
            val targetFile = File(appMusicDir, finalFileName)
            put(MediaStore.MediaColumns.DATA, targetFile.absolutePath)
        }
    }
    val collection = if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {
        MediaStore.Audio.Media.getContentUri(MediaStore.VOLUME_EXTERNAL_PRIMARY)
    } else {
        @Suppress("DEPRECATION")
        MediaStore.Audio.Media.EXTERNAL_CONTENT_URI
    }
    val uri = resolver.insert(collection, contentValues)
    uri?.let { outputUri ->
        try {
            resolver.openOutputStream(outputUri)?.use { outputStream ->
                sourceFile.inputStream().use { inputStream ->
                    inputStream.copyTo(outputStream)
                }
            }
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {
                contentValues.clear()
                contentValues.put(MediaStore.Audio.Media.IS_PENDING, 0)
                resolver.update(outputUri, contentValues, null, null)
            }
            return outputUri
        } catch (e: Exception) {
            Timber.e(e, "Failed to copy file to MediaStore for $finalFileName.")
            resolver.delete(outputUri, null, null)
        }
    }
    return null
}

private fun verifyMetadataInFile(
    file: File,
```

```
        expectedTitle: String,
        expectedArtist: String,
        expectedAlbum: String,
        shouldHaveArtwork: Boolean
    ) {
        var retriever: MediaMetadataRetriever? = null
        try {
            retriever = MediaMetadataRetriever()
            retriever.setDataSource(file.absolutePath)
            val foundTitle = retriever.extractMetadata(MediaMetadataRetriever.METADATA_KEY_TIT
            val foundArtist = retriever.extractMetadata(MediaMetadataRetriever.METADATA_KEY_AR
            val foundAlbum = retriever.extractMetadata(MediaMetadataRetriever.METADATA_KEY_ALB
            val hasArtwork = retriever.embeddedPicture?.isNotEmpty() == true
            val titleMatches = foundTitle == expectedTitle
            val artistMatches = foundArtist == expectedArtist
            val albumMatches = foundAlbum == expectedAlbum
            val artworkMatches = if (shouldHaveArtwork) hasArtwork else true
            if (titleMatches && artistMatches && albumMatches && artworkMatches) {
                Timber.i("Metadata Verification PASSED - All expected metadata found correctly
            } else {
                Timber.w("--- Metadata Verification FAILED ---")
                if (!titleMatches) Timber.w("--> Title Mismatch: Expected '$expectedTitle', Fo
                if (!artistMatches) Timber.w("--> Artist Mismatch: Expected '$expectedArtist',
                if (!albumMatches) Timber.w("--> Album Mismatch: Expected '$expectedAlbum', Fo
                if (!artworkMatches) Timber.w("--> Artwork Mismatch: Expected artwork to be pr
                Timber.w("-----------------------------------")
            }
        } catch (e: Exception) {
            Timber.e(e, "Metadata verification failed due to an exception.")
        } finally {
            try {
                retriever?.release()
            } catch (e: Exception) {
                Timber.w(e, "Failed to release MediaMetadataRetriever")
            }
        }
    }
}

// File: java\com\example\holodex\background\MetadataUpdateWorker.kt
// File: java/com/example/holodex/background/MetadataUpdateWorker.kt

package com.example.holodex.background

import android.content.Context
import android.graphics.Bitmap
import androidx.core.net.toUri
import androidx.hilt.work.HiltWorker
import androidx.work.CoroutineWorker
import androidx.work.WorkerParameters
import coil.imageLoader
import coil.request.ImageRequest
import coil.size.Size
import dagger.assisted.Assisted
import dagger.assisted.AssistedInject
```

```
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.withContext
import org.jaudiotagger.audio.AudioFileIO
import org.jaudiotagger.tag.FieldKey
import org.jaudiotagger.tag.images.ArtworkFactory
import timber.log.Timber
import java.io.ByteArrayOutputStream
import java.io.File
import java.io.IOException

@HiltWorker
class MetadataUpdateWorker @AssistedInject constructor(
    @Assisted private val context: Context,
    @Assisted workerParams: WorkerParameters,
) : CoroutineWorker(context, workerParams) {

    companion object {
        // Keys for WorkManager Data
        const val KEY_ITEM_ID = "ITEM_ID"
        const val KEY_FILE_URI = "FILE_URI"
        const val KEY_SONG_TITLE = "SONG_TITLE"
        const val KEY_ARTIST_NAME = "ARTIST_NAME"
        const val KEY_ALBUM_NAME = "ALBUM_NAME"
        const val KEY_ARTWORK_URI = "ARTWORK_URI"
        const val KEY_TRACK_NUMBER = "TRACK_NUMBER"
        private const val TAG = "MetadataUpdateWorker"
    }

    override suspend fun doWork(): Result = withContext(Dispatchers.IO) {
        val itemId = inputData.getString(KEY_ITEM_ID) ?: return@withContext Result.failure()
        val fileUriString = inputData.getString(KEY_FILE_URI) ?: return@withContext Result.fai
        val songTitle = inputData.getString(KEY_SONG_TITLE)
        val artistName = inputData.getString(KEY_ARTIST_NAME)
        val albumName = inputData.getString(KEY_ALBUM_NAME)
        val artworkUri = inputData.getString(KEY_ARTWORK_URI)
        val trackNumber = inputData.getInt(KEY_TRACK_NUMBER, -1)

        val tempFile = File(context.cacheDir, "metadata_update_${itemId}.m4a")

        try {
            val originalUri = fileUriString.toUri()
            Timber.d("$TAG: Starting metadata update for $itemId at URI: $originalUri")

            // 1. Copy original file to a temporary location for safe editing.
            context.contentResolver.openInputStream(originalUri)?.use { input ->
                tempFile.outputStream().use { output ->
                    input.copyTo(output)
                }
            } ?: throw IOException("Could not open input stream for original file.")

            // 2. Fetch new artwork if available.
            val artworkBytes = fetchArtwork(artworkUri)

            // 3. Use JAudioTagger to write new metadata to the temporary file.
            val audioFile = AudioFileIO.read(tempFile)
```

```
            val tag = audioFile.tagOrCreateAndSetDefault
            if (songTitle != null) tag.setField(FieldKey.TITLE, songTitle)
            if (artistName != null) tag.setField(FieldKey.ARTIST, artistName)
            if (albumName != null) tag.setField(FieldKey.ALBUM, albumName)
            if (trackNumber > 0) tag.setField(FieldKey.TRACK, trackNumber.toString())
            if (artworkBytes != null) {
                tag.deleteArtworkField()
                val artwork = ArtworkFactory.getNew()
                artwork.binaryData = artworkBytes
                artwork.mimeType = "image/jpeg"
                tag.setField(artwork)
            }
            AudioFileIO.write(audioFile)
            Timber.d("$TAG: Successfully wrote new tags to temp file.")

            // 4. Overwrite the original file with the newly tagged temporary file.
            context.contentResolver.openOutputStream(originalUri, "w")?.use { output ->
                tempFile.inputStream().use { input ->
                    input.copyTo(output)
                }
            } ?: throw IOException("Could not open output stream to overwrite original file.")

            Timber.i("$TAG: Successfully updated metadata for $itemId.")
            return@withContext Result.success()

        } catch (e: Exception) {
            Timber.e(e, "$TAG: Failed to update metadata for $itemId.")
            return@withContext Result.failure()
        } finally {
            tempFile.delete()
        }
    }

    private suspend fun fetchArtwork(url: String?): ByteArray? {
        if (url == null) return null
        val highResUrl = url.replace(Regex("""/100x100bb\.jpg$"""), "/1000x1000bb.jpg")
        return try {
            val request = ImageRequest.Builder(context).data(highResUrl).size(Size(1000, 1000)
                .allowHardware(false).build()
            (context.imageLoader.execute(request).drawable as? android.graphics.drawable.Bitma
                val stream = ByteArrayOutputStream()
                bitmap.compress(Bitmap.CompressFormat.JPEG, 95, stream)
                stream.toByteArray()
            }
        } catch (e: Exception) {
            Timber.e(e, "Failed to fetch artwork from $highResUrl for metadata update.")
            null
        }
    }
}

// File: java\com\example\holodex\background\MetadataWriter.kt
package com.example.holodex.background

import android.content.Context
```

```
import android.graphics.Bitmap
import android.graphics.drawable.BitmapDrawable
import coil.imageLoader
import coil.request.ImageRequest
import coil.size.Size
import org.jaudiotagger.audio.AudioFileIO
import org.jaudiotagger.tag.FieldKey
import org.jaudiotagger.tag.TagOptionSingleton
import org.jaudiotagger.tag.images.ArtworkFactory
import timber.log.Timber
import java.io.ByteArrayOutputStream
import java.io.File
import java.io.IOException
import javax.inject.Inject
import javax.inject.Singleton

@Singleton
class MetadataWriter @Inject constructor() {

    private val tag = "MetadataWriter"

    init {
        // Global configuration for JAudioTagger
        TagOptionSingleton.getInstance().isPadNumbers = false
    }

    suspend fun writeMetadata(
        context: Context,
        targetFile: File,
        itemId: String,
        songTitle: String,
        artistName: String,
        albumTitle: String,
        artworkUrl: String?,
        trackNumber: Int
    ) {
        try {
            Timber.d("$tag: Starting metadata tagging for ${targetFile.name}")
            val artworkData = fetchArtwork(context, artworkUrl)

            val audioFile = AudioFileIO.read(targetFile)
            val tag = audioFile.tagOrCreateAndSetDefault

            tag.setField(FieldKey.TITLE, songTitle)
            tag.setField(FieldKey.COMMENT, "holodex_item_id::$itemId")
            tag.setField(FieldKey.ARTIST, artistName)
            tag.setField(FieldKey.ALBUM, albumTitle)
            tag.setField(FieldKey.ALBUM_ARTIST, artistName)
            if (trackNumber > 0) {
                tag.setField(FieldKey.TRACK, trackNumber.toString())
            }

            if (artworkData != null) {
                tag.deleteArtworkField()
                val artwork = ArtworkFactory.getNew()
```

```
                artwork.binaryData = artworkData
                artwork.mimeType = "image/jpeg"
                artwork.description = "Cover"
                tag.setField(artwork)
            }
            AudioFileIO.write(audioFile)
            Timber.i("$tag: Successfully wrote tags to ${targetFile.name}")

        } catch (e: Exception) {
            throw IOException("JAudioTagger tagging failed.", e)
        }
    }

    private suspend fun fetchArtwork(context: Context, url: String?): ByteArray? {
        if (url == null) return null
        val highResUrl = url.replace(Regex("""/100x100bb\.jpg$"""), "/1000x1000bb.jpg")
        return try {
            val request = ImageRequest.Builder(context)
                .data(highResUrl)
                .size(Size(1000, 1000))
                .allowHardware(false)
                .build()
            (context.imageLoader.execute(request).drawable as? BitmapDrawable)?.bitmap?.let {
                val stream = ByteArrayOutputStream()
                bitmap.compress(Bitmap.CompressFormat.JPEG, 95, stream)
                stream.toByteArray()
            }
        } catch (e: Exception) {
            Timber.e(e, "$tag: Failed to fetch artwork from $highResUrl")
            null
        }
    }
}

// File: java\com\example\holodex\background\PlaylistSynchronizer.kt
// File: java/com/example/holodex/background/PlaylistSynchronizer.kt
package com.example.holodex.background

import com.example.holodex.data.db.SyncStatus
import com.example.holodex.data.repository.HolodexRepository
import timber.log.Timber
import javax.inject.Inject

class PlaylistSynchronizer @Inject constructor(
    private val repository: HolodexRepository,
    private val logger: SyncLogger
) : ISynchronizer {
    override val name: String = "PLAYLISTS"

    override suspend fun synchronize(): Boolean {
        logger.startSection(name)
        try {
            // --- PHASE 1: UPSTREAM (Client -> Server) ---
            logger.info("Phase 1: Pushing local changes to server...")
            repository.performUpstreamPlaylistDeletions(logger)
```

```
repository.performUpstreamPlaylistUpserts(logger)
logger.info("Phase 1 complete.")

// --- PHASE 2: FETCH ---
logger.info("Phase 2: Fetching remote and local states...")
val remotePlaylists = repository.getRemotePlaylists()
val localPlaylists = repository.getLocalPlaylists()
logger.info("  -> Fetched ${remotePlaylists.size} remote playlists and ${localPlay

// --- PHASE 3: METADATA RECONCILIATION ---
logger.info("Phase 3: Reconciling playlist metadata...")
val remoteMap = remotePlaylists.associateBy { it.serverId }
val localMap = localPlaylists.filter { it.serverId != null }.associateBy { it.serv

val newFromServer = remotePlaylists.filter { !localMap.containsKey(it.serverId) }
if (newFromServer.isNotEmpty()) {
    logger.info("  Found ${newFromServer.size} new playlists from server:")
    newFromServer.forEach { p -> logger.logItemAction(LogAction.DOWNSTREAM_INSERT_
    repository.insertNewSyncedPlaylists(newFromServer)
}

val deletedOnServer = localPlaylists.filter {
    it.serverId != null && it.syncStatus == SyncStatus.SYNCED && !remoteMap.contai
}
if (deletedOnServer.isNotEmpty()) {
    logger.info("  Found ${deletedOnServer.size} playlists deleted on server:")
    deletedOnServer.forEach { p -> logger.logItemAction(LogAction.DOWNSTREAM_DELET
    repository.deleteLocalPlaylists(deletedOnServer.map { it.playlistId })
}
logger.info("Phase 3 complete.")


// --- PHASE 4: CONTENT SYNCHRONIZATION (Timestamp-based) ---
logger.info("Phase 4: Reconciling song content for all user-owned playlists...")
val finalLocalPlaylists = repository.getLocalPlaylists().filter { it.serverId != n

for (localPlaylist in finalLocalPlaylists) {
    val remotePlaylist = remoteMap[localPlaylist.serverId] ?: continue
    if (localPlaylist.syncStatus == SyncStatus.DIRTY) {
        logger.info("  -> Skipping content sync for DIRTY playlist '${localPlaylis
        continue
    }

    val localTimestamp = localPlaylist.last_modified_at
    val remoteTimestamp = remotePlaylist.last_modified_at

    if (localTimestamp != null && remoteTimestamp != null && remoteTimestamp > loc
        logger.info("  -> Server is newer for playlist '${localPlaylist.name}'. Re

        // CRITICAL FIX: Capture count BEFORE any operations
        val itemCountBefore = repository.getPlaylistItemCount(localPlaylist.playli
        val localOnlyCountBefore = repository.getLocalOnlyItemCount(localPlaylist.

        Timber.tag("SYNC_DEBUG").i(
            "BEFORE sync: Playlist '${localPlaylist.name}' has $itemCountBefore to
```

```
                    )

                    // 1. Get the remote content first.
                    val remoteContent = repository.getRemotePlaylistContent(remotePlaylist.ser

                    // 2. Perform the SAFE reconciliation of items FIRST. This preserves local
                    //    This MUST happen before any metadata updates to avoid triggering cas
                    repository.reconcileLocalPlaylistItems(localPlaylist.playlistId, remoteCon

                    // 3. ONLY AFTER the items are safely reconciled, update the parent playli
                    //    This updates the timestamp to match the server, preventing re-sync o
                    repository.updateLocalPlaylistMetadata(localPlaylist.playlistId, remotePla

                    logger.info("    -> Reconciled local content with ${remoteContent.size} se

                } else {
                    logger.info("  -> Local state for playlist '${localPlaylist.name}' is curr
                }
            }
            logger.info("Phase 4 complete.")

            logger.endSection(name, success = true)
            return true
        } catch (e: Exception) {
            logger.error(e, "Playlist sync failed catastrophically.")
            logger.endSection(name, success = false)
            return false
        }
    }
}

// File: java\com\example\holodex\background\StarredPlaylistSynchronizer.kt
// File: java/com/example/holodex/background/StarredPlaylistSynchronizer.kt (NEW FILE)
package com.example.holodex.background

import com.example.holodex.data.repository.HolodexRepository
import javax.inject.Inject

class StarredPlaylistSynchronizer @Inject constructor(
    private val repository: HolodexRepository,
    private val logger: SyncLogger
) : ISynchronizer {
    override val name: String = "STARRED_PLAYLISTS"

    override suspend fun synchronize(): Boolean {
        logger.startSection(name)
        try {
            // --- PHASE 1: UPSTREAM ---
            logger.info("Phase 1: Pushing local changes to server...")
            repository.performUpstreamStarredPlaylistsSync(logger)
            logger.info("Phase 1 complete.")

            // --- PHASE 2: SMART SERVER SYNC ---
            logger.info("Phase 2: Fetching server state and reconciling local data...")
            val remoteStarred = repository.getRemoteStarredPlaylists()
```

```
            logger.info("  -> Fetched ${remoteStarred.size} starred playlists from server.")

            val unsyncedCount = repository.getLocalUnsyncedStarredPlaylistsCount()
            if (unsyncedCount > 0) {
                logger.info("  -> Preserving $unsyncedCount locally-changed items.")
            }

            val deletedCount = repository.deleteLocalSyncedStarredPlaylists()
            logger.info("  -> Wiped $deletedCount SYNCED items from local database.")

            repository.insertRemoteStarredPlaylistsAsSynced(remoteStarred)
            logger.info("  -> Paved local database with ${remoteStarred.size} fresh items from
            logger.info("Phase 2 complete.")

            logger.endSection(name, success = true)
            return true
        } catch (e: Exception) {
            logger.error(e, "Starred Playlists sync failed catastrophically.")
            logger.endSection(name, success = false)
            return false
        }
    }
}

// File: java\com\example\holodex\background\SyncCoordinator.kt
// File: java/com/example/holodex/background/SyncCoordinator.kt (NEW FILE)
package com.example.holodex.background

import kotlinx.coroutines.async
import kotlinx.coroutines.coroutineScope
import javax.inject.Inject
import javax.inject.Singleton

@Singleton
class SyncCoordinator @Inject constructor(
    private val synchronizers: Set<@JvmSuppressWildcards ISynchronizer>,
    private val logger: SyncLogger
) {
    suspend fun run(): Boolean {
        logger.info("===== Starting Full Synchronization Run =====")
        var allSucceeded = true

        try {
            coroutineScope {
                val results = synchronizers.map { synchronizer ->
                    async {
                        // We will run some syncs sequentially if they are dependent in the fu
                        // For now, all are independent.
                        synchronizer.synchronize()
                    }
                }.map { it.await() }

                if (results.contains(false)) {
                    allSucceeded = false
                }
```

```
            }
        } catch (e: Exception) {
            logger.error(e, "The SyncCoordinator caught an unhandled exception.")
            allSucceeded = false
        }

        logger.info("===== Full Synchronization Run Finished. Overall Success: $allSucceeded =
        return allSucceeded
    }
}

// File: java\com\example\holodex\background\SyncLogger.kt
// File: java/com/example/holodex/background/SyncLogger.kt
package com.example.holodex.background

import timber.log.Timber
import javax.inject.Inject
import javax.inject.Singleton

enum class LogAction {
    // Upstream
    UPSTREAM_DELETE_SUCCESS,
    UPSTREAM_DELETE_FAILED,
    UPSTREAM_UPSERT_SUCCESS,
    UPSTREAM_UPSERT_FAILED,
    // Downstream
    DOWNSTREAM_INSERT_LOCAL,
    DOWNSTREAM_DELETE_LOCAL,
    DOWNSTREAM_UPDATE_LOCAL,
    // Reconciliation
    RECONCILE_SKIP
}

@Singleton
class SyncLogger @Inject constructor() {
    private val TAG = "SYNC"

    fun startSection(name: String) {
        Timber.tag(TAG).i("====== STARTING $name SYNC ======")
    }

    fun endSection(name: String, success: Boolean) {
        val status = if (success) "SUCCESSFUL" else "FAILED"
        Timber.tag(TAG).i("====== $name SYNC $status ======")
    }

    fun info(message: String) {
        Timber.tag(TAG).i(message)
    }

    fun warning(message: String) {
        Timber.tag(TAG).w(message)
    }

    fun error(throwable: Throwable, message: String) {
```

```
        Timber.tag(TAG).e(throwable, message)
    }

    fun logItemAction(
        action: LogAction,
        itemName: String?,
        localId: Long?,
        serverId: String?,
        reason: String? = null
    ) {
        val formattedName = "'${itemName ?: "Unknown"}'"
        val formattedIds = "(LID: ${localId ?: "N/A"}, SID: ${serverId ?: "N/A"})"
        val formattedReason = reason?.let { " | Reason: $it" } ?: ""
        Timber.tag(TAG).d("-> [${action.name}] Item: $formattedName $formattedIds$formattedRea
    }
}

// File: java\com\example\holodex\background\SyncWorker.kt
// File: java/com/example/holodex/background/SyncWorker.kt (MODIFIED)
package com.example.holodex.background

import android.content.Context
import androidx.hilt.work.HiltWorker
import androidx.work.CoroutineWorker
import androidx.work.WorkerParameters
import com.example.holodex.auth.TokenManager
import dagger.assisted.Assisted
import dagger.assisted.AssistedInject
import timber.log.Timber

@HiltWorker
class SyncWorker @AssistedInject constructor(
    @Assisted context: Context,
    @Assisted params: WorkerParameters,
    // --- MODIFICATION: Inject SyncCoordinator, not HolodexRepository ---
    private val syncCoordinator: SyncCoordinator,
    private val tokenManager: TokenManager
) : CoroutineWorker(context, params) {

    companion object {
        const val TAG = "SyncWorker"
        const val MAX_RUN_ATTEMPTS = 3
    }

    override suspend fun doWork(): Result {
        Timber.i("$TAG: Starting synchronization work using SyncCoordinator.")

        if (tokenManager.getJwt() == null) {
            Timber.i("$TAG: User not logged in. Sync work is not required. Finishing successfu
            return Result.success()
        }

        return try {
            // --- MODIFICATION: Call the coordinator ---
            val success = syncCoordinator.run()
```

```
            if (success) {
                Timber.i("$TAG: Synchronization work completed successfully.")
                Result.success()
            } else {
                // The coordinator's log will have the details. We just handle the retry logic
                Timber.w("$TAG: SyncCoordinator reported failure on attempt $runAttemptCount."
                if (runAttemptCount < MAX_RUN_ATTEMPTS) {
                    Timber.w("$TAG: Scheduling a retry.")
                    Result.retry()
                } else {
                    Timber.e("$TAG: Maximum retry attempts reached. Failing the work.")
                    Result.failure()
                }
            }
        } catch (e: Exception) {
            Timber.e(e, "$TAG: SyncWorker caught an unhandled exception.")
            if (runAttemptCount < MAX_RUN_ATTEMPTS) Result.retry() else Result.failure()
        }
    }
}

// File: java\com\example\holodex\data\AppPreferences.kt
package com.example.holodex.data

// Moved from SettingsViewModel.kt to be globally accessible
object AppPreferenceConstants {
    // Image Quality
    const val PREF_IMAGE_QUALITY = "pref_image_quality"
    const val IMAGE_QUALITY_AUTO = "AUTO"
    const val IMAGE_QUALITY_MEDIUM = "MEDIUM"
    const val IMAGE_QUALITY_LOW = "LOW"

    // Audio Quality
    const val PREF_AUDIO_QUALITY = "pref_audio_quality"
    const val AUDIO_QUALITY_BEST = "BEST"
    const val AUDIO_QUALITY_STANDARD = "STANDARD"
    const val AUDIO_QUALITY_SAVER = "SAVER"

    // List Loading
    const val PREF_LIST_LOADING_CONFIG = "pref_list_loading_config"
    const val LIST_LOADING_NORMAL = "NORMAL"
    const val LIST_LOADING_REDUCED = "REDUCED"
    const val LIST_LOADING_MINIMAL = "MINIMAL"

    // Buffering
    const val PREF_BUFFERING_STRATEGY = "pref_buffering_strategy"
    const val BUFFERING_STRATEGY_AGGRESSIVE = "AGGRESSIVE_START"
    const val BUFFERING_STRATEGY_BALANCED = "BALANCED"
    const val BUFFERING_STRATEGY_STABLE = "STABLE_PLAYBACK"

    const val PREF_AUTOPLAY_NEXT_VIDEO = "pref_autoplay_next_video"
    const val PREF_DOWNLOAD_LOCATION = "pref_download_location_uri"
}
```

```
object ThemePreference {
    const val KEY = "app_theme_preference"
    const val LIGHT = "LIGHT"
    const val DARK = "DARK"
    const val SYSTEM = "SYSTEM"
}

// File: java\com\example\holodex\data\api\AuthenticatedMusicdexApiService.kt
// File: java/com/example/holodex/data/api/AuthenticatedMusicdexApiService.kt

package com.example.holodex.data.api

import com.example.holodex.data.model.discovery.DiscoveryResponse
import com.example.holodex.data.model.discovery.FullPlaylist
import com.example.holodex.data.model.discovery.PlaylistStub
import retrofit2.Response
import retrofit2.http.Body
import retrofit2.http.DELETE
import retrofit2.http.GET
import retrofit2.http.HTTP
import retrofit2.http.PATCH
import retrofit2.http.POST
import retrofit2.http.Path
import retrofit2.http.Query

data class LikeRequest(val song_id: String)
// The incorrect DeleteLikeRequest class is removed.

data class StarPlaylistRequest(val playlist_id: String)

data class LikedSongApiDto(
    val id: String, // The song's unique ID
    val channel_id: String,
    val video_id: String,
    val name: String,
    val start: Int,
    val end: Int,
    val original_artist: String?,
    val art: String?,
    val channel: ApiChannelStub?
)

data class ApiChannelStub(
    val name: String,
    val english_name: String?,
    val photo: String?
)

data class PaginatedLikesResponse(
    val page_count: Int,
    val content: List<LikedSongApiDto>
)

data class FavoriteChannelApiDto(
    val id: String,
```

```
    val name: String? = null,
    val english_name: String? = null,
    val photo: String?,
    val org: String?,
    val twitter: String?
)
data class PatchOperation(
    val op: String, // "add" or "remove"
    val channel_id: String
)
typealias PatchFavoriteChannelsRequest = List<PatchOperation>

/**
 * Defines the authenticated endpoints for the music.holodex.net API.
 */
interface AuthenticatedMusicdexApiService {

    @GET("api/v2/musicdex/discovery/favorites")
    suspend fun getDiscoveryForFavorites(): Response<DiscoveryResponse>

    @GET("api/v2/musicdex/like")
    suspend fun getLikes(
        @Query("since") sinceTimestamp: Long? = null,
        @Query("page") page: Int? = 1,
        @Query("paginated") paginated: Boolean = true
    ): Response<PaginatedLikesResponse>

    @GET("api/v2/musicdex/like/check")
    suspend fun checkLikes(@Query("song_id") songIds: String): Response<List<Boolean>>

    @POST("api/v2/musicdex/like")
    suspend fun addLike(@Body request: LikeRequest): Response<Unit>

    // --- FIX: Change to a proper DELETE request with a body ---
    @HTTP(method = "DELETE", path = "api/v2/musicdex/like", hasBody = true)
    suspend fun deleteLike(@Body request: LikeRequest): Response<Unit>
    // --- END OF FIX ---

    @PATCH("api/v2/users/favorites")
    suspend fun patchFavoriteChannels(@Body request: PatchFavoriteChannelsRequest): Response<L

    @GET("api/v2/musicdex/history/{songId}")
    suspend fun trackSongInHistory(@Path("songId") songId: String): Response<Unit>

    @GET("api/v2/musicdex/playlist/{playlistId}")
    suspend fun getPlaylistContent(@Path(value = "playlistId", encoded = true) playlistId: Str

    @GET("api/v2/musicdex/playlist")
    suspend fun getMyPlaylists(): Response<List<PlaylistDto>>

    @POST("api/v2/musicdex/playlist")
    suspend fun createOrUpdatePlaylist(@Body playlist: PlaylistUpdateRequest): Response<List<P

    @DELETE("api/v2/musicdex/playlist/{playlistId}")
    suspend fun deletePlaylist(@Path("playlistId") playlistId: String): Response<Unit>
```

```
    @GET("api/v2/musicdex/playlist/{playlistId}/{songId}")
    suspend fun addSongToPlaylist(
        @Path("playlistId") playlistId: String,
        @Path("songId") songId: String
    ): Response<Unit>

    @DELETE("api/v2/musicdex/playlist/{playlistId}/{songId}")
    suspend fun removeSongFromPlaylist(
        @Path("playlistId") playlistId: String,
        @Path("songId") songId: String
    ): Response<Unit>

    @GET("api/v2/musicdex/star")
    suspend fun getStarredPlaylists(): Response<List<PlaylistStub>>

    @POST("api/v2/musicdex/star")
    suspend fun starPlaylist(@Body request: StarPlaylistRequest): Response<Unit>

    @HTTP(method = "DELETE", path = "api/v2/musicdex/star", hasBody = true)
    suspend fun unstarPlaylist(@Body request: StarPlaylistRequest): Response<Unit>
}

// File: java\com\example\holodex\data\api\HolodexApiService.kt
// File: java\com\example\holodex\data\api\HolodexApiService.kt

package com.example.holodex.data.api

import com.example.holodex.data.model.HolodexVideoItem
import com.example.holodex.data.model.PaginatedVideosResponse
import com.example.holodex.data.model.VideoSearchRequest
import com.example.holodex.data.model.discovery.ChannelDetails
import com.example.holodex.data.model.discovery.MusicdexSong
import com.google.gson.annotations.SerializedName
import retrofit2.Response
import retrofit2.http.Body
import retrofit2.http.GET
import retrofit2.http.POST
import retrofit2.http.Path
import retrofit2.http.Query

// --- Request/Response Models ---
data class LoginRequest(val service: String, val token: String)
data class LoginResponse(val jwt: String) // Assuming user object is handled separately
data class LatestSongsRequest(
    val channel_id: String? = null,
    val paginated: Boolean = true,
    val limit: Int = 25,
    val offset: Int = 0
)

data class PaginatedSongsResponse(
    val total: String,
    val items: List<MusicdexSong>
) {
```

```
    fun getTotalAsInt(): Int? = total.toIntOrNull()
}

// Represents the structure from /statics/orgs.json
data class Organization(
    val name: String,
    @SerializedName("name_jp") val nameJp: String?,
    val short: String?
)

data class PaginatedChannelsResponse(
    @SerializedName("total") val total: String?,
    @SerializedName("items") val items: List<ChannelDetails>
) {
    fun getTotalAsInt(): Int? = total?.toIntOrNull()
}

interface HolodexApiService {

    @GET("api/v2/videos/{videoId}")
    suspend fun getVideoWithSongs(
        @Path("videoId") videoId: String,
        @Query("include") include: String = "songs,live_info,description",
        @Query("lang") lang: String = "en",
        @Query("c") comments: String? = null
    ): Response<HolodexVideoItem>
    @GET("api/v2/videos")
    suspend fun getChannelVideos(
        @Query("channel_id") channelId: String,
        @Query("type") type: String = "stream",
        @Query("include") include: String = "live_info",
        @Query("limit") limit: Int = 10,
        @Query("status") status: String? = null
    ): Response<List<HolodexVideoItem>>
    @POST("api/v2/search/videoSearch")
    suspend fun searchVideosAdvanced(
        @Body request: VideoSearchRequest
    ): Response<PaginatedVideosResponse>

    @POST("api/v2/user/login")
    suspend fun login(@Body request: LoginRequest): Response<LoginResponse>

    @GET("api/v2/user/refresh")
    suspend fun refreshUser(): Response<LoginResponse> // Assuming it returns a new JWT

    @GET("api/v2/channels/{channelId}")
    suspend fun getChannelDetails(@Path("channelId") channelId: String): Response<ChannelDetai

    @GET("api/v2/channels")
    suspend fun getChannels(
        @Query("type") type: String = "vtuber",
        @Query("org") organization: String,
        @Query("limit") limit: Int,
        @Query("offset") offset: Int,
        @Query("sort") sort: String = "suborg"
```

```
    ): Response<List<ChannelDetails>>

    @GET("api/v2/songs/hot")
    suspend fun getHotSongs(
        @Query("org") organization: String? = null,
        @Query("channel_id") channelId: String? = null
    ): Response<List<MusicdexSong>>

    @POST("api/v2/songs/latest")
    suspend fun getLatestSongs(@Body request: LatestSongsRequest): Response<PaginatedSongsResp


    @GET("/statics/orgs.json")
    suspend fun getOrganizations(): Response<List<Organization>>

    @GET("api/v2/users/favorites")
    suspend fun getFavoriteChannels(): Response<List<FavoriteChannelApiDto>>
}

// File: java\com\example\holodex\data\api\MusicdexApiService.kt
// File: java\com\example\holodex\data\api\MusicdexApiService.kt

package com.example.holodex.data.api

import com.example.holodex.data.model.discovery.DiscoveryResponse
import com.example.holodex.data.model.discovery.FullPlaylist
import com.example.holodex.data.model.discovery.MusicdexSong
import com.example.holodex.data.model.discovery.PlaylistStub
import retrofit2.Response
import retrofit2.http.Body
import retrofit2.http.GET
import retrofit2.http.POST
import retrofit2.http.Path
import retrofit2.http.Query

// --- Request/Response Models ---
data class PlaylistListResponse(
    val total: Int,
    val items: List<PlaylistStub>
)

data class ElasticsearchRequest(
    val q: String = "*",
    val query_by: String,
    val sort_by: String,
    val facet_by: String,
    val page: Int = 1,
    val per_page: Int = 25
    // Add other fields as needed for advanced search
)

data class ElasticsearchResponse<T>(
    val found: Int,
    val page: Int,
    val hits: List<Hit<T>>?
```

```
)

data class Hit<T>(
    val document: T
)

/**
 * Defines the public, non-authenticated endpoints for the music.holodex.net API.
 * Requires the X-APIKEY via an interceptor.
 */
interface MusicdexApiService {

    @GET("api/v2/musicdex/discovery/org/{org}")
    suspend fun getDiscoveryForOrg(@Path("org") organization: String): Response<DiscoveryRespo

    @GET("api/v2/musicdex/discovery/channel/{channelId}")
    suspend fun getDiscoveryForChannel(@Path("channelId") channelId: String): Response<Discove

    @GET("api/v2/musicdex/playlist/{playlistId}")
    suspend fun getPlaylistContent(@Path(value = "playlistId", encoded = true) playlistId: Str

    @GET("api/v2/musicdex/radio/{radioId}")
    suspend fun getRadioContent(
        @Path("radioId") radioId: String,
        @Query("offset") offset: Int = 0
    ): Response<FullPlaylist>

    @GET("api/v2/musicdex/discovery/org/{org}/playlists")
    suspend fun getOrgPlaylists(
        @Path("org") org: String,
        @Query("type") type: String,
        @Query("offset") offset: Int,
        @Query("limit") limit: Int
    ): Response<PlaylistListResponse>


    // --- NEW ENDPOINT for Goal 2.1 ---
    @POST("api/v2/musicdex/elasticsearch/search")
    suspend fun searchElasticsearch(
        @Body request: ElasticsearchRequest
    ): Response<ElasticsearchResponse<MusicdexSong>>
}

// File: java\com\example\holodex\data\api\PlaylistDto.kt
// File: java/com/example/holodex/data/api/PlaylistDto.kt (NEW FILE)
package com.example.holodex.data.api

import com.google.gson.annotations.SerializedName

/**
 * A dedicated Data Transfer Object (DTO) for deserializing playlist data from the Holodex API
 * The property names here EXACTLY match the JSON fields from the server.
 */
data class PlaylistDto(
    @SerializedName("id")
```

```
    val id: String,

    @SerializedName("title")
    val title: String?,

    @SerializedName("description")
    val description: String?,

    @SerializedName("owner")
    val owner: Long?,

    @SerializedName("type")
    val type: String?,

    @SerializedName("created_at")
    val createdAt: String?,

    @SerializedName("updated_at")
    val updatedAt: String?
)

// File: java\com\example\holodex\data\api\PlaylistRequestDtos.kt
// File: java/com/example/holodex/data/api/PlaylistRequestDtos.kt (NEW FILE)
package com.example.holodex.data.api

import com.google.gson.annotations.SerializedName

/**
 * A dedicated Data Transfer Object (DTO) for creating or updating a playlist via the API.
 * This class ONLY contains fields the server expects, solving the 500 error caused by
 * sending extra client-side fields like `is_deleted`.
 */
data class PlaylistUpdateRequest(
    @SerializedName("id")
    val id: String?, // Null when creating, non-null when updating

    @SerializedName("owner")
    val owner: Long,

    @SerializedName("title")
    val title: String?,

    @SerializedName("description")
    val description: String?,

    @SerializedName("type")
    val type: String = "ugp",

    /**
     * The complete list of song UUIDs. When sent, this will OVERWRITE the existing content.
     */
    @SerializedName("content")
    val content: List<String>
)
```

```
// File: java\com\example\holodex\data\cache\BrowseListCache.kt
package com.example.holodex.data.cache

import androidx.collection.LruCache
import com.example.holodex.data.db.BrowsePageDao
import com.example.holodex.data.db.CachedBrowsePage
import com.example.holodex.data.model.HolodexVideoItem
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.withContext
import timber.log.Timber
import java.util.concurrent.TimeUnit

class BrowseListCache(
    private val browsePageDao: BrowsePageDao,
    private val maxMemoryPages: Int = 5, // Max number of pages to keep in memory
    private val cacheValidityMs: Long = TimeUnit.HOURS.toMillis(1), // How long a cache entry
    private val staleValidityMs: Long = TimeUnit.DAYS.toMillis(1) // How long stale data can b
) {
    private data class CachePageEntry(
        val result: FetcherResult<HolodexVideoItem>, // Store HolodexVideoItem directly
        val timestamp: Long = System.currentTimeMillis()
    )

    // LruCache stores pages by their string key.
    private val memoryCache = object : LruCache<String, CachePageEntry>(maxMemoryPages) {
        override fun sizeOf(key: String, value: CachePageEntry): Int {
            // For LruCache, size is often 1 per entry if maxMemoryPages is the primary constr
            // If it were based on actual memory footprint, it'd be more complex.
            return 1 // Each entry counts as 1 towards maxMemoryPages
        }
    }

    private fun isFresh(timestamp: Long): Boolean {
        return System.currentTimeMillis() - timestamp < cacheValidityMs
    }

    private fun isStaleButAcceptable(timestamp: Long): Boolean {
        return System.currentTimeMillis() - timestamp < staleValidityMs
    }

    suspend fun get(key: BrowseCacheKey): FetcherResult<HolodexVideoItem>? = withContext(Dispa
        val stringKey = key.stringKey()
        memoryCache[stringKey]?.let { entry ->
            if (isFresh(entry.timestamp)) {
                Timber.d("BrowseListCache: Memory HIT (fresh) for key: $stringKey")
                return@withContext entry.result
            } else {
                Timber.d("BrowseListCache: Memory STALE for key: $stringKey, removing.")
                memoryCache.remove(stringKey) // Remove stale entry from memory
            }
        }

        // Check disk cache
        browsePageDao.getPage(stringKey)?.let { cachedPage ->
            if (isFresh(cachedPage.timestamp)) {
```

```
            Timber.d("BrowseListCache: Disk HIT (fresh) for key: $stringKey")
            val result = FetcherResult(cachedPage.data, cachedPage.totalAvailable)
            memoryCache.put(stringKey, CachePageEntry(result, cachedPage.timestamp)) // Po
            return@withContext result
        } else {
            // Data on disk is stale (older than cacheValidityMs) but not necessarily expi
            // We won't use it as "fresh" here. If network fails, getStale might pick it u
            Timber.d("BrowseListCache: Disk STALE for key: $stringKey. Will rely on networ
            // Optionally, one could delete it here if it's also older than staleValidityM
            // but cleanupExpired is a more global approach.
        }
    }
    Timber.d("BrowseListCache: Cache MISS for key: $stringKey")
    null
}

suspend fun getStale(key: BrowseCacheKey): FetcherResult<HolodexVideoItem>? = withContext(
    val stringKey = key.stringKey()
    // Try memory cache first (even if stale but acceptable)
    memoryCache[stringKey]?.let { entry ->
        if (isStaleButAcceptable(entry.timestamp)) {
            Timber.d("BrowseListCache: Memory HIT (stale but acceptable) for key: $stringK
            return@withContext entry.result
        }
    }

    // Try disk cache (stale but acceptable)
    browsePageDao.getPage(stringKey)?.let { cachedPage ->
        if (isStaleButAcceptable(cachedPage.timestamp)) {
            Timber.d("BrowseListCache: Disk HIT (stale but acceptable) for key: $stringKey
            // Note: Not re-populating memory cache with stale data from disk here,
            // as 'get' should be the one populating with fresh data.
            return@withContext FetcherResult(cachedPage.data, cachedPage.totalAvailable)
        } else {
            // Stale data on disk is too old even for fallback, remove it.
            Timber.d("BrowseListCache: Disk EXPIRED (too stale) for key: $stringKey, delet
            browsePageDao.deletePage(stringKey)
        }
    }
    Timber.d("BrowseListCache: Stale cache MISS for key: $stringKey")
    null
}

suspend fun store(key: BrowseCacheKey, result: FetcherResult<HolodexVideoItem>) = withCont
    val stringKey = key.stringKey()
    val currentTimestamp = System.currentTimeMillis()
    val entry = CachePageEntry(result, currentTimestamp)
    memoryCache.put(stringKey, entry)

    val cachedPage = CachedBrowsePage(
        pageKey = stringKey,
        data = result.data,
        totalAvailable = result.totalAvailable,
        timestamp = currentTimestamp
    )
```

```
        browsePageDao.insertPage(cachedPage)
        Timber.d("BrowseListCache: Stored data for key: $stringKey, items: ${result.data.size}
    }

    suspend fun invalidate(key: BrowseCacheKey) = withContext(Dispatchers.IO) {
        val stringKey = key.stringKey()
        memoryCache.remove(stringKey)
        browsePageDao.deletePage(stringKey)
        Timber.d("BrowseListCache: Invalidated data for key: $stringKey")
    }

    suspend fun clear() = withContext(Dispatchers.IO) {
        memoryCache.evictAll()
        browsePageDao.deleteAllBrowsePages()
        Timber.d("BrowseListCache: Cleared all browse cache data (memory and disk).")
    }

    suspend fun cleanupExpiredEntries() = withContext(Dispatchers.IO) {
        val tooOldTimestamp = System.currentTimeMillis() - staleValidityMs
        browsePageDao.deleteExpiredBrowsePages(tooOldTimestamp)
        // Memory cache entries are evicted by LRU or when found stale during 'get'.
        // Could also iterate memoryCache.snapshot().keys and remove based on timestamp if str
        Timber.d("BrowseListCache: Cleaned up expired disk entries older than $staleValidityMs
    }
}

// File: java\com\example\holodex\data\cache\CacheKey.kt
package com.example.holodex.data.cache

import com.example.holodex.viewmodel.state.BrowseFilterState
import com.google.gson.Gson

/**
 * Base interface for cache keys to ensure a string representation for Room.
 */
interface CacheKey {
    fun stringKey(): String
}

data class BrowseCacheKey(
    val filters: BrowseFilterState,
    val pageOffset: Int
) : CacheKey {
    override fun stringKey(): String {
        val gson = Gson()
        val filterJson = gson.toJson(mapOf(
            "preset" to filters.selectedViewPreset.name,
            "org" to filters.selectedOrganization,
            "topic" to filters.selectedPrimaryTopic,
            "sortField" to filters.sortField.apiValue,
            "sortOrder" to filters.sortOrder.apiValue,
            "status" to filters.status,
            "maxUpcomingHours" to filters.maxUpcomingHours
        ))
        return "browse_${filterJson}_offset=$pageOffset"
```

```
    }
}

data class SearchCacheKey(
    val query: String,
    val pageOffset: Int
) : CacheKey {
    override fun stringKey(): String {
        return "search_query=${query.trim().replace(" ", "_").take(100)}_offset=$pageOffset"
    }
}

// Add other keys as needed, e.g., for Favorites, LikedSegments, PlaylistItems if they use sim
// data class FavoritesCacheKey(val itemType: LikedItemType, val pageOffset: Int) : CacheKey {

// File: java\com\example\holodex\data\cache\CachePolicyAndException.kt
package com.example.holodex.data.cache

enum class CachePolicy {
    /**
     * Try to fetch from cache first.
     * If cache miss or expired, fetch from network.
     * If network fails, attempt to use stale cache.
     */
    CACHE_FIRST,

    /**
     * Try to fetch from network first.
     * If network fails, attempt to use cache (fresh or stale).
     */
    NETWORK_FIRST,

    /**
     * Only fetch from cache. Fails if not found or expired.
     * Might have a sub-policy to allow stale.
     */
    CACHE_ONLY,

    /**
     * Only fetch from network. Do not use cache.
     */
    NETWORK_ONLY
}

sealed class CacheException(message: String, cause: Throwable? = null) : Exception(message, ca
    class NotFound(message: String) : CacheException(message)
    class Expired(message: String) : CacheException(message)
    class StorageError(message: String, cause: Throwable) : CacheException(message, cause)
    class NetworkError(message: String, cause: Throwable?) : CacheException(message, cause)
}

// File: java\com\example\holodex\data\cache\FetcherResult.kt
// File: java/com/example/holodex/data/cache/FetcherResult.kt
package com.example.holodex.data.cache
```

```
data class FetcherResult<V>(
    val data: List<V>,
    val totalAvailable: Int?,
    val nextPageOffset: Int? = null,
    val nextPageCursor: Any? = null // Generic cursor for NewPipe's Page object
)

// File: java\com\example\holodex\data\cache\SearchListCache.kt
package com.example.holodex.data.cache

import androidx.collection.LruCache
import com.example.holodex.data.db.CachedSearchPage
import com.example.holodex.data.db.SearchPageDao
import com.example.holodex.data.model.HolodexVideoItem
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.withContext
import timber.log.Timber
import java.util.concurrent.TimeUnit

class SearchListCache(
    private val searchPageDao: SearchPageDao,
    private val maxMemoryPages: Int = 3, // Search might be less frequently revisited per spec
    private val cacheValidityMs: Long = TimeUnit.MINUTES.toMillis(30),
    private val staleValidityMs: Long = TimeUnit.HOURS.toMillis(12)
) {
    private data class CachePageEntry(
        val result: FetcherResult<HolodexVideoItem>,
        val timestamp: Long = System.currentTimeMillis()
    )

    private val memoryCache = object : LruCache<String, CachePageEntry>(maxMemoryPages) {
        override fun sizeOf(key: String, value: CachePageEntry): Int = 1
    }

    private fun isFresh(timestamp: Long): Boolean = System.currentTimeMillis() - timestamp < c
    private fun isStaleButAcceptable(timestamp: Long): Boolean = System.currentTimeMillis() -

    suspend fun get(key: SearchCacheKey): FetcherResult<HolodexVideoItem>? = withContext(Dispa
        val stringKey = key.stringKey()
        memoryCache[stringKey]?.let { entry ->
            if (isFresh(entry.timestamp)) {
                Timber.d("SearchListCache: Memory HIT (fresh) for key: $stringKey")
                return@withContext entry.result
            } else {
                Timber.d("SearchListCache: Memory STALE for key: $stringKey, removing.")
                memoryCache.remove(stringKey)
            }
        }

        searchPageDao.getPage(stringKey)?.let { cachedPage ->
            if (isFresh(cachedPage.timestamp)) {
                Timber.d("SearchListCache: Disk HIT (fresh) for key: $stringKey")
                val result = FetcherResult(cachedPage.data, cachedPage.totalAvailable)
                memoryCache.put(stringKey, CachePageEntry(result, cachedPage.timestamp))
                return@withContext result
```

```
        }
    }
    Timber.d("SearchListCache: Cache MISS for key: $stringKey")
    null
}

suspend fun getStale(key: SearchCacheKey): FetcherResult<HolodexVideoItem>? = withContext(
    val stringKey = key.stringKey()
    memoryCache[stringKey]?.let { entry ->
        if (isStaleButAcceptable(entry.timestamp)) {
            Timber.d("SearchListCache: Memory HIT (stale acceptable) for key: $stringKey")
            return@withContext entry.result
        }
    }

    searchPageDao.getPage(stringKey)?.let { cachedPage ->
        if (isStaleButAcceptable(cachedPage.timestamp)) {
            Timber.d("SearchListCache: Disk HIT (stale acceptable) for key: $stringKey")
            return@withContext FetcherResult(cachedPage.data, cachedPage.totalAvailable)
        } else {
            Timber.d("SearchListCache: Disk EXPIRED (too stale) for key: $stringKey, delet
            searchPageDao.deletePage(stringKey)
        }
    }
    Timber.d("SearchListCache: Stale cache MISS for key: $stringKey")
    null
}

suspend fun store(key: SearchCacheKey, result: FetcherResult<HolodexVideoItem>) = withCont
    val stringKey = key.stringKey()
    val currentTimestamp = System.currentTimeMillis()
    val entry = CachePageEntry(result, currentTimestamp)
    memoryCache.put(stringKey, entry)

    val cachedPage = CachedSearchPage(
        pageKey = stringKey,
        data = result.data,
        totalAvailable = result.totalAvailable,
        timestamp = currentTimestamp
    )
    searchPageDao.insertPage(cachedPage)
    Timber.d("SearchListCache: Stored data for key: $stringKey, items: ${result.data.size}
}

suspend fun invalidate(key: SearchCacheKey) = withContext(Dispatchers.IO) {
    val stringKey = key.stringKey()
    memoryCache.remove(stringKey)
    searchPageDao.deletePage(stringKey)
    Timber.d("SearchListCache: Invalidated data for key: $stringKey")
}

suspend fun clear() = withContext(Dispatchers.IO) {
    memoryCache.evictAll()
    searchPageDao.deleteAllSearchPages()
    Timber.d("SearchListCache: Cleared all search cache data.")
```

```
    }

    suspend fun cleanupExpiredEntries() = withContext(Dispatchers.IO) {
        val tooOldTimestamp = System.currentTimeMillis() - staleValidityMs
        searchPageDao.deleteExpiredSearchPages(tooOldTimestamp)
        Timber.d("SearchListCache: Cleaned up expired disk entries older than $staleValidityMs
    }
}

// File: java\com\example\holodex\data\db\AppDatabase.kt
package com.example.holodex.data.db

import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase
import androidx.room.TypeConverter
import androidx.room.TypeConverters
import androidx.room.migration.Migration
import androidx.sqlite.db.SupportSQLiteDatabase
import com.example.holodex.playback.data.model.PlaybackDao
import com.example.holodex.playback.data.model.PlaybackQueueRefEntity
import com.example.holodex.playback.data.model.PlaybackStateEntity

@Database(
    entities = [
        // --- NEW UNIFIED ENTITIES ---
        UnifiedMetadataEntity::class,
        UserInteractionEntity::class,
        PlaybackStateEntity::class,
        PlaybackQueueRefEntity::class,
        CachedVideoEntity::class,
        CachedSongEntity::class,
        PlaylistEntity::class,
        PlaylistItemEntity::class,
        CachedBrowsePage::class,
        CachedSearchPage::class,
        ParentVideoMetadataEntity::class,
        CachedDiscoveryResponse::class,
        SyncMetadataEntity::class,
        StarredPlaylistEntity::class
    ],
    version = 26, // INCREMENTED VERSION
    exportSchema = true
)
@TypeConverters(
    HolodexSongListConverter::class,
    LikedItemTypeConverter::class,
    HolodexVideoItemListConverter::class,
    StringListConverter::class,
    DiscoveryResponseConverter::class,
    SyncStatusConverter::class
)
abstract class AppDatabase : RoomDatabase() {
```

```
// --- NEW DAO ---
abstract fun unifiedDao(): UnifiedDao
abstract fun playlistDao(): PlaylistDao
abstract fun browsePageDao(): BrowsePageDao
abstract fun searchPageDao(): SearchPageDao
abstract fun parentVideoMetadataDao(): ParentVideoMetadataDao
abstract fun videoDao(): VideoDao
abstract fun discoveryDao(): DiscoveryDao
abstract fun syncMetadataDao(): SyncMetadataDao
abstract fun starredPlaylistDao(): StarredPlaylistDao
abstract fun playbackDao(): PlaybackDao

companion object {
    @Volatile
    private var INSTANCE: AppDatabase? = null

    val MIGRATION_18_19: Migration = object : Migration(18, 19) {
        override fun migrate(db: SupportSQLiteDatabase) { }
    }
    val MIGRATION_19_20: Migration = object : Migration(19, 20) {
        override fun migrate(db: SupportSQLiteDatabase) {
            db.execSQL("ALTER TABLE `playlist_items` ADD COLUMN `is_local_only` INTEGER NO
            db.execSQL("CREATE TABLE IF NOT EXISTS `local_favorites` (`itemId` TEXT NOT NU
            db.execSQL("CREATE TABLE IF NOT EXISTS `external_channels` (`channelId` TEXT N
            db.execSQL("CREATE TABLE IF NOT EXISTS `local_playlists` (`localPlaylistId` IN
            db.execSQL("CREATE TABLE IF NOT EXISTS `local_playlist_items` (`playlistOwnerI
        }
    }
    val MIGRATION_20_21: Migration = object : Migration(20, 21) {
        override fun migrate(db: SupportSQLiteDatabase) {
            db.execSQL("ALTER TABLE `local_playlists` RENAME TO `local_playlists_old`")
            db.execSQL("CREATE TABLE `local_playlists` (`localPlaylistId` INTEGER PRIMARY
            db.execSQL("INSERT INTO `local_playlists` (localPlaylistId, name, description,
            db.execSQL("DROP TABLE `local_playlists_old`")
            db.execSQL("DROP TABLE IF EXISTS `local_playlist_items`")
            db.execSQL("CREATE TABLE IF NOT EXISTS `local_playlist_items` (`playlistOwnerI
        }
    }

    // *** NEW MIGRATION: 21 -> 22 (FIXED) ***
    val MIGRATION_21_22: Migration = object : Migration(21, 22) {
        override fun migrate(db: SupportSQLiteDatabase) {
            // 1. Create Tables
            db.execSQL("""
                CREATE TABLE IF NOT EXISTS `unified_metadata` (
                    `id` TEXT NOT NULL,
                    `title` TEXT NOT NULL,
                    `artistName` TEXT NOT NULL,
                    `type` TEXT NOT NULL,
                    `specificArtUrl` TEXT,
                    `uploaderAvatarUrl` TEXT,
                    `duration` INTEGER NOT NULL,
                    `startSeconds` INTEGER DEFAULT NULL,
                    `endSeconds` INTEGER DEFAULT NULL,
                    `parentVideoId` TEXT,
```

```
            `channelId` TEXT NOT NULL,
            `org` TEXT,
            `topicId` TEXT,
            `status` TEXT NOT NULL DEFAULT 'past',
            `availableAt` TEXT,
            `publishedAt` TEXT,
            `songCount` INTEGER NOT NULL DEFAULT 0,
            `description` TEXT,
            `lastUpdatedAt` INTEGER NOT NULL DEFAULT 0,
            PRIMARY KEY(`id`)
        )
    """)

    db.execSQL("""
        CREATE TABLE IF NOT EXISTS `user_interactions` (
            `itemId` TEXT NOT NULL,
            `interactionType` TEXT NOT NULL,
            `timestamp` INTEGER NOT NULL,
            `localFilePath` TEXT,
            `downloadStatus` TEXT,
            `downloadFileName` TEXT,
            `downloadTrackNum` INTEGER,
            `downloadTargetFormat` TEXT,
            `downloadProgress` INTEGER NOT NULL DEFAULT 0,
            `serverId` TEXT,
            `syncStatus` TEXT NOT NULL DEFAULT 'SYNCED',
            PRIMARY KEY(`itemId`, `interactionType`),
            FOREIGN KEY(`itemId`) REFERENCES `unified_metadata`(`id`) ON UPDATE NO
        )
    """)
    db.execSQL("CREATE INDEX IF NOT EXISTS `index_user_interactions_itemId` ON `us

    // 3. MIGRATE: Videos -> Metadata
    // FIX: Mapped column names from Room Entity (e.g. 'topic_id') to New Entity (
    db.execSQL("""
        INSERT OR IGNORE INTO unified_metadata
        (id, title, artistName, type, channelId, org, uploaderAvatarUrl, duration,
        SELECT id, title, channel_name, 'VIDEO', channel_id, channel_org, channel_
        FROM videos
    """)

    // 4. MIGRATE: Liked Items
    db.execSQL("""
        INSERT OR IGNORE INTO unified_metadata
        (id, title, artistName, type, specificArtUrl, parentVideoId, channelId, du
        SELECT itemId,
               COALESCE(actual_song_name, title_snapshot),
               COALESCE(actual_song_artist, artist_text_snapshot),
               CASE WHEN item_type = 'SONG_SEGMENT' THEN 'SEGMENT' ELSE 'VIDEO' EN
               actual_song_artwork_url,
               videoId,
               channel_id_snapshot, duration_sec_snapshot,
               song_start_seconds, song_end_seconds, last_modified_at
        FROM liked_items
    """)
```

```
        db.execSQL("""
            INSERT OR IGNORE INTO user_interactions (itemId, interactionType, timestam
            SELECT itemId, 'LIKE', liked_at, server_id, sync_status FROM liked_items
        """)

        // 5. MIGRATE: Downloads
        db.execSQL("""
            INSERT OR IGNORE INTO unified_metadata
            (id, title, artistName, type, specificArtUrl, parentVideoId, channelId, du
            SELECT videoId, title, artistText, 'SEGMENT', artworkUrl,
                   substr(videoId, 0, instr(videoId, '_') - 1),
                   channelId, durationSec,
                   CAST(substr(videoId, instr(videoId, '_') + 1) AS INTEGER),
                   CAST(substr(videoId, instr(videoId, '_') + 1) AS INTEGER) + duratio
                   downloadedAt
            FROM downloaded_items
        """)

        db.execSQL("""
            INSERT OR IGNORE INTO user_interactions
            (itemId, interactionType, timestamp, localFilePath, downloadStatus, downlo
            SELECT videoId, 'DOWNLOAD', downloadedAt, localFileUri, downloadStatus, fi
            FROM downloaded_items
        """)

        // 6. MIGRATE: History
        db.execSQL("""
            INSERT OR IGNORE INTO unified_metadata
            (id, title, artistName, type, specificArtUrl, parentVideoId, channelId, du
            SELECT itemId, title, artistText, 'SEGMENT', artworkUrl, videoId, channelI
                   songStartSeconds, songStartSeconds + durationSec, playedAtTimestamp
            FROM history_items
        """)

        db.execSQL("""
            INSERT OR IGNORE INTO user_interactions (itemId, interactionType, timestam
            SELECT itemId, 'HISTORY', playedAtTimestamp, 'SYNCED' FROM history_items
        """)
    }
}
val MIGRATION_22_23: Migration = object : Migration(22, 23) {
    override fun migrate(db: SupportSQLiteDatabase) {
        val now = System.currentTimeMillis()

        // 1. Migrate Local Favorites -> Unified
        // These are "Local Only", so we mark them DIRTY but they likely won't sync wi
        // We treat them as SEGMENTs or VIDEOs based on flags.
        db.execSQL("""
            INSERT OR IGNORE INTO unified_metadata
            (id, title, artistName, type, specificArtUrl, parentVideoId, channelId, du
            SELECT itemId, title, artistText,
                   CASE WHEN isSegment = 1 THEN 'SEGMENT' ELSE 'VIDEO' END,
                   artworkUrl, videoId, channelId, durationSec,
                   songStartSec, songEndSec, $now
```

```
            FROM local_favorites
        """)

        db.execSQL("""
            INSERT OR IGNORE INTO user_interactions (itemId, interactionType, timestam
            SELECT itemId, 'LIKE', $now, 'DIRTY' FROM local_favorites
        """)

        // 2. Migrate External Channels -> Unified
        db.execSQL("""
            INSERT OR IGNORE INTO unified_metadata
            (id, title, artistName, type, specificArtUrl, uploaderAvatarUrl, duration,
            SELECT channelId, name, 'External', 'CHANNEL', photoUrl, photoUrl, 0, chan
            FROM external_channels
        """)

        db.execSQL("""
            INSERT OR IGNORE INTO user_interactions (itemId, interactionType, timestam
            SELECT channelId, 'FAV_CHANNEL', lastCheckedTimestamp, 'DIRTY' FROM extern
        """)

        // 3. DROP OLD TABLES
        db.execSQL("DROP TABLE IF EXISTS local_favorites")
        db.execSQL("DROP TABLE IF EXISTS external_channels")
        db.execSQL("DROP TABLE IF EXISTS liked_items")
        db.execSQL("DROP TABLE IF EXISTS favorite_channels")
    }
}
val MIGRATION_23_24: Migration = object : Migration(23, 24) {
    override fun migrate(db: SupportSQLiteDatabase) {
        db.execSQL("DROP TABLE IF EXISTS downloaded_items")
    }
}
val MIGRATION_24_25: Migration = object : Migration(24, 25) {
    override fun migrate(db: SupportSQLiteDatabase) {
        db.execSQL("DROP TABLE IF EXISTS history_items")
        // Ensure these are gone if they weren't already
        db.execSQL("DROP TABLE IF EXISTS liked_items")
        db.execSQL("DROP TABLE IF EXISTS downloaded_items")
        db.execSQL("DROP TABLE IF EXISTS local_playlists")
        db.execSQL("DROP TABLE IF EXISTS local_playlist_items")
    }
}
val MIGRATION_25_26: Migration = object : Migration(25, 26) {
    override fun migrate(db: SupportSQLiteDatabase) {
        // Drop the old heavy table
        db.execSQL("DROP TABLE IF EXISTS persisted_playback_items_table")
        db.execSQL("DROP TABLE IF EXISTS persisted_playback_state_table")

        // Create new lightweight tables
        db.execSQL("CREATE TABLE IF NOT EXISTS `playback_state` (`id` INTEGER NOT NULL

        db.execSQL("CREATE TABLE IF NOT EXISTS `playback_queue_ref` (`queue_index` INT

        db.execSQL("CREATE INDEX IF NOT EXISTS `index_playback_queue_ref_item_id` ON `
```

```
            }
        }

        fun getDatabase(context: Context): AppDatabase {
            return INSTANCE ?: synchronized(this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    AppDatabase::class.java,
                    "holodex_music_app_database"
                )
                    .addMigrations(
                        MIGRATION_18_19,
                        MIGRATION_19_20,
                        MIGRATION_20_21,
                        MIGRATION_21_22,
                        MIGRATION_22_23,
                        MIGRATION_23_24,
                        MIGRATION_24_25,
                        MIGRATION_25_26
                    )
                    .build()
                INSTANCE = instance
                instance
            }
        }
    }
}



class LikedItemTypeConverter {
    @TypeConverter
    fun fromLikedItemType(value: LikedItemType?): String? {
        return value?.name
    }

    @TypeConverter
    fun toLikedItemType(value: String?): LikedItemType? {
        return value?.let {
            try {
                LikedItemType.valueOf(it)
            } catch (_: IllegalArgumentException) {
                null
            }
        }
    }
}
class SyncStatusConverter {
    @TypeConverter
    fun fromSyncStatus(value: SyncStatus?): String? {
        return value?.name
    }

    @TypeConverter
    fun toSyncStatus(value: String?): SyncStatus? {
```

```
        return value?.let {
            try {
                SyncStatus.valueOf(it)
            } catch (_: IllegalArgumentException) {
                null
            }
        }
    }
}


// File: java\com\example\holodex\data\db\BrowsePageDao.kt
package com.example.holodex.data.db

import androidx.room.Dao
import androidx.room.Insert
import androidx.room.OnConflictStrategy
import androidx.room.Query

@Dao
interface BrowsePageDao {
    @Query("SELECT * FROM cached_browse_pages WHERE pageKey = :pageKey")
    suspend fun getPage(pageKey: String): CachedBrowsePage?

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertPage(page: CachedBrowsePage)

    @Query("DELETE FROM cached_browse_pages WHERE pageKey = :pageKey")
    suspend fun deletePage(pageKey: String)

    /**
     * Deletes pages older than the given timestamp for browse caches.
     * The pageKey for browse pages might start with a common prefix like "browse_".
     */
    @Query("DELETE FROM cached_browse_pages WHERE timestamp < :expiredTime")
    suspend fun deleteExpiredBrowsePages(expiredTime: Long)

    /**
     * Deletes all pages from the browse cache.
     */
    @Query("DELETE FROM cached_browse_pages")
    suspend fun deleteAllBrowsePages()

    // Optional: Method to get cache size for monitoring
    @Query("SELECT COUNT(pageKey) FROM cached_browse_pages")
    suspend fun getBrowseCacheSize(): Int
}

// File: java\com\example\holodex\data\db\CachedPageEntities.kt
package com.example.holodex.data.db

import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.Index
import com.example.holodex.data.model.HolodexVideoItem // Ensure this import is correct
```

```
import com.example.holodex.data.model.discovery.DiscoveryResponse

/**
 * Entity to store a "page" of browsed video items.
 * The pageKey should uniquely identify the filter set and offset/page number.
 */
@Entity(
    tableName = "cached_browse_pages",
    primaryKeys = ["pageKey"],
    indices = [Index(value = ["timestamp"])]
)
data class CachedBrowsePage(
    val pageKey: String, // Example: "browse_preset=LATEST_STREAMS_org=Hololive_topic=singing_
    @ColumnInfo(name = "data_list")
    val data: List<HolodexVideoItem>, // Will use TypeConverter
    val timestamp: Long = System.currentTimeMillis(),
    val totalAvailable: Int? = null // Total items API reported for this query, to help determ
)

/**
 * Entity to store a "page" of searched video items.
 * The pageKey should uniquely identify the search query and offset/page number.
 */
@Entity(
    tableName = "cached_search_pages",
    primaryKeys = ["pageKey"],
    indices = [Index(value = ["timestamp"])]
)
data class CachedSearchPage(
    val pageKey: String, // Example: "search_query=my_search_term_offset=0"
    @ColumnInfo(name = "data_list")
    val data: List<HolodexVideoItem>, // Will use TypeConverter
    val timestamp: Long = System.currentTimeMillis(),
    val totalAvailable: Int? = null
)

@Entity(
    tableName = "cached_discovery_responses",
    primaryKeys = ["pageKey"],
    indices = [Index(value = ["timestamp"])]
)
data class CachedDiscoveryResponse(
    val pageKey: String, // e.g., "discovery_org_Hololive", "discovery_favorites"
    @ColumnInfo(name = "data_response")
    val data: DiscoveryResponse, // Will use TypeConverter
    val timestamp: Long = System.currentTimeMillis()
)



// File: java\com\example\holodex\data\db\Converters.kt
// File: java/com/example/holodex/data/db/Converters.kt
package com.example.holodex.data.db

import androidx.room.TypeConverter
import com.example.holodex.data.model.HolodexVideoItem
```

```
import com.example.holodex.data.model.discovery.DiscoveryResponse
import com.google.gson.Gson
import com.google.gson.reflect.TypeToken

class HolodexVideoItemListConverter {
    private val gson = Gson()

    @TypeConverter
    fun fromVideoItemList(value: List<HolodexVideoItem>?): String? {
        return value?.let { gson.toJson(it) }
    }

    @TypeConverter
    fun toVideoItemList(value: String?): List<HolodexVideoItem>? {
        return value?.let {
            try {
                val listType = object : TypeToken<List<HolodexVideoItem>>() {}.type
                gson.fromJson(it, listType)
            } catch (e: Exception) {
                // Handle potential GSON parsing errors, e.g., if JSON is malformed
                null // Or throw, or log
            }
        }
    }
}

// Adding the StringListConverter here as well for co-location
class StringListConverter {
    @TypeConverter
    fun fromStringList(value: List<String>?): String? {
        return value?.joinToString(separator = "|||")
    }

    @TypeConverter
    fun toStringList(value: String?): List<String>? {
        return value?.split("|||")?.filter { it.isNotEmpty() }
    }
}
class DiscoveryResponseConverter {
    private val gson = Gson()

    @TypeConverter
    fun fromDiscoveryResponse(value: DiscoveryResponse?): String? {
        return value?.let { gson.toJson(it) }
    }

    @TypeConverter
    fun toDiscoveryResponse(value: String?): DiscoveryResponse? {
        return value?.let {
            try {
                gson.fromJson(it, DiscoveryResponse::class.java)
            } catch (e: Exception) {
                null
            }
        }
```

```
    }
}


// File: java\com\example\holodex\data\db\DiscoveryDao.kt
package com.example.holodex.data.db

import androidx.room.Dao
import androidx.room.Insert
import androidx.room.OnConflictStrategy
import androidx.room.Query

@Dao
interface DiscoveryDao {
    @Query("SELECT * FROM cached_discovery_responses WHERE pageKey = :pageKey")
    suspend fun getResponse(pageKey: String): CachedDiscoveryResponse?

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertResponse(response: CachedDiscoveryResponse)

    @Query("DELETE FROM cached_discovery_responses WHERE timestamp < :expiredTime")
    suspend fun deleteExpiredResponses(expiredTime: Long)
}

// File: java\com\example\holodex\data\db\entities.kt
// File: java/com/example/holodex/data/db/entities.kt
package com.example.holodex.data.db

import androidx.room.ColumnInfo
import androidx.room.Embedded
import androidx.room.Entity
import androidx.room.ForeignKey
import androidx.room.Index
import androidx.room.PrimaryKey
import androidx.room.Relation
import com.example.holodex.data.model.HolodexChannelMin
import com.example.holodex.data.model.HolodexSong
import com.example.holodex.data.model.HolodexVideoItem
import timber.log.Timber


@Entity(tableName = "videos")
data class CachedVideoEntity(
    @PrimaryKey val id: String,
    val title: String,
    val type: String,
    @ColumnInfo(name = "topic_id") val topicId: String?,
    @ColumnInfo(name = "published_at") val publishedAt: String?,
    @ColumnInfo(name = "available_at") val availableAt: String,
    val duration: Long,
    val status: String,
    @ColumnInfo(name = "song_count") val songCount: Int?,
    val description: String?, // This field will store the potentially truncated description
    @Embedded(prefix = "channel_")
    val channel: HolodexChannelMin,
```

```
    @ColumnInfo(
        name = "fetched_at_ms",
        defaultValue = "0"
    ) val fetchedAtMs: Long = System.currentTimeMillis(),
    @ColumnInfo(name = "list_query_key") val listQueryKey: String? = null,
    @ColumnInfo(
        name = "insertion_order",
        defaultValue = "0"
    ) val insertionOrder: Int = 0 // NEW FIELD
)

@Entity(
    tableName = "songs",
    primaryKeys = ["video_id", "start_time_seconds"],
    foreignKeys = [ForeignKey(
        entity = CachedVideoEntity::class,
        parentColumns = ["id"],
        childColumns = ["video_id"],
        onDelete = ForeignKey.CASCADE
    )],
    indices = [Index(value = ["video_id"])]
)
data class CachedSongEntity(
    @ColumnInfo(name = "video_id") val videoId: String,
    val name: String,
    @ColumnInfo(name = "start_time_seconds") val startTimeSeconds: Int,
    @ColumnInfo(name = "end_time_seconds") val endTimeSeconds: Int,
    @ColumnInfo(name = "original_artist") val originalArtist: String?,
    @ColumnInfo(name = "itunes_id") val itunesId: Int?,
    @ColumnInfo(name = "art_url") val artUrl: String?
)


// --- Mappers ---

fun HolodexVideoItem.toEntity(
    queryKey: String? = null,
    currentTimestamp: Long = System.currentTimeMillis(),
    orgNameFromRequest: String? = null,
    insertionOrder: Int = 0 // NEW PARAMETER
): CachedVideoEntity {
    val effectiveOrg = orgNameFromRequest ?: this.channel.org

    // Truncate description to a reasonable length
    val maxDescriptionLength = 1000 // Max 1000 characters for DB storage
    var truncatedDescription = this.description
    if (this.description?.length ?: 0 > maxDescriptionLength) {
        // Ensure truncation doesn't break multi-byte characters if present, though it's less
        truncatedDescription = this.description!!.substring(0, maxDescriptionLength) + "..."
        Timber.tag("HoloVideoItem.toEntity")
            .w("Description for video ${this.id} was truncated from ${this.description.length}
    }

    return CachedVideoEntity(
        id = this.id,
```

```
        title = this.title,
        type = this.type,
        topicId = this.topicId,
        publishedAt = this.publishedAt,
        availableAt = this.availableAt,
        duration = this.duration,
        status = this.status,
        songCount = this.songcount,
        description = truncatedDescription, // Use the (potentially) truncated description
        channel = HolodexChannelMin(
            id = this.channel.id,
            name = this.channel.name,
            englishName = this.channel.englishName,
            photoUrl = this.channel.photoUrl,
            type = this.channel.type,
            org = effectiveOrg
        ),
        fetchedAtMs = currentTimestamp,
        listQueryKey = queryKey,
        insertionOrder = insertionOrder // Populate new field
    )
}

fun HolodexSong.toEntity(videoIdParam: String): CachedSongEntity {
    return CachedSongEntity(
        videoId = videoIdParam,
        name = this.name,
        startTimeSeconds = this.start,
        endTimeSeconds = this.end,
        originalArtist = null, // This field was not in HolodexSong, keep as null or populate
        itunesId = this.itunesId,
        artUrl = this.artUrl
    )
}

fun CachedVideoEntity.toDomain(songsList: List<HolodexSong>? = null): HolodexVideoItem {
    return HolodexVideoItem(
        id = this.id,
        title = this.title,
        type = this.type,
        topicId = this.topicId,
        publishedAt = this.publishedAt,
        availableAt = this.availableAt,
        duration = this.duration,
        status = this.status,
        songcount = this.songCount,
        description = this.description, // This will be the (potentially) truncated descriptio
        channel = this.channel,
        songs = songsList
    )
}

fun CachedSongEntity.toDomain(): HolodexSong {
    return HolodexSong(
        name = this.name,
```

```
        start = this.startTimeSeconds,
        end = this.endTimeSeconds,
        itunesId = this.itunesId,
        videoId = this.videoId, // Ensure this is mapped back
        artUrl = this.artUrl
    )
}

data class VideoWithSongs(
    @Embedded val video: CachedVideoEntity,
    @Relation(
        parentColumn = "id",
        entityColumn = "video_id"
    )
    val songs: List<CachedSongEntity>
) {
    fun toDomain(): HolodexVideoItem {
        return video.toDomain(songsList = songs.map { it.toDomain() })
    }
}
```

```
// --- START REPLACEMENT ---
/**
 * Represents a user-created playlist in the local Room database.
 * This class is now decoupled from the network layer and contains only the fields
 * necessary for persistence and client-side sync logic.
 */
@Entity(
    tableName = "playlists",
    indices = [Index(value = ["server_id"])]
)
data class PlaylistEntity(
    @PrimaryKey(autoGenerate = true)
    var playlistId: Long = 0, // Local DB primary key, auto-generated

    // Server-side fields
    @ColumnInfo(name = "server_id")
    var serverId: String?, // The UUID from the server, nullable for local-only playlists

    var name: String?,
    var description: String?,
    var owner: Long?,
    var type: String = "ugp",

    @ColumnInfo(name = "created_at")
    var createdAt: String?,

    @ColumnInfo(name = "updated_at")
    var last_modified_at: String?, // Cleaned up property name to match column
```

```
    // Client-side sync state fields, ignored by network serialization
    @ColumnInfo(name = "is_deleted", defaultValue = "0")
    var isDeleted: Boolean = false,

    @ColumnInfo(name = "sync_status", defaultValue = "'DIRTY'")
    var syncStatus: SyncStatus = SyncStatus.DIRTY
)
// --- END REPLACEMENT ---


@Entity(
    tableName = "playlist_items",
    primaryKeys = ["playlist_owner_id", "item_id_in_playlist"],
    foreignKeys = [
        ForeignKey(
            entity = PlaylistEntity::class,
            parentColumns = ["playlistId"],
            childColumns = ["playlist_owner_id"],
            onDelete = ForeignKey.CASCADE
        )

    ],
    indices = [Index(value = ["playlist_owner_id"])]
)
data class PlaylistItemEntity(
    @ColumnInfo(name = "playlist_owner_id") val playlistOwnerId: Long,
    @ColumnInfo(name = "item_id_in_playlist") val itemIdInPlaylist: String, // Mirrors structu
    @ColumnInfo(name = "video_id_for_item") val videoIdForItem: String, // Parent video ID
    @ColumnInfo(name = "item_type_in_playlist") val itemTypeInPlaylist: LikedItemType,
    @ColumnInfo(name = "is_local_only", defaultValue = "0") val isLocalOnly: Boolean = false,
    // Optional: Snapshot of data for quick display, similar to LikedItemEntity
    @ColumnInfo(name = "song_start_seconds_playlist") val songStartSecondsPlaylist: Int? = nul
    @ColumnInfo(name = "song_end_seconds_playlist") val songEndSecondsPlaylist: Int? = null,
    @ColumnInfo(name = "song_name_playlist") val songNamePlaylist: String? = null,
    @ColumnInfo(name = "song_artist_text_playlist") val songArtistTextPlaylist: String? = null
    @ColumnInfo(name = "song_artwork_url_playlist") val songArtworkUrlPlaylist: String? = null
    // Could also store video title snapshot if it's a video item

    @ColumnInfo(name = "added_at", defaultValue = "CURRENT_TIMESTAMP")
    val addedAt: Long = System.currentTimeMillis(),
    @ColumnInfo(name = "item_order") val itemOrder: Int, // Order of the item within this spec
    @ColumnInfo(name = "last_modified_at", defaultValue = "0")
    val lastModifiedAt: Long = System.currentTimeMillis(),

    @ColumnInfo(name = "sync_status", defaultValue = "'DIRTY'")
    var syncStatus: SyncStatus = SyncStatus.DIRTY
)



@Entity(tableName = "sync_metadata")
data class SyncMetadataEntity(
    @PrimaryKey val dataType: String, // e.g., "likes", "history"
    val lastSyncTimestamp: Long
```

```
)
@Entity(tableName = "parent_video_metadata")
data class ParentVideoMetadataEntity(
    @PrimaryKey val videoId: String,
    val title: String,
    val channelName: String,
    val channelId: String,
    val thumbnailUrl: String?,
    val description: String?,
    val totalDurationSec: Long
)

// File: java\com\example\holodex\data\db\Enums.kt
package com.example.holodex.data.db

enum class LikedItemType {
    VIDEO,
    SONG_SEGMENT
}

enum class DownloadStatus {
    NOT_DOWNLOADED,
    ENQUEUED,
    DOWNLOADING,
    COMPLETED,
    FAILED,
    PROCESSING,
    EXPORT_FAILED,
    PAUSED,
    DELETING
}

enum class SyncStatus {
    SYNCED,
    DIRTY,
    PENDING_DELETE
}

// File: java\com\example\holodex\data\db\HolodexSongListConverter.kt
// File: com/example/holodex/data/db/HolodexSongListConverter.kt
package com.example.holodex.data.db

import androidx.room.TypeConverter
import com.example.holodex.data.model.HolodexSong
import com.google.gson.Gson
import com.google.gson.reflect.TypeToken

class HolodexSongListConverter {
    private val gson = Gson()

    @TypeConverter
    fun fromHolodexSongList(songs: List<HolodexSong>?): String? {
        return songs?.let { gson.toJson(it) }
    }
```

```
    @TypeConverter
    fun toHolodexSongList(songsJson: String?): List<HolodexSong>? {
        return songsJson?.let {
            val listType = object : TypeToken<List<HolodexSong>>() {}.type
            gson.fromJson(it, listType)
        }
    }
}

// File: java\com\example\holodex\data\db\LocalEntities.kt
// File: java/com/example/holodex/data/db/LocalEntities.kt

package com.example.holodex.data.db

import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity(tableName = "local_playlists")
data class LocalPlaylistEntity(
    @PrimaryKey(autoGenerate = true) val localPlaylistId: Long = 0,
    val name: String,
    val description: String?,
    val createdAt: Long = System.currentTimeMillis()
)

@Entity(
    tableName = "local_playlist_items",
    primaryKeys = ["playlistOwnerId", "itemId"]
)
data class LocalPlaylistItemEntity(
    val playlistOwnerId: Long,
    val itemId: String, // The composite ID: "videoId_startTime"
    val videoId: String,
    val itemOrder: Int,

    // Snapshot data for quick display
    val title: String,
    val artistText: String,
    val artworkUrl: String?,
    val durationSec: Long,
    val channelId: String,
    val isSegment: Boolean,
    val songStartSec: Int?,
    val songEndSec: Int?
)


// File: java\com\example\holodex\data\db\ParentVideoMetadataDao.kt
// File: java/com/example/holodex/data/db/ParentVideoMetadataDao.kt
package com.example.holodex.data.db

import androidx.room.Dao
import androidx.room.Insert
import androidx.room.OnConflictStrategy
import androidx.room.Query
```

```
@Dao
interface ParentVideoMetadataDao {
    @Insert(onConflict = OnConflictStrategy.IGNORE)
    suspend fun insert(metadata: ParentVideoMetadataEntity)

    @Query("SELECT * FROM parent_video_metadata WHERE videoId = :videoId")
    suspend fun getById(videoId: String): ParentVideoMetadataEntity?
}

// File: java\com\example\holodex\data\db\PlaylistDao.kt
// File: java/com/example/holodex/data/db/PlaylistDao.kt

package com.example.holodex.data.db

import androidx.room.Dao
import androidx.room.Insert
import androidx.room.OnConflictStrategy
import androidx.room.Query
import androidx.room.Transaction
import androidx.room.Update
import kotlinx.coroutines.flow.Flow

@Dao
interface PlaylistDao {
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertPlaylist(playlist: PlaylistEntity): Long

    // --- START OF FIX: Add a dedicated @Update function ---
    @Update
    suspend fun updatePlaylist(playlist: PlaylistEntity)
    // --- END OF FIX ---

    @Transaction
    suspend fun updatePlaylistAndItems(playlist: PlaylistEntity, items: List<PlaylistItemEntit
        updatePlaylist(playlist)
        deleteAllItemsForPlaylist(playlist.playlistId)
        if (items.isNotEmpty()) {
            upsertPlaylistItems(items)
        }
    }

    @Query("SELECT * FROM playlists WHERE is_deleted = 0 ORDER BY name ASC")
    fun getAllPlaylists(): Flow<List<PlaylistEntity>>

    @Query("SELECT * FROM playlists WHERE playlistId = :playlistId")
    suspend fun getPlaylistById(playlistId: Long): PlaylistEntity?

    @Query("UPDATE playlists SET is_deleted = 1, sync_status = 'PENDING_DELETE' WHERE playlist
    suspend fun softDeletePlaylist(playlistId: Long)

    @Query("DELETE FROM playlists WHERE playlistId = :playlistId")
    suspend fun deletePlaylist(playlistId: Long)

    // --- START OF FIX: Change strategy to REPLACE for robustness ---
```

```
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertPlaylistItem(playlistItem: PlaylistItemEntity)
    // --- END OF FIX ---

    @Query("UPDATE playlist_items SET sync_status = 'PENDING_DELETE' WHERE playlist_owner_id =
    suspend fun softDeletePlaylistItem(playlistId: Long, itemIdInPlaylist: String)

    @Query("SELECT * FROM playlist_items WHERE playlist_owner_id = :playlistId AND sync_status
    fun getItemsForPlaylist(playlistId: Long): Flow<List<PlaylistItemEntity>>

    // --- START OF FIX: Make query more robust by excluding soft-deleted items ---
    @Query("SELECT MAX(item_order) FROM playlist_items WHERE playlist_owner_id = :playlistId A
    suspend fun getLastItemOrder(playlistId: Long): Int?
    // --- END OF FIX ---

    @Query("SELECT * FROM playlists")
    suspend fun getAllPlaylistsOnce(): List<PlaylistEntity>

    @Query("SELECT * FROM playlists WHERE sync_status != 'SYNCED'")
    suspend fun getUnsyncedPlaylists(): List<PlaylistEntity>

    @Query("SELECT * FROM playlist_items WHERE sync_status != 'SYNCED'")
    suspend fun getUnsyncedPlaylistItems(): List<PlaylistItemEntity>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun upsertPlaylists(playlists: List<PlaylistEntity>)

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun upsertPlaylistItems(items: List<PlaylistItemEntity>)

    @Query("DELETE FROM playlists WHERE is_deleted = 1")
    suspend fun deleteSoftDeletedPlaylists()

    @Query("DELETE FROM playlist_items WHERE playlist_owner_id = :playlistId")
    suspend fun deleteAllItemsForPlaylist(playlistId: Long)

    @Query("UPDATE playlist_items SET sync_status = 'SYNCED' WHERE playlist_owner_id = :playli
    suspend fun markItemsAsSynced(playlistId: Long, itemIds: List<String>)

    @Query("DELETE FROM playlist_items WHERE sync_status = 'PENDING_DELETE' AND playlist_owner
    suspend fun deleteSyncedSoftDeletedItemsForPlaylist(playlistId: Long)

    @Query("UPDATE playlists SET name = :name, description = :description, updated_at = :times
    suspend fun updatePlaylistMetadata(playlistId: Long, name: String?, description: String?,

}

// File: java\com\example\holodex\data\db\SearchPageDao.kt
package com.example.holodex.data.db

import androidx.room.Dao
import androidx.room.Insert
import androidx.room.OnConflictStrategy
import androidx.room.Query
```

```
@Dao
interface SearchPageDao {
    @Query("SELECT * FROM cached_search_pages WHERE pageKey = :pageKey")
    suspend fun getPage(pageKey: String): CachedSearchPage?

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertPage(page: CachedSearchPage)

    @Query("DELETE FROM cached_search_pages WHERE pageKey = :pageKey")
    suspend fun deletePage(pageKey: String)

    /**
     * Deletes pages older than the given timestamp for search caches.
     */
    @Query("DELETE FROM cached_search_pages WHERE timestamp < :expiredTime")
    suspend fun deleteExpiredSearchPages(expiredTime: Long)

    /**
     * Deletes all pages from the search cache.
     */
    @Query("DELETE FROM cached_search_pages")
    suspend fun deleteAllSearchPages()

    @Query("SELECT COUNT(pageKey) FROM cached_search_pages")
    suspend fun getSearchCacheSize(): Int
}

// File: java\com\example\holodex\data\db\StarredPlaylistDao.kt
// File: java/com/example/holodex/data/db/StarredPlaylistDao.kt (NEW FILE)
package com.example.holodex.data.db

import androidx.room.Dao
import androidx.room.Insert
import androidx.room.OnConflictStrategy
import androidx.room.Query
import kotlinx.coroutines.flow.Flow

@Dao
interface StarredPlaylistDao {
    @Query("SELECT * FROM starred_playlists")
    fun getStarredPlaylists(): Flow<List<StarredPlaylistEntity>>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insert(starredPlaylist: StarredPlaylistEntity)

    @Query("DELETE FROM starred_playlists WHERE playlist_id = :playlistId")
    suspend fun deleteById(playlistId: String)

    // --- Methods for Sync Logic ---
    @Query("SELECT * FROM starred_playlists WHERE sync_status != 'SYNCED'")
    suspend fun getUnsyncedItems(): List<StarredPlaylistEntity>

    @Query("DELETE FROM starred_playlists WHERE sync_status = 'SYNCED'")
    suspend fun deleteAllSyncedItems()
```

```
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun upsertAll(items: List<StarredPlaylistEntity>)
}

// File: java\com\example\holodex\data\db\StarredPlaylistEntity.kt
// File: java/com/example/holodex/data/db/StarredPlaylistEntity.kt (NEW FILE)
package com.example.holodex.data.db

import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity(tableName = "starred_playlists")
data class StarredPlaylistEntity(
    @PrimaryKey
    @ColumnInfo(name = "playlist_id")
    val playlistId: String,

    @ColumnInfo(name = "sync_status")
    val syncStatus: SyncStatus
)

// File: java\com\example\holodex\data\db\SyncMetadataDao.kt
// File: java/com/example/holodex/data/db/SyncMetadataDao.kt
// (Create this new file)

package com.example.holodex.data.db

import androidx.room.Dao
import androidx.room.Insert
import androidx.room.OnConflictStrategy
import androidx.room.Query

@Dao
interface SyncMetadataDao {
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun setLastSyncTimestamp(metadata: SyncMetadataEntity)

    @Query("SELECT lastSyncTimestamp FROM sync_metadata WHERE dataType = :dataType")
    suspend fun getLastSyncTimestamp(dataType: String): Long?
}

// File: java\com\example\holodex\data\db\UnifiedDao.kt
// File: java/com/example/holodex/data/db/UnifiedDao.kt
package com.example.holodex.data.db

import androidx.room.Dao
import androidx.room.Insert
import androidx.room.OnConflictStrategy
import androidx.room.Query
import androidx.room.Transaction
import androidx.room.Update
import kotlinx.coroutines.flow.Flow

@Dao
```

```
interface UnifiedDao {

    // --- WRITES ---

    @Insert(onConflict = OnConflictStrategy.IGNORE)
    suspend fun insertMetadataIgnore(metadata: UnifiedMetadataEntity): Long

    @Update
    suspend fun updateMetadataRaw(metadata: UnifiedMetadataEntity)

    @Transaction
    suspend fun upsertMetadata(metadata: UnifiedMetadataEntity) {
        val rowId = insertMetadataIgnore(metadata)
        if (rowId == -1L) {
            updateMetadataRaw(metadata)
        }
    }

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun upsertInteraction(interaction: UserInteractionEntity)

    @Query("DELETE FROM user_interactions WHERE itemId = :itemId AND interactionType = :type")
    suspend fun deleteInteraction(itemId: String, type: String)

    // --- NEW METHOD FOR HISTORY SYNC ---
    @Query("DELETE FROM user_interactions WHERE interactionType = :type")
    suspend fun deleteAllInteractionsByType(type: String)
    // -----------------------------------

    @Query("UPDATE user_interactions SET syncStatus = 'PENDING_DELETE' WHERE itemId = :itemId
    suspend fun softDeleteInteraction(itemId: String, type: String)

    @Query("UPDATE user_interactions SET serverId = :serverId, syncStatus = 'SYNCED' WHERE ite
    suspend fun updateServerId(itemId: String, type: String, serverId: String)

    @Query("UPDATE user_interactions SET syncStatus = 'SYNCED' WHERE itemId IN (:ids) AND inte
    suspend fun markAsSynced(ids: List<String>, type: String)

    @Query("DELETE FROM user_interactions WHERE itemId IN (:ids) AND interactionType = :type A
    suspend fun deleteSyncedInteractions(ids: List<String>, type: String)

    // --- CHECKS ---

    @Query("SELECT serverId FROM user_interactions WHERE itemId = :itemId AND interactionType
    suspend fun getLikeServerId(itemId: String): String?

    @Query("SELECT COUNT(*) FROM user_interactions WHERE itemId = :itemId AND interactionType
    suspend fun isLiked(itemId: String): Int

    @Query("SELECT COUNT(*) FROM user_interactions WHERE itemId = :itemId AND interactionType
    suspend fun isChannelLiked(itemId: String): Int

    @Query("SELECT serverId FROM user_interactions WHERE itemId = :itemId AND interactionType
    suspend fun getChannelLikeServerId(itemId: String): String?
```

```
// --- READS (UI) ---

@Transaction
@Query(
    """
    SELECT M.* FROM unified_metadata M
    INNER JOIN user_interactions I ON M.id = I.itemId
    WHERE I.interactionType = 'LIKE' AND I.syncStatus != 'PENDING_DELETE'
    ORDER BY I.timestamp DESC
"""
)
fun getFavorites(): Flow<List<UnifiedItemProjection>>

@Transaction
@Query(
    """
    SELECT M.* FROM unified_metadata M
    INNER JOIN user_interactions I ON M.id = I.itemId
    WHERE I.interactionType = 'FAV_CHANNEL' AND I.syncStatus != 'PENDING_DELETE'
    ORDER BY I.timestamp DESC
"""
)
fun getFavoriteChannels(): Flow<List<UnifiedItemProjection>>

@Query("SELECT itemId FROM user_interactions WHERE interactionType = 'LIKE' AND syncStatus
fun getLikedItemIds(): Flow<List<String>>

@Transaction
@Query(
    """
    SELECT M.* FROM unified_metadata M
    INNER JOIN user_interactions I ON M.id = I.itemId
    WHERE I.interactionType = 'DOWNLOAD'
    ORDER BY I.timestamp DESC
"""
)
fun getDownloads(): Flow<List<UnifiedItemProjection>>

@Transaction
@Query(
    """
    SELECT M.* FROM unified_metadata M
    INNER JOIN user_interactions I ON M.id = I.itemId
    WHERE I.interactionType = 'HISTORY'
    ORDER BY I.timestamp DESC
"""
)
fun getHistory(): Flow<List<UnifiedItemProjection>>

@Insert(onConflict = OnConflictStrategy.IGNORE)
suspend fun insertMetadataBatch(metadataList: List<UnifiedMetadataEntity>): List<Long>
// --- READS (Sync Worker) ---

@Query("SELECT * FROM user_interactions WHERE interactionType = :type AND syncStatus = 'DI
suspend fun getDirtyInteractions(type: String): List<UserInteractionEntity>
```

```
@Query("SELECT * FROM user_interactions WHERE interactionType = :type AND syncStatus = 'PE
suspend fun getPendingDeleteInteractions(type: String): List<UserInteractionEntity>

@Query("SELECT * FROM unified_metadata WHERE id = :id")
suspend fun getItemByIdOneShot(id: String): UnifiedItemProjection?

// --- SYNC SPECIFIC QUERIES ---

@Query("SELECT * FROM user_interactions WHERE interactionType = :type AND syncStatus = 'DI
suspend fun getDirtyItems(type: String): List<UserInteractionEntity>

@Query("SELECT * FROM user_interactions WHERE interactionType = :type AND syncStatus = 'PE
suspend fun getPendingDeleteItems(type: String): List<UserInteractionEntity>

@Query("SELECT * FROM user_interactions WHERE interactionType = :type AND syncStatus = 'SY
suspend fun getSyncedItems(type: String): List<UserInteractionEntity>

@Query("UPDATE user_interactions SET serverId = :serverId, syncStatus = 'SYNCED' WHERE ite
suspend fun confirmUpload(itemId: String, type: String, serverId: String)

@Query("DELETE FROM user_interactions WHERE itemId = :itemId AND interactionType = :type")
suspend fun confirmDeletion(itemId: String, type: String)

@Query("DELETE FROM user_interactions WHERE itemId = :itemId AND interactionType = :type A
suspend fun deleteSyncedItem(itemId: String, type: String)

@Query("UPDATE user_interactions SET syncStatus = 'SYNCED' WHERE itemId IN (:ids) AND inte
suspend fun markBatchAsSynced(ids: List<String>, type: String)

@Query("DELETE FROM user_interactions WHERE itemId IN (:ids) AND interactionType = :type A
suspend fun deleteBatchPending(ids: List<String>, type: String)

@Query("SELECT * FROM user_interactions WHERE interactionType = 'DOWNLOAD' AND downloadSta
suspend fun getCompletedDownloadsBatch(ids: List<String>): List<UserInteractionEntity>

// --- DOWNLOAD SPECIFIC UPDATES ---

@Query("UPDATE user_interactions SET downloadStatus = :status WHERE itemId = :itemId AND i
suspend fun updateDownloadStatus(itemId: String, status: String)

@Query("UPDATE user_interactions SET downloadProgress = :progress WHERE itemId = :itemId A
suspend fun updateDownloadProgress(itemId: String, progress: Int)

@Query("UPDATE user_interactions SET localFilePath = :path, downloadStatus = 'COMPLETED',
suspend fun completeDownload(itemId: String, path: String)

@Query("SELECT * FROM user_interactions WHERE interactionType = 'DOWNLOAD'")
suspend fun getAllDownloadsOneShot(): List<UserInteractionEntity>

@Query("SELECT * FROM user_interactions WHERE itemId = :itemId AND interactionType = 'DOWN
suspend fun getDownloadInteraction(itemId: String): UserInteractionEntity?

@Query("SELECT * FROM user_interactions WHERE itemId = :itemId AND interactionType = 'DOWN
fun getDownloadInteractionSync(itemId: String): UserInteractionEntity?
```

```
}

// File: java\com\example\holodex\data\db\UnifiedItemProjection.kt
package com.example.holodex.data.db

import androidx.room.Embedded
import androidx.room.Relation
import com.example.holodex.viewmodel.UnifiedDisplayItem

data class UnifiedItemProjection(
    @Embedded val metadata: UnifiedMetadataEntity,

    @Relation(
        parentColumn = "id",
        entityColumn = "itemId"
    )
    val interactions: List<UserInteractionEntity>
) {
    fun toUnifiedDisplayItem(): UnifiedDisplayItem {
        val downloadInteraction = interactions.find { it.interactionType == "DOWNLOAD" }
        val likeInteraction = interactions.find { it.interactionType == "LIKE" }

        val isSegment = metadata.type == "SEGMENT"

        return UnifiedDisplayItem(
            stableId = "${metadata.type.lowercase()}_${metadata.id}",
            playbackItemId = metadata.id,
            videoId = metadata.parentVideoId ?: metadata.id,
            channelId = metadata.channelId,
            title = metadata.title,
            artistText = metadata.artistName,
            artworkUrls = metadata.getComputedArtworkList(),
            durationText = com.example.holodex.playback.util.formatDurationSeconds(metadata.du

            isSegment = isSegment,
            songCount = if(isSegment) null else metadata.songCount,

            isDownloaded = downloadInteraction?.downloadStatus == "COMPLETED",

            downloadStatus = downloadInteraction?.downloadStatus,

            // --- FIX IS HERE: PASS THE PATH ---
            localFilePath = downloadInteraction?.localFilePath,
            // ---------------------------------

            isLiked = likeInteraction != null,

            itemTypeForPlaylist = if (isSegment) LikedItemType.SONG_SEGMENT else LikedItemType
            songStartSec = metadata.startSeconds?.toInt(),
            songEndSec = metadata.endSeconds?.toInt(),
            originalArtist = null,
```

```
            isExternal = metadata.type == "CHANNEL" || metadata.org == "External"
        )
    }
}

// File: java\com\example\holodex\data\db\UnifiedMetadataEntity.kt
package com.example.holodex.data.db

import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey
import com.example.holodex.util.ThumbnailQuality
import com.example.holodex.util.getYouTubeThumbnailUrl

@Entity(tableName = "unified_metadata")
data class UnifiedMetadataEntity(
    @PrimaryKey val id: String,

    val title: String,
    val artistName: String,
    val type: String,

    val specificArtUrl: String?,
    val uploaderAvatarUrl: String?,

    val duration: Long,

    @ColumnInfo(defaultValue = "NULL") val startSeconds: Long? = null,
    @ColumnInfo(defaultValue = "NULL") val endSeconds: Long? = null,
    val parentVideoId: String? = null,

    val channelId: String,
    val org: String? = null,
    val topicId: String? = null,

    // *** FIX: Explicit default value to match SQL ***
    @ColumnInfo(defaultValue = "'past'")
    val status: String = "past",

    val availableAt: String? = null,
    val publishedAt: String? = null,

    // *** FIX: Explicit default value to match SQL ***
    @ColumnInfo(defaultValue = "0")
    val songCount: Int = 0,

    val description: String? = null,

    @ColumnInfo(defaultValue = "0")
    val lastUpdatedAt: Long = System.currentTimeMillis()
) {
    fun getComputedArtworkList(): List<String> {
        // If specific art exists, just return that 1 item list.
        // Coil will handle the fallback if it fails, we don't need to send 4 URLs to the UI e
        if (!specificArtUrl.isNullOrBlank()) {
```

```
            return listOf(specificArtUrl)
        }

        // Fallback only if needed
        val targetId = parentVideoId ?: if (type != "CHANNEL") id else null
        return if (targetId != null) {
            getYouTubeThumbnailUrl(targetId, ThumbnailQuality.MEDIUM)
        } else {
            emptyList()
        }
    }
}

// File: java\com\example\holodex\data\db\UserInteractionEntity.kt
package com.example.holodex.data.db

import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.ForeignKey
import androidx.room.Index

@Entity(
    tableName = "user_interactions",
    primaryKeys = ["itemId", "interactionType"],
    foreignKeys = [
        ForeignKey(
            entity = UnifiedMetadataEntity::class,
            parentColumns = ["id"],
            childColumns = ["itemId"],
            onDelete = ForeignKey.CASCADE
        )
    ],
    indices = [Index("itemId")]
)
data class UserInteractionEntity(
    val itemId: String,
    val interactionType: String,
    val timestamp: Long,

    val localFilePath: String? = null,
    val downloadStatus: String? = null,
    val downloadFileName: String? = null,
    val downloadTrackNum: Int? = null,
    val downloadTargetFormat: String? = null,

    @ColumnInfo(defaultValue = "0")
    val downloadProgress: Int = 0,

    val serverId: String? = null,

    @ColumnInfo(defaultValue = "'SYNCED'")
    val syncStatus: String = "SYNCED"
)

// File: java\com\example\holodex\data\db\VideoDao.kt
```

```
package com.example.holodex.data.db

import androidx.room.Dao
import androidx.room.Insert
import androidx.room.OnConflictStrategy
import androidx.room.Query
import androidx.room.Transaction
import kotlinx.coroutines.flow.Flow

@Dao
interface VideoDao {

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertVideo(video: CachedVideoEntity)

    @Transaction
    @Query("SELECT * FROM videos WHERE id = :videoId")
    suspend fun getVideoWithSongsOnce(videoId: String): VideoWithSongs?

    @Query("SELECT * FROM videos WHERE id = :videoId LIMIT 1")
    suspend fun getVideoByIdOnce(videoId: String): CachedVideoEntity?


    @Query("DELETE FROM videos")
    fun clearAllVideos()

    @Query("DELETE FROM songs WHERE video_id = :videoId")
    suspend fun deleteSongsForVideo(videoId: String)

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertSongs(songs: List<CachedSongEntity>)

    @Query("SELECT * FROM videos WHERE id IN (:videoIds)")
    fun getVideosByIds(videoIds: List<String>): Flow<List<CachedVideoEntity>>


    @Query("DELETE FROM songs")
    fun clearAllSongs()
}



// File: java\com\example\holodex\data\db\mappers\SyncMappers.kt
// File: java\com\example\holodex\data\db\mappers\SyncMappers.kt (NEW FILE)
package com.example.holodex.data.db.mappers

import com.example.holodex.data.api.PlaylistDto
import com.example.holodex.data.db.PlaylistEntity
import com.example.holodex.data.db.SyncStatus


fun PlaylistDto.toEntity(): PlaylistEntity {
    return PlaylistEntity(
        playlistId = 0, // Let Room auto-generate the local ID
        serverId = this.id,
```

```
        name = this.title,
        description = this.description,
        owner = this.owner,
        type = this.type ?: "ugp",
        createdAt = this.createdAt,
        last_modified_at = this.updatedAt,
        isDeleted = false,
        syncStatus = SyncStatus.SYNCED // Data from server is considered synced
    )
}

// File: java\com\example\holodex\data\download\DownloadCompletionObserver.kt
package com.example.holodex.data.download

import android.content.Context
import androidx.annotation.OptIn
import androidx.media3.common.util.UnstableApi
import androidx.media3.exoplayer.offline.Download
import androidx.media3.exoplayer.offline.DownloadManager
import androidx.work.Data
import androidx.work.ExistingWorkPolicy
import androidx.work.OneTimeWorkRequestBuilder
import androidx.work.WorkManager
import com.example.holodex.background.M4AExportWorker
import com.example.holodex.data.db.DownloadStatus
import com.example.holodex.data.db.UnifiedDao
import dagger.hilt.android.qualifiers.ApplicationContext
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.SupervisorJob
import kotlinx.coroutines.launch
import timber.log.Timber
import javax.inject.Inject
import javax.inject.Singleton

@OptIn(UnstableApi::class)
@Singleton
class DownloadCompletionObserver @Inject constructor(
    @ApplicationContext private val context: Context,
    private val downloadManager: DownloadManager,
    private val unifiedDao: UnifiedDao // <--- UPDATED: Uses UnifiedDao
) : DownloadManager.Listener {

    companion object {
        private const val TAG = "DownloadCompletionObs"
    }
    private val scope = CoroutineScope(Dispatchers.IO + SupervisorJob())
    private val workManager = WorkManager.getInstance(context)

    fun initialize() {
        downloadManager.addListener(this)
    }

    override fun onDownloadChanged(
        manager: DownloadManager,
```

```
    download: Download,
    finalException: Exception?
) {
    scope.launch {
        val itemId = download.request.id

        // 1. Check Unified DB
        val currentEntry = unifiedDao.getDownloadInteraction(itemId)
        if (currentEntry == null) {
            Timber.w("$TAG: onDownloadChanged for ID: $itemId, but no DB entry found! Igno
            return@launch
        }

        // 2. Get Metadata for Export logic
        val projection = unifiedDao.getItemByIdOneShot(itemId) ?: return@launch

        when (download.state) {
            Download.STATE_COMPLETED -> {
                Timber.i("$TAG: Media3 download COMPLETED for ID: $itemId. Enqueueing expo
                unifiedDao.updateDownloadStatus(itemId, DownloadStatus.PROCESSING.name)

                val clipStartMs = (projection.metadata.startSeconds ?: 0) * 1000L
                val clipEndMs = (projection.metadata.endSeconds ?: 0) * 1000L

                val workData = Data.Builder()
                    .putString(M4AExportWorker.KEY_ITEM_ID, itemId)
                    .putString(M4AExportWorker.KEY_ORIGINAL_URI, download.request.uri.toSt
                    .putString(M4AExportWorker.KEY_SONG_TITLE, projection.metadata.title)
                    .putString(M4AExportWorker.KEY_ARTIST_NAME, projection.metadata.artist
                    .putString(M4AExportWorker.KEY_ALBUM_NAME, projection.metadata.title)
                    .putString(M4AExportWorker.KEY_ARTWORK_URI, projection.metadata.specif
                    .putLong(M4AExportWorker.KEY_CLIP_START_MS, clipStartMs)
                    .putLong(M4AExportWorker.KEY_CLIP_END_MS, clipEndMs)
                    .apply {
                        currentEntry.downloadTrackNum?.let { putInt(M4AExportWorker.KEY_TR
                    }
                    .build()

                val exportRequest = OneTimeWorkRequestBuilder<M4AExportWorker>()
                    .setInputData(workData)
                    .build()

                workManager.enqueueUniqueWork("export_$itemId", ExistingWorkPolicy.REPLACE
            }
            Download.STATE_FAILED -> {
                Timber.e(finalException, "$TAG: Download FAILED for ID: $itemId.")
                unifiedDao.updateDownloadStatus(itemId, DownloadStatus.FAILED.name)
            }
            Download.STATE_STOPPED -> {
                if (download.stopReason != 0 || finalException != null) {
                    unifiedDao.updateDownloadStatus(itemId, DownloadStatus.FAILED.name)
                }
            }
            Download.STATE_RESTARTING -> {
                if (currentEntry.downloadStatus != DownloadStatus.COMPLETED.name) {
```

```
                        unifiedDao.updateDownloadStatus(itemId, DownloadStatus.DOWNLOADING.nam
                    }
                }
                else -> { /* Other states like QUEUED/DOWNLOADING handled by initial insert */
            }
        }
    }
}

// File: java\com\example\holodex\data\download\DownloadExceptions.kt
// File: java/com/example/holodex/data/download/DownloadExceptions.kt
package com.example.holodex.data.download

/**
 * A specific exception thrown when a download is initiated without a
 * download location being configured in the app's settings.
 */
class NoDownloadLocationException(message: String) : Exception(message)

// File: java\com\example\holodex\data\download\LegacyDownloadScanner.kt
// File: java/com/example/holodex/data/download/LegacyDownloadScanner.kt
package com.example.holodex.data.download

import android.content.Context
import android.media.MediaMetadataRetriever
import android.net.Uri
import android.provider.MediaStore
import com.example.holodex.data.db.DownloadStatus
import com.example.holodex.data.db.UnifiedDao
import com.example.holodex.data.db.UnifiedMetadataEntity
import com.example.holodex.data.db.UserInteractionEntity
import com.example.holodex.data.model.HolodexSong
import com.example.holodex.data.model.HolodexVideoItem
import com.example.holodex.data.model.SearchCondition
import com.example.holodex.data.model.VideoSearchRequest
import com.example.holodex.data.repository.HolodexRepository
import dagger.hilt.android.qualifiers.ApplicationContext
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.async
import kotlinx.coroutines.awaitAll
import kotlinx.coroutines.coroutineScope
import kotlinx.coroutines.sync.Semaphore
import kotlinx.coroutines.withContext
import timber.log.Timber
import java.io.File
import javax.inject.Inject
import javax.inject.Singleton
import kotlin.math.max

@Singleton
class LegacyDownloadScanner @Inject constructor(
    @ApplicationContext private val context: Context,
    private val holodexRepository: HolodexRepository,
    private val unifiedDao: UnifiedDao // Injected UnifiedDao instead of removed DAO
) {
```

```
companion object {
    private const val TAG = "LegacyDownloadScanner"
    private const val DOWNLOAD_FOLDER_NAME = "HolodexMusic"
    private const val SIMILARITY_THRESHOLD = 0.85 // 85% similarity needed for a match
    private const val CONCURRENT_SCANS_LIMIT = 4 // Process up to 4 videos at a time
}

private data class FileInfo(val uri: Uri, val name: String, val lastModified: Long)
private data class FileMetadata(val title: String, val artist: String, val album: String)
private data class FileWithMetadata(val fileInfo: FileInfo, val metadata: FileMetadata)

suspend fun scanAndImportLegacyDownloads(): Int = withContext(Dispatchers.IO) {
    var totalImportedCount = 0
    try {
        val potentialLegacyFiles = queryMediaStoreForAppDownloads()
        if (potentialLegacyFiles.isEmpty()) {
            Timber.d("$TAG: No .m4a files found via MediaStore.")
            return@withContext 0
        }

        // FIX: Use UnifiedDao to get existing download filenames
        val existingDbFiles = unifiedDao.getAllDownloadsOneShot()
            .mapNotNull { it.downloadFileName }
            .toSet()

        val filesToProcess = potentialLegacyFiles.filterNot { existingDbFiles.contains(it.

        if (filesToProcess.isEmpty()) {
            Timber.i("$TAG: All ${potentialLegacyFiles.size} files are already in the data
            return@withContext 0
        }

        Timber.i("$TAG: Found ${filesToProcess.size} new potential legacy files. Grouping

        val filesGroupedByAlbum = filesToProcess.mapNotNull { fileInfo ->
            extractMetadata(fileInfo.uri)?.let { metadata -> FileWithMetadata(fileInfo, me
        }.groupBy { it.metadata.album }

        Timber.d("$TAG: Grouped files into ${filesGroupedByAlbum.size} potential videos. S

        val semaphore = Semaphore(CONCURRENT_SCANS_LIMIT)

        totalImportedCount = coroutineScope {
            val deferredImports = filesGroupedByAlbum.map { (albumTitle, filesInAlbum) ->
                async {
                    var groupImportCount = 0
                    semaphore.acquire() // Wait for a permit to start
                    try {
                        val artist = filesInAlbum.first().metadata.artist
                        val videoWithSongs = findVideoForGroup(albumTitle, artist)

                        if (videoWithSongs != null) {
                            for (fileWithMeta in filesInAlbum) {
                                val matchedSong = findMatchingSong(fileWithMeta.metadata.t
                                if (matchedSong != null) {
```

```
                                    importSong(fileWithMeta.fileInfo, videoWithSongs, matc
                                    groupImportCount++
                                } else {
                                    Timber.w("$TAG: FAILED to find song match for '${fileW
                                }
                            }
                            if (groupImportCount > 0) {
                                Timber.i("$TAG: Imported $groupImportCount songs for video
                            }
                        } else {
                            Timber.w("$TAG: FAILED to find a parent video for group with a
                        }
                    } finally {
                        semaphore.release() // Always release the permit
                    }
                    groupImportCount // Return the count for this group
                }
            }
            deferredImports.awaitAll().sum() // Wait for all groups to finish and sum the
        }

    } catch (e: Exception) {
        Timber.e(e, "$TAG: An error occurred during the scan process.")
    }
    Timber.i("$TAG: Scan complete. Imported a total of $totalImportedCount new files.")
    return@withContext totalImportedCount
}

private suspend fun findVideoForGroup(albumTitle: String, artist: String): HolodexVideoIte
    val organization = extractOrgFromArtist(artist)
    val coreTitle = extractCoreTitle(albumTitle)

    // Search for both the unique part of the title AND the artist name
    val searchRequest = VideoSearchRequest(
        target = listOf("stream", "clip"),
        conditions = listOf(
            SearchCondition(text = coreTitle),
            SearchCondition(text = artist)
        ),
        org = organization?.let { listOf(it) }
    )

    try {
        val searchResult = holodexRepository.holodexApiService.searchVideosAdvanced(search

        val potentialVideos = searchResult.body()?.items
        if (potentialVideos.isNullOrEmpty()) {
            return null // API found no candidates
        }

        // Client-side filter: find the video in the results that is most similar to our f
        val bestVideoMatch = potentialVideos.maxByOrNull {
            calculateSimilarity(it.title, albumTitle)
        } ?: return null
```

```
        // Confidence check: If the best match is still not very similar, reject it.
        if (calculateSimilarity(bestVideoMatch.title, albumTitle) < 0.6) {
            Timber.w("$TAG: Found a potential video ('${bestVideoMatch.title}'), but it wa
            return null
        }

        val videoWithSongsResult = holodexRepository.getVideoWithSongs(bestVideoMatch.id)
        return videoWithSongsResult.getOrNull()
    } catch (e: Exception) {
        Timber.e(e, "$TAG: API error while finding video for group: $albumTitle")
        return null
    }
}

private fun findMatchingSong(fileTitle: String, apiSongs: List<HolodexSong>?): HolodexSong
    if (apiSongs.isNullOrEmpty()) return null

    val normalizedFileTitle = normalize(fileTitle)
    return apiSongs
        .map { apiSong ->
            val normalizedApiTitle = normalize(apiSong.name)
            val similarity = calculateSimilarity(normalizedFileTitle, normalizedApiTitle)
            apiSong to similarity
        }
        .filter { it.second >= SIMILARITY_THRESHOLD }
        .maxByOrNull { it.second }
        ?.first
}

// FIX: Updated to use UnifiedDao with Metadata and Interaction tables
private suspend fun importSong(
    fileInfo: FileInfo,
    video: HolodexVideoItem,
    song: HolodexSong
) {
    val itemId = "${video.id}_${song.start}"

    // 1. Upsert Metadata
    val metadata = UnifiedMetadataEntity(
        id = itemId,
        title = song.name,
        artistName = video.channel.name,
        type = "SEGMENT",
        specificArtUrl = song.artUrl ?: video.channel.photoUrl,
        uploaderAvatarUrl = video.channel.photoUrl,
        duration = (song.end - song.start).toLong(),
        channelId = video.channel.id ?: "unknown",
        parentVideoId = video.id,
        startSeconds = song.start.toLong(),
        endSeconds = song.end.toLong(),
        lastUpdatedAt = System.currentTimeMillis()
    )
    unifiedDao.upsertMetadata(metadata)

    // 2. Upsert Interaction (Download Status)
```

```
        val interaction = UserInteractionEntity(
            itemId = itemId,
            interactionType = "DOWNLOAD",
            timestamp = fileInfo.lastModified,
            localFilePath = fileInfo.uri.toString(),
            downloadStatus = DownloadStatus.COMPLETED.name,
            downloadFileName = fileInfo.name,
            downloadTargetFormat = "M4A",
            downloadProgress = 100,
            syncStatus = "SYNCED" // Legacy imports are local, but we mark them synced to avoi
        )
        unifiedDao.upsertInteraction(interaction)
    }

    private fun queryMediaStoreForAppDownloads(): List<FileInfo> {
        val files = mutableListOf<FileInfo>()
        val projection = arrayOf(
            MediaStore.Audio.Media._ID,
            MediaStore.Audio.Media.DISPLAY_NAME,
            MediaStore.Audio.Media.DATE_MODIFIED
        )

        val selection = "${MediaStore.Audio.Media.RELATIVE_PATH} LIKE ?"
        val selectionArgs = arrayOf("%${File.separator}$DOWNLOAD_FOLDER_NAME${File.separator}%

        val sortOrder = "${MediaStore.Audio.Media.DATE_MODIFIED} DESC"
        val queryUri = MediaStore.Audio.Media.EXTERNAL_CONTENT_URI

        try {
            context.contentResolver.query(
                queryUri,
                projection,
                selection,
                selectionArgs,
                sortOrder
            )?.use { cursor ->
                val idColumn = cursor.getColumnIndexOrThrow(MediaStore.Audio.Media._ID)
                val nameColumn = cursor.getColumnIndexOrThrow(MediaStore.Audio.Media.DISPLAY_N
                val dateModifiedColumn = cursor.getColumnIndexOrThrow(MediaStore.Audio.Media.D

                while (cursor.moveToNext()) {
                    val id = cursor.getLong(idColumn)
                    val name = cursor.getString(nameColumn)
                    val dateModified = cursor.getLong(dateModifiedColumn)
                    val contentUri = Uri.withAppendedPath(queryUri, id.toString())
                    files.add(FileInfo(contentUri, name, dateModified * 1000))
                }
            }
        } catch (e: Exception) {
            Timber.e(e, "$TAG: Failed to query MediaStore.")
        }
        return files
    }

    private fun extractMetadata(uri: Uri): FileMetadata? {
```

```
    return try {
        MediaMetadataRetriever().use { retriever ->
            retriever.setDataSource(context, uri)
            val title = retriever.extractMetadata(MediaMetadataRetriever.METADATA_KEY_TITL
            val artist = retriever.extractMetadata(MediaMetadataRetriever.METADATA_KEY_ART
            val album = retriever.extractMetadata(MediaMetadataRetriever.METADATA_KEY_ALBU

            if (title.isNullOrBlank() || artist.isNullOrBlank() || album.isNullOrBlank())
                Timber.w("$TAG: File at uri '$uri' is missing essential metadata.")
                null
            } else {
                FileMetadata(title, artist, album)
            }
        }
    } catch (e: Exception) {
        Timber.e(e, "$TAG: Failed to extract metadata from uri '$uri'")
        null
    }
}

private fun extractOrgFromArtist(artist: String): String? {
    return when {
        artist.contains("?????", ignoreCase = true) -> "Nijisanji"
        artist.contains("hololive", ignoreCase = true) -> "Hololive"
        else -> null
    }
}

private fun normalize(input: String): String {
    return input
        .lowercase()
        .replace(Regex("[\\s.,?/()!\\[\\]_{}\"']"), "")
}

private fun calculateSimilarity(s1: String, s2: String): Double {
    val jaro = jaroDistance(s1, s2)
    if (jaro < 0.7) return jaro
    var prefix = 0
    for (i in 0 until minOf(s1.length, s2.length, 4)) {
        if (s1[i] == s2[i]) prefix++ else break
    }
    return jaro + 0.1 * prefix * (1.0 - jaro)
}

private fun jaroDistance(s1: String, s2: String): Double {
    if (s1 == s2) return 1.0
    val len1 = s1.length
    val len2 = s2.length
    if (len1 == 0 || len2 == 0) return 0.0
    val matchDistance = max(len1, len2) / 2 - 1
    val s1Matches = BooleanArray(len1)
    val s2Matches = BooleanArray(len2)
    var matches = 0
    for (i in 0 until len1) {
        val start = max(0, i - matchDistance)
```

```
            val end = minOf(i + matchDistance + 1, len2)
            for (j in start until end) {
                if (!s2Matches[j] && s1[i] == s2[j]) {
                    s1Matches[i] = true
                    s2Matches[j] = true
                    matches++
                    break
                }
            }
        }
        if (matches == 0) return 0.0
        var t = 0.0
        var k = 0
        for (i in 0 until len1) {
            if (s1Matches[i]) {
                while (!s2Matches[k]) k++
                if (s1[i] != s2[k]) t++
                k++
            }
        }
        val transpositions = t / 2
        return (matches.toDouble() / len1 + matches.toDouble() / len2 + (matches - transpositi
    }

    private fun extractCoreTitle(albumTitle: String): String {
        // Priority 1: Extract content from Japanese brackets ??
        val bracketRegex = Regex("?(.*?)?")
        val bracketMatch = bracketRegex.find(albumTitle)
        if (bracketMatch != null && bracketMatch.groupValues[1].isNotBlank()) {
            return bracketMatch.groupValues[1]
                .replace("#", "")
                .trim()
        }

        // Priority 2: Split by Japanese punctuation to find meaningful phrases
        val punctuationRegex = Regex("[??]")
        val phrases = albumTitle.split(punctuationRegex)
            .map { it.trim() }
            .filter { it.length > 2 }

        if (phrases.isNotEmpty()) {
            return phrases.maxByOrNull { it.length } ?: albumTitle
        }

        // Fallback: Return full title if no punctuation found
        return albumTitle
    }
}

// File: java\com\example\holodex\data\model\AudioStreamDetails.kt
package com.example.holodex.data.model // Make sure this package name matches your structure

import com.google.gson.annotations.SerializedName

// Represents what Musicdex or a similar service might return
```

```
data class AudioStreamDetails(
    @SerializedName("url") val streamUrl: String, // Direct audio stream URL
    @SerializedName("format") val format: String?, // e.g., "m4a", "opus"
    @SerializedName("quality") val quality: String? // e.g., "128kbps"
)

// File: java\com\example\holodex\data\model\ChannelSearchResult.kt
// File: java/com/example/holodex/data/model/ChannelSearchResult.kt
package com.example.holodex.data.model

data class ChannelSearchResult(
    val channelId: String,
    val name: String,
    val thumbnailUrl: String?,
    val subscriberCount: String?
)

// File: java\com\example\holodex\data\model\HolodexSong.kt
package com.example.holodex.data.model

import android.os.Parcelable
import com.google.gson.annotations.SerializedName
import kotlinx.parcelize.Parcelize

@Parcelize
data class HolodexSong(
    @SerializedName("name") val name: String,
    @SerializedName("start") val start: Int,
    @SerializedName("end") val end: Int,
    @SerializedName("itunesid") val itunesId: Int?,
    @SerializedName("art") val artUrl: String? = null,
    @SerializedName("original_artist") val originalArtist: String? = null,
    var videoId: String? = null
) : Parcelable

// File: java\com\example\holodex\data\model\HolodexVideoItem.kt
package com.example.holodex.data.model

import com.google.gson.annotations.SerializedName

data class HolodexVideoItem(
    @SerializedName("id") val id: String,
    @SerializedName("title") val title: String,
    @SerializedName("type") val type: String,
    @SerializedName("topic_id") val topicId: String?,
    @SerializedName("available_at") val availableAt: String,
    @SerializedName("published_at") val publishedAt: String?,
    @SerializedName("duration") val duration: Long,
    @SerializedName("status") val status: String,
    @SerializedName("channel") val channel: HolodexChannelMin,
    @SerializedName("songcount") val songcount: Int?,
    @SerializedName("description") val description: String?,

    // This field will be populated when fetching a single video with "include=songs"
    @SerializedName("songs") val songs: List<HolodexSong>? = null
```

```
)

// HolodexChannelMin remains the same
data class HolodexChannelMin(
    @SerializedName("id") val id: String?,
    @SerializedName("name") val name: String,
    @SerializedName("english_name") val englishName: String?,
    @SerializedName("org") var org: String?,
    @SerializedName("type") val type: String?,
    @SerializedName("photo") val photoUrl: String?
)

// File: java\com\example\holodex\data\model\PaginatedVideosResponse.kt
package com.example.holodex.data.model

import com.google.gson.annotations.SerializedName

data class PaginatedVideosResponse(
    @SerializedName("total") val total: String?, // API spec says number, but examples sometim
    @SerializedName("items") val items: List<HolodexVideoItem>
) {
    // Convenience getter for total as Int
    fun getTotalAsInt(): Int? {
        return total?.toIntOrNull()
    }
}

// File: java\com\example\holodex\data\model\VideoSearchRequest.kt
package com.example.holodex.data.model

import com.google.gson.annotations.SerializedName

data class VideoSearchRequest(
    @SerializedName("sort") val sort: String = "newest",
    @SerializedName("lang") val lang: List<String>? = null,
    @SerializedName("target") val target: List<String>, // e.g., ["stream", "clip"]
    @SerializedName("conditions") val conditions: List<SearchCondition>? = null, // For text s
    @SerializedName("topic") val topic: List<String>? = null, // For topic filtering
    @SerializedName("vch") val vch: List<String>? = null, // For channel ID search
    @SerializedName("org") val org: List<String>? = null, // For organization filtering
    @SerializedName("paginated") val paginated: Boolean = true,
    @SerializedName("offset") val offset: Int = 0,
    @SerializedName("limit") val limit: Int = 25,
    // @SerializedName("status") val status: List<String>? = null // REMOVE if /search/videoSe
    // 'comment' field was also in the original openapi spec example but not in your data clas
    @SerializedName("comment") val comment: List<String>? = null
)

data class SearchCondition(
    @SerializedName("text") val text: String
)

// File: java\com\example\holodex\data\model\discovery\ChannelDetails.kt
package com.example.holodex.data.model.discovery
```

```
import com.google.gson.annotations.SerializedName

/**
 * Represents the full details of a channel from the /channels/{id} endpoint.
 */
data class ChannelDetails(
    @SerializedName("id") val id: String,
    @SerializedName("name") val name: String,
    @SerializedName("english_name") val englishName: String?,
    @SerializedName("description") val description: String?,
    @SerializedName("photo") val photoUrl: String?,
    @SerializedName("banner") val bannerUrl: String?,
    @SerializedName("org") val org: String?,
    @SerializedName("suborg") val suborg: String?,
    @SerializedName("twitter") val twitter: String?,
    @SerializedName("group") val group: String? // Add the correct field for grouping
)

// File: java\com\example\holodex\data\model\discovery\DiscoveryResponse.kt
package com.example.holodex.data.model.discovery

import com.example.holodex.data.model.HolodexVideoItem
import com.google.gson.annotations.SerializedName


//Represents the entire aggregated response from the /musicdex/discovery/ endpoints.

data class DiscoveryResponse(
    @SerializedName("recentSingingStreams") val recentSingingStreams: List<SingingStreamShelfI
    @SerializedName("channels") val channels: List<DiscoveryChannel>?,
    @SerializedName("recommended") val recommended: RecommendedPlaylists?
)

data class SingingStreamShelfItem(
    @SerializedName("video") val video: HolodexVideoItem,
    // The playlist object here is a full playlist with content
    @SerializedName("playlist") val playlist: FullPlaylist
)

data class DiscoveryChannel(
    @SerializedName("id") val id: String,
    @SerializedName("name") val name: String,
    @SerializedName("english_name") val englishName: String?,
    @SerializedName("photo") val photoUrl: String?,
    @SerializedName("song_count") val songCount: Int?,
    @SerializedName("suborg") val suborg: String? // Add the missing property
)

data class RecommendedPlaylists(
    @SerializedName("playlists") val playlists: List<PlaylistStub>
)

// File: java\com\example\holodex\data\model\discovery\FullPlaylist.kt
// File: java/com/example/holodex/data/model/discovery/FullPlaylist.kt
package com.example.holodex.data.model.discovery
```

```
import com.google.gson.annotations.SerializedName

data class FullPlaylist(
    @SerializedName("id") val id: String,
    @SerializedName("title") val title: String,
    @SerializedName("description") val description: String?,
    @SerializedName("type") val type: String?,
    @SerializedName("created_at") val createdAt: String?,
    @SerializedName("updated_at") val updatedAt: String?,
    @SerializedName("content") val content: List<MusicdexSong>?
)

// File: java\com\example\holodex\data\model\discovery\MusicdexSong.kt
package com.example.holodex.data.model.discovery

import com.google.gson.annotations.SerializedName

data class MusicdexSong(
    @SerializedName("id") val id: String?,
    @SerializedName("song_id") val songId: String,
    @SerializedName("name") val name: String,
    @SerializedName("original_artist") val originalArtist: String?,
    @SerializedName("art") val artUrl: String?,
    @SerializedName("video_id") val videoId: String,
    @SerializedName("start") val start: Int,
    @SerializedName("end") val end: Int,
    @SerializedName("available_at") val available_at: String?,

    @SerializedName("channel_id") val channelId: String?,
    @SerializedName("channel") val channel: MusicdexChannel,
    @SerializedName("ts") val ts: String? = null
)

data class MusicdexChannel(
    @SerializedName("id") val id: String?,
    @SerializedName("name") val name: String,
    @SerializedName("english_name") val englishName: String?,
    @SerializedName("photo") val photoUrl: String?,
    @SerializedName("suborg") val suborg: String?
)

// File: java\com\example\holodex\data\model\discovery\PlaylistStub.kt
package com.example.holodex.data.model.discovery

import com.google.gson.annotations.SerializedName

/**
 * Represents a playlist "stub" as returned in discovery carousels.
 * It contains metadata but not the full list of songs.
 */
data class PlaylistStub(
    @SerializedName("id") val id: String,
    @SerializedName("title") val title: String,
    @SerializedName("type") val type: String, // e.g., "ugp", "radio/artist"
```

```
    @SerializedName("art_context") val artContext: ArtContext?,
    @SerializedName("description") val description: String?
)

data class ArtContext(
    @SerializedName("videos") val videos: List<String>?, // Changed name and type
    @SerializedName("channels") val channels: List<String>?,
    @SerializedName("channel_photo") val channelPhotoUrl: String?
)

// File: java\com\example\holodex\data\repository\DownloadRepository.kt
package com.example.holodex.data.repository

import android.content.Context
import android.net.Uri
import android.os.Environment
import androidx.annotation.OptIn
import androidx.media3.common.MediaItem
import androidx.media3.common.util.UnstableApi
import androidx.media3.datasource.DataSource
import androidx.media3.datasource.cache.CacheDataSource
import androidx.media3.datasource.cache.SimpleCache
import androidx.media3.exoplayer.offline.DownloadHelper
import androidx.media3.exoplayer.offline.DownloadManager
import androidx.media3.exoplayer.offline.DownloadRequest
import androidx.media3.exoplayer.offline.DownloadService
import androidx.work.Data
import androidx.work.ExistingWorkPolicy
import androidx.work.OneTimeWorkRequestBuilder
import androidx.work.WorkManager
import com.example.holodex.background.M4AExportWorker
import com.example.holodex.data.db.DownloadStatus
import com.example.holodex.data.db.UnifiedDao
import com.example.holodex.data.db.UnifiedMetadataEntity
import com.example.holodex.data.db.UserInteractionEntity
import com.example.holodex.data.model.HolodexSong
import com.example.holodex.data.model.HolodexVideoItem
import com.example.holodex.di.ApplicationScope
import com.example.holodex.di.DownloadCache
import com.example.holodex.di.UpstreamDataSource
import com.example.holodex.playback.data.source.StreamResolutionCoordinator
import com.example.holodex.service.HolodexDownloadService
import dagger.hilt.android.qualifiers.ApplicationContext
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.flow.MutableSharedFlow
import kotlinx.coroutines.flow.SharedFlow
import kotlinx.coroutines.flow.asSharedFlow
import kotlinx.coroutines.launch
import kotlinx.coroutines.suspendCancellableCoroutine
import kotlinx.coroutines.withContext
import kotlinx.coroutines.withTimeout
import org.jaudiotagger.audio.AudioFileIO
import org.jaudiotagger.tag.FieldKey
import timber.log.Timber
```

```
import java.io.File
import java.io.IOException
import java.nio.charset.StandardCharsets
import javax.inject.Inject
import javax.inject.Singleton
import kotlin.coroutines.resume
import kotlin.coroutines.resumeWithException

@UnstableApi
interface DownloadRepository {
    suspend fun startDownload(video: HolodexVideoItem, song: HolodexSong)
    suspend fun cancelDownload(itemId: String)
    suspend fun deleteDownloadById(itemId: String)
    suspend fun resumeDownload(itemId: String)
    suspend fun retryExport(itemId: String)
    suspend fun reconcileAllDownloads()
    suspend fun rescanStorageForDownloads()
    val downloadCompletedEvents: SharedFlow<DownloadCompletedEvent>
    suspend fun postDownloadCompletedEvent(event: DownloadCompletedEvent)
    data class DownloadCompletedEvent(val itemId: String, val localFileUri: String)
}

@UnstableApi
@Singleton
@OptIn(UnstableApi::class)
class DownloadRepositoryImpl @Inject constructor(
    @ApplicationContext private val context: Context,
    private val unifiedDao: UnifiedDao,
    private val youtubeStreamRepository: YouTubeStreamRepository,
    @DownloadCache private val downloadCache: SimpleCache,
    @UpstreamDataSource private val upstreamDataSourceFactory: DataSource.Factory,
    private val media3DownloadManager: DownloadManager,
    @ApplicationScope private val applicationScope: CoroutineScope,
    private val workManager: WorkManager,
    private val streamResolutionCoordinator: StreamResolutionCoordinator
) : DownloadRepository {

    companion object {
        private const val TAG = "DownloadRepositoryImpl"
        private const val DOWNLOAD_FOLDER_NAME = "HolodexMusic"
    }

    private val _downloadCompletedEvents = MutableSharedFlow<DownloadRepository.DownloadComple
    override val downloadCompletedEvents: SharedFlow<DownloadRepository.DownloadCompletedEvent
        _downloadCompletedEvents.asSharedFlow()

    override suspend fun startDownload(video: HolodexVideoItem, song: HolodexSong) {
        val itemId = "${video.id}_${song.start}"
        val displayTitle = song.name.ifBlank { video.title }
        val durationSec = (song.end - song.start).toLong()

        val existing = unifiedDao.getDownloadInteraction(itemId)
        if (existing?.downloadStatus in listOf(DownloadStatus.ENQUEUED.name, DownloadStatus.DO
            Timber.w("$TAG: Download for $itemId is already active/done. Skipping.")
            return
```

```
}

Timber.d("$TAG: Initiating download for: $itemId")

applicationScope.launch(Dispatchers.IO) {
    var downloadHelper: DownloadHelper? = null
    try {
        // 1. Upsert Metadata & Interaction State (Existing code)
        val metadata = UnifiedMetadataEntity(
            id = itemId, title = displayTitle, artistName = video.channel.name, type =
            specificArtUrl = song.artUrl, uploaderAvatarUrl = video.channel.photoUrl,
            channelId = video.channel.id ?: "unknown", parentVideoId = video.id,
            startSeconds = song.start.toLong(), endSeconds = song.end.toLong(),
            lastUpdatedAt = System.currentTimeMillis()
        )
        unifiedDao.upsertMetadata(metadata)

        val interaction = UserInteractionEntity(
            itemId = itemId, interactionType = "DOWNLOAD", timestamp = System.currentT
            downloadStatus = DownloadStatus.ENQUEUED.name, downloadTargetFormat = "M4A
            downloadProgress = 0, downloadFileName = "${displayTitle.take(50)}.m4a"
        )
        unifiedDao.upsertInteraction(interaction)

        // 2. Resolve Stream URL (Existing code)
        val streamUrl = streamResolutionCoordinator.getCachedUrl(video.id) ?: withTime
            // Pass true here to force M4A selection
            youtubeStreamRepository.getAudioStreamDetails(video.id, preferM4a = true).
        }




        // 4. Create MediaItem WITH ClippingConfiguration (*** THE FIX ***)
        val mediaItem = MediaItem.fromUri(streamUrl)

        val cacheDataSourceFactory = CacheDataSource.Factory()
            .setCache(downloadCache)
            .setUpstreamDataSourceFactory(upstreamDataSourceFactory)

        // 4. Create Helper
        val downloadHelperFactory = DownloadHelper.Factory().setDataSourceFactory(cach
        downloadHelper = downloadHelperFactory.create(mediaItem)


        // 6. Prepare and Get Request
        val request = suspendCancellableCoroutine<DownloadRequest> { continuation ->
            downloadHelper.prepare(object : DownloadHelper.Callback {
                override fun onPrepared(
                    helper: DownloadHelper,
                    tracksInfoAvailable: Boolean
                ) {
                    try {
                        // Calculate milliseconds
                        val startMs = song.start * 1000L
```

```
                            val durationMs = (song.end - song.start) * 1000L

                            // Use the overload explicitly designed for progressive stream
                            // This forces the helper to calculate the byte range for this
                            val req = helper.getDownloadRequest(
                                itemId,                                     // id
                                displayTitle.toByteArray(StandardCharsets.UTF_8), // data
                                startMs,                                    // startPositi
                                durationMs                                  // durationMs
                            )
                            continuation.resume(req)
                        } catch (e: Exception) {
                            continuation.resumeWithException(e)
                        }
                    }

                    override fun onPrepareError(helper: DownloadHelper, e: IOException) {
                        continuation.resumeWithException(e)
                    }
                })
                continuation.invokeOnCancellation {
                    downloadHelper.release()
                }
            }

            // 6. Dispatch to Service
            DownloadService.sendAddDownload(context, HolodexDownloadService::class.java, r
            Timber.i("$TAG: Download dispatched with Partial Range: ${song.start}s to ${so

        } catch (e: Exception) {
            Timber.e(e, "Download setup failed for $itemId")
            unifiedDao.updateDownloadStatus(itemId, DownloadStatus.FAILED.name)
        } finally {
            downloadHelper?.release()
        }
    }
}

override suspend fun retryExport(itemId: String) {
    val projection = unifiedDao.getItemByIdOneShot(itemId) ?: return
    val downloadInt = projection.interactions.find { it.interactionType == "DOWNLOAD" } ?:
    if (downloadInt.downloadStatus != DownloadStatus.EXPORT_FAILED.name) return

    Timber.i("Retrying export for $itemId")

    val workData = Data.Builder()
        .putString(M4AExportWorker.KEY_ITEM_ID, itemId)
        .putString(M4AExportWorker.KEY_ORIGINAL_URI, "cache://$itemId")
        .putString(M4AExportWorker.KEY_SONG_TITLE, projection.metadata.title)
        .putString(M4AExportWorker.KEY_ARTIST_NAME, projection.metadata.artistName)
        .putLong(M4AExportWorker.KEY_CLIP_START_MS, 0) // Clipping is already handled by t
        .putLong(M4AExportWorker.KEY_CLIP_END_MS, projection.metadata.duration * 1000L)
        .build()

    val exportRequest = OneTimeWorkRequestBuilder<M4AExportWorker>().setInputData(workData
```

```
        workManager.enqueueUniqueWork("export_$itemId", ExistingWorkPolicy.REPLACE, exportRequ
        unifiedDao.updateDownloadStatus(itemId, DownloadStatus.PROCESSING.name)
    }

    override suspend fun deleteDownloadById(itemId: String) {
        unifiedDao.deleteInteraction(itemId, "DOWNLOAD")
        downloadCache.removeResource(itemId)
        DownloadService.sendRemoveDownload(context, HolodexDownloadService::class.java, itemId
    }

    override suspend fun cancelDownload(itemId: String) {
        DownloadService.sendRemoveDownload(context, HolodexDownloadService::class.java, itemId
        unifiedDao.deleteInteraction(itemId, "DOWNLOAD")
    }

    override suspend fun resumeDownload(itemId: String) {
        DownloadService.sendResumeDownloads(context, HolodexDownloadService::class.java, true)
        unifiedDao.updateDownloadStatus(itemId, DownloadStatus.ENQUEUED.name)
    }

    override suspend fun reconcileAllDownloads() {
        withContext(Dispatchers.IO) {
            Timber.i("Reconciling downloads: Checking for zombie states...")

            // 1. Get all items the DB thinks are downloading
            val activeInDb = unifiedDao.getAllDownloadsOneShot().filter {
                it.downloadStatus == DownloadStatus.DOWNLOADING.name ||
                        it.downloadStatus == DownloadStatus.ENQUEUED.name ||
                        it.downloadStatus == DownloadStatus.PROCESSING.name
            }

            // 2. Get what Media3 actually knows about
            val actuallyRunning = media3DownloadManager.currentDownloads.map { it.request.id }

            var fixedCount = 0
            for (item in activeInDb) {
                // If DB says downloading, but Media3 doesn't know about it, it's a zombie.
                if (!actuallyRunning.contains(item.itemId)) {
                    Timber.w("Found zombie download: ${item.itemId}. Marking as FAILED.")
                    unifiedDao.updateDownloadStatus(item.itemId, DownloadStatus.FAILED.name)
                    fixedCount++
                }
            }

            if (fixedCount > 0) {
                Timber.i("Reconciliation complete. Fixed $fixedCount zombie downloads.")
            }
        }
    }


    override suspend fun rescanStorageForDownloads() {
        withContext(Dispatchers.IO) {
            val musicDir = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY
            val appMusicDir = File(musicDir, DOWNLOAD_FOLDER_NAME)
```

```
            if (!appMusicDir.exists() || !appMusicDir.isDirectory) return@withContext
            val mediaFiles = appMusicDir.listFiles { _, name -> name.endsWith(".m4a") } ?: ret
            val existingIds = unifiedDao.getAllDownloadsOneShot().map { it.itemId }.toSet()
            for (file in mediaFiles) {
                try {
                    val audioFile = AudioFileIO.read(file)
                    val comment = audioFile.tag?.getFirst(FieldKey.COMMENT)
                    if (comment != null && comment.startsWith("holodex_item_id::")) {
                        val itemId = comment.substringAfter("holodex_item_id::")
                        if (!existingIds.contains(itemId)) {
                            val title = audioFile.tag?.getFirst(FieldKey.TITLE) ?: "Unknown"
                            val artist = audioFile.tag?.getFirst(FieldKey.ARTIST) ?: "Unknown"
                            val duration = audioFile.audioHeader.trackLength.toLong()
                            val parentId = itemId.split("_").firstOrNull() ?: itemId
                            val start = itemId.split("_").getOrNull(1)?.toLongOrNull() ?: 0L

                            val meta = UnifiedMetadataEntity(
                                id = itemId, title = title, artistName = artist, type = "SEGME
                                specificArtUrl = null, uploaderAvatarUrl = null, duration = du
                                channelId = "", parentVideoId = parentId, startSeconds = start
                                lastUpdatedAt = System.currentTimeMillis()
                            )
                            unifiedDao.upsertMetadata(meta)

                            val interaction = UserInteractionEntity(
                                itemId = itemId, interactionType = "DOWNLOAD", timestamp = fil
                                localFilePath = Uri.fromFile(file).toString(),
                                downloadStatus = DownloadStatus.COMPLETED.name,
                                downloadFileName = file.name,
                                downloadProgress = 100
                            )
                            unifiedDao.upsertInteraction(interaction)
                        }
                    }
                } catch (e: Exception) { Timber.e(e, "Failed to scan file: ${file.name}") }
            }
        }
    }

    override suspend fun postDownloadCompletedEvent(event: DownloadRepository.DownloadComplete
        _downloadCompletedEvents.emit(event)
    }
}

// File: java\com\example\holodex\data\repository\HolodexRepository.kt
// File: java/com/example/holodex/data/repository/HolodexRepository.kt
package com.example.holodex.data.repository

import androidx.media3.common.util.UnstableApi
import androidx.room.withTransaction
import com.example.holodex.auth.TokenManager
import com.example.holodex.background.LogAction
import com.example.holodex.background.SyncLogger
import com.example.holodex.data.api.AuthenticatedMusicdexApiService
import com.example.holodex.data.api.HolodexApiService
```

```
import com.example.holodex.data.api.LatestSongsRequest
import com.example.holodex.data.api.MusicdexApiService
import com.example.holodex.data.api.Organization
import com.example.holodex.data.api.PaginatedChannelsResponse
import com.example.holodex.data.api.PaginatedSongsResponse
import com.example.holodex.data.api.PlaylistListResponse
import com.example.holodex.data.api.PlaylistUpdateRequest
import com.example.holodex.data.api.StarPlaylistRequest
import com.example.holodex.data.cache.BrowseCacheKey
import com.example.holodex.data.cache.BrowseListCache
import com.example.holodex.data.cache.CacheException
import com.example.holodex.data.cache.CachePolicy
import com.example.holodex.data.cache.FetcherResult
import com.example.holodex.data.cache.SearchCacheKey
import com.example.holodex.data.cache.SearchListCache
import com.example.holodex.data.db.AppDatabase
import com.example.holodex.data.db.CachedDiscoveryResponse
import com.example.holodex.data.db.DiscoveryDao
import com.example.holodex.data.db.LikedItemType
import com.example.holodex.data.db.PlaylistDao
import com.example.holodex.data.db.PlaylistEntity
import com.example.holodex.data.db.PlaylistItemEntity
import com.example.holodex.data.db.StarredPlaylistDao
import com.example.holodex.data.db.StarredPlaylistEntity
import com.example.holodex.data.db.SyncMetadataDao
import com.example.holodex.data.db.SyncStatus
import com.example.holodex.data.db.UnifiedDao
import com.example.holodex.data.db.VideoDao
import com.example.holodex.data.db.mappers.toEntity
import com.example.holodex.data.db.toEntity
import com.example.holodex.data.model.HolodexChannelMin
import com.example.holodex.data.model.HolodexSong
import com.example.holodex.data.model.HolodexVideoItem
import com.example.holodex.data.model.SearchCondition
import com.example.holodex.data.model.VideoSearchRequest
import com.example.holodex.data.model.discovery.ChannelDetails
import com.example.holodex.data.model.discovery.DiscoveryResponse
import com.example.holodex.data.model.discovery.FullPlaylist
import com.example.holodex.data.model.discovery.MusicdexSong
import com.example.holodex.data.model.discovery.PlaylistStub
import com.example.holodex.di.ApplicationScope
import com.example.holodex.di.DefaultDispatcher
import com.example.holodex.util.VideoFilteringUtil
import com.example.holodex.viewmodel.UnifiedDisplayItem
import com.example.holodex.viewmodel.VideoListViewModel
import com.example.holodex.viewmodel.state.BrowseFilterState
import com.example.holodex.viewmodel.state.ViewTypePreset
import kotlinx.coroutines.CoroutineDispatcher
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.async
import kotlinx.coroutines.awaitAll
import kotlinx.coroutines.coroutineScope
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.MutableStateFlow
```

```
import kotlinx.coroutines.flow.SharingStarted
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.flow.first
import kotlinx.coroutines.flow.map
import kotlinx.coroutines.flow.stateIn
import kotlinx.coroutines.launch
import kotlinx.coroutines.sync.Mutex
import kotlinx.coroutines.sync.withLock
import kotlinx.coroutines.sync.withPermit
import kotlinx.coroutines.withContext
import org.schabi.newpipe.extractor.NewPipe
import org.schabi.newpipe.extractor.ServiceList
import org.schabi.newpipe.extractor.StreamingService
import org.schabi.newpipe.extractor.stream.StreamInfoItem
import timber.log.Timber
import java.io.IOException
import java.net.URLEncoder
import java.nio.charset.StandardCharsets
import java.time.Instant
import java.util.concurrent.TimeUnit
import javax.inject.Inject
import javax.inject.Singleton

@UnstableApi
@Singleton
class HolodexRepository @Inject constructor(
    val holodexApiService: HolodexApiService,
    private val musicdexApiService: MusicdexApiService,
    private val authenticatedMusicdexApiService: AuthenticatedMusicdexApiService,
    private val discoveryDao: DiscoveryDao,
    private val browseListCache: BrowseListCache,
    private val searchListCache: SearchListCache,
    private val videoDao: VideoDao,
    val playlistDao: PlaylistDao,
    private val appDatabase: AppDatabase,
    @DefaultDispatcher private val defaultDispatcher: CoroutineDispatcher, // Use Qualifier
    internal val syncMetadataDao: SyncMetadataDao,
    private val starredPlaylistDao: StarredPlaylistDao,
    private val tokenManager: TokenManager,
    private val unifiedDao: UnifiedDao,
    private val unifiedRepository: UnifiedVideoRepository, // Add here
    @ApplicationScope private val applicationScope: CoroutineScope
) {

    companion object {
        private const val TAG = "HolodexRepository"
        private val DISCOVERY_CACHE_TTL_MS = TimeUnit.HOURS.toMillis(1)
        const val DEFAULT_PAGE_SIZE = 50
        val DEFAULT_MUSIC_TOPICS =
            listOf("singing", "Music_Cover", "Original_Song", "3D_Stream")
        val CACHE_STALE_DURATION_MS = TimeUnit.HOURS.toMillis(1)
        private val TAG_SYNC = "SYNC_DEBUG"
    }
```

```
private val browseNetworkMutex = Mutex()
private val searchNetworkMutex = Mutex()
private val videoDetailMutex = Mutex()

val likedItemIds: StateFlow<Set<String>> =
    unifiedDao.getLikedItemIds() // Uses new DAO
        .map { it.toSet() }
        .stateIn(
            scope = applicationScope,
            started = SharingStarted.WhileSubscribed(5000L),
            initialValue = emptySet()
        )

private val _availableOrganizations = MutableStateFlow<List<Pair<String, String?>>>(
    listOf("All Vtubers" to null, "Favorites" to "Favorites") // Initial default value
)
val availableOrganizations: StateFlow<List<Pair<String, String?>>> =
    _availableOrganizations.asStateFlow()

init {


    // Fetch the dynamic organization list as soon as the repository is created.
    applicationScope.launch {
        fetchOrganizationList()
    }
}

private suspend fun fetchOrganizationList() {
    getOrganizationList() // This is the existing function that calls the API
        .onSuccess { orgs ->
            val orgList = orgs.map { org -> (org.name to org.name) }
            // Prepend the static options to the dynamic list
            _availableOrganizations.value =
                listOf("All Vtubers" to null, "Favorites" to "Favorites") + orgList
        }
        .onFailure {
            Timber.e(it, "Failed to load dynamic organization list. Using hardcoded fallba
            // Populate with a fallback list on failure
            _availableOrganizations.value = listOf(
                "All Vtubers" to null,
                "Favorites" to "Favorites",
                "Hololive" to "Hololive",
                "Nijisanji" to "Nijisanji",
                "Independents" to "Independents"
            )
        }
}

suspend fun fetchBrowseList(
    key: BrowseCacheKey,
    forceNetwork: Boolean = false,
    cachePolicy: CachePolicy = CachePolicy.CACHE_FIRST
): Result<FetcherResult<HolodexVideoItem>> = withContext(defaultDispatcher) {
    Timber.d("$TAG: fetchBrowseList called. Key: ${key.stringKey()}, forceNetwork: $forceN
```

```
    try {
        when (cachePolicy) {
            CachePolicy.CACHE_FIRST -> fetchBrowseWithCacheFirst(key, forceNetwork)
            CachePolicy.NETWORK_FIRST -> fetchBrowseWithNetworkFirst(key)
            CachePolicy.CACHE_ONLY -> fetchBrowseFromCacheOnly(key)
            CachePolicy.NETWORK_ONLY -> fetchBrowseFromNetworkOnly(key)
        }
    } catch (e: Exception) {
        Timber.e(e, "$TAG: Unhandled exception in fetchBrowseList for key ${key.stringKey(
        Result.failure(
            CacheException.StorageError(
                "Failed to fetch browse list for key ${key.stringKey()}",
                e
            )
        )
    }
}

private suspend fun fetchBrowseWithCacheFirst(
    key: BrowseCacheKey,
    forceNetwork: Boolean
): Result<FetcherResult<HolodexVideoItem>> {
    if (!forceNetwork) {
        browseListCache.get(key)?.let {
            Timber.d("$TAG: Browse CACHE_FIRST hit for key: ${key.stringKey()}")
            return Result.success(it)
        }
    }
    Timber.d("$TAG: Browse CACHE_FIRST miss or forceNetwork for key: ${key.stringKey()}. F
    return fetchBrowseFromNetworkWithFallback(key)
}

private suspend fun fetchBrowseWithNetworkFirst(key: BrowseCacheKey): Result<FetcherResult
    Timber.d("$TAG: Browse NETWORK_FIRST for key: ${key.stringKey()}. Fetching from networ
    return fetchBrowseFromNetworkWithFallback(key)
}

private suspend fun fetchBrowseFromCacheOnly(key: BrowseCacheKey): Result<FetcherResult<Ho
    return browseListCache.get(key)?.let {
        Timber.d("$TAG: Browse CACHE_ONLY hit for key: ${key.stringKey()}")
        Result.success(it)
    }
        ?: Result.failure(CacheException.NotFound("No cached browse data for key ${key.str
}

private suspend fun fetchBrowseFromNetworkOnly(key: BrowseCacheKey): Result<FetcherResult<
    Timber.d("$TAG: Browse NETWORK_ONLY for key: ${key.stringKey()}. Fetching directly fro
    return fetchBrowseFromNetwork(key)
}

private suspend fun fetchBrowseFromNetworkWithFallback(key: BrowseCacheKey): Result<Fetche
    val networkResult = fetchBrowseFromNetwork(key)
    if (networkResult.isSuccess) {
        return networkResult
    } else {
```

```
        val networkError = networkResult.exceptionOrNull() ?: CacheException.NetworkError(
            "Unknown browse network error for ${key.stringKey()}",
            null
        )
        Timber.w(
            networkError,
            "$TAG: Browse network fetch failed for ${key.stringKey()}. Trying stale cache.
        )
        browseListCache.getStale(key)?.let { staleData ->
            Timber.d("$TAG: Browse using STALE cache for ${key.stringKey()} after network
            return Result.success(staleData)
        } ?: return Result.failure(networkError)
    }
}

private suspend fun fetchBrowseFromNetwork(key: BrowseCacheKey): Result<FetcherResult<Holo
    return browseNetworkMutex.withLock {
        try {
            val apiRequest = VideoSearchRequest(
                sort = key.filters.sortField.apiValue,
                target = listOf("stream", "clip"),
                topic = key.filters.selectedPrimaryTopic?.let { listOf(it) } ?: DEFAULT_MU
                org = key.filters.selectedOrganization?.let { listOf(it) },
                paginated = true,
                offset = key.pageOffset,
                limit = DEFAULT_PAGE_SIZE
            )
            val response = holodexApiService.searchVideosAdvanced(apiRequest)
            if (!response.isSuccessful || response.body() == null) {
                throw IOException("API Error")
            }

            val videosFromApi = response.body()!!.items

            // --- NEW SIMPLIFIED FILTERING LOGIC ---
            val filteredVideos = videosFromApi.filter { video ->
                // Must be music content AND longer than 60 seconds
                VideoFilteringUtil.isMusicContent(video) && video.duration > 60
            }
            // ------------------------------------

            val fetcherResult = FetcherResult(
                filteredVideos,
                response.body()?.getTotalAsInt(),
                key.pageOffset + filteredVideos.size
            )
            browseListCache.store(key, fetcherResult)
            Result.success(fetcherResult)
        } catch (e: Exception) {
            Result.failure(e)
        }
    }
}
```

```
suspend fun fetchSearchList(
    key: SearchCacheKey,
    forceNetwork: Boolean = false,
    cachePolicy: CachePolicy = CachePolicy.CACHE_FIRST
): Result<FetcherResult<HolodexVideoItem>> = withContext(defaultDispatcher) {
    Timber.d("$TAG: fetchSearchList called. Key: ${key.stringKey()}, forceNetwork: $forceN
    try {
        when (cachePolicy) {
            CachePolicy.CACHE_FIRST -> fetchSearchWithCacheFirst(key, forceNetwork)
            CachePolicy.NETWORK_FIRST -> fetchSearchWithNetworkFirst(key)
            CachePolicy.CACHE_ONLY -> fetchSearchFromCacheOnly(key)
            CachePolicy.NETWORK_ONLY -> fetchSearchFromNetworkOnly(key)
        }
    } catch (e: Exception) {
        Timber.e(e, "$TAG: Unhandled exception in fetchSearchList for key ${key.stringKey(
        Result.failure(
            CacheException.StorageError(
                "Failed to fetch search list for key ${key.stringKey()}",
                e
            )
        )
    }
}

private suspend fun fetchSearchWithCacheFirst(
    key: SearchCacheKey,
    forceNetwork: Boolean
): Result<FetcherResult<HolodexVideoItem>> {
    if (!forceNetwork) {
        searchListCache.get(key)?.let {
            Timber.d("$TAG: Search CACHE_FIRST hit for key: ${key.stringKey()}")
            return Result.success(it)
        }
    }
    Timber.d("$TAG: Search CACHE_FIRST miss or forceNetwork for key: ${key.stringKey()}. F
    return fetchSearchFromNetworkWithFallback(key)
}

private suspend fun fetchSearchWithNetworkFirst(key: SearchCacheKey): Result<FetcherResult
    Timber.d("$TAG: Search NETWORK_FIRST for key: ${key.stringKey()}. Fetching from networ
    return fetchSearchFromNetworkWithFallback(key)
}

private suspend fun fetchSearchFromCacheOnly(key: SearchCacheKey): Result<FetcherResult<Ho
    return searchListCache.get(key)?.let {
        Timber.d("$TAG: Search CACHE_ONLY hit for key: ${key.stringKey()}")
        Result.success(it)
    }
        ?: Result.failure(CacheException.NotFound("No cached search data for key ${key.str
}

private suspend fun fetchSearchFromNetworkOnly(key: SearchCacheKey): Result<FetcherResult<
    Timber.d("$TAG: Search NETWORK_ONLY for key: ${key.stringKey()}. Fetching directly fro
    return fetchSearchFromNetwork(key)
}
```

```
private suspend fun fetchSearchFromNetworkWithFallback(key: SearchCacheKey): Result<Fetche
    val networkResult = fetchSearchFromNetwork(key)
    if (networkResult.isSuccess) {
        return networkResult
    } else {
        val networkError = networkResult.exceptionOrNull() ?: CacheException.NetworkError(
            "Unknown search network error for ${key.stringKey()}",
            null
        )
        Timber.w(
            networkError,
            "$TAG: Search network fetch failed for ${key.stringKey()}. Trying stale cache.
        )
        searchListCache.getStale(key)?.let { staleData ->
            Timber.d("$TAG: Search using STALE cache for ${key.stringKey()} after network
            return Result.success(staleData)
        } ?: return Result.failure(networkError)
    }
}

fun getStarredPlaylistsFlow(): Flow<List<StarredPlaylistEntity>> {
    return starredPlaylistDao.getStarredPlaylists()
}

@UnstableApi
private suspend fun fetchSearchFromNetwork(key: SearchCacheKey): Result<FetcherResult<Holo
    return searchNetworkMutex.withLock {
        Timber.d("$TAG: Fetching SEARCH from network: Key=${key.stringKey()}")
        try {
            val actualTextSearchConditions: List<SearchCondition>?
            val actualChannelIdForVch: List<String>?

            if (key.query.startsWith(VideoListViewModel.CHANNEL_ID_SEARCH_PREFIX)) {
                val actualChannelId =
                    key.query.removePrefix(VideoListViewModel.CHANNEL_ID_SEARCH_PREFIX)
                actualChannelIdForVch =
                    if (actualChannelId.isNotBlank()) listOf(actualChannelId) else null
                actualTextSearchConditions = null
            } else {
                actualTextSearchConditions = listOf(SearchCondition(text = key.query))
                actualChannelIdForVch = null
            }

            val apiRequest = VideoSearchRequest(
                sort = "newest",
                target = listOf("stream", "clip"),
                conditions = actualTextSearchConditions,
                topic = DEFAULT_MUSIC_TOPICS,
                vch = actualChannelIdForVch,
                paginated = true,
                offset = key.pageOffset,
                limit = DEFAULT_PAGE_SIZE
            )
            val response = holodexApiService.searchVideosAdvanced(apiRequest)
```

```
            if (!response.isSuccessful || response.body() == null) {
                throw IOException("API Error (Search) for '${key.query}': ${response.code(
            }

            val videosFromApi = response.body()!!.items
            val musicallyRelevantVideos =
                videosFromApi.filter { VideoFilteringUtil.isMusicContent(it) }
            Timber.d(
                "$TAG: Search network fetch successful for ${key.stringKey()}. Items: ${mu
                    response.body()?.getTotalAsInt()
                }"
            )
            val fetcherResult = FetcherResult(
                musicallyRelevantVideos,
                response.body()?.getTotalAsInt(),
                key.pageOffset + musicallyRelevantVideos.size
            )
            searchListCache.store(key, fetcherResult)
            Result.success(fetcherResult)
        } catch (e: Exception) {
            Timber.e(
                e,
                "$TAG: Exception during search network fetch for key ${key.stringKey()}"
            )
            Result.failure(
                CacheException.NetworkError(
                    "Search network fetch failed for ${key.stringKey()}",
                    e
                )
            )
        }
    }
}

suspend fun getVideoWithSongs(
    videoId: String,
    forceRefresh: Boolean = false
): Result<HolodexVideoItem> = withContext(defaultDispatcher) {
    videoDetailMutex.withLock {
        if (!forceRefresh) {
            val cachedVideoWithSongs = videoDao.getVideoWithSongsOnce(videoId)
            if (cachedVideoWithSongs != null && System.currentTimeMillis() - cachedVideoWi
                Timber.d("$TAG: getVideoWithSongs (ID: $videoId) - Returning FRESH network
                return@withLock Result.success(cachedVideoWithSongs.toDomain())
            }
        }

        Timber.d("$TAG: getVideoWithSongs (ID: $videoId) - No suitable cached version foun
        try {
            val response = holodexApiService.getVideoWithSongs(
                videoId = videoId,
                include = "songs,live_info,description",
                lang = "en"
            )
```

```
            if (response.isSuccessful && response.body() != null) {
                val videoFromApi = response.body()!!
                videoFromApi.songs?.forEach { it.videoId = videoFromApi.id }

                val existingVideoEntity = videoDao.getVideoByIdOnce(videoId)
                val entityToSave = videoFromApi.toEntity(
                    queryKey = existingVideoEntity?.listQueryKey,
                    insertionOrder = existingVideoEntity?.insertionOrder ?: 0,
                    currentTimestamp = System.currentTimeMillis()
                )
                val songEntitiesToSave =
                    videoFromApi.songs?.map { it.toEntity(videoFromApi.id) } ?: emptyList(

                appDatabase.withTransaction {
                    videoDao.insertVideo(entityToSave)
                    videoDao.deleteSongsForVideo(videoFromApi.id)
                    if (songEntitiesToSave.isNotEmpty()) {
                        videoDao.insertSongs(songEntitiesToSave)
                    }
                }
                Result.success(videoFromApi)
            } else {
                val errorBody = response.errorBody()?.string() ?: "Unknown API error"
                Timber.e("$TAG: API Error ${response.code()} for getVideoWithSongs ($video
                Result.failure(IOException("API Error ${response.code()} for $videoId: $er
            }
        } catch (e: Exception) {
            Timber.e(e, "$TAG: Network Exception for getVideoWithSongs ($videoId)")
            val staleVideo = videoDao.getVideoWithSongsOnce(videoId)
            if (staleVideo != null) {
                Timber.w("$TAG: Network failed, but returning STALE cached data for $video
                Result.success(staleVideo.toDomain())
            } else {
                Result.failure(
                    CacheException.NetworkError(
                        "Network fetch failed for $videoId and no cache is available.",
                        e
                    )
                )
            }
        }
    }
}

fun getItemsForPlaylist(playlistId: Long): Flow<List<PlaylistItemEntity>> =
    playlistDao.getItemsForPlaylist(playlistId)

fun getAllPlaylists(): Flow<List<PlaylistEntity>> = playlistDao.getAllPlaylists()
suspend fun getLastItemOrderInPlaylist(playlistId: Long): Int? =
    withContext(defaultDispatcher) { playlistDao.getLastItemOrder(playlistId) }

suspend fun getPlaylistById(playlistId: Long): PlaylistEntity? =
    withContext(defaultDispatcher) { playlistDao.getPlaylistById(playlistId) }
```

```
suspend fun clearAllCachedData() = withContext(Dispatchers.IO) {
    Timber.i("$TAG: Clearing temporary application caches.")
    browseListCache.clear()
    searchListCache.clear()
    appDatabase.withTransaction {
        videoDao.clearAllSongs()
        videoDao.clearAllVideos()
    }
    Timber.i("$TAG: Temporary application caches cleared from repository.")
}

suspend fun cleanupExpiredCacheEntries() = withContext(Dispatchers.IO) {
    Timber.d("$TAG: Cleaning up expired cache entries.")
    browseListCache.cleanupExpiredEntries()
    searchListCache.cleanupExpiredEntries()
}



suspend fun getFavoritesFeed(
    channels: List<UnifiedDisplayItem>,
    filters: BrowseFilterState,
    offset: Int
): Result<FetcherResult<HolodexVideoItem>> = withContext(defaultDispatcher) {

    Timber.e("========== DEBUG: FAVORITES FEED INPUT ==========")
    Timber.e("Total Input Channels: ${channels.size}")
    channels.forEachIndexed { index, item ->
        // We log the Title and isExternal flag for every channel to identify the culprit
        Timber.e("[$index] ${item.title} | ID: ${item.playbackItemId} | isExternal: ${item
    }
    Timber.e("================================================")

    if (channels.isEmpty()) {
        return@withContext Result.success(FetcherResult(emptyList(), 0))
    }

    try {
        val holodexChannels = channels.filter { !it.isExternal }.map { it.playbackItemId }
        val externalChannels = channels.filter { it.isExternal }.map { it.playbackItemId }

        Timber.d("FavoritesFeed: Input -> Holodex IDs: ${holodexChannels.size}, External I

        // If Holodex IDs are 0, the Mapper 'isExternal' logic is still the root cause.

        val semaphore = kotlinx.coroutines.sync.Semaphore(10)

        // 1. Fetch Holodex
        val holodexResults = if (holodexChannels.isNotEmpty()) {
            coroutineScope {
                holodexChannels.map { channelId ->
                    async {
                        semaphore.withPermit {
                            try {
                                val request = VideoSearchRequest(
                                    sort = filters.sortField.apiValue,
```

```
                            vch = listOf(channelId),
                            topic = filters.selectedPrimaryTopic?.let { listOf(it)
                            paginated = true,
                            offset = 0,
                            limit = 15,
                            target = listOf("stream", "clip")
                        )
                        val res = holodexApiService.searchVideosAdvanced(request).
                        // Debug log per channel
                         Timber.v("Holodex Fetch $channelId: ${res.size} items")
                        res
                    } catch (e: Exception) {
                        Timber.e(e, "Failed to fetch Holodex favorites for $channe
                        emptyList()
                    }
                }
            }
        }.awaitAll().flatten()
    }
} else emptyList()

// 2. Fetch External
val externalResults = if (externalChannels.isNotEmpty()) {
    // LIMIT CONCURRENCY FOR NEWPIPE TO 4
    // NewPipe uses heavy HTML parsing which can cause OOM/Native crashes if run t
    val externalSemaphore = kotlinx.coroutines.sync.Semaphore(4)

    coroutineScope {
        externalChannels.map { extId ->
            async {
                externalSemaphore.withPermit {
                    getMusicFromExternalChannel(extId, null).getOrNull()?.data ?:
                }
            }
        }.awaitAll().flatten()
    }
} else emptyList()

Timber.d("FavoritesFeed: Raw Results -> Holodex: ${holodexResults.size}, External:

// 3. Merge & Filter
val allVideos = holodexResults + externalResults

// Filter duration > 60s
val validVideos = allVideos.filter { it.duration > 60 }

// 4. Robust Sorting
// Parse the 'availableAt' string to Instant/Millis for correct comparison
val sortedList = validVideos
    .distinctBy { it.id }
    .sortedByDescending { video ->
        try {
            // Handle ISO 8601 string
            if (video.availableAt.isNotEmpty()) {
                java.time.Instant.parse(video.availableAt).toEpochMilli()
```

```
                    } else 0L
                } catch (e: Exception) {
                    0L // Fallback for bad dates
                }
            }

        // 5. Paginate
        val paginatedList = if (sortedList.size > offset) {
            sortedList.drop(offset).take(DEFAULT_PAGE_SIZE)
        } else {
            emptyList()
        }

        Result.success(FetcherResult(paginatedList, totalAvailable = null))

    } catch (e: Exception) {
        Timber.e(e, "Error in getFavoritesFeed")
        Result.failure(e)
    }
}

suspend fun getUpcomingMusicPaginated(
    org: String?,
    offset: Int
): Result<FetcherResult<HolodexVideoItem>> {
    val filters = BrowseFilterState.create(
        preset = ViewTypePreset.UPCOMING_STREAMS,
        organization = org,
    )
    val key = BrowseCacheKey(filters, offset)
    return fetchBrowseList(key, forceNetwork = true)
}

suspend fun getDiscoveryHubContent(org: String): Result<DiscoveryResponse> =
    withContext(defaultDispatcher) {
        val cacheKey = "discovery_org_$org"
        try {
            val cachedResponse = discoveryDao.getResponse(cacheKey)
            if (cachedResponse != null) {
                val isStale =
                    System.currentTimeMillis() - cachedResponse.timestamp > DISCOVERY_CACH
                Timber.d("$TAG: Discovery Hub cache HIT for key '$cacheKey'. Is stale: $is
                if (!isStale) {
                    return@withContext Result.success(cachedResponse.data)
                } else {
                    launch { fetchAndCacheDiscoveryContent(org) }
                    return@withContext Result.success(cachedResponse.data)
                }
            }
            Timber.d("$TAG: Discovery Hub cache MISS for key '$cacheKey'. Fetching from ne
            return@withContext fetchAndCacheDiscoveryContent(org)

        } catch (e: Exception) {
            Timber.e(e, "$TAG: Exception in getDiscoveryHubContent for org '$org'")
            Result.failure(e)
```

```
        }
    }

    suspend fun getFavoritesHubContent(): Result<DiscoveryResponse> =
        withContext(defaultDispatcher) {
            val cacheKey = "discovery_favorites"
            try {
                val cachedResponse = discoveryDao.getResponse(cacheKey)
                if (cachedResponse != null) {
                    val isStale =
                        System.currentTimeMillis() - cachedResponse.timestamp > DISCOVERY_CACH
                    Timber.d("$TAG: Favorites Hub cache HIT for key '$cacheKey'. Is stale: $is
                    if (!isStale) {
                        return@withContext Result.success(cachedResponse.data)
                    } else {
                        launch { fetchAndCacheFavoritesContent() }
                        return@withContext Result.success(cachedResponse.data)
                    }
                }
                Timber.d("$TAG: Favorites Hub cache MISS for key '$cacheKey'. Fetching from ne
                return@withContext fetchAndCacheFavoritesContent()

            } catch (e: Exception) {
                Timber.e(e, "$TAG: Exception in getFavoritesHubContent")
                Result.failure(e)
            }
        }

    suspend fun getHotSongsForCarousel(org: String?): Result<List<MusicdexSong>> =
        withContext(defaultDispatcher) {
            try {
                val response = holodexApiService.getHotSongs(organization = org, channelId = n
                if (response.isSuccessful && response.body() != null) {
                    Result.success(response.body()!!)
                } else {
                    Result.failure(IOException("API Error fetching hot songs for org '$org': $
                }
            } catch (e: Exception) {
                Result.failure(e)
            }
        }

    @JvmName("getHotSongsForCarouselByChannelId")
    suspend fun getHotSongsForCarousel(channelId: String): Result<List<MusicdexSong>> =
        withContext(defaultDispatcher) {
            try {
                val response =
                    holodexApiService.getHotSongs(organization = null, channelId = channelId)
                if (response.isSuccessful && response.body() != null) {
                    Result.success(response.body()!!)
                } else {
                    Result.failure(IOException("API Error fetching hot songs for channel '$cha
                }
            } catch (e: Exception) {
                Result.failure(e)
```

```
        }
    }

    suspend fun getFullPlaylistContent(playlistId: String): Result<FullPlaylist> =
        withContext(defaultDispatcher) {
            try {
                Timber.d("$TAG: Fetching full playlist content from network for ID: $playlistI

                val isSystemPlaylist = playlistId.startsWith(":")

                val response = if (isSystemPlaylist) {
                    // System playlists like :history and :video require authentication
                    authenticatedMusicdexApiService.getPlaylistContent(playlistId)
                } else {
                    musicdexApiService.getPlaylistContent(playlistId)
                }

                if (response.isSuccessful && response.body() != null) {
                    Result.success(response.body()!!)
                } else {
                    Result.failure(IOException("API Error fetching playlist content for '$play
                }
            } catch (e: Exception) {
                Result.failure(e)
            }
        }

    private suspend fun fetchAndCacheDiscoveryContent(org: String): Result<DiscoveryResponse>
        return try {
            val response = musicdexApiService.getDiscoveryForOrg(org)
            if (response.isSuccessful && response.body() != null) {
                val discoveryResponse = response.body()!!
                val cacheEntry = CachedDiscoveryResponse(
                    pageKey = "discovery_org_$org",
                    data = discoveryResponse
                )
                discoveryDao.insertResponse(cacheEntry)
                Timber.i("$TAG: Successfully fetched and cached discovery content for org '$or
                Result.success(discoveryResponse)
            } else {
                Result.failure(IOException("API Error fetching discovery content for org '$org
            }
        } catch (e: Exception) {
            Result.failure(e)
        }
    }

    suspend fun getDiscoveryForChannel(channelId: String): Result<DiscoveryResponse> =
        withContext(defaultDispatcher) {
            try {
                val response = musicdexApiService.getDiscoveryForChannel(channelId)
                if (response.isSuccessful && response.body() != null) {
                    Result.success(response.body()!!)
                } else {
                    Result.failure(IOException("API Error fetching channel discovery: ${respon
```

```
            }
        } catch (e: Exception) {
            Result.failure(e)
        }
    }

private suspend fun fetchAndCacheFavoritesContent(): Result<DiscoveryResponse> {
    return try {
        val response = authenticatedMusicdexApiService.getDiscoveryForFavorites()
        if (response.isSuccessful && response.body() != null) {
            val discoveryResponse = response.body()!!
            val cacheEntry = CachedDiscoveryResponse(
                pageKey = "discovery_favorites",
                data = discoveryResponse
            )
            discoveryDao.insertResponse(cacheEntry)
            Timber.i("$TAG: Successfully fetched and cached favorites discovery content.")
            Result.success(discoveryResponse)
        } else {
            Result.failure(IOException("API Error fetching favorites discovery content: ${
        }
    } catch (e: Exception) {
        Result.failure(e)
    }
}

suspend fun getLatestSongsPaginated(
    offset: Int,
    limit: Int = 25
): Result<PaginatedSongsResponse> {
    return try {
        val request = LatestSongsRequest(offset = offset, limit = limit, paginated = true)
        val response = holodexApiService.getLatestSongs(request)
        if (response.isSuccessful && response.body() != null) {
            Result.success(response.body()!!)
        } else {
            Result.failure(IOException("API Error fetching latest songs: ${response.code()}
        }
    } catch (e: Exception) {
        Timber.e(e, "$TAG: Failed to fetch latest songs.")
        Result.failure(e)
    }
}

suspend fun getOrgChannelsPaginated(
    org: String,
    offset: Int,
    limit: Int = 25
): Result<PaginatedChannelsResponse> { // The return type to the ViewModel remains the sam
    return try {
        val response = holodexApiService.getChannels(
            organization = org,
            offset = offset,
            limit = limit
        )
```

```
        if (response.isSuccessful && response.body() != null) {
            val channelsList = response.body()!!
            // Manually construct the PaginatedChannelsResponse object.
            // The 'total' will be null, but the view model already handles this.
            val paginatedResponse = PaginatedChannelsResponse(
                total = null, // The API doesn't provide a total in this format
                items = channelsList
            )
            Result.success(paginatedResponse)
        } else {
            Result.failure(IOException("API Error fetching org channels for '$org': ${resp
        }
    } catch (e: Exception) {
        Timber.e(e, "Failed to fetch org channels for: $org")
        Result.failure(e)
    }
}

suspend fun getRadioContent(radioId: String): Result<FullPlaylist> =
    withContext(defaultDispatcher) {
        try {
            Timber.d("$TAG: Fetching radio content from network for ID: $radioId")
            val response = musicdexApiService.getRadioContent(radioId)
            if (response.isSuccessful && response.body() != null) {
                Result.success(response.body()!!)
            } else {
                Result.failure(IOException("API Error fetching radio content for '$radioId
            }
        } catch (e: Exception) {
            Result.failure(e)
        }
    }

suspend fun getOrgPlaylistsPaginated(
    org: String,
    type: String,
    offset: Int,
    limit: Int = 25
): Result<PlaylistListResponse> {
    return try {
        val response = musicdexApiService.getOrgPlaylists(
            org = org.ifBlank { "All_Vtubers" },
            type = type,
            offset = offset,
            limit = limit
        )
        if (response.isSuccessful && response.body() != null) {
            Result.success(response.body()!!)
        } else {
            Result.failure(IOException("API Error fetching org playlists (type: $type): ${
        }
    } catch (e: Exception) {
        Timber.e(e, "$TAG: Failed to fetch org playlists for org: $org, type: $type")
        Result.failure(e)
    }
```

```
}

suspend fun getChannelDetails(channelId: String): Result<ChannelDetails> =
    withContext(defaultDispatcher) {
        try {
            val response = holodexApiService.getChannelDetails(channelId)
            if (response.isSuccessful && response.body() != null) {
                Result.success(response.body()!!)
            } else {
                Result.failure(IOException("API Error fetching channel details: ${response
            }
        } catch (e: Exception) {
            Result.failure(e)
        }
    }

suspend fun fetchVideoAndFindSong(
    videoId: String,
    startTime: Int
): Pair<HolodexVideoItem, MusicdexSong?>? {
    val sgpId = withContext(Dispatchers.IO) {
        URLEncoder.encode(":video[id=$videoId]", StandardCharsets.UTF_8.toString())
    }
    val result = musicdexApiService.getPlaylistContent(sgpId)
    if (result.isSuccessful && result.body() != null) {
        val fullPlaylist = result.body()!!
        val matchingSong = fullPlaylist.content?.find { it.start == startTime }

        val videoItemShell = HolodexVideoItem(
            id = videoId,
            title = fullPlaylist.title,
            description = fullPlaylist.description,
            type = "stream",
            topicId = null,
            availableAt = matchingSong?.available_at?.toString() ?: "",
            publishedAt = null,
            duration = 0,
            status = "past",
            channel = HolodexChannelMin(
                id = matchingSong?.channel?.id ?: matchingSong?.channelId,
                name = matchingSong?.channel?.name ?: "Unknown",
                englishName = matchingSong?.channel?.englishName,
                org = null,
                type = "vtuber",
                photoUrl = matchingSong?.channel?.photoUrl
            ),
            songcount = fullPlaylist.content?.size,
            songs = fullPlaylist.content?.map { it.toHolodexSong() }
        )
        return Pair(videoItemShell, matchingSong)
    }
    return null
}
```

```
// --- Playlists ---
suspend fun getLocalPlaylists(): List<PlaylistEntity> = playlistDao.getAllPlaylistsOnce()
suspend fun getLocalPlaylistsByStatus(status: SyncStatus): List<PlaylistEntity> =
    playlistDao.getUnsyncedPlaylists().filter { it.syncStatus == status }

suspend fun performUpstreamPlaylistDeletions(logger: SyncLogger) {
    val pendingDeletes = getLocalPlaylistsByStatus(SyncStatus.PENDING_DELETE)
    if (pendingDeletes.isNotEmpty()) logger.info("  Processing ${pendingDeletes.size} pend
    pendingDeletes.forEach { playlist ->
        if (playlist.serverId != null) {
            val response = authenticatedMusicdexApiService.deletePlaylist(playlist.serverI
            if (response.isSuccessful || response.code() == 404) {
                logger.logItemAction(
                    LogAction.UPSTREAM_DELETE_SUCCESS,
                    playlist.name,
                    playlist.playlistId,
                    playlist.serverId
                )
                playlistDao.deletePlaylist(playlist.playlistId)
            } else {
                logger.logItemAction(
                    LogAction.UPSTREAM_DELETE_FAILED,
                    playlist.name,
                    playlist.playlistId,
                    playlist.serverId,
                    "Code: ${response.code()}"
                )
            }
        } else {
            playlistDao.deletePlaylist(playlist.playlistId) // Local-only item
        }
    }
}

suspend fun performUpstreamPlaylistUpserts(logger: SyncLogger) {
    val dirtyPlaylists = getLocalPlaylistsByStatus(SyncStatus.DIRTY)
    val userId = tokenManager.getUserId()?.toLongOrNull() ?: return
    if (dirtyPlaylists.isNotEmpty()) logger.info("  Processing ${dirtyPlaylists.size} dirt

    dirtyPlaylists.forEach { playlist ->
        val songServerIds = getLocalPlaylistItemServerIds(playlist.playlistId)
        val requestDto = PlaylistUpdateRequest(
            id = playlist.serverId,
            owner = userId,
            title = playlist.name,
            description = playlist.description,
            content = songServerIds
        )
        val response = authenticatedMusicdexApiService.createOrUpdatePlaylist(requestDto)

        if (response.isSuccessful) {
            if (playlist.serverId != null) {
                logger.logItemAction(
                    LogAction.UPSTREAM_UPSERT_SUCCESS,
                    playlist.name,
```

```
                    playlist.playlistId,
                    playlist.serverId
                )
                val updatedEntity = playlist.copy(syncStatus = SyncStatus.SYNCED)
                playlistDao.updatePlaylist(updatedEntity)
            } else {
                val newServerPlaylist = response.body()?.firstOrNull()
                if (newServerPlaylist != null) {
                    logger.logItemAction(
                        LogAction.UPSTREAM_UPSERT_SUCCESS,
                        newServerPlaylist.title,
                        playlist.playlistId,
                        newServerPlaylist.id
                    )
                    val finalEntity = newServerPlaylist.toEntity().copy(
                        playlistId = playlist.playlistId, // Keep the original local ID
                        syncStatus = SyncStatus.SYNCED
                    )
                    playlistDao.updatePlaylist(finalEntity)
                } else {
                    logger.logItemAction(
                        LogAction.UPSTREAM_UPSERT_FAILED,
                        playlist.name,
                        playlist.playlistId,
                        null,
                        "Server returned success but no playlist data."
                    )
                }
            }
        } else {
            logger.logItemAction(
                LogAction.UPSTREAM_UPSERT_FAILED,
                playlist.name,
                playlist.playlistId,
                playlist.serverId,
                "Code: ${response.code()}"
            )
        }
    }
}

suspend fun getRemotePlaylists(): List<PlaylistEntity> {
    val dtoList = authenticatedMusicdexApiService.getMyPlaylists().body() ?: emptyList()
    return dtoList.map { it.toEntity() } // Map the whole list
}

suspend fun insertNewSyncedPlaylists(playlists: List<PlaylistEntity>) {
    playlistDao.upsertPlaylists(playlists.map { it.copy(syncStatus = SyncStatus.SYNCED) })
}

suspend fun deleteLocalPlaylists(localIds: List<Long>) =
    localIds.forEach { playlistDao.deletePlaylist(it) }

suspend fun updateLocalPlaylistMetadata(localId: Long, remotePlaylist: PlaylistEntity) {
    // Use the surgical UPDATE query - this should NOT trigger CASCADE
```

```
    playlistDao.updatePlaylistMetadata(
        playlistId = localId,
        name = remotePlaylist.name,
        description = remotePlaylist.description,
        timestamp = remotePlaylist.last_modified_at
    )

    // Diagnostic: Verify items still exist after metadata update
    val itemsAfterUpdate = playlistDao.getItemsForPlaylist(localId).first()
    Timber.tag("SYNC_DEBUG").i(
        "After updateLocalPlaylistMetadata: Playlist $localId has ${itemsAfterUpdate.size}
    )
}


suspend fun getRemotePlaylistContent(serverId: String): List<MusicdexSong> =
    authenticatedMusicdexApiService.getPlaylistContent(serverId).body()?.content ?: emptyL

// --- MODIFIED TO FILTER LOCAL-ONLY ITEMS ---
suspend fun getLocalPlaylistItemServerIds(localPlaylistId: Long): List<String> =
    coroutineScope {
        val items = playlistDao.getItemsForPlaylist(localPlaylistId).first()

        // *** THE CRITICAL FILTER ***
        // Only consider items that are meant to be synced to the server.
        val syncedItems = items.filter { !it.isLocalOnly }

        // Asynchronously fetch the server ID for each item in the playlist.
        // This is necessary because the playlist only stores a reference (videoId + start
        // not the song's unique server UUID needed for the sync.
        syncedItems.map { playlistItem ->
            async {
                playlistItem.songStartSecondsPlaylist?.let { startTime ->
                    // This helper function fetches the video details from the API and fin
                    // song with the matching start time to get its server ID.
                    fetchVideoAndFindSong(playlistItem.videoIdForItem, startTime)?.second?
                }
            }
        }.awaitAll().filterNotNull() // Launch all lookups in parallel, wait for them, and
    }
// --- END OF MODIFICATION ---

suspend fun reconcileLocalPlaylistItems(localPlaylistId: Long, remoteSongs: List<MusicdexS
    appDatabase.withTransaction {
        // IMPORTANT: Read items at the START of the transaction to get fresh data
        // Using .first() on a Flow inside a transaction should give us the current state
        val localItems = playlistDao.getItemsForPlaylist(localPlaylistId).first()

        Timber.tag("SYNC_DEBUG").i(
            "reconcileLocalPlaylistItems START: Found ${localItems.size} local items for p
        )

        val localOnlyItems = localItems.filter { it.isLocalOnly }

        Timber.tag("SYNC_DEBUG").i(
```

```
            "reconcileLocalPlaylistItems: Filtered ${localOnlyItems.size} local-only items
        )

        // --- Step 2: Prepare the "Server Truth" ---
        val remoteItemEntities = remoteSongs.mapIndexedNotNull { index, song ->
            if (song.channelId.isNullOrBlank()) {
                Timber.w("Skipping playlist song ('${song.name}') because its top-level 'c
                return@mapIndexedNotNull null
            }

            // FIX: Manually construct the ID string.
            // LikedItemEntity.generateSongItemId(song.videoId, song.start) was just this:
            val compositeId = "${song.videoId}_${song.start}"

            PlaylistItemEntity(
                playlistOwnerId = localPlaylistId,
                itemIdInPlaylist = compositeId, // <--- Fixed here
                videoIdForItem = song.videoId,
                itemTypeInPlaylist = LikedItemType.SONG_SEGMENT,
                songStartSecondsPlaylist = song.start,
                songEndSecondsPlaylist = song.end,
                songNamePlaylist = song.name,
                songArtistTextPlaylist = song.channel.name,
                songArtworkUrlPlaylist = song.artUrl,
                itemOrder = index,
                syncStatus = SyncStatus.SYNCED,
                isLocalOnly = false
            )
        }

        // --- Step 3: Construct the new "Final Truth" by merging ---
        val finalMergedList = remoteItemEntities.toMutableList()
        finalMergedList.addAll(localOnlyItems)
        val finalListWithCorrectOrder = finalMergedList.mapIndexed { index, item ->
            item.copy(itemOrder = index)
        }

        // --- Step 4: Execute Database Operations ---
        // Delete all items first, then insert the merged list
        playlistDao.deleteAllItemsForPlaylist(localPlaylistId)

        if (finalListWithCorrectOrder.isNotEmpty()) {
            playlistDao.upsertPlaylistItems(finalListWithCorrectOrder)
        }

        Timber.tag("SYNC_DEBUG").i(
            "Reconciled playlist ID $localPlaylistId. Kept ${localOnlyItems.size} local-on
        )
    }
}

suspend fun savePlaylistEdits(
    editedPlaylist: PlaylistEntity,
    finalItems: List<PlaylistItemEntity>
) = withContext(defaultDispatcher) {
```

```
    val itemsToSave = finalItems.mapIndexed { index, item ->
        item.copy(itemOrder = index)
    }

    playlistDao.updatePlaylistAndItems(editedPlaylist, itemsToSave)

    if (editedPlaylist.syncStatus == SyncStatus.DIRTY) {
        Timber.tag(TAG_SYNC).i("Saved edits for playlist '${editedPlaylist.name}'. Marked
    } else {
        Timber.tag(TAG_SYNC).i("Saved local-only edits for playlist '${editedPlaylist.name
    }
}

suspend fun createNewPlaylist(name: String, description: String? = null): Long =
    withContext(defaultDispatcher) {
        val now = Instant.now().toString()
        val userId = tokenManager.getUserId()
        if (userId == null) {
            Timber.e("Cannot create playlist: User is not logged in.")
            throw IOException("User not logged in.")
        }
        playlistDao.insertPlaylist(
            PlaylistEntity(
                name = name.trim(),
                description = description?.trim(),
                syncStatus = SyncStatus.DIRTY,
                owner = userId.toLongOrNull(),
                createdAt = now,
                last_modified_at = now,
                isDeleted = false,
                serverId = null
            )
        )
    }

suspend fun deletePlaylist(playlistId: Long) = withContext(defaultDispatcher) {
    playlistDao.softDeletePlaylist(playlistId)
}

suspend fun addPlaylistItem(playlistItem: PlaylistItemEntity) = withContext(defaultDispatc
    appDatabase.withTransaction {
        // Only mark the playlist as dirty if the item being added is a syncable item.
        if (!playlistItem.isLocalOnly) {
            val parentPlaylist = playlistDao.getPlaylistById(playlistItem.playlistOwnerId)
            if (parentPlaylist != null) {
                playlistDao.updatePlaylist(
                    parentPlaylist.copy(
                        syncStatus = SyncStatus.DIRTY,
                        last_modified_at = Instant.now().toString()
                    )
                )
            }
        }

        // Always insert the new item, whether it's local or not.
```

```
        playlistDao.insertPlaylistItem(playlistItem)
    }
}
// Add these helper methods for diagnostics
suspend fun getPlaylistItemCount(localPlaylistId: Long): Int {
    return playlistDao.getItemsForPlaylist(localPlaylistId).first().size
}

suspend fun getLocalOnlyItemCount(localPlaylistId: Long): Int {
    return playlistDao.getItemsForPlaylist(localPlaylistId).first().count { it.isLocalOnly
}

// --- Starred Playlists ---
suspend fun performUpstreamStarredPlaylistsSync(logger: SyncLogger) {
    val toRemove = starredPlaylistDao.getUnsyncedItems()
        .filter { it.syncStatus == SyncStatus.PENDING_DELETE }
    toRemove.forEach {
        val response =
            authenticatedMusicdexApiService.unstarPlaylist(StarPlaylistRequest(it.playlist
        if (response.isSuccessful) {
            logger.info("  -> Successfully UNSTARRED playlist ${it.playlistId} on server."
            starredPlaylistDao.deleteById(it.playlistId)
        } else {
            logger.warning("  -> FAILED to unstar playlist ${it.playlistId}. Code: ${respo
        }
    }

    val toAdd =
        starredPlaylistDao.getUnsyncedItems().filter { it.syncStatus == SyncStatus.DIRTY }
    toAdd.forEach {
        val response =
            authenticatedMusicdexApiService.starPlaylist(StarPlaylistRequest(it.playlistId
        if (response.isSuccessful) {
            logger.info("  -> Successfully STARRED playlist ${it.playlistId} on server.")
            starredPlaylistDao.insert(it.copy(syncStatus = SyncStatus.SYNCED))
        } else {
            logger.warning("  -> FAILED to star playlist ${it.playlistId}. Code: ${respons
        }
    }
}

suspend fun getRemoteStarredPlaylists(): List<PlaylistStub> =
    authenticatedMusicdexApiService.getStarredPlaylists().body() ?: emptyList()

suspend fun getLocalUnsyncedStarredPlaylistsCount(): Int =
    starredPlaylistDao.getUnsyncedItems().size

suspend fun deleteLocalSyncedStarredPlaylists(): Int {
    val syncedItems = starredPlaylistDao.getStarredPlaylists().first()
        .filter { it.syncStatus == SyncStatus.SYNCED }
    if (syncedItems.isNotEmpty()) {
        starredPlaylistDao.deleteAllSyncedItems() // Assuming this method deletes where sy
    }
    return syncedItems.size
}
```

```
suspend fun insertRemoteStarredPlaylistsAsSynced(starred: List<PlaylistStub>) {
    val entities = starred.map { StarredPlaylistEntity(it.id, SyncStatus.SYNCED) }
    starredPlaylistDao.upsertAll(entities)
}

private fun MusicdexSong.toHolodexSong(): HolodexSong {
    return HolodexSong(
        name = this.name,
        start = this.start,
        end = this.end,
        itunesId = null,
        artUrl = this.artUrl,
        originalArtist = this.originalArtist,
        videoId = this.videoId
    )
}


suspend fun searchMusicOnChannels(
    query: String,
    channelIds: List<String>
): Result<List<HolodexVideoItem>> = withContext(defaultDispatcher) {
    if (channelIds.isEmpty()) return@withContext Result.success(emptyList())

    try {
        val ytService = NewPipe.getService(ServiceList.YouTube.serviceId)
        val allResults = coroutineScope {
            channelIds.map { channelId ->
                async {
                    searchSingleChannel(ytService, channelId, query)
                }
            }.awaitAll()
        }

        Result.success(allResults.flatten())
    } catch (e: Exception) {
        Timber.e(e, "Failed to perform external channel search")
        Result.failure(e)
    }
}

private suspend fun searchSingleChannel(
    ytService: StreamingService,
    channelId: String,
    query: String
): List<HolodexVideoItem> = try {
    val channelUrl = "https://www.youtube.com/channel/$channelId"

    val channelExtractor = ytService.getChannelExtractor(channelUrl)
    channelExtractor.fetchPage()

    val avatarUrl = channelExtractor.avatars.firstOrNull()?.url.orEmpty()
    val channelName = channelExtractor.name
```

```
    val videosTab = channelExtractor.tabs.find { tab ->
        tab.contentFilters.contains("videos")
    }

    if (videosTab != null) {
        val tabExtractor = ytService.getChannelTabExtractor(videosTab)
        tabExtractor.fetchPage()

        tabExtractor.initialPage.items
            .mapNotNull { it as? StreamInfoItem }
            .filter { it.name.contains(query, ignoreCase = true) }
            .map { item ->
                mapStreamInfoItemToHolodexVideoItem(
                    item,
                    channelId,
                    channelName,
                    avatarUrl
                )
            }
            .filter { VideoFilteringUtil.isMusicContent(it) }
    } else {
        emptyList()
    }
} catch (e: Exception) {
    Timber.e(e, "Failed to search within channel $channelId")
    emptyList()
}



suspend fun getMusicFromExternalChannel(
    channelId: String,
    nextPage: org.schabi.newpipe.extractor.Page?
): Result<FetcherResult<HolodexVideoItem>> = withContext(defaultDispatcher) {
    try {
        // ... (Keep existing NewPipe setup code) ...
        val ytService = NewPipe.getService(ServiceList.YouTube.serviceId)
        val channelUrl = "https://www.youtube.com/channel/$channelId"
        val channelExtractor = ytService.getChannelExtractor(channelUrl)
        channelExtractor.fetchPage()

        val videosTab = channelExtractor.tabs.firstOrNull { it.getUrl().contains("/videos"
            ?: return@withContext Result.failure(Exception("Could not find Videos tab"))

        val tabExtractor = ytService.getChannelTabExtractor(videosTab)
        val itemsPage = if (nextPage == null) {
            tabExtractor.fetchPage()
            tabExtractor.initialPage
        } else {
            tabExtractor.getPage(nextPage)
        }

        if (itemsPage == null || itemsPage.items.isEmpty()) {
            return@withContext Result.success(FetcherResult(emptyList(), null, null, null)
        }
```

```
        val videos = itemsPage.items.mapNotNull { it as? StreamInfoItem }
        val avatarUrl = channelExtractor.avatars.firstOrNull()?.url.orEmpty()

        val holodexItems = videos.map { item ->
            mapStreamInfoItemToHolodexVideoItem(item, channelId, channelExtractor.name, av
        }

        // --- NEW FILTERING LOGIC ---
        // Filter > 60s and Music Content
        val musicContent = holodexItems.filter {
            VideoFilteringUtil.isMusicContent(it) && it.duration > 60
        }
        // ---------------------------

        Result.success(
            FetcherResult(
                data = musicContent,
                totalAvailable = null,
                nextPageCursor = if (itemsPage.hasNextPage()) itemsPage.nextPage else null
            )
        )
    } catch (e: Exception) {
        Result.failure(e)
    }
}


private fun mapStreamInfoItemToHolodexVideoItem(
    item: StreamInfoItem,
    channelId: String,
    channelName: String,
    channelAvatar: String
): HolodexVideoItem {
    val timestamp = try {
        item.uploadDate?.offsetDateTime()?.toInstant() ?: Instant.now()
    } catch (e: Exception) { Instant.now() }

    val videoId = try {
        item.url.substringAfter("watch?v=").substringBefore("&")
    } catch (e: Exception) { item.url }

    return HolodexVideoItem(
        id = videoId,
        title = item.name,
        type = "stream",
        topicId = null, // External items don't have Holodex topics
        availableAt = timestamp.toString(),
        publishedAt = timestamp.toString(),
        duration = item.duration.takeIf { it > 0 } ?: 0,
        status = "past", // Assume external videos are past
        channel = HolodexChannelMin(
            id = channelId,
            name = channelName,
            englishName = null,
```

```
                org = "External", // Mark as External
                type = "vtuber",
                photoUrl = channelAvatar
            ),
            songcount = 0, // External items don't have a song count from Holodex
            description = null, // StreamInfoItem doesn't have a full description, StreamExtra
            songs = null
        )
    }

    suspend fun searchForExternalChannels(query: String): Result<List<com.example.holodex.data
        try {
            val ytService = NewPipe.getService(ServiceList.YouTube.serviceId)
            val extractor = ytService.getSearchExtractor(query, listOf("channels"), "")
            extractor.fetchPage()

            val results = extractor.initialPage.items.mapNotNull { it as? org.schabi.newpipe.e
                .map { infoItem ->
                    com.example.holodex.data.model.ChannelSearchResult(
                        channelId = infoItem.url.substringAfter("/channel/"),
                        name = infoItem.name,
                        thumbnailUrl = infoItem.thumbnails.firstOrNull()?.url,
                        subscriberCount = if (infoItem.subscriberCount > 0) "${infoItem.subscr
                    )
                }
            Result.success(results)
        } catch (e: Exception) {
            Timber.e(e, "Failed to search for external channels with query: $query")
            Result.failure(e)
        }
    }

    suspend fun getOrganizationList(): Result<List<Organization>> = withContext(defaultDispatc
        try {
            val response = holodexApiService.getOrganizations()
            if (response.isSuccessful && response.body() != null) {
                Result.success(response.body()!!)
            } else {
                Result.failure(IOException("API Error fetching organizations: ${response.code(
            }
        } catch (e: Exception) {
            Result.failure(e)
        }
    }
}

// File: java\com\example\holodex\data\repository\SearchHistoryRepository.kt
// File: java/com/example/holodex/data/repository/SearchHistoryRepository.kt
package com.example.holodex.data.repository

import android.content.SharedPreferences
import androidx.core.content.edit
import com.google.gson.Gson
import com.google.gson.reflect.TypeToken
import kotlinx.coroutines.CoroutineDispatcher
```

```
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.withContext
import timber.log.Timber

interface SearchHistoryRepository {
    val searchHistory: StateFlow<List<String>>
    suspend fun addSearchQueryToHistory(query: String)
    suspend fun loadSearchHistory()
    suspend fun clearSearchHistory()
}

class SharedPreferencesSearchHistoryRepository(
    private val sharedPreferences: SharedPreferences,
    private val gson: Gson,
    private val ioDispatcher: CoroutineDispatcher = Dispatchers.IO
) : SearchHistoryRepository {

    private companion object {
        const val PREF_SEARCH_HISTORY = "search_history_list_v1" // Moved from ViewModel
        const val MAX_SEARCH_HISTORY_SIZE = 10 // Moved from ViewModel
        const val TAG = "SearchHistoryRepo"
    }

    private val _searchHistoryFlow = MutableStateFlow<List<String>>(emptyList())
    override val searchHistory: StateFlow<List<String>> = _searchHistoryFlow.asStateFlow()

    override suspend fun loadSearchHistory() {
        withContext(ioDispatcher) {
            try {
                val historyJson = sharedPreferences.getString(PREF_SEARCH_HISTORY, null)
                _searchHistoryFlow.value = if (historyJson != null) {
                    gson.fromJson(historyJson, object : TypeToken<List<String>>() {}.type) ?:
                } else {
                    emptyList()
                }
                Timber.tag(TAG).d("Search history loaded: ${_searchHistoryFlow.value.size} ite
            } catch (e: Exception) {
                Timber.tag(TAG).e(e, "Failed to load search history")
                _searchHistoryFlow.value = emptyList()
            }
        }
    }

    private suspend fun saveSearchHistory(newHistory: List<String>) {
        withContext(ioDispatcher) {
            try {
                sharedPreferences.edit { putString(PREF_SEARCH_HISTORY, gson.toJson(newHistory
                Timber.tag(TAG).d("Search history saved.")
            } catch (e: Exception) {
                Timber.tag(TAG).e(e, "Failed to save search history")
            }
        }
```

```
    }

    override suspend fun addSearchQueryToHistory(query: String) {
        val currentHistory = _searchHistoryFlow.value.toMutableList().apply {
            remove(query) // Remove if exists to move to top
            add(0, query) // Add to the beginning
        }
        _searchHistoryFlow.value = currentHistory.take(MAX_SEARCH_HISTORY_SIZE) // Trim to max
        saveSearchHistory(_searchHistoryFlow.value)
    }

    override suspend fun clearSearchHistory() {
        _searchHistoryFlow.value = emptyList()
        withContext(ioDispatcher) {
            sharedPreferences.edit { remove(PREF_SEARCH_HISTORY) }
        }
        Timber.tag(TAG).d("Search history cleared.")
    }
}

// File: java\com\example\holodex\data\repository\SyncRepository.kt
package com.example.holodex.data.repository

import com.example.holodex.data.db.SyncStatus
import com.example.holodex.data.db.UnifiedDao
import com.example.holodex.data.db.UnifiedMetadataEntity
import com.example.holodex.data.db.UserInteractionEntity
import javax.inject.Inject
import javax.inject.Singleton

@Singleton
class SyncRepository @Inject constructor(
    private val unifiedDao: UnifiedDao
) {

    // --- GENERIC SYNC HELPERS ---

    suspend fun getDirtyItems(type: String): List<UserInteractionEntity> {
        return unifiedDao.getDirtyInteractions(type)
    }

    suspend fun getPendingDeleteItems(type: String): List<UserInteractionEntity> {
        return unifiedDao.getPendingDeleteInteractions(type)
    }

    suspend fun getSyncedItems(type: String): List<UserInteractionEntity> {
        return unifiedDao.getSyncedItems(type)
    }
    suspend fun getMetadata(id: String): UnifiedMetadataEntity? {
        return unifiedDao.getItemByIdOneShot(id)?.metadata
    }
    /**
     * Called after a successful upstream POST.
     * Updates local item with the Server ID and marks as SYNCED.
     */
```

```
suspend fun markAsSynced(itemId: String, type: String, serverId: String) {
    unifiedDao.confirmUpload(itemId, type, serverId)
}

/**
 * Updates the Server ID for an item without changing its sync status to SYNCED.
 * Useful for repairing orphans that need to remain DIRTY for the next sync pass.
 */
suspend fun updateServerId(itemId: String, type: String, serverId: String) {
    unifiedDao.updateServerId(itemId, type, serverId)
}

/**
 * Called after a successful upstream DELETE.
 * Removes the PENDING_DELETE row from the DB.
 */
suspend fun confirmDeletion(itemId: String, type: String) {
    unifiedDao.confirmDeletion(itemId, type)
}

// --- DOWNSTREAM HELPERS ---

/**
 * Inserts a new item found on the server.
 * NOTE: You must provide metadata because Foreign Keys require it.
 */
suspend fun insertRemoteItem(
    itemId: String,
    type: String,
    serverId: String,
    metadata: UnifiedMetadataEntity
) {
    // 1. Ensure Metadata exists (Upsert prevents crashes if it exists)
    unifiedDao.upsertMetadata(metadata)

    // 2. Insert Interaction
    val interaction = UserInteractionEntity(
        itemId = itemId,
        interactionType = type,
        timestamp = System.currentTimeMillis(), // Or server timestamp if available
        serverId = serverId,
        syncStatus = SyncStatus.SYNCED.name
    )
    unifiedDao.upsertInteraction(interaction)
}

/**
 * Removes an item locally because it was deleted on the server.
 * SAFE: Only deletes if status is SYNCED. Ignores DIRTY/PENDING items (User changes win).
 */
suspend fun removeRemoteItem(itemId: String, type: String) {
    unifiedDao.deleteSyncedItem(itemId, type)
}

// --- Batch Operations for Efficiency ---
```

```
    suspend fun markBatchSynced(ids: List<String>, type: String) {
        unifiedDao.markBatchAsSynced(ids, type)
    }

    suspend fun confirmBatchDeletion(ids: List<String>, type: String) {
        unifiedDao.deleteBatchPending(ids, type)
    }
}

// File: java\com\example\holodex\data\repository\UnifiedVideoRepository.kt
package com.example.holodex.data.repository

import com.example.holodex.data.db.SyncStatus
import com.example.holodex.data.db.UnifiedDao
import com.example.holodex.data.db.UnifiedMetadataEntity
import com.example.holodex.data.db.UserInteractionEntity
import com.example.holodex.data.model.discovery.ChannelDetails
import com.example.holodex.di.IoDispatcher
import com.example.holodex.playback.domain.model.PlaybackItem
import com.example.holodex.viewmodel.UnifiedDisplayItem
import kotlinx.coroutines.CoroutineDispatcher
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.distinctUntilChanged
import kotlinx.coroutines.flow.map
import kotlinx.coroutines.withContext
import javax.inject.Inject
import javax.inject.Singleton

@Singleton
class UnifiedVideoRepository @Inject constructor(
    private val unifiedDao: UnifiedDao,
    @IoDispatcher private val ioDispatcher: CoroutineDispatcher = Dispatchers.IO // Use Qualif
) {

    // ================================================================================
    // 1. UI DATA STREAMS
    // ================================================================================

    fun getFavorites(): Flow<List<UnifiedDisplayItem>> {
        return unifiedDao.getFavorites().map { list ->
            list.map { it.toUnifiedDisplayItem() }
        }
    }

    fun getFavoriteChannels(): Flow<List<UnifiedDisplayItem>> {
        return unifiedDao.getFavoriteChannels().map { list ->
            list.map { it.toUnifiedDisplayItem() }
        }
    }
    fun getFavoriteChannelIds(): Flow<List<String>> {
        return unifiedDao.getFavoriteChannels().map { list ->
            list.map { projection -> projection.metadata.id }
        }
    }
```

```
fun observeLikedItemIds(): Flow<Set<String>> {
    return unifiedDao.getLikedItemIds().map { it.toSet() }
}
fun getDownloads(): Flow<List<UnifiedDisplayItem>> {
    return unifiedDao.getDownloads().map { list ->
        list.map { it.toUnifiedDisplayItem() }
    }
}
fun getHistory(): Flow<List<UnifiedDisplayItem>> {
    return unifiedDao.getHistory().map { list -> list.map { it.toUnifiedDisplayItem() } }
}

// --- NEW HELPER FOR CRASH FIX ---
suspend fun observeDownloadedIds(): Flow<Set<String>> {
    // We use the interaction table directly for speed and stability
    return unifiedDao.getAllDownloadsOneShot().let {
        // This logic needs to be a Flow to use .first() safely in Vms without blocking ma
        // Actually, Room's Flow always emits. Let's use the DAO flow for safety.
        unifiedDao.getDownloads().map { list ->
            list.filter {
                it.interactions.any { i -> i.interactionType == "DOWNLOAD" && i.downloadSt
            }.map { it.metadata.id }.toSet()
        }
    }.distinctUntilChanged()
}

// Alternative: A suspend function if you just need a snapshot
suspend fun getDownloadedIdsSnapshot(): Set<String> {
    return unifiedDao.getAllDownloadsOneShot()
        .filter { it.downloadStatus == "COMPLETED" }
        .map { it.itemId }
        .toSet()
}

suspend fun getDownloadedItemsIdToPathMap(): Map<String, String> {
    return unifiedDao.getAllDownloadsOneShot()
        .filter { it.downloadStatus == "COMPLETED" && !it.localFilePath.isNullOrBlank() }
        .associate { it.itemId to it.localFilePath!! }
}
// ================================================================================
// 2. USER ACTIONS (TOGGLES)
// ================================================================================

suspend fun toggleLike(item: PlaybackItem) = withContext(Dispatchers.IO) {
    val isCurrentlyLiked = unifiedDao.isLiked(item.id) > 0

    if (isCurrentlyLiked) {
        // Check if synced before hard deleting
        val serverId = unifiedDao.getLikeServerId(item.id)
        if (serverId != null) {
            unifiedDao.softDeleteInteraction(item.id, "LIKE")
        } else {
            unifiedDao.deleteInteraction(item.id, "LIKE")
        }
    } else {
```

```
        // 1. Metadata
        val isSegment = item.songId != null
        val type = if (isSegment) "SEGMENT" else "VIDEO"
        val parentId = if (isSegment) item.videoId else null

        val metadata = UnifiedMetadataEntity(
            id = item.id,
            title = item.title,
            artistName = item.artistText,
            type = type,
            specificArtUrl = item.artworkUri,
            uploaderAvatarUrl = null,
            duration = item.durationSec,
            channelId = item.channelId,
            description = item.description,
            startSeconds = item.clipStartSec,
            endSeconds = item.clipEndSec,
            parentVideoId = parentId,
            lastUpdatedAt = System.currentTimeMillis()
        )
        unifiedDao.upsertMetadata(metadata)

        // 2. Interaction
        val interaction = UserInteractionEntity(
            itemId = item.id,
            interactionType = "LIKE",
            timestamp = System.currentTimeMillis(),
            syncStatus = SyncStatus.DIRTY.name,
            serverId = item.serverUuid
        )
        unifiedDao.upsertInteraction(interaction)
    }
}

suspend fun toggleChannelLike(channel: ChannelDetails) = withContext(Dispatchers.IO) {
    // We check if ANY interaction of type FAV_CHANNEL exists for this ID
    val isLiked = unifiedDao.isChannelLiked(channel.id) > 0

    if (isLiked) {
        // Soft delete if synced, hard delete if not (logic inside dao or here)
        // For simplicity, let's assume soft delete support for channels too
        val serverId = unifiedDao.getChannelLikeServerId(channel.id)
        if (serverId != null) {
            unifiedDao.softDeleteInteraction(channel.id, "FAV_CHANNEL")
        } else {
            unifiedDao.deleteInteraction(channel.id, "FAV_CHANNEL")
        }
    } else {
        val meta = UnifiedMetadataEntity(
            id = channel.id,
            title = channel.name,
            artistName = channel.org ?: "",
            type = "CHANNEL",
            specificArtUrl = channel.photoUrl,
            uploaderAvatarUrl = channel.photoUrl,
```

```
                duration = 0,
                channelId = channel.id,
                description = channel.description
            )
            unifiedDao.upsertMetadata(meta)

            val interaction = UserInteractionEntity(
                itemId = channel.id,
                interactionType = "FAV_CHANNEL",
                timestamp = System.currentTimeMillis(),
                syncStatus = SyncStatus.DIRTY.name
            )
            unifiedDao.upsertInteraction(interaction)
        }
    }

    suspend fun getChannel(channelId: String): ChannelDetails? = withContext(Dispatchers.IO) {
        // We look for metadata of type 'CHANNEL' with this ID
        // We can use the existing DAO method 'getItemByIdOneShot' if you added it,
        // or just filter the flow (slower), or add a specific query.
        // Ideally, add 'getMetadataById(id)' to UnifiedDao.

        // Assuming you added getItemByIdOneShot to UnifiedDao as per previous steps:
        val projection = unifiedDao.getItemByIdOneShot(channelId) ?: return@withContext null

        if (projection.metadata.type != "CHANNEL") return@withContext null

        ChannelDetails(
            id = projection.metadata.id,
            name = projection.metadata.title,
            englishName = null, // Metadata only stores one title
            description = projection.metadata.description,
            photoUrl = projection.metadata.specificArtUrl,
            bannerUrl = null,
            org = projection.metadata.artistName,
            suborg = null,
            twitter = null,
            group = null
        )
    }

}

// File: java\com\example\holodex\data\repository\UserPreferencesRepository.kt
package com.example.holodex.data.repository

import android.content.Context
import androidx.datastore.core.DataStore
import androidx.datastore.preferences.core.Preferences
import androidx.datastore.preferences.core.booleanPreferencesKey
import androidx.datastore.preferences.core.edit
import androidx.datastore.preferences.preferencesDataStore
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.map
import timber.log.Timber
```

```
// Extension property to get the DataStore instance
val Context.userPreferencesDataStore: DataStore<Preferences> by preferencesDataStore(name = "u

class UserPreferencesRepository(private val dataStore: DataStore<Preferences>) {

    companion object {
        val AUTOPLAY_NEXT_VIDEO = booleanPreferencesKey("autoplay_next_video")
        val SHUFFLE_ON_PLAY_START = booleanPreferencesKey("shuffle_on_play_start")
        private const val TAG = "UserPrefsRepo"
    }

    val autoplayEnabled: Flow<Boolean> = dataStore.data
        .map { preferences ->
            preferences[AUTOPLAY_NEXT_VIDEO] != false // Default to true (autoplay is on)
        }
        .also { Timber.d("$TAG: Observing autoplayEnabled preference.") }

    val shuffleOnPlayStartEnabled: Flow<Boolean> = dataStore.data
        .map { preferences ->
            preferences[SHUFFLE_ON_PLAY_START] == true // Default to false (OFF)
        }

    suspend fun setAutoplayEnabled(enabled: Boolean) {
        dataStore.edit { preferences ->
            preferences[AUTOPLAY_NEXT_VIDEO] = enabled
        }
        Timber.i("$TAG: Autoplay preference set to: $enabled")
    }
    suspend fun setShuffleOnPlayStartEnabled(enabled: Boolean) {
        dataStore.edit { preferences ->
            preferences[SHUFFLE_ON_PLAY_START] = enabled
        }
        Timber.i("$TAG: Shuffle on play start preference set to: $enabled")
    }
}


// File: java\com\example\holodex\data\repository\YouTubeStreamRepository.kt
// File: java\com\example\holodex\data\repository\YouTubeStreamRepository.kt
package com.example.holodex.data.repository

import android.content.SharedPreferences
import com.example.holodex.data.AppPreferenceConstants
import com.example.holodex.data.model.AudioStreamDetails
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.withContext
import org.schabi.newpipe.extractor.MediaFormat
import org.schabi.newpipe.extractor.NewPipe
import org.schabi.newpipe.extractor.ServiceList
import org.schabi.newpipe.extractor.stream.AudioStream
import org.schabi.newpipe.extractor.stream.AudioTrackType
import org.schabi.newpipe.extractor.stream.StreamInfo
import timber.log.Timber
import java.util.Locale
```

```
import javax.inject.Inject
import javax.inject.Singleton

@Singleton
class YouTubeStreamRepository @Inject constructor(
    private val sharedPreferences: SharedPreferences,
) {

    companion object {
        private const val TAG = "YouTubeStreamRepo"
    }

    private val saverMaxBitrate = 96
    private val standardMaxBitrate = 140

    suspend fun getAudioStreamDetails(videoId: String, preferM4a: Boolean = false): Result<Aud
        return withContext(Dispatchers.IO) {
            try {
                val youtubeUrl = "https://www.youtube.com/watch?v=$videoId"
                val ytService = NewPipe.getService(ServiceList.YouTube.serviceId)
                    ?: return@withContext Result.failure(Exception("YouTube service not found.

                val streamInfo: StreamInfo = StreamInfo.getInfo(ytService, youtubeUrl)
                val allAudioStreams: List<AudioStream> = streamInfo.audioStreams

                if (allAudioStreams.isEmpty()) {
                    return@withContext Result.failure(Exception("No audio streams found for vi
                }

                // 1. Determine which pool of streams to choose from
                val candidateStreams = if (preferM4a) {
                    val m4aStreams = allAudioStreams.filter { it.format == MediaFormat.M4A }
                    // Fallback to all streams if no M4A is found, to prevent crash
                    if (m4aStreams.isNotEmpty()) m4aStreams else allAudioStreams
                } else {
                    allAudioStreams
                }

                val audioQualityPref = sharedPreferences.getString(
                    AppPreferenceConstants.PREF_AUDIO_QUALITY,
                    AppPreferenceConstants.AUDIO_QUALITY_BEST
                ) ?: AppPreferenceConstants.AUDIO_QUALITY_BEST

                // Helper to apply quality/bitrate filters
                val applyQualityFilterAndSort: (List<AudioStream>) -> AudioStream? = { streams
                    val qualityFiltered = when (audioQualityPref) {
                        AppPreferenceConstants.AUDIO_QUALITY_SAVER ->
                            streams.filter { it.averageBitrate in 1..saverMaxBitrate }
                                .ifEmpty { streams.filter { it.averageBitrate in 1..standardMa
                                .ifEmpty { streams }
                        AppPreferenceConstants.AUDIO_QUALITY_STANDARD ->
                            streams.filter { it.averageBitrate in 1..standardMaxBitrate }
                                .ifEmpty { streams }
                        else -> streams
                    }
```

```
                    qualityFiltered.maxByOrNull { it.averageBitrate }
                }

                // 2. Apply your specific logic to the candidate list
                // We run your exact logic on 'candidateStreams'
                val bestAudioStream =
                    // Priority 1: Original Japanese track
                    applyQualityFilterAndSort(candidateStreams.filter {
                        it.audioTrackType == AudioTrackType.ORIGINAL && it.audioLocale?.langua
                    })
                    // Priority 2: Any Original track
                        ?: applyQualityFilterAndSort(candidateStreams.filter {
                            it.audioTrackType == AudioTrackType.ORIGINAL
                        })
                        // Priority 3: Any Japanese track
                        ?: applyQualityFilterAndSort(candidateStreams.filter {
                            it.audioLocale?.language == Locale.JAPANESE.language
                        })
                        // Final Fallback: The best of whatever is left
                        ?: applyQualityFilterAndSort(candidateStreams)


                if (bestAudioStream != null) {
                    val finalUrl = bestAudioStream.content
                        ?: return@withContext Result.failure(Exception("Selected best audio st
                    val finalFormat = bestAudioStream.format?.getName()?.uppercase() ?: "UNKNO
                    val qualityDesc = "${bestAudioStream.averageBitrate}kbps"

                    Timber.i("$TAG: Resolved stream for $videoId. PreferM4a: $preferM4a. Resul

                    return@withContext Result.success(
                        AudioStreamDetails(
                            streamUrl = finalUrl,
                            format = finalFormat,
                            quality = qualityDesc
                        )
                    )
                } else {
                    return@withContext Result.failure(Exception("No suitable audio stream foun
                }

            } catch (e: Exception) {
                Timber.e(e, "$TAG: Unexpected error for $videoId")
                return@withContext Result.failure(e)
            }
        }
    }
}

// File: java\com\example\holodex\data\source\CacheSchemeDataSourceFactory.kt
// File: java/com/example/holodex/data/source/CacheSchemeDataSourceFactory.kt

package com.example.holodex.data.source

import androidx.media3.common.util.UnstableApi
```

```
import androidx.media3.datasource.DataSource
import androidx.media3.datasource.DataSpec // <-- ADD THIS IMPORT
import androidx.media3.datasource.DefaultHttpDataSource
import androidx.media3.datasource.cache.CacheDataSource
import androidx.media3.datasource.cache.SimpleCache
import timber.log.Timber

/**
 * A custom DataSource.Factory that routes requests based on the URI scheme.
 * - Handles "http" and "https" by delegating to a standard HttpDataSource.
 * - Handles a custom "cache" scheme by delegating to a CacheDataSource,
 *   using the URI's authority as the cache key.
 */
@UnstableApi
class CacheSchemeDataSourceFactory(
    private val downloadCache: SimpleCache,
    private val upstreamFactory: DefaultHttpDataSource.Factory
) : DataSource.Factory {

    override fun createDataSource(): DataSource {
        val defaultCacheDataSource = CacheDataSource(
            downloadCache,
            upstreamFactory.createDataSource(),
            CacheDataSource.FLAG_BLOCK_ON_CACHE or CacheDataSource.FLAG_IGNORE_CACHE_ON_ERROR
        )

        val cacheOnlyDataSource = CacheDataSource(downloadCache, null)

        return object : DataSource by defaultCacheDataSource {
            // The method signature is now corrected to match the DataSource interface.
            // It takes a single `DataSpec` parameter.
            override fun open(dataSpec: DataSpec): Long {
                return if (dataSpec.uri.scheme == "cache") {
                    Timber.d("CacheScheme: Routing 'cache://' URI to cache-only source. Key: $
                    // The logic inside remains correct. We create a new DataSpec using the au
                    val newSpec = dataSpec.buildUpon().setKey(dataSpec.uri.authority).build()
                    cacheOnlyDataSource.open(newSpec)
                } else {
                    // For all other schemes (http, https), use the default CacheDataSource.
                    Timber.d("CacheScheme: Routing '${dataSpec.uri.scheme}' URI to default cac
                    defaultCacheDataSource.open(dataSpec)
                }
            }
        }
    }
}

// File: java\com\example\holodex\di\AppModule.kt
// File: java\com\example\holodex\di\AppModule.kt
package com.example.holodex.di

import android.app.Application
import android.content.Context
import android.content.SharedPreferences
import androidx.annotation.OptIn
```

```
import androidx.media3.exoplayer.offline.DownloadNotificationHelper
import androidx.work.WorkManager
import coil.ImageLoader
import coil.disk.DiskCache
import coil.memory.MemoryCache
import com.example.holodex.util.PaletteExtractor
import com.google.gson.Gson
import dagger.Module
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.android.UnstableApi
import dagger.hilt.android.qualifiers.ApplicationContext
import dagger.hilt.components.SingletonComponent
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.SupervisorJob
import javax.inject.Singleton

@Module
@InstallIn(SingletonComponent::class)
object AppModule {

    @Provides
    @Singleton
    fun provideWorkManager(@ApplicationContext context: Context): WorkManager {
        return WorkManager.getInstance(context)
    }

    @Provides
    @Singleton
    fun provideSharedPreferences(app: Application): SharedPreferences {
        return app.getSharedPreferences("UserPrefs", Context.MODE_PRIVATE)
    }

    @Provides
    @Singleton
    fun provideGson(): Gson = Gson()



    @Provides
    @Singleton
    @ApplicationScope // We'll create this annotation for clarity
    fun provideApplicationScope(): CoroutineScope =
        CoroutineScope(SupervisorJob() + Dispatchers.Default)

    @Provides
    @Singleton
    fun provideImageLoader(@ApplicationContext context: Context): ImageLoader {
        return ImageLoader.Builder(context)
            .memoryCache {
                MemoryCache.Builder(context)
                    .maxSizePercent(0.25)
                    .build()
            }
```

```
            .diskCache {
                DiskCache.Builder()
                    .directory(context.cacheDir.resolve("image_cache_v1"))
                    .maxSizeBytes(50L * 1024L * 1024L) // Increased to 100MB for better offlin
                    .build()
            }
            .crossfade(true) // Reduces visual jitter on load
            .allowHardware(true) // CRITICAL: Uses GPU for bitmaps, saving Main Thread CPU
            .respectCacheHeaders(false) // Aggressively cache images regardless of server head
            .build()
    }

    @OptIn(androidx.media3.common.util.UnstableApi::class)
    @Provides
    @Singleton
    @UnstableApi
    fun provideDownloadNotificationHelper(@ApplicationContext context: Context): DownloadNotif
        return DownloadNotificationHelper(context, "download_channel")
    }
    @Provides
    @Singleton
    fun providePaletteExtractor(@ApplicationContext context: Context): PaletteExtractor {
        return PaletteExtractor(context)
    }
}

// File: java\com\example\holodex\di\AuthModule.kt
package com.example.holodex.di

import android.content.Context
import com.example.holodex.auth.TokenManager
import dagger.Module
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.android.qualifiers.ApplicationContext
import dagger.hilt.components.SingletonComponent
import net.openid.appauth.AuthorizationService
import javax.inject.Singleton

@Module
@InstallIn(SingletonComponent::class)
object AuthModule {

    @Provides
    @Singleton
    fun provideTokenManager(@ApplicationContext context: Context): TokenManager {
        return TokenManager(context)
    }

    @Provides
    // Not a singleton, as it should be created and disposed with the component that uses it (
    fun provideAuthorizationService(@ApplicationContext context: Context): AuthorizationServic
        return AuthorizationService(context)
    }
}
```

```
// File: java\com\example\holodex\di\CacheModule.kt
package com.example.holodex.di

import android.content.Context
import androidx.media3.common.util.UnstableApi
import androidx.media3.database.StandaloneDatabaseProvider
import androidx.media3.datasource.cache.LeastRecentlyUsedCacheEvictor
import androidx.media3.datasource.cache.NoOpCacheEvictor
import androidx.media3.datasource.cache.SimpleCache
import com.example.holodex.data.cache.BrowseListCache
import com.example.holodex.data.cache.SearchListCache
import com.example.holodex.data.db.BrowsePageDao
import com.example.holodex.data.db.SearchPageDao
import dagger.Module
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.android.qualifiers.ApplicationContext
import dagger.hilt.components.SingletonComponent
import java.io.File
import javax.inject.Singleton

@Module
@InstallIn(SingletonComponent::class)
object CacheModule {

    @Provides
    @Singleton
    fun provideBrowseListCache(browsePageDao: BrowsePageDao): BrowseListCache {
        return BrowseListCache(browsePageDao)
    }

    @Provides
    @Singleton
    fun provideSearchListCache(searchPageDao: SearchPageDao): SearchListCache {
        return SearchListCache(searchPageDao)
    }

    @Provides
    @Singleton
    @DownloadCache
    @UnstableApi
    fun provideDownloadCache(@ApplicationContext context: Context): SimpleCache {
        val downloadDirectory = File(context.getExternalFilesDir(null), "downloads")
        val databaseProvider = StandaloneDatabaseProvider(context)
        return SimpleCache(downloadDirectory, NoOpCacheEvictor(), databaseProvider)
    }

    @Provides
    @Singleton
    @MediaCache
    @UnstableApi
    fun provideMediaCache(@ApplicationContext context: Context): SimpleCache {
        val mediaCacheDirectory = File(context.cacheDir, "exoplayer_media_cache")
        val cacheEvictor = LeastRecentlyUsedCacheEvictor(150L * 1024L * 1024L) // 150MB
```

```
        val databaseProvider = StandaloneDatabaseProvider(context)
        return SimpleCache(mediaCacheDirectory, cacheEvictor, databaseProvider)
    }
}

// File: java\com\example\holodex\di\DatabaseModule.kt
package com.example.holodex.di

import android.app.Application
import com.example.holodex.data.db.AppDatabase
import com.example.holodex.data.db.BrowsePageDao
import com.example.holodex.data.db.DiscoveryDao
import com.example.holodex.data.db.ParentVideoMetadataDao
import com.example.holodex.data.db.PlaylistDao
import com.example.holodex.data.db.SearchPageDao
import com.example.holodex.data.db.StarredPlaylistDao
import com.example.holodex.data.db.SyncMetadataDao
import com.example.holodex.data.db.UnifiedDao
import com.example.holodex.data.db.VideoDao
import dagger.Module
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.components.SingletonComponent
import javax.inject.Singleton

@Module
@InstallIn(SingletonComponent::class)
object DatabaseModule {

    @Provides
    @Singleton
    fun provideAppDatabase(app: Application): AppDatabase = AppDatabase.getDatabase(app)

    // *** THE FIX: Add the provider for the new DAO ***
    @Provides
    fun provideUnifiedDao(db: AppDatabase): UnifiedDao = db.unifiedDao()

    // Keep the rest of the DAOs that are still in use
    @Provides fun provideVideoDao(db: AppDatabase): VideoDao = db.videoDao()
    @Provides fun providePlaylistDao(db: AppDatabase): PlaylistDao = db.playlistDao()
    @Provides fun provideBrowsePageDao(db: AppDatabase): BrowsePageDao = db.browsePageDao()
    @Provides fun provideSearchPageDao(db: AppDatabase): SearchPageDao = db.searchPageDao()
    @Provides fun provideDiscoveryDao(db: AppDatabase): DiscoveryDao = db.discoveryDao()
    @Provides fun provideParentVideoMetadataDao(db: AppDatabase): ParentVideoMetadataDao = db.
    @Provides fun provideSyncMetadataDao(db: AppDatabase): SyncMetadataDao = db.syncMetadataDa
    @Provides fun provideStarredPlaylistDao(db: AppDatabase): StarredPlaylistDao = db.starredP
}

// File: java\com\example\holodex\di\DispatchersModule.kt
package com.example.holodex.di

import dagger.Module
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.components.SingletonComponent
```

```
import kotlinx.coroutines.CoroutineDispatcher
import kotlinx.coroutines.Dispatchers

@Module
@InstallIn(SingletonComponent::class)
object DispatchersModule {

    @Provides
    @IoDispatcher
    fun provideIoDispatcher(): CoroutineDispatcher = Dispatchers.IO

    @Provides
    @DefaultDispatcher
    fun provideDefaultDispatcher(): CoroutineDispatcher = Dispatchers.Default
}

// File: java\com\example\holodex\di\NetworkModule.kt
// File: java/com/example/holodex/di/NetworkModule.kt

package com.example.holodex.di

import android.content.SharedPreferences
import com.example.holodex.auth.TokenManager
import com.example.holodex.data.api.AuthenticatedMusicdexApiService
import com.example.holodex.data.api.HolodexApiService
import com.example.holodex.data.api.MusicdexApiService
import com.google.gson.Gson
import dagger.Module
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.components.SingletonComponent
import okhttp3.OkHttpClient
import okhttp3.logging.HttpLoggingInterceptor
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory
import timber.log.Timber
import java.util.concurrent.TimeUnit
import javax.inject.Singleton

@Module
@InstallIn(SingletonComponent::class)
object NetworkModule {

    private const val BROWSER_USER_AGENT = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWeb

    @Provides
    @Singleton
    fun provideLoggingInterceptor(): HttpLoggingInterceptor {
        return HttpLoggingInterceptor { message ->
            Timber.tag("OkHttp-Holodex").i(message)
        }.apply { level = HttpLoggingInterceptor.Level.BODY }
    }

    // CLIENT FOR HOLODEX.NET API (GET requests, etc.)
    @Provides
```

```
@Singleton
@HolodexHttpClient
fun provideHolodexOkHttpClient(
    sharedPreferences: SharedPreferences,
    tokenManager: TokenManager,
    loggingInterceptor: HttpLoggingInterceptor
): OkHttpClient {
    return OkHttpClient.Builder()
        .addInterceptor { chain ->
            val requestBuilder = chain.request().newBuilder()
            requestBuilder.header("User-Agent", BROWSER_USER_AGENT)
            val apiKey = sharedPreferences.getString("API_KEY", "") ?: ""
            if (apiKey.isNotEmpty()) {
                requestBuilder.header("X-APIKEY", apiKey)
            }
            // Also add JWT if available, for endpoints like GET /users/favorites
            tokenManager.getJwt()?.let { jwt ->
                requestBuilder.header("Authorization", "Bearer $jwt")
            }
            chain.proceed(requestBuilder.build())
        }
        .addInterceptor(loggingInterceptor)
        .connectTimeout(90, TimeUnit.SECONDS)
        .readTimeout(90, TimeUnit.SECONDS)
        .build()
}

// CLIENT FOR PUBLIC MUSICDEX.NET API (API Key only)
@Provides
@Singleton
@MusicdexHttpClient
fun provideMusicdexOkHttpClient(
    sharedPreferences: SharedPreferences,
    loggingInterceptor: HttpLoggingInterceptor
): OkHttpClient {
    return OkHttpClient.Builder()
        .addInterceptor { chain ->
            val requestBuilder = chain.request().newBuilder()
            requestBuilder.header("User-Agent", BROWSER_USER_AGENT)
            val apiKey = sharedPreferences.getString("API_KEY", "") ?: ""
            if (apiKey.isNotEmpty()) {
                requestBuilder.header("X-APIKEY", apiKey)
            }
            chain.proceed(requestBuilder.build())
        }
        .addInterceptor(loggingInterceptor)
        .connectTimeout(90, TimeUnit.SECONDS)
        .readTimeout(90, TimeUnit.SECONDS)
        .build()
}

// CLIENT FOR AUTHENTICATED MUSICDEX.NET API (Likes, History, Playlists, AND NOW FAVORITES
// --- FIX: This client now sends both the API Key and the JWT ---
// This makes it compatible with the favorites PATCH endpoint without breaking the likes e
@Provides
```

```
@Singleton
@AuthenticatedMusicdexHttpClient
fun provideAuthenticatedMusicdexOkHttpClient(
    sharedPreferences: SharedPreferences, // <-- ADDED
    tokenManager: TokenManager,
    loggingInterceptor: HttpLoggingInterceptor
): OkHttpClient {
    return OkHttpClient.Builder()
        .addInterceptor { chain ->
            val requestBuilder = chain.request().newBuilder()
            requestBuilder.header("User-Agent", BROWSER_USER_AGENT)

            // Add API Key
            val apiKey = sharedPreferences.getString("API_KEY", "") ?: ""
            if (apiKey.isNotEmpty()) {
                requestBuilder.header("X-APIKEY", apiKey)
            }

            // Add JWT
            tokenManager.getJwt()?.let { jwt ->
                requestBuilder.header("Authorization", "Bearer $jwt")
            }
            requestBuilder.header("Referer", "https://music.holodex.net/")

            chain.proceed(requestBuilder.build())
        }
        .addInterceptor(loggingInterceptor)
        .connectTimeout(90, TimeUnit.SECONDS)
        .readTimeout(90, TimeUnit.SECONDS)
        .build()
}

// --- RETROFIT SERVICE PROVIDERS (unchanged) ---

@Provides
@Singleton
fun provideHolodexApiService(@HolodexHttpClient okHttpClient: OkHttpClient): HolodexApiSer
    return Retrofit.Builder()
        .baseUrl("https://holodex.net/")
        .client(okHttpClient)
        .addConverterFactory(GsonConverterFactory.create())
        .build()
        .create(HolodexApiService::class.java)
}

@Provides
@Singleton
fun provideMusicdexApiService(@MusicdexHttpClient okHttpClient: OkHttpClient, gson: Gson):
    return Retrofit.Builder()
        .baseUrl("https://music.holodex.net/")
        .client(okHttpClient)
        .addConverterFactory(GsonConverterFactory.create(gson))
        .build()
        .create(MusicdexApiService::class.java)
}
```

```
    @Provides
    @Singleton
    fun provideAuthenticatedMusicdexApiService(@AuthenticatedMusicdexHttpClient okHttpClient:
        return Retrofit.Builder()
            .baseUrl("https://music.holodex.net/")
            .client(okHttpClient)
            .addConverterFactory(GsonConverterFactory.create(gson))
            .build()
            .create(AuthenticatedMusicdexApiService::class.java)
    }
}

// File: java\com\example\holodex\di\PlaybackModule.kt
// File: java/com/example/holodex/di/PlaybackModule.kt
package com.example.holodex.di

import android.content.Context
import android.content.SharedPreferences
import androidx.media3.common.C
import androidx.media3.common.Player
import androidx.media3.common.util.UnstableApi
import androidx.media3.database.StandaloneDatabaseProvider
import androidx.media3.datasource.DataSource
import androidx.media3.datasource.DataSpec
import androidx.media3.datasource.DefaultDataSource
import androidx.media3.datasource.DefaultHttpDataSource
import androidx.media3.datasource.ResolvingDataSource
import androidx.media3.datasource.cache.CacheDataSource
import androidx.media3.datasource.cache.SimpleCache
import androidx.media3.exoplayer.DefaultLoadControl
import androidx.media3.exoplayer.ExoPlayer
import androidx.media3.exoplayer.offline.DownloadManager
import androidx.media3.exoplayer.source.DefaultMediaSourceFactory
import androidx.media3.exoplayer.source.preload.DefaultPreloadManager
import androidx.media3.exoplayer.upstream.DefaultAllocator
import com.example.holodex.data.AppPreferenceConstants
import com.example.holodex.data.db.AppDatabase
import com.example.holodex.data.db.UnifiedDao
import com.example.holodex.data.repository.HolodexRepository
import com.example.holodex.data.repository.UserPreferencesRepository
import com.example.holodex.playback.data.mapper.MediaItemMapper
import com.example.holodex.playback.data.model.PlaybackDao
import com.example.holodex.playback.data.preload.PreloadConfiguration
import com.example.holodex.playback.data.preload.PreloadStatusController
import com.example.holodex.playback.data.queue.ShuffleOrderProvider
import com.example.holodex.playback.data.repository.HolodexStreamResolverRepositoryImpl
import com.example.holodex.playback.data.source.HolodexResolvingDataSource
import com.example.holodex.playback.data.source.StreamResolutionCoordinator
import com.example.holodex.playback.domain.repository.StreamResolverRepository
import com.example.holodex.playback.player.Media3PlayerController
import com.example.holodex.playback.player.PlaybackController
import com.example.holodex.viewmodel.autoplay.AutoplayItemProvider
import com.example.holodex.viewmodel.autoplay.ContinuationManager
import dagger.Module
```

```
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.android.qualifiers.ApplicationContext
import dagger.hilt.components.SingletonComponent
import kotlinx.coroutines.CoroutineScope
import timber.log.Timber
import java.util.concurrent.Executors
import javax.inject.Singleton

private data class BufferSettings(
    val minBufferMs: Int,
    val maxBufferMs: Int,
    val bufferForPlaybackMs: Int,
    val bufferForPlaybackAfterRebufferMs: Int
)

@Module
@InstallIn(SingletonComponent::class)
@UnstableApi
object PlaybackModule {

    @Provides
    @Singleton
    fun provideLoadControl(sharedPreferences: SharedPreferences): DefaultLoadControl {
        val bufferingStrategy = sharedPreferences.getString(
            AppPreferenceConstants.PREF_BUFFERING_STRATEGY,
            AppPreferenceConstants.BUFFERING_STRATEGY_AGGRESSIVE
        ) ?: AppPreferenceConstants.BUFFERING_STRATEGY_AGGRESSIVE

        val settings = when (bufferingStrategy) {
            AppPreferenceConstants.BUFFERING_STRATEGY_BALANCED -> {
                Timber.d("ExoPlayer LoadControl: BALANCED")
                BufferSettings(20000, 60000, 3000, 5000)
            }

            AppPreferenceConstants.BUFFERING_STRATEGY_STABLE -> {
                Timber.d("ExoPlayer LoadControl: STABLE")
                BufferSettings(30000, 120000, 7500, 10000)
            }

            else -> {
                Timber.d("ExoPlayer LoadControl: AGGRESSIVE (default)")
                BufferSettings(10000, 60000, 1000, 2500)
            }
        }

        return DefaultLoadControl.Builder()
            .setAllocator(DefaultAllocator(true, C.DEFAULT_BUFFER_SEGMENT_SIZE))
            .setBufferDurationsMs(
                settings.minBufferMs,
                settings.maxBufferMs,
                settings.bufferForPlaybackMs,
                settings.bufferForPlaybackAfterRebufferMs
            )
            .build()
```

```
}

@Provides
@Singleton
fun provideExoPlayer(
    @ApplicationContext context: Context,
    loadControl: DefaultLoadControl,
    mediaSourceFactory: DefaultMediaSourceFactory
): ExoPlayer {
    return ExoPlayer.Builder(context)
        .setMediaSourceFactory(mediaSourceFactory)
        .setLoadControl(loadControl)
        .build()
}

@Provides
@Singleton
fun providePlayer(exoPlayer: ExoPlayer): Player = exoPlayer

@Provides
@Singleton
fun provideDownloadManager(
    @ApplicationContext context: Context,
    @DownloadCache downloadCache: SimpleCache
): DownloadManager {
    val databaseProvider = StandaloneDatabaseProvider(context)
    val downloadManagerDataSourceFactory =
        DefaultHttpDataSource.Factory().setUserAgent("HolodexAppDownloader/1.0")
    return DownloadManager(
        context,
        databaseProvider,
        downloadCache,
        downloadManagerDataSourceFactory,
        Executors.newFixedThreadPool(3)
    ).apply { resumeDownloads() }
}

@Provides
@Singleton
fun provideDataSourceFactory(
    @ApplicationContext context: Context,
    @DownloadCache downloadCache: SimpleCache,
    @MediaCache mediaCache: SimpleCache,
    holodexResolver: HolodexResolvingDataSource // <--- INJECT THIS
): DataSource.Factory {

    // 1. The base factory for network requests.
    val upstreamFactory = DefaultHttpDataSource.Factory()
        .setUserAgent("HolodexApp/1.0")
        .setConnectTimeoutMs(30000)
        .setReadTimeoutMs(30000)
        .setAllowCrossProtocolRedirects(true)

    // 2. The default factory that handles most schemes (http, https, content, file, etc.)
    val defaultDataSourceFactory = DefaultDataSource.Factory(context, upstreamFactory)
```

```
    // 3. The factory that handles streaming from the media cache.
    val mediaCacheDataSourceFactory = CacheDataSource.Factory()
        .setCache(mediaCache)
        .setUpstreamDataSourceFactory(defaultDataSourceFactory)

    // 4. Our final, all-encompassing factory.
    return object : DataSource.Factory {
        override fun createDataSource(): DataSource {
            // The source for handling downloads via the "cache://" scheme.
            val downloadCacheDataSource = CacheDataSource(downloadCache, null) // Cache-on

            // The source for everything else.
            val defaultSource = mediaCacheDataSourceFactory.createDataSource()

            // *** FIX IS HERE ***
            // Wrap the default source in the ResolvingDataSource.
            // This intercepts 'holodex://' -> resolves to 'https://' -> passes 'https://'
            val resolvingDataSource = ResolvingDataSource(defaultSource, holodexResolver)

            return object : DataSource by resolvingDataSource {
                override fun open(dataSpec: DataSpec): Long {
                    return when (dataSpec.uri.scheme) {
                        "cache" -> {
                            Timber.d("DataSource: Routing 'cache://' to download cache. Ke
                            val newSpec =
                                dataSpec.buildUpon().setKey(dataSpec.uri.authority).build(
                            downloadCacheDataSource.open(newSpec)
                        }

                        "placeholder" -> {
                            Timber.d("DataSource: Intercepting 'placeholder://' URI. Retur
                            0
                        }

                        else -> {
                            // Pass 'holodex://', 'http', 'https', 'file' etc. to the reso
                            // The resolving source will check if it needs to modify the U
                            resolvingDataSource.open(dataSpec)
                        }
                    }
                }
            }
        }
    }
}


@Provides
@Singleton
fun provideDefaultMediaSourceFactory(
    @ApplicationContext context: Context,
    dataSourceFactory: DataSource.Factory
): DefaultMediaSourceFactory {
    return DefaultMediaSourceFactory(context).setDataSourceFactory(dataSourceFactory)
```

```
}

@Provides
@Singleton
fun providePreloadManager(
    @ApplicationContext context: Context,
    statusController: PreloadStatusController,
    mediaSourceFactory: DefaultMediaSourceFactory,
    loadControl: DefaultLoadControl
): DefaultPreloadManager {
    return DefaultPreloadManager.Builder(context, statusController)
        .setMediaSourceFactory(mediaSourceFactory)
        .setLoadControl(loadControl)
        .build()
}

@Provides
@Singleton
fun provideMedia3PlayerController(
    @ApplicationContext context: Context,
    exoPlayer: ExoPlayer,
    preloadManager: DefaultPreloadManager
): Media3PlayerController {
    return Media3PlayerController(exoPlayer, preloadManager)
}


@Provides
@Singleton
fun provideStreamResolverRepository(repo: com.example.holodex.data.repository.YouTubeStrea
    return HolodexStreamResolverRepositoryImpl(repo)
}

@Provides
@Singleton
fun provideStreamResolutionCoordinator(
    repo: StreamResolverRepository,
    unifiedDao: UnifiedDao
): StreamResolutionCoordinator = StreamResolutionCoordinator(repo, unifiedDao)

@Provides
@Singleton
fun provideShuffleOrderProvider(): ShuffleOrderProvider = ShuffleOrderProvider()


@Provides
@Singleton
fun provideAutoplayItemProvider(holodexRepository: HolodexRepository): AutoplayItemProvide
    return AutoplayItemProvider(holodexRepository)
}

@Provides
@Singleton
fun provideContinuationManager(
    holodexRepository: HolodexRepository,
```

```
        userPreferencesRepository: UserPreferencesRepository,
        autoplayItemProvider: AutoplayItemProvider
    ): ContinuationManager {
        return ContinuationManager(
            holodexRepository,
            userPreferencesRepository,
            autoplayItemProvider
        )
    }

    @Provides
    @Singleton
    fun providePreloadConfig(): PreloadConfiguration = PreloadConfiguration()

    @Provides
    fun providePlaybackDao(db: AppDatabase): PlaybackDao = db.playbackDao()

    @Provides
    @Singleton
    fun providePlaybackController(
        exoPlayer: ExoPlayer,
        playbackDao: PlaybackDao,
        unifiedDao: UnifiedDao,
        mapper: MediaItemMapper,
        @ApplicationScope scope: CoroutineScope,
        continuationManager: ContinuationManager
    ): PlaybackController {
        return PlaybackController(
            exoPlayer, playbackDao, unifiedDao, mapper, continuationManager, scope
        )
    }

    @Provides
    @Singleton
    fun provideMediaItemMapper(
        @ApplicationContext context: Context,
        sharedPreferences: SharedPreferences
    ): MediaItemMapper {
        return MediaItemMapper(context, sharedPreferences)
    }

}

// File: java\com\example\holodex\di\Qualifiers.kt
// File: java\com\example\holodex\di\Qualifiers.kt
package com.example.holodex.di

import javax.inject.Qualifier

@Qualifier
@Retention(AnnotationRetention.BINARY)
annotation class ApplicationScope

@Qualifier
@Retention(AnnotationRetention.BINARY)
```

```
annotation class PublicClient

@Qualifier
@Retention(AnnotationRetention.BINARY)
annotation class AuthenticatedClient

@Qualifier
@Retention(AnnotationRetention.BINARY)
annotation class DownloadCache

@Qualifier
@Retention(AnnotationRetention.BINARY)
annotation class MediaCache

@Qualifier
@Retention(AnnotationRetention.BINARY)
annotation class UpstreamDataSource

@Qualifier
@Retention(AnnotationRetention.BINARY)
annotation class HolodexHttpClient

@Qualifier
@Retention(AnnotationRetention.BINARY)
annotation class MusicdexHttpClient

@Qualifier
@Retention(AnnotationRetention.BINARY)
annotation class AuthenticatedMusicdexHttpClient

@Qualifier
@Retention(AnnotationRetention.BINARY)
annotation class IoDispatcher

@Qualifier
@Retention(AnnotationRetention.BINARY)
annotation class DefaultDispatcher

// File: java\com\example\holodex\di\RepositoryModule.kt
package com.example.holodex.di

import android.content.Context
import android.content.SharedPreferences
import androidx.annotation.OptIn
import androidx.media3.common.util.UnstableApi
import androidx.media3.datasource.DataSource
import androidx.media3.datasource.DefaultHttpDataSource
import com.example.holodex.auth.AuthRepository
import com.example.holodex.auth.TokenManager
import com.example.holodex.data.api.AuthenticatedMusicdexApiService
import com.example.holodex.data.api.HolodexApiService
import com.example.holodex.data.api.MusicdexApiService
import com.example.holodex.data.cache.BrowseListCache
import com.example.holodex.data.cache.SearchListCache
import com.example.holodex.data.db.AppDatabase
```

```
import com.example.holodex.data.db.DiscoveryDao
import com.example.holodex.data.db.PlaylistDao
import com.example.holodex.data.db.StarredPlaylistDao
import com.example.holodex.data.db.SyncMetadataDao
import com.example.holodex.data.db.UnifiedDao // Added
import com.example.holodex.data.db.VideoDao
import com.example.holodex.data.repository.DownloadRepository
import com.example.holodex.data.repository.DownloadRepositoryImpl
import com.example.holodex.data.repository.HolodexRepository
// LocalRepository import removed
import com.example.holodex.data.repository.SearchHistoryRepository
import com.example.holodex.data.repository.SharedPreferencesSearchHistoryRepository
import com.example.holodex.data.repository.UnifiedVideoRepository
import com.example.holodex.data.repository.UserPreferencesRepository
import com.example.holodex.data.repository.YouTubeStreamRepository
import com.example.holodex.data.repository.userPreferencesDataStore
import com.google.gson.Gson
import dagger.Binds
import dagger.Module
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.android.qualifiers.ApplicationContext
import dagger.hilt.components.SingletonComponent
import kotlinx.coroutines.CoroutineDispatcher
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import net.openid.appauth.AuthorizationService
import javax.inject.Singleton

@Module
@InstallIn(SingletonComponent::class)
abstract class RepositoryModule {

    @Binds
    @Singleton
    abstract fun bindSearchHistoryRepository(
        impl: SharedPreferencesSearchHistoryRepository
    ): SearchHistoryRepository

    @Binds
    @Singleton
    @UnstableApi
    abstract fun bindDownloadRepository(
        impl: DownloadRepositoryImpl
    ): DownloadRepository

    companion object {

        @Provides
        @Singleton
        fun provideHolodexRepository(
            holodexApiService: HolodexApiService,
            musicdexApiService: MusicdexApiService,
            authenticatedMusicdexApiService: AuthenticatedMusicdexApiService,
            discoveryDao: DiscoveryDao,
```

```
    browseListCache: BrowseListCache,
    searchListCache: SearchListCache,
    videoDao: VideoDao,
    playlistDao: PlaylistDao,
    appDatabase: AppDatabase,
    @DefaultDispatcher defaultDispatcher: CoroutineDispatcher,
    syncMetadataDao: SyncMetadataDao,
    starredPlaylistDao: StarredPlaylistDao,
    tokenManager: TokenManager,
    unifiedDao: UnifiedDao,
    unifiedRepository: UnifiedVideoRepository,
    @ApplicationScope applicationScope: CoroutineScope
): HolodexRepository {
    return HolodexRepository(
        holodexApiService,
        musicdexApiService,
        authenticatedMusicdexApiService,
        discoveryDao,
        browseListCache,
        searchListCache,
        videoDao,
        playlistDao,
        appDatabase,
        defaultDispatcher,
        syncMetadataDao,
        starredPlaylistDao,
        tokenManager,
        unifiedDao,
        unifiedRepository,
        applicationScope
    )
}


@Provides
@Singleton
fun provideYouTubeStreamRepository(
    sharedPreferences: SharedPreferences,
): YouTubeStreamRepository {
    return YouTubeStreamRepository(sharedPreferences)
}

@Provides
@Singleton
fun provideAuthRepository(
    holodexApiService: HolodexApiService,
    authService: AuthorizationService
): AuthRepository {
    return AuthRepository(holodexApiService, authService)
}

@Provides
@Singleton
fun provideUserPreferencesRepository(@ApplicationContext context: Context): UserPrefer
    return UserPreferencesRepository(context.userPreferencesDataStore)
```

```
        }

        @Provides
        @Singleton
        fun provideSharedPreferencesSearchHistoryRepository(
            sharedPreferences: SharedPreferences,
            gson: Gson
        ): SharedPreferencesSearchHistoryRepository {
            return SharedPreferencesSearchHistoryRepository(sharedPreferences, gson, Dispatche
        }

        @OptIn(UnstableApi::class)
        @Provides
        @Singleton
        @UpstreamDataSource
        fun provideUpstreamDataSourceFactory(): DataSource.Factory {
            return DefaultHttpDataSource.Factory()
                .setUserAgent("HolodexAppDownloader/1.0")
                .setConnectTimeoutMs(30000)
                .setReadTimeoutMs(30000)
                .setAllowCrossProtocolRedirects(true)
        }
    }
}

// File: java\com\example\holodex\di\SyncModule.kt
// File: java/com/example/holodex/di/SyncModule.kt (MODIFIED)
package com.example.holodex.di

import com.example.holodex.background.FavoriteChannelSynchronizer
import com.example.holodex.background.HistorySynchronizer
import com.example.holodex.background.ISynchronizer
import com.example.holodex.background.LikesSynchronizer
import com.example.holodex.background.PlaylistSynchronizer
import com.example.holodex.background.StarredPlaylistSynchronizer
import dagger.Binds
import dagger.Module
import dagger.hilt.InstallIn
import dagger.hilt.components.SingletonComponent
import dagger.multibindings.IntoSet

@Module
@InstallIn(SingletonComponent::class)
abstract class SyncModule {

    @Binds
    @IntoSet
    abstract fun bindLikesSynchronizer(impl: LikesSynchronizer): ISynchronizer

    @Binds
    @IntoSet
    abstract fun bindPlaylistSynchronizer(impl: PlaylistSynchronizer): ISynchronizer

    @Binds
    @IntoSet
```

```
    abstract fun bindFavoriteChannelSynchronizer(impl: FavoriteChannelSynchronizer): ISynchron

    @Binds
    @IntoSet
    abstract fun bindStarredPlaylistSynchronizer(impl: StarredPlaylistSynchronizer): ISynchron

    @Binds
    @IntoSet
    abstract fun bindHistorySynchronizer(impl: HistorySynchronizer): ISynchronizer
}

// File: java\com\example\holodex\di\UseCaseModule.kt
package com.example.holodex.di

import com.example.holodex.data.repository.HolodexRepository
import com.example.holodex.playback.domain.usecase.AddItemsToQueueUseCase
import com.example.holodex.playback.domain.usecase.AddOrFetchAndAddUseCase
import com.example.holodex.playback.player.PlaybackController
import dagger.Module
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.components.SingletonComponent

@Module
@InstallIn(SingletonComponent::class)
object UseCaseModule {

    // We only keep complex UseCases.
    // Simple ones (Play, Pause) are handled directly by Controller in ViewModels now.

    @Provides
    fun provideAddItemsToQueueUseCase(controller: PlaybackController): AddItemsToQueueUseCase
        return AddItemsToQueueUseCase(controller)
    }

    @Provides
    fun provideAddOrFetchAndAddUseCase(
        repo: HolodexRepository,
        addItemsUseCase: AddItemsToQueueUseCase
    ) = AddOrFetchAndAddUseCase(repo, addItemsUseCase)
}

// File: java\com\example\holodex\extractor\DownloaderImpl.kt
// Location: com.example.holodex.extractor/DownloaderImpl.kt
package com.example.holodex.extractor

import okhttp3.Headers.Companion.toHeaders
import okhttp3.OkHttpClient
import okhttp3.RequestBody.Companion.toRequestBody
import org.schabi.newpipe.extractor.downloader.Downloader
import org.schabi.newpipe.extractor.downloader.Request
import org.schabi.newpipe.extractor.downloader.Response // Ensure this is org.schabi.newpipe.e
import org.schabi.newpipe.extractor.exceptions.ReCaptchaException
import java.io.IOException
```

```
class DownloaderImpl(private val okHttpClient: OkHttpClient) : Downloader() {

    @Throws(IOException::class, ReCaptchaException::class)
    override fun execute(request: Request): Response {
        val okHttpRequestBuilder = okhttp3.Request.Builder()
            .url(request.url())
            .method(request.httpMethod(), request.dataToSend()?.toRequestBody(null))

        for ((headerName, headerValueList) in request.headers()) {
            if (headerValueList.size > 1) {
                headerValueList.forEach { headerValue ->
                    okHttpRequestBuilder.addHeader(headerName, headerValue)
                }
            } else if (headerValueList.size == 1) {
                okHttpRequestBuilder.header(headerName, headerValueList[0])
            }
        }

        val call = okHttpClient.newCall(okHttpRequestBuilder.build())
        val okHttpResponse = call.execute() // This is a synchronous call

        // It's crucial to read the body string only once if you need to inspect it AND pass i
        // If the body is very large, this could be memory inefficient.
        // For ReCaptcha detection, we often only need to check a small part or rely on status
        // However, many ReCaptcha pages are full HTML, so checking content is common.
        val responseBodyString = okHttpResponse.body?.string() // Read body once

        // Simplified ReCaptcha check
        if (okHttpResponse.code == 429 || (responseBodyString != null &&
                    (responseBodyString.contains("consent.youtube.com", ignoreCase = true) ||
                            responseBodyString.contains("??????????", ignoreCase = true) ||
                            responseBodyString.contains("before you continue to youtube", igno
                            responseBodyString.contains("/sorry/index?continue=", ignoreCase =
                            responseBodyString.contains("www.google.com/recaptcha", ignoreCase
                            responseBodyString.contains("??? ??????????, ???????????, ?? ?? ??
                            ))) {
            okHttpResponse.close() // Ensure response is closed if not using its body stream
            throw ReCaptchaException("ReCaptcha Challenge Found", okHttpResponse.request.url.t
        }

        // Pass the already read responseBodyString to the Response constructor
        return Response(
            okHttpResponse.code,
            okHttpResponse.message,
            okHttpResponse.headers.toMultimap(), // Convert OkHttp Headers to Map<String, List
            responseBodyString,                  // Pass the String body directly
            okHttpResponse.request.url.toString()
        )
    }

    // As per your diff, the getCookies/setCookies overrides are removed
    // as they are not part of the current Downloader base class.
    // Cookie management, if needed, should be handled via OkHttp's CookieJar
    // configured on the OkHttpClient instance passed to this DownloaderImpl.
}
```

```
// File: java\com\example\holodex\playback\data\mapper\MediaItemMapper.kt
package com.example.holodex.playback.data.mapper

import android.content.Context
import android.content.SharedPreferences
import android.os.Bundle
import androidx.core.net.toUri
import androidx.media3.common.MediaItem
import androidx.media3.common.MediaMetadata
import com.example.holodex.R
import com.example.holodex.data.AppPreferenceConstants
import com.example.holodex.playback.domain.model.PlaybackItem
import com.example.holodex.util.getHighResArtworkUrl
import javax.inject.Inject

// ... (Keep the const keys at the top of the file) ...
internal const val EXTRA_KEY_HOLODEX_VIDEO_ID = "com.example.holodex.EXTRA_VIDEO_ID"
internal const val EXTRA_KEY_HOLODEX_SONG_ID = "com.example.holodex.EXTRA_SONG_ID"
internal const val EXTRA_KEY_HOLODEX_SERVER_UUID = "com.example.holodex.EXTRA_SERVER_UUID"
internal const val EXTRA_KEY_ORIGINAL_DURATION_SEC = "com.example.holodex.EXTRA_DURATION_SEC"
internal const val EXTRA_KEY_ARTIST_TEXT = "com.example.holodex.EXTRA_ARTIST_TEXT"
internal const val EXTRA_KEY_ALBUM_TEXT = "com.example.holodex.EXTRA_ALBUM_TEXT"
internal const val EXTRA_KEY_DESCRIPTION_TEXT = "com.example.holodex.EXTRA_DESCRIPTION_TEXT"
internal const val EXTRA_KEY_HOLODEX_CHANNEL_ID = "com.example.holodex.EXTRA_CHANNEL_ID"
private const val MAPPER_TAG = "MediaItemMapper"

class MediaItemMapper @Inject constructor(
    private val context: Context,
    private val sharedPreferences: SharedPreferences
) {

    fun toMedia3MediaItem(playbackItem: PlaybackItem): MediaItem {
        // 1. Determine URI
        // Controller has already populated streamUri if local.
        val uriToUse = if (!playbackItem.streamUri.isNullOrBlank()) {
            playbackItem.streamUri!!.toUri()
        } else {
            "holodex://resolve/${playbackItem.id}".toUri()
        }

        // 2. Build Metadata
        val extras = Bundle().apply {
            putString(EXTRA_KEY_HOLODEX_VIDEO_ID, playbackItem.videoId)
            playbackItem.songId?.let { putString(EXTRA_KEY_HOLODEX_SONG_ID, it) }
            playbackItem.serverUuid?.let { putString(EXTRA_KEY_HOLODEX_SERVER_UUID, it) }
            putLong(EXTRA_KEY_ORIGINAL_DURATION_SEC, playbackItem.durationSec)
            putString(EXTRA_KEY_ARTIST_TEXT, playbackItem.artistText)
            playbackItem.albumText?.let { putString(EXTRA_KEY_ALBUM_TEXT, it) }
            playbackItem.description?.let { putString(EXTRA_KEY_DESCRIPTION_TEXT, it) }
            putString(EXTRA_KEY_HOLODEX_CHANNEL_ID, playbackItem.channelId)
        }

        val imageQualityPref = sharedPreferences.getString(
            AppPreferenceConstants.PREF_IMAGE_QUALITY,
```

```
            AppPreferenceConstants.IMAGE_QUALITY_AUTO
        ) ?: AppPreferenceConstants.IMAGE_QUALITY_AUTO

        val highResArtworkUriString = getHighResArtworkUrl(playbackItem.artworkUri, imageQuali

        val mediaMetadata = MediaMetadata.Builder()
            .setTitle(playbackItem.title.ifBlank { context.getString(R.string.unknown_title) }
            .setArtist(playbackItem.artistText.ifBlank { context.getString(R.string.unknown_ar
            .setAlbumTitle(playbackItem.albumText ?: playbackItem.title)
            .setArtworkUri(highResArtworkUriString?.toUri())
            .setExtras(extras)
            .build()

        val mediaItemBuilder = MediaItem.Builder()
            .setMediaId(playbackItem.id)
            .setMediaMetadata(mediaMetadata)
            .setUri(uriToUse)

        // 3. Apply Clipping
        // Controller clears clip start/end if file is local.
        // So we simply check if clip params exist. Simpler logic.
        if (playbackItem.clipStartSec != null && playbackItem.clipEndSec != null && playbackIt
            mediaItemBuilder.setClippingConfiguration(
                MediaItem.ClippingConfiguration.Builder()
                    .setStartPositionMs(playbackItem.clipStartSec * 1000L)
                    .setEndPositionMs(playbackItem.clipEndSec * 1000L)
                    .setRelativeToDefaultPosition(false)
                    .build()
            )
        }

        return mediaItemBuilder.build()
    }

    fun toPlaybackItem(mediaItem: MediaItem): PlaybackItem? {
        val mediaId = mediaItem.mediaId
        if (mediaId.isBlank()) return null

        val metadata = mediaItem.mediaMetadata
        val extras = metadata.extras ?: Bundle.EMPTY
        val durationSec = extras.getLong(EXTRA_KEY_ORIGINAL_DURATION_SEC, 0L)

        // Note: We don't recover clipping config here because the UI uses the ID to look up m

        return PlaybackItem(
            id = mediaId,
            videoId = extras.getString(EXTRA_KEY_HOLODEX_VIDEO_ID) ?: "unknown",
            serverUuid = extras.getString(EXTRA_KEY_HOLODEX_SERVER_UUID),
            songId = extras.getString(EXTRA_KEY_HOLODEX_SONG_ID),
            title = metadata.title?.toString() ?: "Unknown",
            artistText = extras.getString(EXTRA_KEY_ARTIST_TEXT) ?: metadata.artist?.toString(
            albumText = extras.getString(EXTRA_KEY_ALBUM_TEXT),
            artworkUri = metadata.artworkUri?.toString(),
            durationSec = durationSec,
            streamUri = mediaItem.localConfiguration?.uri?.toString(),
```

```
            clipStartSec = null,
            clipEndSec = null,
            description = extras.getString(EXTRA_KEY_DESCRIPTION_TEXT),
            channelId = extras.getString(EXTRA_KEY_HOLODEX_CHANNEL_ID) ?: "unknown",
            originalArtist = null
        )
    }

}

// File: java\com\example\holodex\playback\data\model\PlaybackDao.kt
package com.example.holodex.playback.data.model

import androidx.room.Dao
import androidx.room.Insert
import androidx.room.OnConflictStrategy
import androidx.room.Query
import androidx.room.Transaction
import com.example.holodex.data.db.UnifiedItemProjection

@Dao
interface PlaybackDao {

    // --- WRITES ---

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun setPlaybackState(state: PlaybackStateEntity)

    @Query("DELETE FROM playback_queue_ref")
    suspend fun clearQueue()

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertQueueRefs(refs: List<PlaybackQueueRefEntity>)

    @Transaction
    suspend fun saveFullState(
        state: PlaybackStateEntity,
        activeQueue: List<PlaybackQueueRefEntity>,
        backupQueue: List<PlaybackQueueRefEntity>
    ) {
        // Atomic transaction: Clear old state -> Save new state
        clearQueue()
        setPlaybackState(state)
        insertQueueRefs(activeQueue)
        if (backupQueue.isNotEmpty()) {
            insertQueueRefs(backupQueue)
        }
    }

    // --- READS ---

    @Query("SELECT * FROM playback_state WHERE id = 0 LIMIT 1")
    suspend fun getState(): PlaybackStateEntity?

    /**
```

```
     * Fetches the Active Queue (Shuffled or Normal)
     * Joins with UnifiedMetadata so we get Titles/Images instantly.
     */
    @Transaction
    @Query("""
        SELECT M.* FROM unified_metadata M
        INNER JOIN playback_queue_ref Q ON M.id = Q.item_id
        WHERE Q.is_backup = 0
        ORDER BY Q.queue_index ASC
    """)
    suspend fun getActiveQueueWithMetadata(): List<UnifiedItemProjection>

    /**
     * Fetches the Backup Queue (Original Order)
     * Used to restore order when un-shuffling.
     */
    @Transaction
    @Query("""
        SELECT M.* FROM unified_metadata M
        INNER JOIN playback_queue_ref Q ON M.id = Q.item_id
        WHERE Q.is_backup = 1
        ORDER BY Q.queue_index ASC
    """)
    suspend fun getBackupQueueWithMetadata(): List<UnifiedItemProjection>
}

// File: java\com\example\holodex\playback\data\model\PlaybackEntities.kt
package com.example.holodex.playback.data.model

import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.ForeignKey
import androidx.room.Index
import androidx.room.PrimaryKey
import com.example.holodex.data.db.UnifiedMetadataEntity

/**
 * Stores the "Head" of the state: Index, Position, Shuffle Mode.
 * Lightweight snapshot of the player settings.
 */
@Entity(tableName = "playback_state")
data class PlaybackStateEntity(
    @PrimaryKey val id: Int = 0, // Single row architecture
    @ColumnInfo(name = "current_index") val currentIndex: Int,
    @ColumnInfo(name = "position_ms") val positionMs: Long,
    @ColumnInfo(name = "shuffle_mode_enabled") val isShuffleEnabled: Boolean,
    @ColumnInfo(name = "repeat_mode") val repeatMode: Int // 0=OFF, 1=ONE, 2=ALL
)

/**
 * Stores the Queue as a list of IDs pointing to the Unified Table.
 * This is the "Database Diet" - no duplicated strings/urls.
 */
@Entity(
    tableName = "playback_queue_ref",
```

```
    primaryKeys = ["queue_index", "is_backup"], // unique item at specific position
    foreignKeys = [
        ForeignKey(
            entity = UnifiedMetadataEntity::class,
            parentColumns = ["id"],
            childColumns = ["item_id"],
            onDelete = ForeignKey.CASCADE
        )
    ],
    indices = [Index("item_id")]
)
data class PlaybackQueueRefEntity(
    @ColumnInfo(name = "queue_index") val sortOrder: Int,
    @ColumnInfo(name = "item_id") val itemId: String,

    // TRUE = The original order (Used when shuffle is OFF)
    // FALSE = The current active order (Used by Player right now)
    @ColumnInfo(name = "is_backup") val isBackup: Boolean
)

// File: java\com\example\holodex\playback\data\preload\PreloadConfiguration.kt
// File: java/com/example/holodex/playback/data/preload/PreloadConfiguration.kt
package com.example.holodex.playback.data.preload

data class PreloadConfiguration(
    val preloadDurationMs: Long = 10_000L,
    val maxConcurrentPreloads: Int = 2,
    val isEnabled: Boolean = true
)

// File: java\com\example\holodex\playback\data\preload\PreloadStatusController.kt
// File: java/com/example/holodex/playback/data/preload/PreloadStatusController.kt
package com.example.holodex.playback.data.preload

import androidx.media3.common.util.UnstableApi
import androidx.media3.exoplayer.source.preload.DefaultPreloadManager
import androidx.media3.exoplayer.source.preload.TargetPreloadStatusControl
import timber.log.Timber

@UnstableApi
class PreloadStatusController(
    private val getCurrentIndex: () -> Int,
    private val preloadDurationMs: Long = 10_000L
) : TargetPreloadStatusControl<Int, DefaultPreloadManager.PreloadStatus> {

    companion object {
        private const val TAG = "PreloadStatusController"
    }

    override fun getTargetPreloadStatus(rankingData: Int): DefaultPreloadManager.PreloadStatus
        val currentIndex = getCurrentIndex()
        val ranking = rankingData - currentIndex

        return when (ranking) {
            1 -> {
```

```
                Timber.i("$TAG: Preloading next item (index $rankingData) for ${preloadDuratio
                DefaultPreloadManager.PreloadStatus.specifiedRangeLoaded(preloadDurationMs)
            }
            2 -> {
                Timber.i("$TAG: Preloading second item (index $rankingData) for ${preloadDurat
                DefaultPreloadManager.PreloadStatus.specifiedRangeLoaded(preloadDurationMs / 2
            }
            else -> {
                null
            }
        }
    }
}

// File: java\com\example\holodex\playback\data\queue\PlaybackQueueState.kt
// File: java/com/example/holodex/playback/data/queue/PlaybackQueueState.kt
package com.example.holodex.playback.data.queue

import com.example.holodex.playback.domain.model.DomainRepeatMode
import com.example.holodex.playback.domain.model.DomainShuffleMode
import com.example.holodex.playback.domain.model.PlaybackItem

data class PlaybackQueueState(
    val originalList: List<PlaybackItem> = emptyList(),
    val activeList: List<PlaybackItem> = emptyList(),
    val currentIndex: Int = -1,
    val shuffleMode: DomainShuffleMode = DomainShuffleMode.OFF,
    val repeatMode: DomainRepeatMode = DomainRepeatMode.NONE,
    val transientStartPositionMs: Long = 0L
) {
    val currentItem: PlaybackItem?
        get() = activeList.getOrNull(currentIndex)
}

// File: java\com\example\holodex\playback\data\queue\QueueAction.kt
// File: java/com/example/holodex/playback/data/queue/QueueAction.kt
package com.example.holodex.playback.data.queue

import com.example.holodex.playback.domain.model.DomainRepeatMode
import com.example.holodex.playback.domain.model.DomainShuffleMode
import com.example.holodex.playback.domain.model.PlaybackItem

sealed class QueueAction {
    data class SetQueue(
        val items: List<PlaybackItem>,
        val startIndex: Int,
        val startPositionMs: Long = 0L,
        val shouldShuffle: Boolean = false,
        val restoredShuffleMode: DomainShuffleMode? = null,
        val restoredRepeatMode: DomainRepeatMode? = null,
        val restoredShuffledList: List<PlaybackItem>? = null
    ) : QueueAction()

    object ToggleShuffle : QueueAction()
    data class SetRepeatMode(val repeatMode: DomainRepeatMode) : QueueAction()
```

```
    data class ReorderItem(val fromIndex: Int, val toIndex: Int) : QueueAction()
    data class AddItem(val item: PlaybackItem, val index: Int?) : QueueAction()
    data class AddItems(val items: List<PlaybackItem>, val index: Int?) : QueueAction()
    data class RemoveItem(val index: Int) : QueueAction()
    data class SetCurrentIndex(val newIndex: Int) : QueueAction()
    object ClearQueue : QueueAction()
    data class UpdateItemInQueue(val updatedItem: PlaybackItem) : QueueAction()
    object SkipToNext : QueueAction()
    object SkipToPrevious : QueueAction()
}

// File: java\com\example\holodex\playback\data\queue\ShuffleOrderProvider.kt
// File: java/com/example/holodex/playback/data/queue/ShuffleOrderProvider.kt
package com.example.holodex.playback.data.queue

import com.example.holodex.playback.domain.model.PlaybackItem

class ShuffleOrderProvider {
    fun createShuffledList(
        originalItems: List<PlaybackItem>,
        currentItem: PlaybackItem?
    ): List<PlaybackItem> {
        if (originalItems.isEmpty()) return emptyList()

        val mutableList = originalItems.toMutableList()
        val currentIndex = currentItem?.let { originalItems.indexOf(it) } ?: -1

        if (currentIndex >= 0) {
            val current = mutableList.removeAt(currentIndex)
            mutableList.shuffle()
            return listOf(current) + mutableList
        } else {
            mutableList.shuffle()
            return mutableList
        }
    }
}

// File: java\com\example\holodex\playback\data\repository\HolodexStreamResolverRepositoryImpl
// File: java/com/example/holodex/playback/data/repository/HolodexStreamResolverRepositoryImpl
package com.example.holodex.playback.data.repository

import com.example.holodex.data.model.AudioStreamDetails
import com.example.holodex.data.repository.YouTubeStreamRepository
import com.example.holodex.playback.domain.model.StreamDetails
import com.example.holodex.playback.domain.repository.StreamResolverRepository
import javax.inject.Inject
import javax.inject.Singleton

@Singleton
class HolodexStreamResolverRepositoryImpl @Inject constructor(
    private val youtubeStreamRepository: YouTubeStreamRepository
) : StreamResolverRepository {

    override suspend fun resolveStreamUrl(videoId: String): Result<StreamDetails> {
```

```
        return try {
            val youtubeResult: Result<AudioStreamDetails> = youtubeStreamRepository.getAudioSt

            youtubeResult.map { oldDetails ->
                val streamUrl = oldDetails.streamUrl ?: throw IllegalStateException("Stream UR
                StreamDetails(
                    url = streamUrl,
                    format = oldDetails.format,
                    quality = oldDetails.quality
                )
            }
        } catch (e: Exception) {
            Result.failure(e)
        }
    }
}

// File: java\com\example\holodex\playback\data\source\HolodexResolvingDataSource.kt
package com.example.holodex.playback.data.source

import android.net.Uri
import androidx.collection.LruCache
import androidx.media3.common.util.UnstableApi
import androidx.media3.datasource.DataSpec
import androidx.media3.datasource.ResolvingDataSource
import com.example.holodex.data.db.DownloadStatus
import com.example.holodex.data.db.UnifiedDao
import com.example.holodex.data.repository.YouTubeStreamRepository
import kotlinx.coroutines.runBlocking
import timber.log.Timber
import java.io.File
import java.util.concurrent.TimeUnit
import javax.inject.Inject
import javax.inject.Singleton

@UnstableApi
@Singleton
class HolodexResolvingDataSource @Inject constructor(
    private val streamRepository: YouTubeStreamRepository,
    private val unifiedDao: UnifiedDao
) : ResolvingDataSource.Resolver {

    private data class CachedUrl(val url: String, val timestamp: Long)
    private val urlCache = LruCache<String, CachedUrl>(20)
    private val CACHE_TTL_MS = TimeUnit.MINUTES.toMillis(60)

    override fun resolveDataSpec(dataSpec: DataSpec): DataSpec {
        val uri = dataSpec.uri

        if (uri.scheme == "holodex" && uri.host == "resolve") {
            val rawId = uri.lastPathSegment ?: return dataSpec

            // ---------------------------------------------------------------
            // 1. CHECK LOCAL DATABASE (PURE SYNC - NO COROUTINES)
            // ---------------------------------------------------------------
```

```
val localPath = try {
    // A. Try exact match
    var interaction = unifiedDao.getDownloadInteractionSync(rawId)

    // B. If not found, try parent ID
    if (interaction == null && rawId.contains("_")) {
        val parentId = rawId.substringBeforeLast("_")
        interaction = unifiedDao.getDownloadInteractionSync(parentId)
    }

    if (interaction != null &&
        interaction.downloadStatus == DownloadStatus.COMPLETED.name &&
        !interaction.localFilePath.isNullOrBlank()
    ) {
        val path = interaction.localFilePath!!
        if (path.startsWith("content://") || File(path).exists()) {
            path
        } else {
            Timber.w("ResolvingDataSource: DB path missing on disk: $path")
            null
        }
    } else {
        null
    }
} catch (e: Exception) {
    Timber.e(e, "Error checking DB for local file (Sync)")
    null
}

if (localPath != null) {
    Timber.i("ResolvingDataSource: Using LOCAL file for $rawId")
    return dataSpec.buildUpon()
        .setUri(Uri.parse(localPath))
        .build()
}

// ---------------------------------------------------------------
// 2. NETWORK RESOLUTION
// ---------------------------------------------------------------
val videoIdForNetwork = if (rawId.contains("_") && rawId.substringAfterLast("_").a
    rawId.substringBeforeLast("_")
} else {
    rawId
}

synchronized(urlCache) {
    val cachedEntry = urlCache[videoIdForNetwork]
    if (cachedEntry != null) {
        if (System.currentTimeMillis() - cachedEntry.timestamp < CACHE_TTL_MS) {
            return dataSpec.buildUpon().setUri(Uri.parse(cachedEntry.url)).build()
        } else {
            urlCache.remove(videoIdForNetwork)
        }
    }
}
```

```
            Timber.i("ResolvingDataSource: Resolving network for $videoIdForNetwork")

            // Network calls inherently require blocking or async.
            // Since we can't use async here easily without blocking (which causes the issue),
            // and runBlocking is what failed, we need a safe way.
            // Ideally, we shouldn't do network calls in resolveDataSpec if we can avoid it,
            // but NewPipe requires extraction.

            // We will try runBlocking ONLY for network, as it takes longer.
            // But if it fails/cancels, we return original spec to let ExoPlayer handle the er
            val streamDetails = try {
                runBlocking {
                    streamRepository.getAudioStreamDetails(videoIdForNetwork).getOrNull()
                }
            } catch (e: Exception) {
                Timber.w("Network resolution cancelled or failed for $videoIdForNetwork")
                null
            }

            if (streamDetails != null) {
                synchronized(urlCache) {
                    urlCache.put(videoIdForNetwork, CachedUrl(streamDetails.streamUrl, System.
                }
                return dataSpec.buildUpon().setUri(Uri.parse(streamDetails.streamUrl)).build()
            } else {
                // Don't throw exception here. Return the original Spec.
                // ExoPlayer will try to load "holodex://..." and fail with a standard HttpErr
                // which might trigger a retry policy instead of a hard crash/skip.
                return dataSpec
            }
        }

        return dataSpec
    }
}

// File: java\com\example\holodex\playback\data\source\StreamResolutionCoordinator.kt
package com.example.holodex.playback.data.source

import androidx.collection.LruCache
import androidx.media3.common.util.UnstableApi
import com.example.holodex.data.db.DownloadStatus
import com.example.holodex.data.db.UnifiedDao
import com.example.holodex.data.db.UserInteractionEntity
import com.example.holodex.playback.domain.model.PlaybackItem
import com.example.holodex.playback.domain.model.StreamDetails
import com.example.holodex.playback.domain.repository.StreamResolverRepository
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.Job
import kotlinx.coroutines.isActive
import kotlinx.coroutines.withContext
import java.util.concurrent.TimeUnit
import kotlin.coroutines.coroutineContext
```

```
@UnstableApi
class StreamResolutionCoordinator(
    private val streamResolverRepository: StreamResolverRepository,
    private val unifiedDao: UnifiedDao // <-- REPLACED DownloadedItemDao
) {
    companion object {
        private const val TAG = "StreamResolutionCoord"
    }

    private var currentResolutionJob: Job? = null
    private data class ResolvedStreamCacheEntry(val streamDetails: StreamDetails, val timestam
    private val streamUrlCache = LruCache<String, ResolvedStreamCacheEntry>(50)
    private val cacheExpiryMs = TimeUnit.MINUTES.toMillis(10)

    suspend fun resolveSingleStream(item: PlaybackItem): PlaybackItem? {
        return withContext(Dispatchers.IO) {
            resolveSingleStreamInternal(item.copy(), null)
        }
    }

    fun clearMemoryCache() {
        streamUrlCache.evictAll()
    }

    fun getCachedUrl(videoId: String): String? {
        return getStreamFromCache(videoId)?.url
    }

    private suspend fun resolveSingleStreamInternal(
        item: PlaybackItem,
        prewarmedDownloads: Map<String, UserInteractionEntity>? // Type changed
    ): PlaybackItem? {
        if (!item.streamUri.isNullOrBlank()) {
            return item
        }

        // --- THE FIX ---
        // Check for download interaction in the unified table
        val downloadInteraction = prewarmedDownloads?.get(item.id) ?: unifiedDao.getDownloadIn
        if (downloadInteraction != null &&
            downloadInteraction.downloadStatus == DownloadStatus.COMPLETED.name &&
            !downloadInteraction.localFilePath.isNullOrBlank()) {
            return item.copy(streamUri = downloadInteraction.localFilePath)
        }
        // --- END FIX ---

        getStreamFromCache(item.videoId)?.let { cachedDetails ->
            return item.copy(streamUri = cachedDetails.url)
        }

        if (!coroutineContext.isActive) {
            return null
        }

        return try {
```

```
            val result = streamResolverRepository.resolveStreamUrl(item.videoId)
            if (result.isSuccess) {
                val streamDetails = result.getOrThrow()
                putStreamInCache(item.videoId, streamDetails)
                item.copy(streamUri = streamDetails.url)
            } else {
                null
            }
        } catch (e: Exception) {
            null
        }
    }

    private fun getStreamFromCache(videoId: String): StreamDetails? {
        val entry = streamUrlCache[videoId]
        if (entry != null) {
            if (System.currentTimeMillis() - entry.timestamp < cacheExpiryMs) {
                return entry.streamDetails
            } else {
                streamUrlCache.remove(videoId)
            }
        }
        return null
    }

    private fun putStreamInCache(videoId: String, streamDetails: StreamDetails) {
        streamUrlCache.put(videoId, ResolvedStreamCacheEntry(streamDetails, System.currentTime
    }

    fun cancelOngoingResolutions() {
        synchronized(this) {
            currentResolutionJob?.cancel()
            currentResolutionJob = null
        }
    }
}

// File: java\com\example\holodex\playback\domain\model\DomainPlaybackProgress.kt
// File: java/com/example/holodex/playback/domain/model/DomainPlaybackProgress.kt
package com.example.holodex.playback.domain.model

data class DomainPlaybackProgress(
    val positionSec: Long = 0L,
    val durationSec: Long = 0L,
    val bufferedPositionSec: Long = 0L
) {
    companion object {
        val NONE = DomainPlaybackProgress()
    }
}

// File: java\com\example\holodex\playback\domain\model\DomainPlaybackState.kt
// File: java/com/example/holodex/playback/domain/model/DomainPlaybackState.kt
package com.example.holodex.playback.domain.model
```

```
enum class DomainPlaybackState {
    IDLE,
    BUFFERING,
    PLAYING,
    PAUSED,
    ENDED,
    ERROR
}

// File: java\com\example\holodex\playback\domain\model\DomainRepeatMode.kt
// File: java/com/example/holodex/playback/domain/model/DomainRepeatMode.kt
package com.example.holodex.playback.domain.model

enum class DomainRepeatMode {
    NONE,
    ONE,
    ALL
}

// File: java\com\example\holodex\playback\domain\model\DomainShuffleMode.kt
// File: java/com/example/holodex/playback/domain/model/DomainShuffleMode.kt
package com.example.holodex.playback.domain.model

enum class DomainShuffleMode {
    OFF,
    ON
}

// File: java\com\example\holodex\playback\domain\model\PersistedPlaybackData.kt
// File: java/com/example/holodex/playback/domain/model/PersistedPlaybackData.kt
package com.example.holodex.playback.domain.model

data class PersistedPlaybackData(
    val queueId: String,
    val queueItems: List<PersistedPlaybackItem>,
    val currentIndex: Int,
    val currentPositionSec: Long,
    val currentItemId: String?,
    val repeatMode: DomainRepeatMode,
    val shuffleMode: DomainShuffleMode,
    val shuffledQueueItemIds: List<String>? = null
)

data class PersistedPlaybackItem(
    val id: String,
    val videoId: String,
    val songId: String?,
    val title: String,
    val artistText: String,
    val albumText: String?,
    val artworkUri: String?,
    val durationSec: Long,
    val description: String? = null,
    val channelId: String,
    val clipStartSec: Long? = null,
```

```
    val clipEndSec: Long? = null
)

// File: java\com\example\holodex\playback\domain\model\PlaybackItem.kt
// File: java/com/example/holodex/playback/domain/model/PlaybackItem.kt
package com.example.holodex.playback.domain.model

import android.os.Parcelable
import kotlinx.parcelize.Parcelize

@Parcelize
data class PlaybackItem(
    /** The unique composite ID for this item, used for playback and UI state. (e.g., "videoId
    val id: String,

    /** The ID of the parent YouTube video. */
    val videoId: String,

    /** The server's unique UUID for this specific song segment. Null for full videos. */
    val serverUuid: String?,

    /** DEPRECATED USAGE: This was used ambiguously. Now primarily for internal reference if n
    val songId: String?,

    val title: String,
    val artistText: String,
    val albumText: String?,
    val artworkUri: String?,
    val durationSec: Long,
    var streamUri: String? = null,
    val clipStartSec: Long? = null,
    val clipEndSec: Long? = null,
    val description: String? = null,
    val channelId: String,
    val originalArtist: String? = null,

    // *** THIS IS THE MISSING PROPERTY ***
    val isExternal: Boolean = false
) : Parcelable

// File: java\com\example\holodex\playback\domain\model\PlaybackQueue.kt
// File: java/com/example/holodex/playback/domain/model/PlaybackQueue.kt
package com.example.holodex.playback.domain.model

data class PlaybackQueue(
    val items: List<PlaybackItem> = emptyList(),
    val currentIndex: Int = -1,
    val repeatMode: DomainRepeatMode = DomainRepeatMode.NONE,
    val shuffleMode: DomainShuffleMode = DomainShuffleMode.OFF,
    val queueId: String = "default_queue"
) {
    val currentItem: PlaybackItem?
        get() = items.getOrNull(currentIndex)
}
```

```
// File: java\com\example\holodex\playback\domain\model\StreamDetails.kt
// File: java/com/example/holodex/playback/domain/model/StreamDetails.kt
package com.example.holodex.playback.domain.model

data class StreamDetails(
    val url: String,
    val format: String?,
    val quality: String?
)

// File: java\com\example\holodex\playback\domain\repository\PlaybackStateRepository.kt
// File: java/com/example/holodex/playback/domain/repository/PlaybackStateRepository.kt
package com.example.holodex.playback.domain.repository

import com.example.holodex.playback.domain.model.PersistedPlaybackData

interface PlaybackStateRepository {
    suspend fun saveState(data: PersistedPlaybackData)
    suspend fun loadState(): PersistedPlaybackData?
    suspend fun clearState()
}

// File: java\com\example\holodex\playback\domain\repository\StreamResolverRepository.kt
// File: java/com/example/holodex/playback/domain/repository/StreamResolverRepository.kt
package com.example.holodex.playback.domain.repository

import com.example.holodex.playback.domain.model.StreamDetails

interface StreamResolverRepository {
    suspend fun resolveStreamUrl(videoId: String): Result<StreamDetails>
}

// File: java\com\example\holodex\playback\domain\usecase\AddItemsToQueueUseCase.kt
package com.example.holodex.playback.domain.usecase

import com.example.holodex.playback.domain.model.PlaybackItem
import com.example.holodex.playback.player.PlaybackController
import javax.inject.Inject

class AddItemsToQueueUseCase @Inject constructor(
    private val controller: PlaybackController
) {
    /**
     * Adds items to the playback queue.
     * @param items The items to add.
     * @param index The index to insert at. If null, appends to end (or next in shuffle).
     */
    operator fun invoke(items: List<PlaybackItem>, index: Int? = null) {
        controller.addItemsToQueue(items, index)
    }
}

// File: java\com\example\holodex\playback\domain\usecase\AddItemToQueueUseCase.kt
package com.example.holodex.playback.domain.usecase
```

```
import com.example.holodex.playback.domain.model.PlaybackItem
import com.example.holodex.playback.player.PlaybackController
import javax.inject.Inject

class AddItemToQueueUseCase @Inject constructor(
    private val controller: PlaybackController
) {
    operator fun invoke(item: PlaybackItem, index: Int? = null) {
        controller.addItemsToQueue(listOf(item), index)
    }
}

// File: java\com\example\holodex\playback\domain\usecase\AddOrFetchAndAddUseCase.kt
// File: java/com/example/holodex/playback/domain/usecase/AddOrFetchAndAddUseCase.kt
package com.example.holodex.playback.domain.usecase

import com.example.holodex.data.repository.HolodexRepository
import com.example.holodex.playback.domain.model.PlaybackItem
import com.example.holodex.viewmodel.mappers.toPlaybackItem
import timber.log.Timber
import javax.inject.Inject

/**
 * A use case that intelligently adds items to the playback queue.
 * - If the item is a song segment, it's added directly.
 * - If the item is a full video, it fetches the video's segments and adds all of them.
 * - If a video has no segments, it adds the full video itself.
 * @return A Result containing a user-friendly message for a Toast/Snackbar.
 */
class AddOrFetchAndAddUseCase @Inject constructor(
    private val holodexRepository: HolodexRepository,
    private val addItemsToQueueUseCase: AddItemsToQueueUseCase
) {
    suspend operator fun invoke(item: PlaybackItem): Result<String> {
        return try {
            if (item.songId != null) {
                // It's a single song segment, add it directly.
                addItemsToQueueUseCase(listOf(item))
                Timber.d("AddOrFetchAndAddUseCase: Added single segment '${item.title}' to que
                Result.success("Added '${item.title}' to queue.")
            } else {
                // It's a full video, fetch its details to get all segments.
                Timber.d("AddOrFetchAndAddUseCase: Item is a full video ('${item.title}'). Fet
                val videoResult = holodexRepository.getVideoWithSongs(item.videoId, forceRefre

                videoResult.fold(
                    onSuccess = { videoWithSongs ->
                        if (!videoWithSongs.songs.isNullOrEmpty()) {
                            // Video has segments, add them all.
                            val segmentItems = videoWithSongs.songs.map { song ->
                                song.toPlaybackItem(videoWithSongs)
                            }
                            addItemsToQueueUseCase(segmentItems)
                            Timber.d("AddOrFetchAndAddUseCase: Added ${segmentItems.size} segm
                            Result.success("Added ${segmentItems.size} songs from '${item.titl
```

```
                        } else {
                            // Video has no segments, add the full video itself.
                            addItemsToQueueUseCase(listOf(item))
                            Timber.d("AddOrFetchAndAddUseCase: Video has no segments. Added fu
                            Result.success("Added '${item.title}' to queue.")
                        }
                    },
                    onFailure = {
                        Timber.e(it, "AddOrFetchAndAddUseCase: Failed to fetch video details f
                        Result.failure(it)
                    }
                )
            }
        } catch (e: Exception) {
            Timber.e(e, "AddOrFetchAndAddUseCase: An unexpected error occurred.")
            Result.failure(e)
        }
    }
}

// File: java\com\example\holodex\playback\domain\usecase\LoadPlaybackStateUseCase.kt
package com.example.holodex.playback.domain.usecase

import com.example.holodex.playback.domain.model.PersistedPlaybackData
import com.example.holodex.playback.domain.repository.PlaybackStateRepository

class LoadPlaybackStateUseCase(
    private val playbackStateRepository: PlaybackStateRepository
) {
    suspend operator fun invoke(): PersistedPlaybackData? {
        return playbackStateRepository.loadState()
    }
}

// File: java\com\example\holodex\playback\domain\usecase\ResolveStreamUrlUseCase.kt
package com.example.holodex.playback.domain.usecase

import com.example.holodex.playback.domain.model.StreamDetails
import com.example.holodex.playback.domain.repository.StreamResolverRepository


class ResolveStreamUrlUseCase(
    private val streamResolverRepository: StreamResolverRepository
) {
    suspend operator fun invoke(videoId: String): Result<StreamDetails> {
        return streamResolverRepository.resolveStreamUrl(videoId)
    }
}

// File: java\com\example\holodex\playback\domain\usecase\SavePlaybackStateUseCase.kt
package com.example.holodex.playback.domain.usecase

import com.example.holodex.playback.domain.model.PersistedPlaybackData
import com.example.holodex.playback.domain.repository.PlaybackStateRepository
```

```
class SavePlaybackStateUseCase(
    private val playbackStateRepository: PlaybackStateRepository
) {
    suspend operator fun invoke(data: PersistedPlaybackData) {
        playbackStateRepository.saveState(data)
    }
}

// File: java\com\example\holodex\playback\player\Media3PlayerController.kt
// File: java/com/example/holodex/playback/player/Media3PlayerController.kt
package com.example.holodex.playback.player

import androidx.media3.common.AudioAttributes
import androidx.media3.common.C
import androidx.media3.common.MediaItem
import androidx.media3.common.PlaybackException
import androidx.media3.common.Player
import androidx.media3.common.Timeline
import androidx.media3.common.util.UnstableApi
import androidx.media3.exoplayer.ExoPlayer
import androidx.media3.exoplayer.source.preload.DefaultPreloadManager
import androidx.media3.exoplayer.util.EventLogger
import com.example.holodex.playback.domain.model.DomainPlaybackState
import com.example.holodex.playback.util.PlayerStateMapper
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.SupervisorJob
import kotlinx.coroutines.flow.MutableSharedFlow
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.SharedFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asSharedFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.launch
import timber.log.Timber

data class PlayerMediaItemTransition(val mediaItem: MediaItem?, val newIndex: Int, val reason:
data class PlayerErrorEvent(val error: PlaybackException)
data class PlayerTimelineChangedEvent(val timeline: Timeline, val reason: Int)
data class PlayerIsPlayingChangedEvent(val isPlaying: Boolean)
data class PlayerDiscontinuityEvent(
    val oldPosition: Player.PositionInfo,
    val newPosition: Player.PositionInfo,
    val reason: Int
)

@UnstableApi
class Media3PlayerController(
    val exoPlayer: ExoPlayer,
    private val preloadManager: DefaultPreloadManager
) : Player.Listener {

    companion object {
        private const val TAG = "Media3PlayerController"
    }
```

```
private val eventLogger: EventLogger = EventLogger(TAG)
private val controllerScope = CoroutineScope(Dispatchers.Main.immediate + SupervisorJob())

private val _playerPlaybackStateFlow = MutableStateFlow(DomainPlaybackState.IDLE)
val playerPlaybackStateFlow: StateFlow<DomainPlaybackState> = _playerPlaybackStateFlow.asS

private val _mediaItemTransitionEventFlow = MutableSharedFlow<PlayerMediaItemTransition>(r
val mediaItemTransitionEventFlow: SharedFlow<PlayerMediaItemTransition> = _mediaItemTransi

private val _isPlayingChangedEventFlow = MutableSharedFlow<PlayerIsPlayingChangedEvent>(re
val isPlayingChangedEventFlow: SharedFlow<PlayerIsPlayingChangedEvent> = _isPlayingChanged

private val _discontinuityEventFlow = MutableSharedFlow<PlayerDiscontinuityEvent>(replay =
val discontinuityEventFlow: SharedFlow<PlayerDiscontinuityEvent> = _discontinuityEventFlow

init {
    exoPlayer.addListener(this)
    exoPlayer.addAnalyticsListener(eventLogger)
    setupAudioAttributes()

    _playerPlaybackStateFlow.value = PlayerStateMapper.mapExoPlayerStateToDomain(exoPlayer
}

private fun setupAudioAttributes() {
    val audioAttributes = AudioAttributes.Builder()
        .setContentType(C.AUDIO_CONTENT_TYPE_MUSIC)
        .setUsage(C.USAGE_MEDIA)
        .build()
    exoPlayer.setAudioAttributes(audioAttributes, true)
}

fun play() {
    if (exoPlayer.playbackState == Player.STATE_IDLE && exoPlayer.mediaItemCount > 0) {
        exoPlayer.prepare()
    }
    if (exoPlayer.playbackState == Player.STATE_ENDED) {
        exoPlayer.seekToDefaultPosition()
    }
    exoPlayer.play()
}

fun stop() {
    exoPlayer.stop()
    exoPlayer.clearMediaItems()
}

fun pause() = exoPlayer.pause()
fun seekTo(positionMs: Long) {
    if (exoPlayer.isCommandAvailable(Player.COMMAND_SEEK_IN_CURRENT_MEDIA_ITEM)) {
        exoPlayer.seekTo(positionMs.coerceAtLeast(0L))
    }
}

fun seekToItem(itemIndex: Int, positionMs: Long) {
```

```
    if (exoPlayer.isCommandAvailable(Player.COMMAND_SEEK_TO_MEDIA_ITEM)) {
        exoPlayer.seekTo(itemIndex, positionMs.coerceAtLeast(0L))
    }
}

fun setRepeatMode(@Player.RepeatMode exoPlayerMode: Int) {
    exoPlayer.repeatMode = exoPlayerMode
}

fun setMediaItems(items: List<MediaItem>, startIndex: Int, startPositionMs: Long) {
    updatePreloadManagerPlaylist(items)
    if (items.isNotEmpty()) {
        exoPlayer.setMediaItems(items, startIndex, startPositionMs)
        exoPlayer.prepare()
    } else {
        _playerPlaybackStateFlow.value = PlayerStateMapper.mapExoPlayerStateToDomain(exoPl
    }
}

fun clearMediaItemsAndStop() {
    exoPlayer.stop()
    exoPlayer.clearMediaItems()
    preloadManager.reset()
}

internal  fun getMediaItemsFromPlayerTimeline(): List<MediaItem> {
    val timeline = exoPlayer.currentTimeline
    if (timeline.isEmpty) return emptyList()
    return (0 until timeline.windowCount).map {
        timeline.getWindow(it, Timeline.Window()).mediaItem
    }
}

private fun updatePreloadManagerPlaylist(mediaItems: List<MediaItem>) {
    try {
        preloadManager.reset()
        if (mediaItems.isEmpty()) {
            preloadManager.setCurrentPlayingIndex(C.INDEX_UNSET)
            return
        }
        val preloadableItems = mediaItems.filter { it.localConfiguration?.uri?.scheme != "
        preloadableItems.forEach { item ->
            val originalIndex = mediaItems.indexOf(item)
            if (originalIndex != -1) {
                preloadManager.add(item, originalIndex)
            }
        }
        preloadManager.setCurrentPlayingIndex(exoPlayer.currentMediaItemIndex)
    } catch (e: Exception) {
        Timber.e(e, "$TAG: Error updating preload manager playlist.")
    }
}

fun releasePlayer() {
    exoPlayer.removeAnalyticsListener(eventLogger)
```

```
        exoPlayer.removeListener(this)
        exoPlayer.release()
    }

    override fun onPlaybackStateChanged(playbackState: Int) = onStateChanged()
    override fun onPlayWhenReadyChanged(playWhenReady: Boolean, reason: Int) = onStateChanged(
    private fun onStateChanged() {
        _playerPlaybackStateFlow.value = PlayerStateMapper.mapExoPlayerStateToDomain(exoPlayer
    }

    override fun onIsPlayingChanged(isPlaying: Boolean) {
        controllerScope.launch { _isPlayingChangedEventFlow.emit(PlayerIsPlayingChangedEvent(i
    }

    override fun onMediaItemTransition(mediaItem: MediaItem?, reason: Int) {
        val newIndex = exoPlayer.currentMediaItemIndex.takeIf { it != C.INDEX_UNSET } ?: -1
        controllerScope.launch { _mediaItemTransitionEventFlow.emit(PlayerMediaItemTransition(
    }

    override fun onTimelineChanged(timeline: Timeline, reason: Int) {
        if (reason == Player.TIMELINE_CHANGE_REASON_PLAYLIST_CHANGED) {
            updatePreloadManagerPlaylist(getMediaItemsFromPlayerTimeline())
        }
    }

    override fun onPlayerError(error: PlaybackException) {
        _playerPlaybackStateFlow.value = DomainPlaybackState.ERROR
    }

    override fun onPositionDiscontinuity(oldPosition: Player.PositionInfo, newPosition: Player
        controllerScope.launch { _discontinuityEventFlow.emit(PlayerDiscontinuityEvent(oldPosi
    }

    fun updatePreloadIndex(newIndex: Int) {
        try {
            preloadManager.setCurrentPlayingIndex(newIndex)
        } catch (e: Exception) {
            Timber.e(e, "$TAG: Error updating preload manager index")
        }
    }
}

// File: java\com\example\holodex\playback\player\MediaControllerManager.kt
// File: java/com/example/holodex/playback/player/MediaControllerManager.kt (NEW FILE)
package com.example.holodex.playback.player

import android.content.ComponentName
import android.content.Context
import androidx.media3.session.MediaController
import androidx.media3.session.SessionToken
import com.example.holodex.service.MediaPlaybackService
import com.google.common.util.concurrent.ListenableFuture
import com.google.common.util.concurrent.MoreExecutors
import dagger.hilt.android.qualifiers.ApplicationContext
import kotlinx.coroutines.CompletableDeferred
```

```
import timber.log.Timber
import javax.inject.Inject
import javax.inject.Singleton

/**
 * A Singleton manager that provides a single, app-wide instance of MediaController.
 * It handles the asynchronous connection to the MediaSessionService.
 */
@Singleton
class MediaControllerManager @Inject constructor(
    @ApplicationContext private val context: Context
) {
    private val sessionToken = SessionToken(context, ComponentName(context, MediaPlaybackServi
    private val controllerFuture: ListenableFuture<MediaController> =
        MediaController.Builder(context, sessionToken).buildAsync()

    private val deferredController = CompletableDeferred<MediaController>()

    init {
        controllerFuture.addListener({
            try {
                val controller = controllerFuture.get()
                deferredController.complete(controller)
                Timber.d("MediaControllerManager: MediaController connected successfully.")
            } catch (e: Exception) {
                deferredController.completeExceptionally(e)
                Timber.e(e, "MediaControllerManager: Failed to connect MediaController.")
            }
        }, MoreExecutors.directExecutor())
    }

    /**
     * Suspends until the MediaController is connected, then returns the instance.
     */
    suspend fun awaitController(): MediaController {
        return deferredController.await()
    }

    fun release() {
        if (controllerFuture.isDone) {
            MediaController.releaseFuture(controllerFuture)
        }
    }
}

// File: java\com\example\holodex\playback\player\PlaybackController.kt
package com.example.holodex.playback.player

import androidx.media3.common.C
import androidx.media3.common.MediaItem
import androidx.media3.common.Player
import androidx.media3.common.util.UnstableApi
import androidx.media3.exoplayer.ExoPlayer
import com.example.holodex.data.db.UnifiedDao
import com.example.holodex.playback.data.mapper.MediaItemMapper
```

```
import com.example.holodex.playback.data.model.PlaybackDao
import com.example.holodex.playback.data.model.PlaybackQueueRefEntity
import com.example.holodex.playback.data.model.PlaybackStateEntity
import com.example.holodex.playback.domain.model.DomainRepeatMode
import com.example.holodex.playback.domain.model.DomainShuffleMode
import com.example.holodex.playback.domain.model.PlaybackItem
import com.example.holodex.viewmodel.autoplay.ContinuationManager
import com.example.holodex.viewmodel.mappers.toPlaybackItem
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.Job
import kotlinx.coroutines.delay
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.flow.update
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext
import timber.log.Timber
import javax.inject.Inject
import javax.inject.Singleton

// The One True State Object
data class PlaybackState(
    val activeQueue: List<PlaybackItem> = emptyList(),
    val currentIndex: Int = -1,
    val isPlaying: Boolean = false,
    val repeatMode: DomainRepeatMode = DomainRepeatMode.NONE,
    val shuffleMode: DomainShuffleMode = DomainShuffleMode.OFF,
    val progressMs: Long = 0L,
    val durationMs: Long = 0L,
    val isLoading: Boolean = false
)

@UnstableApi
@Singleton
class PlaybackController @Inject constructor(
     val exoPlayer: ExoPlayer,
    private val playbackDao: PlaybackDao,
    private val unifiedDao: UnifiedDao,
    private val mapper: MediaItemMapper,
     private val continuationManager: ContinuationManager,
     private val scope: CoroutineScope
) : Player.Listener {

    private val _state = MutableStateFlow(PlaybackState())
    val state: StateFlow<PlaybackState> = _state.asStateFlow()

    // The Backup Queue (Source of Truth for Un-Shuffling)
    private var backupQueue: List<PlaybackItem> = emptyList()

    private var saveJob: Job? = null
    private var progressJob: Job? = null

    init {
```

```
    exoPlayer.addListener(this)
    restoreState()
}

// --- PUBLIC API (Called by ViewModel) ---

fun loadAndPlay(items: List<PlaybackItem>, startIndex: Int = 0, startPositionMs: Long = 0L
    scope.launch {
        // 1. Resolve Local Files (THE FIX)
        val resolvedItems = resolveLocalPaths(items)

        // 2. Update State & Backup
        backupQueue = resolvedItems

        // 3. Set to Player (on Main Thread)
        withContext(Dispatchers.Main) {
            val mediaItems = resolvedItems.mapNotNull { mapper.toMedia3MediaItem(it) }
            exoPlayer.setMediaItems(mediaItems, startIndex, startPositionMs)
            exoPlayer.prepare()
            exoPlayer.play()

            _state.update { it.copy(
                activeQueue = resolvedItems,
                currentIndex = startIndex,
                isPlaying = true
            )}
        }
        scheduleSave()
    }
}

fun togglePlayPause() {
    if (exoPlayer.isPlaying) exoPlayer.pause() else exoPlayer.play()
}

fun seekTo(positionMs: Long) {
    exoPlayer.seekTo(positionMs)
}

fun skipToNext() {
    if (exoPlayer.hasNextMediaItem()) exoPlayer.seekToNextMediaItem()
}

fun skipToPrevious() {
    if (exoPlayer.hasPreviousMediaItem()) exoPlayer.seekToPreviousMediaItem()
}

fun toggleShuffle() {
    scope.launch {
        val currentMode = _state.value.shuffleMode
        val newMode = if (currentMode == DomainShuffleMode.ON) DomainShuffleMode.OFF else

        val currentItem = _state.value.activeQueue.getOrNull(exoPlayer.currentMediaItemInd
        val currentPos = exoPlayer.currentPosition
```

```
        val newQueue = if (newMode == DomainShuffleMode.ON) {
            // SHUFFLE ON: Randomize backup, keep current song first
            val shuffled = backupQueue.toMutableList()
            shuffled.shuffle()
            if (currentItem != null) {
                shuffled.remove(currentItem)
                shuffled.add(0, currentItem)
            }
            shuffled
        } else {
            // SHUFFLE OFF: Restore backup
            backupQueue
        }

        withContext(Dispatchers.Main) {
            val mediaItems = newQueue.mapNotNull { mapper.toMedia3MediaItem(it) }
            val newIndex = newQueue.indexOfFirst { it.id == currentItem?.id }.coerceAtLeas

            exoPlayer.setMediaItems(mediaItems, newIndex, currentPos)

            _state.update { it.copy(
                activeQueue = newQueue,
                shuffleMode = newMode,
                currentIndex = newIndex
            )}
        }
        scheduleSave()
    }
}

// --- INTERNAL LOGIC ---

/**
 * The "Local File Fix".
 * Checks DB for every item. If downloaded, sets streamUri = file:// and clears clipping.
 */
private suspend fun resolveLocalPaths(items: List<PlaybackItem>): List<PlaybackItem> {
    val itemIds = items.map { it.id }
    // Use the Batch Query we planned
    val downloads = unifiedDao.getCompletedDownloadsBatch(itemIds)
    val downloadMap = downloads.associateBy { it.itemId }

    return items.map { item ->
        val localDownload = downloadMap[item.id]
        if (localDownload != null && !localDownload.localFilePath.isNullOrBlank()) {
            item.copy(
                streamUri = localDownload.localFilePath,
                clipStartSec = null, // DISABLE CLIPPING for local files
                clipEndSec = null
            )
        } else {
            item // Network item, keep clipping
        }
    }
}
```

```
private fun restoreState() {
    scope.launch {
        val savedState = playbackDao.getState() ?: return@launch
        val activeProjections = playbackDao.getActiveQueueWithMetadata()
        val backupProjections = playbackDao.getBackupQueueWithMetadata()

        val activeItems = activeProjections.map { it.toUnifiedDisplayItem().toPlaybackItem
        val backupItems = backupProjections.map { it.toUnifiedDisplayItem().toPlaybackItem

        // Resolve paths again in case files moved/deleted while app was closed
        val resolvedActive = resolveLocalPaths(activeItems)
        backupQueue = resolveLocalPaths(backupItems)

        withContext(Dispatchers.Main) {
            if (resolvedActive.isNotEmpty()) {
                val mediaItems = resolvedActive.mapNotNull { mapper.toMedia3MediaItem(it)
                exoPlayer.setMediaItems(mediaItems, savedState.currentIndex, savedState.po
                exoPlayer.repeatMode = savedState.repeatMode
                // Don't auto-play on restore
                exoPlayer.prepare()

                _state.update { it.copy(
                    activeQueue = resolvedActive,
                    currentIndex = savedState.currentIndex,
                    shuffleMode = if(savedState.isShuffleEnabled) DomainShuffleMode.ON els
                    repeatMode = when(savedState.repeatMode) {
                        1 -> DomainRepeatMode.ONE
                        2 -> DomainRepeatMode.ALL
                        else -> DomainRepeatMode.NONE
                    }
                )}
            }
        }
    }
}

private fun scheduleSave() {
    saveJob?.cancel()
    saveJob = scope.launch {
        delay(2000) // Debounce 2s

        // --- FIX: Read state on Main Thread first ---
        val playerStateSnapshot = withContext(Dispatchers.Main) {
            Triple(
                exoPlayer.currentMediaItemIndex,
                exoPlayer.currentPosition,
                exoPlayer.repeatMode
            )
        }

        // Now pass the snapshot to the background save function
        saveStateToDb(playerStateSnapshot)
    }
}
```

```
private suspend fun saveStateToDb(snapshot: Triple<Int, Long, Int>) {
    val (index, position, repeat) = snapshot
    val s = _state.value

    // 1. Prepare Metadata
    // We combine active and backup queues, and distinct by ID to avoid duplicate work
    val allItems = (s.activeQueue + backupQueue).distinctBy { it.id }

    val metadataEntities = allItems.map { item ->
        val isSegment = item.songId != null
        val type = if (isSegment) "SEGMENT" else "VIDEO"
        // If it's a segment, the parent video ID is usually videoId.
        // If it's a video, parentVideoId is null.
        val parentId = if (isSegment) item.videoId else null

        com.example.holodex.data.db.UnifiedMetadataEntity(
            id = item.id,
            title = item.title,
            artistName = item.artistText,
            type = type,
            specificArtUrl = item.artworkUri,
            uploaderAvatarUrl = null,
            duration = item.durationSec,
            channelId = item.channelId,
            description = item.description,
            startSeconds = item.clipStartSec,
            endSeconds = item.clipEndSec,
            parentVideoId = parentId,
            lastUpdatedAt = System.currentTimeMillis()
        )
    }

    // 2. Batch Insert (Optimized)
    // We use IGNORE strategy. If it exists, we do nothing (fast). If missing, we insert (
    if (metadataEntities.isNotEmpty()) {
        unifiedDao.insertMetadataBatch(metadataEntities)
    }

    // 3. Save Queue Refs
    val stateEntity = PlaybackStateEntity(
        currentIndex = index,
        positionMs = position,
        isShuffleEnabled = s.shuffleMode == DomainShuffleMode.ON,
        repeatMode = repeat
    )

    val activeRefs = s.activeQueue.mapIndexed { i, item ->
        PlaybackQueueRefEntity(i, item.id, isBackup = false)
    }
    val backupRefs = backupQueue.mapIndexed { i, item ->
        PlaybackQueueRefEntity(i, item.id, isBackup = true)
    }

    playbackDao.saveFullState(stateEntity, activeRefs, backupRefs)
```

```
}

// --- PLAYER LISTENERS ---

override fun onPlaybackStateChanged(playbackState: Int) {
    if (playbackState == Player.STATE_READY && _state.value.isPlaying) {
        startProgressLoop()
    } else {
        stopProgressLoop()
    }

    // Update loading state
    _state.update { it.copy(isLoading = playbackState == Player.STATE_BUFFERING) }
}

override fun onIsPlayingChanged(isPlaying: Boolean) {
    _state.update { it.copy(isPlaying = isPlaying) }
    if (isPlaying) startProgressLoop() else stopProgressLoop()
    scheduleSave()
}

override fun onMediaItemTransition(mediaItem: MediaItem?, reason: Int) {
    _state.update { it.copy(currentIndex = exoPlayer.currentMediaItemIndex) }
    scheduleSave()
}

private fun startProgressLoop() {
    stopProgressLoop()
    progressJob = scope.launch {
        while (true) {
            // --- FIX: Switch to Main thread to read player properties ---
            val (pos, dur) = withContext(Dispatchers.Main) {
                exoPlayer.currentPosition to exoPlayer.duration
            }

            _state.update { it.copy(
                progressMs = pos,
                durationMs = dur
            )}
            delay(500)
        }
    }
}

private fun stopProgressLoop() {
    progressJob?.cancel()
    progressJob = null
}



fun addItemsToQueue(items: List<PlaybackItem>) {
    scope.launch {
        // 1. Resolve paths
        val resolvedItems = resolveLocalPaths(items)
```

```
            // 2. Add to Backup
            backupQueue = backupQueue + resolvedItems

            // 3. Add to Player
            withContext(Dispatchers.Main) {
                val mediaItems = resolvedItems.mapNotNull { mapper.toMedia3MediaItem(it) }
                exoPlayer.addMediaItems(mediaItems)

                // State update happens via onTimelineChanged listener automatically,
                // but we trigger save explicitly
                scheduleSave()
            }
        }
    }

    fun addItemsToQueue(items: List<PlaybackItem>, index: Int? = null) {
        if (items.isEmpty()) return

        scope.launch {
            // 1. Resolve paths
            val resolvedItems = resolveLocalPaths(items)
            val mediaItems = resolvedItems.mapNotNull { mapper.toMedia3MediaItem(it) }

            withContext(Dispatchers.Main) {
                val currentQueueSize = exoPlayer.mediaItemCount
                val currentShuffleMode = _state.value.shuffleMode == DomainShuffleMode.ON

                // Determine Insertion Index for Active Queue
                // If index is provided, use it.
                // If null and Shuffling, add after current item (or at end).
                // If null and Not Shuffling, add to end.
                val insertIndex = index ?: if (currentShuffleMode) {
                    val current = exoPlayer.currentMediaItemIndex
                    if (current != C.INDEX_UNSET) current + 1 else currentQueueSize
                } else {
                    currentQueueSize
                }

                // 2. Add to Player (Active Queue)
                exoPlayer.addMediaItems(insertIndex, mediaItems)

                // 3. Add to Backup Queue
                // We always append to the END of the backup queue to preserve "Album Order" l
                // If the user wants to reorder the backup, they must unshuffle first.
                backupQueue = backupQueue + resolvedItems

                // 4. Save
                scheduleSave()
            }
        }
    }
    fun loadRadio(radioId: String) {
        scope.launch {
            // 1. Clear existing state
```

```
        withContext(Dispatchers.Main) {
            exoPlayer.stop()
            exoPlayer.clearMediaItems()
        }

        // 2. Start Radio Session via ContinuationManager
        // Note: You might need to update startRadioSession signature if it depended on Re
        // Assuming it returns List<PlaybackItem>?
        val initialItems = continuationManager.startRadioSession(radioId, scope, this@Play

        if (initialItems.isNullOrEmpty()) {
            Timber.e("PlaybackController: Failed to start radio session for $radioId")
            return@launch
        }

        // 3. Load items like normal playback
        // Resolve local files
        val resolvedItems = resolveLocalPaths(initialItems)
        backupQueue = resolvedItems // Radio doesn't really use backup/unshuffle, but we k

        withContext(Dispatchers.Main) {
            val mediaItems = resolvedItems.mapNotNull { mapper.toMedia3MediaItem(it) }
            exoPlayer.setMediaItems(mediaItems, 0, 0L)
            exoPlayer.prepare()
            exoPlayer.play()

            _state.update { it.copy(
                activeQueue = resolvedItems,
                currentIndex = 0,
                isPlaying = true,
                shuffleMode = DomainShuffleMode.OFF // Radios are usually linear streams
            )}
            scheduleSave()
        }
    }
}

fun removeItem(index: Int) {
    // 1. Get the item ID from the Active Queue BEFORE we delete it from the player
    val activeQueue = _state.value.activeQueue
    if (index !in activeQueue.indices) return

    val itemToRemove = activeQueue[index]
    val idToRemove = itemToRemove.id

    // 2. Remove from ExoPlayer (Active Queue)
    // This triggers onTimelineChanged which updates _state.activeQueue
    exoPlayer.removeMediaItem(index)

    // 3. Remove from Backup Queue
    // We reconstruct the list excluding the item with the matching ID
    val currentBackup = backupQueue
    backupQueue = currentBackup.filter { it.id != idToRemove }

    Timber.d("Removed item $idToRemove. Backup size: ${currentBackup.size} -> ${backupQueu
```

```
        scheduleSave()
    }

    fun moveItem(from: Int, to: Int) {
        exoPlayer.moveMediaItem(from, to)
    }

    fun clearQueue() {
        exoPlayer.clearMediaItems()
        exoPlayer.stop()
        backupQueue = emptyList()
        scope.launch { playbackDao.clearQueue() }
        _state.update { PlaybackState() } // Reset state
    }
}

// File: java\com\example\holodex\playback\util\PlaybackUtil.kt
// File: java/com/example/holodex/playback/util/PlaybackUtil.kt
package com.example.holodex.playback.util

import androidx.media3.common.Player
import java.util.Locale
import java.util.concurrent.TimeUnit

/** Returns a human-readable name for the given playback state. */
fun playbackStateToString(state: Int): String = when (state) {
    Player.STATE_IDLE -> "STATE_IDLE"
    Player.STATE_BUFFERING -> "STATE_BUFFERING"
    Player.STATE_READY -> "STATE_READY"
    Player.STATE_ENDED -> "STATE_ENDED"
    else -> "UNKNOWN_PLAYBACK_STATE($state)"
}

/** Returns a readable name for media-item transition reasons. */
fun mediaItemTransitionReasonToString(reason: Int): String = when (reason) {
    Player.MEDIA_ITEM_TRANSITION_REASON_AUTO -> "AUTO"
    Player.MEDIA_ITEM_TRANSITION_REASON_PLAYLIST_CHANGED -> "PLAYLIST_CHANGED"
    Player.MEDIA_ITEM_TRANSITION_REASON_REPEAT -> "REPEAT_MODE"
    Player.MEDIA_ITEM_TRANSITION_REASON_SEEK -> "SEEK"
    else -> "UNKNOWN_TRANSITION_REASON($reason)"
}

/** Returns a readable name for timeline-change reasons. */
fun timelineChangeReasonToString(reason: Int): String = when (reason) {
    Player.TIMELINE_CHANGE_REASON_PLAYLIST_CHANGED -> "PLAYLIST_CHANGED"
    Player.TIMELINE_CHANGE_REASON_SOURCE_UPDATE -> "SOURCE_UPDATE"
    else -> "UNKNOWN_TIMELINE_REASON($reason)"
}

/** Returns a readable name for discontinuity reasons. */
fun discontinuityReasonToString(reason: Int): String = when (reason) {
    Player.DISCONTINUITY_REASON_AUTO_TRANSITION -> "AUTO_TRANSITION"
    Player.DISCONTINUITY_REASON_SEEK -> "SEEK"
    Player.DISCONTINUITY_REASON_SEEK_ADJUSTMENT -> "SEEK_ADJUSTMENT"
```

```
    Player.DISCONTINUITY_REASON_REMOVE -> "REMOVE"
    Player.DISCONTINUITY_REASON_INTERNAL -> "INTERNAL"
    else -> "UNKNOWN_DISCONTINUITY_REASON($reason)"
}

/** Formats a duration in seconds into a MM:SS string. */
fun formatSongTimestamp(seconds: Long): String {
    if (seconds < 0) return "--:--"
    val minutes = TimeUnit.SECONDS.toMinutes(seconds) % 60
    val secs = seconds % 60
    return String.format(Locale.US, "%02d:%02d", minutes, secs)
}

/** Formats a total duration in seconds into a HH:MM:SS or MM:SS string. */
fun formatDurationSecondsToString(totalSeconds: Long): String {
    if (totalSeconds < 0) return "--:--"
    if (totalSeconds == 0L) return "00:00"

    val hours = TimeUnit.SECONDS.toHours(totalSeconds)
    val minutes = TimeUnit.SECONDS.toMinutes(totalSeconds) % 60
    val secs = totalSeconds % 60

    return if (hours > 0) {
        String.format(Locale.US, "%d:%02d:%02d", hours, minutes, secs)
    } else {
        String.format(Locale.US, "%02d:%02d", minutes, secs)
    }
}

/** Formats a total duration in seconds into a HH:MM:SS or MM:SS string. */
fun formatDurationSeconds(totalSecondsLong: Long): String {
    if (totalSecondsLong <= 0) return ""
    val hours = TimeUnit.SECONDS.toHours(totalSecondsLong)
    val minutes = TimeUnit.SECONDS.toMinutes(totalSecondsLong) % 60
    val secs = totalSecondsLong % 60

    return if (hours > 0) {
        String.format(Locale.US, "%d:%02d:%02d", hours, minutes, secs)
    } else {
        String.format(Locale.US, "%02d:%02d", minutes, secs)
    }
}

// File: java\com\example\holodex\playback\util\PlayerStateMapper.kt
// File: java/com/example/holodex/playback/util/PlayerStateMapper.kt
package com.example.holodex.playback.util

import androidx.media3.common.Player
import com.example.holodex.playback.domain.model.DomainPlaybackState
import com.example.holodex.playback.domain.model.DomainRepeatMode
import com.example.holodex.playback.domain.model.DomainShuffleMode

object PlayerStateMapper {

    fun mapExoPlayerStateToDomain(playbackState: Int, playWhenReady: Boolean): DomainPlaybackS
```

```
        return when (playbackState) {
            Player.STATE_IDLE -> DomainPlaybackState.IDLE
            Player.STATE_BUFFERING -> DomainPlaybackState.BUFFERING
            Player.STATE_READY -> if (playWhenReady) DomainPlaybackState.PLAYING else DomainPl
            Player.STATE_ENDED -> DomainPlaybackState.ENDED
            else -> DomainPlaybackState.IDLE
        }
    }
    @Player.RepeatMode
    fun mapDomainRepeatModeToExoPlayer(domainMode: DomainRepeatMode): Int {
        return when (domainMode) {
            DomainRepeatMode.NONE -> Player.REPEAT_MODE_OFF
            DomainRepeatMode.ONE -> Player.REPEAT_MODE_ONE
            DomainRepeatMode.ALL -> Player.REPEAT_MODE_ALL
        }
    }

    fun mapExoPlayerRepeatModeToDomain(@Player.RepeatMode exoPlayerMode: Int): DomainRepeatMod
        return when (exoPlayerMode) {
            Player.REPEAT_MODE_OFF -> DomainRepeatMode.NONE
            Player.REPEAT_MODE_ONE -> DomainRepeatMode.ONE
            Player.REPEAT_MODE_ALL -> DomainRepeatMode.ALL
            else -> DomainRepeatMode.NONE
        }
    }

    fun mapDomainShuffleModeToExoPlayer(domainMode: DomainShuffleMode): Boolean {
        return domainMode == DomainShuffleMode.ON
    }

    fun mapExoPlayerShuffleModeToDomain(exoPlayerShuffleEnabled: Boolean): DomainShuffleMode {
        return if (exoPlayerShuffleEnabled) DomainShuffleMode.ON else DomainShuffleMode.OFF
    }
}

// File: java\com\example\holodex\service\HolodexDownloadService.kt
package com.example.holodex.service

import android.app.Notification
import androidx.annotation.OptIn
import androidx.media3.common.util.UnstableApi
import androidx.media3.exoplayer.offline.Download
import androidx.media3.exoplayer.offline.DownloadManager
import androidx.media3.exoplayer.offline.DownloadNotificationHelper
import androidx.media3.exoplayer.offline.DownloadService
import androidx.media3.exoplayer.scheduler.Scheduler
import androidx.media3.exoplayer.workmanager.WorkManagerScheduler
import com.example.holodex.R
import dagger.hilt.android.AndroidEntryPoint
import timber.log.Timber
import javax.inject.Inject

@AndroidEntryPoint
@OptIn(UnstableApi::class)
class HolodexDownloadService : DownloadService(
```

```
    FOREGROUND_NOTIFICATION_ID,
    DEFAULT_FOREGROUND_NOTIFICATION_UPDATE_INTERVAL,
    DOWNLOAD_NOTIFICATION_CHANNEL_ID,
    R.string.download_notification_channel_name,
    R.string.download_notification_channel_description
) {

    // --- Injected fields remain the same ---
    @Inject
    lateinit var downloadManagerInstance: DownloadManager

    @Inject
    lateinit var notificationHelper: DownloadNotificationHelper

    // --- All listener and scope code is REMOVED from here down to onDestroy ---

    companion object {
        private const val FOREGROUND_NOTIFICATION_ID = 2
        private const val DOWNLOAD_NOTIFICATION_CHANNEL_ID = "download_channel"
        private const val DOWNLOAD_WORK_MANAGER_JOB_ID = "holodex_download_job"
    }

    override fun getDownloadManager(): DownloadManager {
        // Just return the injected instance. No need to add/remove listeners.
        return downloadManagerInstance
    }

    override fun getScheduler(): Scheduler {
        return WorkManagerScheduler(this, DOWNLOAD_WORK_MANAGER_JOB_ID)
    }

    override fun getForegroundNotification(
        downloads: MutableList<Download>,
        notMetRequirements: Int
    ): Notification {
        return notificationHelper.buildProgressNotification(
            this, R.drawable.ic_notification_small, null, null, downloads, notMetRequirements
        )
    }

    // --- No need for onDestroy to cancel a job anymore ---
    override fun onDestroy() {
        super.onDestroy()
        Timber.d("HolodexDownloadService destroyed.")
    }
}

// File: java\com\example\holodex\service\MediaPlaybackService.kt
// File: java/com/example/holodex/service/MediaPlaybackService.kt
package com.example.holodex.service

import android.app.Notification
import android.app.PendingIntent
import android.content.Intent
import android.graphics.Bitmap
```

```
import android.graphics.Canvas
import android.graphics.drawable.BitmapDrawable
import android.graphics.drawable.Drawable
import androidx.core.content.ContextCompat
import androidx.core.graphics.createBitmap
import androidx.media3.common.Player
import androidx.media3.common.util.UnstableApi
import androidx.media3.session.MediaSession
import androidx.media3.session.MediaSession.ControllerInfo
import androidx.media3.session.MediaSessionService
import androidx.media3.ui.PlayerNotificationManager
import coil.imageLoader
import coil.request.ImageRequest
import com.example.holodex.R
import com.example.holodex.playback.util.playbackStateToString
import com.example.holodex.ui.MainActivity
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.SupervisorJob
import kotlinx.coroutines.cancel
import timber.log.Timber
import javax.inject.Inject

// --- Utility function for Bitmap conversion ---
fun Drawable?.toBitmapSafe(): Bitmap {
    if (this == null) {
        Timber.tag("BmpSafe").w("Drawable was null, returning 1x1 bitmap.")
        return createBitmap(1, 1, Bitmap.Config.ARGB_8888)
    }
    if (this is BitmapDrawable && this.bitmap != null) { return this.bitmap }
    val width = intrinsicWidth.coerceAtLeast(1)
    val height = intrinsicHeight.coerceAtLeast(1)
    val bitmap = createBitmap(width, height, Bitmap.Config.ARGB_8888)
    val canvas = Canvas(bitmap)
    this.setBounds(0, 0, canvas.width, canvas.height)
    this.draw(canvas)
    return bitmap
}

// --- Constants ---
const val CUSTOM_COMMAND_PREPARE_FROM_REQUEST = "com.example.holodex.PREPARE_FROM_REQUEST"
const val ARG_PLAYBACK_ITEMS_LIST = "playback_items_list"
const val ARG_START_INDEX = "start_index"
const val ARG_START_POSITION_SEC = "start_position_sec"
const val ARG_SHOULD_SHUFFLE = "should_shuffle_playlist"

private const val SERVICE_NOTIFICATION_ID = 123
private const val PLAYBACK_NOTIFICATION_CHANNEL_ID = "holodex_playback_channel_v3"
private const val SERVICE_TAG = "MediaPlaybackService"

@UnstableApi
@AndroidEntryPoint
class MediaPlaybackService : MediaSessionService() {
```

```
private var mediaSession: MediaSession? = null
private val serviceJob = SupervisorJob()
private val serviceScope = CoroutineScope(Dispatchers.Main.immediate + serviceJob)

@Inject lateinit var player: Player

private lateinit var notificationManager: PlayerNotificationManager
private var defaultNotificationBitmap: Bitmap? = null
private var isServiceInForeground = false

private val playerListener = PlayerStateListener()

override fun onCreate() {
    super.onCreate()
    Timber.tag(SERVICE_TAG).i("onCreate: Service creating...")

    defaultNotificationBitmap = (ContextCompat.getDrawable(this, R.drawable.ic_default_alb
        ?.toBitmapSafe()) ?: createBitmap(64, 64, Bitmap.Config.ARGB_8888)

    player.addListener(playerListener)

    initializeMediaSession()
    initializeNotificationManager()
    Timber.tag(SERVICE_TAG).i("onCreate: Service creation complete.")
}

private fun initializeMediaSession() {
    mediaSession = MediaSession.Builder(this, player)
        .setSessionActivity(getSingleTopActivityPendingIntent())
        .setId("holodex_music_media_session_${System.currentTimeMillis()}")
        .build()

    addSession(mediaSession!!)
}

private fun initializeNotificationManager() {
    val mediaDescriptionAdapter = ServiceMediaDescriptionAdapter()
    val notificationListener = ServiceNotificationListener()

    notificationManager = PlayerNotificationManager.Builder(
        this,
        SERVICE_NOTIFICATION_ID,
        PLAYBACK_NOTIFICATION_CHANNEL_ID
    )
        .setChannelNameResourceId(R.string.playback_notification_channel_name)
        .setChannelDescriptionResourceId(R.string.playback_notification_channel_descriptio
        .setMediaDescriptionAdapter(mediaDescriptionAdapter)
        .setNotificationListener(notificationListener)
        .setSmallIconResourceId(R.drawable.ic_stat_music_note)
        .build().apply {
            setUseRewindAction(false)
            setUseFastForwardAction(false)
            setUseNextAction(true)
            setUsePreviousAction(true)
            setColorized(true)
```

```
            setUseNextActionInCompactView(true)
            setUsePreviousActionInCompactView(true)
            setPlayer(player)
        }
    }

    private fun getSingleTopActivityPendingIntent(): PendingIntent {
        val intent = Intent(this, MainActivity::class.java).apply {
            flags = Intent.FLAG_ACTIVITY_SINGLE_TOP or Intent.FLAG_ACTIVITY_CLEAR_TOP
        }
        val flags = PendingIntent.FLAG_IMMUTABLE or PendingIntent.FLAG_UPDATE_CURRENT
        return PendingIntent.getActivity(this, 0, intent, flags)
    }

    @Suppress("OVERRIDE_DEPRECATION")
    override fun onUpdateNotification(session: MediaSession, startInForegroundRequired: Boolea
        // Deprecated but still called. Handled by PlayerNotificationManager.
    }

    private inner class PlayerStateListener : Player.Listener {
        override fun onPlaybackStateChanged(playbackState: Int) {
            Timber.tag(SERVICE_TAG).i("PlayerListener.onPlaybackStateChanged: %s", playbackSta
        }

        override fun onIsPlayingChanged(isPlaying: Boolean) {
            Timber.tag(SERVICE_TAG).i("PlayerListener.onIsPlayingChanged: %b", isPlaying)
        }
    }

    private inner class ServiceMediaDescriptionAdapter : PlayerNotificationManager.MediaDescri
        override fun getCurrentContentTitle(player: Player): CharSequence {
            return player.currentMediaItem?.mediaMetadata?.title ?: getString(R.string.unknown
        }

        override fun createCurrentContentIntent(player: Player): PendingIntent? {
            return getSingleTopActivityPendingIntent()
        }

        override fun getCurrentContentText(player: Player): CharSequence? {
            return player.currentMediaItem?.mediaMetadata?.artist
        }

        override fun getCurrentLargeIcon(player: Player, callback: PlayerNotificationManager.B
            val artworkUri = player.currentMediaItem?.mediaMetadata?.artworkUri
            if (artworkUri != null) {
                val request = ImageRequest.Builder(applicationContext)
                    .data(artworkUri)
                    .allowHardware(false)
                    .target(
                        onSuccess = { drawable -> callback.onBitmap(drawable.toBitmapSafe()) }
                        onError = { defaultNotificationBitmap?.let { callback.onBitmap(it) } }
                    ).build()
                applicationContext.imageLoader.enqueue(request)
                return defaultNotificationBitmap
            }
```

```
            return defaultNotificationBitmap
        }
    }

    private inner class ServiceNotificationListener : PlayerNotificationManager.NotificationLi
        override fun onNotificationPosted(notificationId: Int, notification: Notification, ong
            if (ongoing) {
                if (!isServiceInForeground) {
                    try {
                        startForeground(notificationId, notification)
                        isServiceInForeground = true
                    } catch (e: Exception) {
                        Timber.e(e, "CRITICAL EXCEPTION during startForeground().")
                    }
                }
            } else {
                if (isServiceInForeground) {
                    stopForeground(STOP_FOREGROUND_REMOVE)
                    isServiceInForeground = false
                }
            }
        }

        override fun onNotificationCancelled(notificationId: Int, dismissedByUser: Boolean) {
            if (dismissedByUser) {
                player.stop()
                stopSelf()
            }
            isServiceInForeground = false
        }
    }

    override fun onGetSession(controllerInfo: ControllerInfo): MediaSession = mediaSession!!



    override fun onTaskRemoved(rootIntent: Intent?) {
        if (!player.playWhenReady && player.playbackState != Player.STATE_BUFFERING) {
            stopSelf()
        }
    }

    override fun onDestroy() {
        player.removeListener(playerListener)
        if (::notificationManager.isInitialized) {
            notificationManager.setPlayer(null)
        }
        mediaSession?.release()
        mediaSession = null
        serviceScope.cancel()
        super.onDestroy()
    }
}

// File: java\com\example\holodex\ui\MainActivity.kt
```

```
package com.example.holodex.ui

import android.Manifest
import android.content.ComponentName
import android.content.pm.PackageManager
import android.os.Build
import android.os.Bundle
import android.widget.Toast
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.activity.result.contract.ActivityResultContracts
import androidx.core.content.ContextCompat
import androidx.core.splashscreen.SplashScreen.Companion.installSplashScreen
import androidx.core.view.WindowCompat
import androidx.media3.common.util.UnstableApi
import androidx.media3.exoplayer.ExoPlayer
import androidx.media3.session.MediaController
import androidx.media3.session.SessionCommand
import androidx.media3.session.SessionResult
import androidx.media3.session.SessionToken
import androidx.navigation.compose.rememberNavController
import com.example.holodex.R
import com.example.holodex.playback.domain.model.PlaybackItem
import com.example.holodex.service.ARG_PLAYBACK_ITEMS_LIST
import com.example.holodex.service.ARG_SHOULD_SHUFFLE
import com.example.holodex.service.ARG_START_INDEX
import com.example.holodex.service.ARG_START_POSITION_SEC
import com.example.holodex.service.CUSTOM_COMMAND_PREPARE_FROM_REQUEST
import com.example.holodex.service.MediaPlaybackService
import com.google.common.util.concurrent.ListenableFuture
import com.google.common.util.concurrent.MoreExecutors
import dagger.hilt.android.AndroidEntryPoint
import timber.log.Timber
import javax.inject.Inject

@UnstableApi
@AndroidEntryPoint
class MainActivity : ComponentActivity() {


    @Inject lateinit var player: ExoPlayer

    private var mediaController: MediaController? = null
    private lateinit var sessionToken: SessionToken

    private val requestPermissionLauncher =
        registerForActivityResult(ActivityResultContracts.RequestMultiplePermissions()) { perm
            permissions.entries.forEach {
                Timber.d("Permission [${it.key}] granted: ${it.value}")
            }
        }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        WindowCompat.setDecorFitsSystemWindows(window, false)
```

```
    installSplashScreen()

    sessionToken = SessionToken(this, ComponentName(this, MediaPlaybackService::class.java
    checkAndRequestPermissions()

    setContent {
        val navController = rememberNavController()
        MainScreenScaffold(
            navController = navController,
            activity = this,
            player = player // Pass it here
        )
    }
}

private fun checkAndRequestPermissions() {
    val permissionsToRequest = mutableListOf<String>()
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
        if (ContextCompat.checkSelfPermission(
                this,
                Manifest.permission.POST_NOTIFICATIONS
            ) != PackageManager.PERMISSION_GRANTED
        ) {
            permissionsToRequest.add(Manifest.permission.POST_NOTIFICATIONS)
        }
    }
    if (Build.VERSION.SDK_INT <= Build.VERSION_CODES.P) {
        if (ContextCompat.checkSelfPermission(
                this,
                Manifest.permission.WRITE_EXTERNAL_STORAGE
            ) != PackageManager.PERMISSION_GRANTED
        ) {
            permissionsToRequest.add(Manifest.permission.WRITE_EXTERNAL_STORAGE)
        }
    }
    if (permissionsToRequest.isNotEmpty()) {
        requestPermissionLauncher.launch(permissionsToRequest.toTypedArray())
    }
}

override fun onStart() {
    super.onStart()

}

override fun onStop() {
    super.onStop()

}

internal fun sendPlaybackRequestToService(
    items: List<PlaybackItem>,
    startIndex: Int,
    startPositionSec: Long,
    shouldShuffle: Boolean = false
```

```
    ) {
        if (items.isEmpty()) return

        val controller = mediaController ?: run {
            Toast.makeText(
                this,
                getString(R.string.error_player_service_not_ready),
                Toast.LENGTH_SHORT
            ).show()
            return
        }

        val commandArgs = Bundle().apply {
            putParcelableArrayList(ARG_PLAYBACK_ITEMS_LIST, ArrayList(items))
            putInt(ARG_START_INDEX, startIndex)
            putLong(ARG_START_POSITION_SEC, startPositionSec)
            putBoolean(ARG_SHOULD_SHUFFLE, shouldShuffle)
        }
        val command = SessionCommand(CUSTOM_COMMAND_PREPARE_FROM_REQUEST, Bundle.EMPTY)
        val resultFuture: ListenableFuture<SessionResult> =
            controller.sendCustomCommand(command, commandArgs)

        resultFuture.addListener({
            try {
                val result = resultFuture.get()
                if (result.resultCode != SessionResult.RESULT_SUCCESS) {
                    Timber.w("Custom playback command failed: ${result.resultCode}")
                }
            } catch (e: Exception) {
                Timber.e(e, "Error processing playback command result")
            }
        }, MoreExecutors.directExecutor())
    }
}

// File: java\com\example\holodex\ui\MainScreenScaffold.kt
package com.example.holodex.ui

import android.annotation.SuppressLint
import android.util.Log
import android.widget.Toast
import androidx.activity.ComponentActivity
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.ModalBottomSheet
import androidx.compose.material3.NavigationBar
import androidx.compose.material3.NavigationBarItem
import androidx.compose.material3.Text
import androidx.compose.material3.rememberModalBottomSheetState
import androidx.compose.runtime.Composable
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.graphics.Color
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.res.stringResource
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.media3.common.Player
import androidx.media3.exoplayer.ExoPlayer
import androidx.navigation.NavGraph.Companion.findStartDestination
import androidx.navigation.NavHostController
import androidx.navigation.compose.currentBackStackEntryAsState
import com.example.holodex.ui.composables.FullPlayerActions
import com.example.holodex.ui.composables.FullPlayerScreenContent
import com.example.holodex.ui.composables.MainScreenLayout
import com.example.holodex.ui.composables.MiniPlayerWithProgressBar
import com.example.holodex.ui.composables.PlaylistManagementDialogs
import com.example.holodex.ui.navigation.AppDestinations
import com.example.holodex.ui.navigation.HolodexNavHost
import com.example.holodex.ui.screens.navigation.BottomNavItem
import com.example.holodex.ui.theme.HolodexMusicTheme
import com.example.holodex.viewmodel.FavoritesViewModel
import com.example.holodex.viewmodel.PlaybackViewModel
import com.example.holodex.viewmodel.PlaylistManagementViewModel
import com.example.holodex.viewmodel.SettingsViewModel
import com.example.holodex.viewmodel.VideoListSideEffect
import com.example.holodex.viewmodel.VideoListViewModel
import kotlinx.coroutines.launch
import org.orbitmvi.orbit.compose.collectSideEffect

private const val TAG = "MainScreenScaffold"

@SuppressLint("UnstableApi")
@androidx.annotation.OptIn(androidx.media3.common.util.UnstableApi::class)
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun MainScreenScaffold(
    navController: NavHostController,
    activity: ComponentActivity,
    player: ExoPlayer
) {

    Log.d(TAG, "MainScreenScaffold: Composing")

    val coroutineScope = rememberCoroutineScope()
    val context = LocalContext.current

    // ViewModels
    val settingsViewModel: SettingsViewModel = hiltViewModel()
    val playbackViewModel: PlaybackViewModel = hiltViewModel()
    val playlistManagementViewModel: PlaylistManagementViewModel = hiltViewModel(activity)
    val videoListViewModel: VideoListViewModel = hiltViewModel(activity)
```

```
Log.d(TAG, "MainScreenScaffold: ViewModels created")

// UI State
var showFullPlayerSheet by remember { mutableStateOf(false) }
val fullPlayerSheetState = rememberModalBottomSheetState(skipPartiallyExpanded = true)

// Navigation State
val navBackStackEntry by navController.currentBackStackEntryAsState()
val currentRoute = navBackStackEntry?.destination?.route

// --- Orbit Side Effects ---
videoListViewModel.collectSideEffect { sideEffect ->
    when (sideEffect) {
        is VideoListSideEffect.NavigateTo -> {
            when (val destination = sideEffect.destination) {
                is VideoListViewModel.NavigationDestination.VideoDetails -> {
                    navController.navigate(AppDestinations.videoDetailRoute(destination.vi
                }

                is VideoListViewModel.NavigationDestination.HomeScreenWithSearch -> {
                    navController.navigate(AppDestinations.HOME_ROUTE) {
                        popUpTo(navController.graph.startDestinationId) { saveState = true
                        launchSingleTop = true
                    }
                    videoListViewModel.setSearchActive(true)
                }
            }
        }

        is VideoListSideEffect.ShowToast -> {
            Toast.makeText(context, sideEffect.message, Toast.LENGTH_SHORT).show()
        }
    }
}



HolodexMusicTheme(settingsViewModel = settingsViewModel) {
    MainScreenLayout(
        modifier = Modifier.fillMaxSize(),
        bottomBar = {
            Column {
                // The MiniPlayer logic remains the same
                MiniPlayerWithProgressBar(
                    playbackViewModel = playbackViewModel,
                    onTapped = { showFullPlayerSheet = true }
                )
                NavigationBar {
                    val navItems = listOf(
                        BottomNavItem.Discover,
                        BottomNavItem.Browse,
                        BottomNavItem.Library,
                        BottomNavItem.Downloads
                    )
                    navItems.forEach { item ->
```

```
                    val isSelected = currentRoute == item.route
                    NavigationBarItem(
                        icon = { Icon(item.icon, contentDescription = null) },
                        label = { Text(stringResource(item.titleResId)) },
                        selected = isSelected,
                        onClick = {
                            if (item.route == AppDestinations.DISCOVERY_ROUTE &&
                                navController.graph.startDestinationId == navControlle
                                    AppDestinations.DISCOVERY_ROUTE
                                )?.id
                            ) {
                                navController.navigate(item.route) {
                                    popUpTo(0) { inclusive = true }
                                    launchSingleTop = true
                                }
                            } else {
                                navController.navigate(item.route) {
                                    popUpTo(navController.graph.findStartDestination()
                                        saveState = true
                                    }
                                    launchSingleTop = true
                                    restoreState = true
                                }
                            }
                        }
                    )
                }
            }
            Log.d(TAG, "bottomBar Column: NavigationBar composed")
        }
    }
) { dynamicPadding ->
    // dynamicPadding now contains the EXACT height of (MiniPlayer + NavBar)

    Box(modifier = Modifier.fillMaxSize()) {
        HolodexNavHost(
            navController = navController,
            videoListViewModel = videoListViewModel,
            playlistManagementViewModel = playlistManagementViewModel,
            activity = activity,
            // Pass this padding down to your screens!
            contentPadding = dynamicPadding
        )
    }
}

if (showFullPlayerSheet) {
    ModalBottomSheet(
        onDismissRequest = { showFullPlayerSheet = false },
        sheetState = fullPlayerSheetState,
        containerColor = Color.Transparent,
        shape = RoundedCornerShape(0),
        scrimColor = Color.Black.copy(alpha = 0.6f),
        dragHandle = null
    ) {
```

```
                FullPlayerScreenDestination(
                    player = player,
                    navController = navController,
                    onNavigateUp = {
                        coroutineScope.launch { fullPlayerSheetState.hide() }.invokeOnCompleti
                            if (!fullPlayerSheetState.isVisible) showFullPlayerSheet = false
                        }
                    }
                )
            }
        }

        PlaylistManagementDialogs(playlistManagementViewModel)
    }
}

@androidx.annotation.OptIn(androidx.media3.common.util.UnstableApi::class)
@Composable
private fun FullPlayerScreenDestination(
    player: Player?,
    navController: NavHostController,
    onNavigateUp: () -> Unit,
    modifier: Modifier = Modifier
) {
    val playbackViewModel: PlaybackViewModel = hiltViewModel()
    val favoritesViewModel: FavoritesViewModel = hiltViewModel()
    val playlistManagementViewModel: PlaylistManagementViewModel = hiltViewModel()
    val videoListViewModel: VideoListViewModel = hiltViewModel()
    val context = LocalContext.current

    val playerActions = remember(
        playbackViewModel,
        favoritesViewModel,
        videoListViewModel,
        playlistManagementViewModel
    ) {
        FullPlayerActions(
            onNavigateUp = onNavigateUp,
            onTogglePlayPause = { playbackViewModel.togglePlayPause() },
            onSeekTo = { positionSec -> playbackViewModel.seekTo(positionSec) },
            onSkipToNext = { playbackViewModel.skipToNext() },
            onSkipToPrevious = { playbackViewModel.skipToPrevious() },
            onToggleRepeatMode = { playbackViewModel.toggleRepeatMode() },
            onToggleShuffleMode = { playbackViewModel.toggleShuffleMode() },
            onPlayQueueItemAtIndex = { index -> playbackViewModel.playQueueItemAtIndex(index)
            onReorderQueueItem = { from, to -> playbackViewModel.reorderQueueItem(from, to) },
            onRemoveQueueItem = { index -> playbackViewModel.removeItemFromQueue(index) },
            onClearQueue = { playbackViewModel.clearCurrentQueue() },
            onToggleLike = { playbackItem -> favoritesViewModel.toggleLike(playbackItem) },
            onFindArtist = { channelId -> videoListViewModel.setBrowseContextAndNavigate(chann
            onOpenAudioSettings = { audioSessionId ->
                Toast.makeText(context, "Audio FX Session ID: $audioSessionId", Toast.LENGTH_S
                    .show()
            },
            onAddToPlaylist = { playbackItem ->
```

```
                playlistManagementViewModel.prepareItemForPlaylistAdditionFromPlaybackItem(
                    playbackItem
                )
            }
        )
    }

    FullPlayerScreenContent(
        player = player,
        navController = navController,
        actions = playerActions,
        modifier = modifier
    )
}

// File: java\com\example\holodex\ui\composables\ApiKeyInputScreen.kt
package com.example.holodex.ui.composables

import android.widget.Toast
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.foundation.text.KeyboardActions
import androidx.compose.foundation.text.KeyboardOptions
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.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.platform.LocalFocusManager
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.input.ImeAction
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.text.input.PasswordVisualTransformation
import androidx.compose.ui.unit.dp
import androidx.hilt.navigation.compose.hiltViewModel
import com.example.holodex.R
import com.example.holodex.viewmodel.ApiKeySaveResult
import com.example.holodex.viewmodel.SettingsViewModel
import org.orbitmvi.orbit.compose.collectAsState // <--- Import

@Composable
fun ApiKeyInputScreen(
    settingsViewModel: SettingsViewModel = hiltViewModel(),
    onApiKeySavedSuccessfully: () -> Unit,
    modifier: Modifier = Modifier,
) {
```

```
// FIX: Collect state first
val state by settingsViewModel.collectAsState()

// FIX: Access properties from state
val currentApiKey = state.currentApiKey
val apiKeySaveResult = state.apiKeySaveResult

var apiKeyInputText by remember(currentApiKey) { mutableStateOf(currentApiKey) }

val context = LocalContext.current
val focusManager = LocalFocusManager.current

LaunchedEffect(apiKeySaveResult) {
    when (val result = apiKeySaveResult) {
        is ApiKeySaveResult.Success -> {
            Toast.makeText(context, R.string.toast_api_key_saved, Toast.LENGTH_SHORT).show
            onApiKeySavedSuccessfully()
            settingsViewModel.resetApiKeySaveResult()
        }
        is ApiKeySaveResult.Empty -> {
            Toast.makeText(context, R.string.toast_api_key_empty, Toast.LENGTH_SHORT).show
            settingsViewModel.resetApiKeySaveResult()
        }
        is ApiKeySaveResult.Error -> {
            Toast.makeText(context, result.message, Toast.LENGTH_LONG).show()
            settingsViewModel.resetApiKeySaveResult()
        }
        is ApiKeySaveResult.Idle -> { /* Do nothing */ }
    }
}

Column(
    modifier = modifier.padding(bottom = 8.dp),
    verticalArrangement = Arrangement.spacedBy(8.dp)
) {
    OutlinedTextField(
        value = apiKeyInputText,
        onValueChange = { apiKeyInputText = it },
        label = { Text(stringResource(id = R.string.hint_api_key)) },
        modifier = Modifier.fillMaxWidth(),
        singleLine = true,
        visualTransformation = PasswordVisualTransformation(),
        keyboardOptions = KeyboardOptions.Default.copy(
            keyboardType = KeyboardType.Password,
            imeAction = ImeAction.Done
        ),
        keyboardActions = KeyboardActions(onDone = {
            focusManager.clearFocus()
            settingsViewModel.saveApiKey(apiKeyInputText)
        })
    )
    Button(
        onClick = {
            focusManager.clearFocus()
            settingsViewModel.saveApiKey(apiKeyInputText)
```

```
            },
            modifier = Modifier.align(Alignment.End)
        ) {
            Text(stringResource(id = R.string.button_save_key))
        }
    }
}

// File: java\com\example\holodex\ui\composables\CarouselShelf.kt
// File: java/com/example/holodex/ui/composables/CarouselShelf.kt

package com.example.holodex.ui.composables

import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.width
import androidx.compose.foundation.lazy.LazyRow
import androidx.compose.foundation.lazy.items
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.ErrorOutline
import androidx.compose.material3.Icon
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import com.example.holodex.viewmodel.state.UiState

@Composable
fun <T> CarouselShelf(
    title: String,
    uiState: UiState<List<T>>,
    itemContent: @Composable (T) -> Unit,
    modifier: Modifier = Modifier,
    actionContent: (@Composable () -> Unit)? = null
) {
    Column(modifier = modifier) {
        Row(
            modifier = Modifier
                .fillMaxWidth()
                .padding(horizontal = 16.dp),
            verticalAlignment = Alignment.CenterVertically,
            horizontalArrangement = Arrangement.SpaceBetween
        ) {
            Text(text = title, style = MaterialTheme.typography.titleLarge)
            actionContent?.invoke()
```

```
        }

        when (uiState) {
            is UiState.Loading -> {
                LazyRow(
                    modifier = Modifier.fillMaxWidth(),
                    contentPadding = PaddingValues(horizontal = 16.dp),
                    horizontalArrangement = Arrangement.spacedBy(12.dp)
                ) {
                    items(5) {
                        Box(modifier = Modifier.width(140.dp).height(180.dp).background(Materi
                    }
                }
            }
            is UiState.Success -> {
                if (uiState.data.isEmpty()) {
                    Text(
                        text = "No content available.",
                        modifier = Modifier.padding(16.dp),
                        style = MaterialTheme.typography.bodyMedium,
                        color = MaterialTheme.colorScheme.onSurfaceVariant
                    )
                } else {
                    LazyRow(
                        modifier = Modifier.fillMaxWidth(),
                        contentPadding = PaddingValues(horizontal = 16.dp),
                        horizontalArrangement = Arrangement.spacedBy(12.dp)
                    ) {
                        items(uiState.data) { item ->
                            itemContent(item)
                        }
                    }
                }
            }
            is UiState.Error -> {
                Row(
                    modifier = Modifier.padding(16.dp),
                    verticalAlignment = Alignment.CenterVertically
                ) {
                    Icon(Icons.Default.ErrorOutline, contentDescription = "Error", tint = Mate
                    Spacer(Modifier.width(8.dp))
                    Text(
                        text = uiState.message,
                        color = MaterialTheme.colorScheme.error,
                        style = MaterialTheme.typography.bodyMedium
                    )
                }
            }
        }
    }
}

// File: java\com\example\holodex\ui\composables\ChannelCard.kt
package com.example.holodex.ui.composables
```

```
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.layout.width
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.material3.Card
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.remember
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.text.style.TextOverflow
import androidx.compose.ui.unit.dp
import coil.compose.AsyncImage
import coil.request.ImageRequest
import com.example.holodex.R
import com.example.holodex.data.model.discovery.DiscoveryChannel
import com.example.holodex.util.ArtworkResolver

@Composable
fun ChannelCard(
    channel: DiscoveryChannel,
    onChannelClicked: (String) -> Unit,
    modifier: Modifier = Modifier
) {
    // --- START OF IMPLEMENTATION ---
    val artworkUrl = remember(channel.id, channel.photoUrl) {
        // Prioritize the photoUrl if it exists, otherwise construct it from the ID.
        channel.photoUrl?.takeIf { it.isNotBlank() }
            ?: ArtworkResolver.getChannelPhotoUrl(channel.id)
    }
    // --- END OF IMPLEMENTATION ---

    Card(
        modifier = modifier
            .width(140.dp)
            .clickable { onChannelClicked(channel.id) }
    ) {
        Column(
            modifier = Modifier.padding(12.dp),
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.spacedBy(8.dp)
        ) {
            AsyncImage(
                // --- MODIFICATION: Use the new artworkUrl variable ---
                model = ImageRequest.Builder(LocalContext.current)
                    .data(artworkUrl)
                    .placeholder(R.drawable.ic_placeholder_image)
```

```
                    .error(R.drawable.ic_error_image)
                    .crossfade(true).build(),
                contentDescription = "Avatar for ${channel.name}",
                modifier = Modifier.size(96.dp).clip(CircleShape),
                contentScale = ContentScale.Crop
            )
            Text(
                text = channel.englishName ?: channel.name,
                style = MaterialTheme.typography.titleSmall,
                textAlign = TextAlign.Center,
                maxLines = 2,
                overflow = TextOverflow.Ellipsis
            )
        }
    }
}

// File: java\com\example\holodex\ui\composables\CustomPagedUnifiedList.kt
@file:kotlin.OptIn(ExperimentalMaterial3Api::class)

package com.example.holodex.ui.composables

import androidx.annotation.OptIn
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.PaddingValues
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.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.LazyListState
import androidx.compose.foundation.lazy.items
import androidx.compose.material3.CircularProgressIndicator
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.material3.pulltorefresh.PullToRefreshBox
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.derivedStateOf
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.res.stringResource
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.media3.common.util.UnstableApi
import androidx.navigation.NavController
import com.example.holodex.R
import com.example.holodex.viewmodel.FavoritesViewModel
import com.example.holodex.viewmodel.PlaylistManagementViewModel
import com.example.holodex.viewmodel.UnifiedDisplayItem
```

```
import com.example.holodex.viewmodel.VideoListViewModel
import timber.log.Timber

@OptIn(UnstableApi::class, ExperimentalMaterial3Api::class) // Fixed: Removed duplicate annota
@Composable
fun CustomPagedUnifiedList(
    listKeyPrefix: String,
    items: List<UnifiedDisplayItem>,
    listState: LazyListState,
    onItemClicked: (UnifiedDisplayItem) -> Unit,
    videoListViewModel: VideoListViewModel,
    favoritesViewModel: FavoritesViewModel,
    playlistManagementViewModel: PlaylistManagementViewModel,
    navController: NavController,
    isLoadingMore: Boolean,
    endOfList: Boolean,
    isRefreshing: Boolean,
    onRefresh: () -> Unit,
    onLoadMore: () -> Unit,
    modifier: Modifier = Modifier,
    contentPadding: PaddingValues = PaddingValues(bottom = 80.dp),
    header: (@Composable () -> Unit)? = null,
) {
    // Track initial load state to fix scroll-to-end bug
    var hasPerformedInitialScroll by remember { mutableStateOf(false) }

    // Fix for scroll-to-end bug: Ensure we start at top after initial load
    LaunchedEffect(items.size) {
        if (items.isNotEmpty() && !hasPerformedInitialScroll) {
            // Only scroll to top if we're not already there (avoids unnecessary animation)
            if (listState.firstVisibleItemIndex != 0 || listState.firstVisibleItemScrollOffset
                listState.scrollToItem(0)
            }
            hasPerformedInitialScroll = true
        } else if (items.isEmpty()) {
            // Reset flag if list becomes empty (e.g., after refresh)
            hasPerformedInitialScroll = false
        }
    }

    // Improved load-more logic with better performance and stability
    val shouldLoadMore by remember {
        derivedStateOf {
            // Early exit conditions for better performance
            if (isLoadingMore || endOfList || items.isEmpty()) {
                false
            } else {
                val layoutInfo = listState.layoutInfo
                val visibleItemsInfo = layoutInfo.visibleItemsInfo

                // More robust check
                if (visibleItemsInfo.isEmpty()) {
                    false
                } else {
                    val lastVisibleItem = visibleItemsInfo.last()
```

```
                val threshold = 3
                val totalItems = layoutInfo.totalItemsCount

                // Account for header in total count if present
                val adjustedTotalItems = if (header != null) totalItems - 1 else totalItem

                lastVisibleItem.index >= adjustedTotalItems - 1 - threshold
            }
        }
    }
}

// More efficient LaunchedEffect that only triggers when actually needed
LaunchedEffect(shouldLoadMore, isLoadingMore, endOfList) {
    if (shouldLoadMore && !isLoadingMore && !endOfList) {
        Timber.i("CustomPagedUnifiedList ($listKeyPrefix): >>> LOAD MORE UI TRIGGERED <<<"
        onLoadMore()
    }
}

PullToRefreshBox(
    isRefreshing = isRefreshing,
    onRefresh = {
        // Reset initial scroll flag on refresh
        hasPerformedInitialScroll = false
        onRefresh()
    },
    modifier = modifier
) {
    LazyColumn(
        state = listState,
        modifier = Modifier.fillMaxSize(), // Removed redundant modifier parameter
        contentPadding = contentPadding
    ) {
        header?.let {
            item(key = "${listKeyPrefix}_header") { it() }
        }

        items(
            items = items,
            key = { item -> item.stableId }
        ) { item ->
            UnifiedListItem(
                item = item,
                onItemClicked = { onItemClicked(item) }, // Fixed formatting
                videoListViewModel = videoListViewModel,
                favoritesViewModel = favoritesViewModel,
                playlistManagementViewModel = playlistManagementViewModel,
                navController = navController,
            )
        }

        item(key = "${listKeyPrefix}_footer") {
            if (isLoadingMore) {
                Box(
```

```
                        modifier = Modifier
                            .fillMaxWidth()
                            .padding(vertical = 16.dp),
                        contentAlignment = Alignment.Center
                    ) {
                        CircularProgressIndicator(modifier = Modifier.size(36.dp))
                    }
                } else if (endOfList && items.isNotEmpty()) {
                    Text(
                        text = stringResource(R.string.message_youve_reached_the_end),
                        modifier = Modifier
                            .fillMaxWidth()
                            .padding(16.dp), // Fixed chaining
                        textAlign = TextAlign.Center,
                        style = MaterialTheme.typography.bodySmall,
                        color = MaterialTheme.colorScheme.onSurfaceVariant
                    )
                }
            }
        }
    }
}

// File: java\com\example\holodex\ui\composables\FullPlayerScreen.kt
// File: java/com/example/holodex/ui/composables/FullPlayerScreen.kt
package com.example.holodex.ui.composables

import android.content.Context
import android.media.AudioManager
import android.widget.Toast
import androidx.compose.animation.AnimatedContent
import androidx.compose.animation.AnimatedVisibility
import androidx.compose.animation.animateColorAsState
import androidx.compose.animation.core.Animatable
import androidx.compose.animation.core.EaseOutCubic
import androidx.compose.animation.core.animateDpAsState
import androidx.compose.animation.core.tween
import androidx.compose.animation.fadeIn
import androidx.compose.animation.fadeOut
import androidx.compose.animation.scaleIn
import androidx.compose.animation.scaleOut
import androidx.compose.animation.togetherWith
import androidx.compose.foundation.BorderStroke
import androidx.compose.foundation.ExperimentalFoundationApi
import androidx.compose.foundation.background
import androidx.compose.foundation.combinedClickable
import androidx.compose.foundation.gestures.detectDragGestures
import androidx.compose.foundation.interaction.MutableInteractionSource
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.aspectRatio
```

```
import androidx.compose.foundation.layout.fillMaxHeight
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.navigationBarsPadding
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.layout.systemBarsPadding
import androidx.compose.foundation.layout.width
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.itemsIndexed
import androidx.compose.foundation.lazy.rememberLazyListState
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.foundation.verticalScroll
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.automirrored.filled.ArrowBack
import androidx.compose.material.icons.automirrored.filled.PlaylistAdd
import androidx.compose.material.icons.automirrored.filled.PlaylistPlay
import androidx.compose.material.icons.automirrored.filled.VolumeDown
import androidx.compose.material.icons.automirrored.filled.VolumeMute
import androidx.compose.material.icons.automirrored.filled.VolumeUp
import androidx.compose.material.icons.automirrored.outlined.QueueMusic
import androidx.compose.material.icons.filled.Delete
import androidx.compose.material.icons.filled.DragHandle
import androidx.compose.material.icons.filled.Equalizer
import androidx.compose.material.icons.filled.Favorite
import androidx.compose.material.icons.filled.FavoriteBorder
import androidx.compose.material.icons.filled.MoreVert
import androidx.compose.material.icons.filled.MusicNote
import androidx.compose.material.icons.filled.PersonSearch
import androidx.compose.material.icons.filled.PlayArrow
import androidx.compose.material.icons.filled.PlaylistRemove
import androidx.compose.material.icons.filled.TextFields
import androidx.compose.material3.CircularProgressIndicator
import androidx.compose.material3.DropdownMenu
import androidx.compose.material3.DropdownMenuItem
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.FilledTonalIconButton
import androidx.compose.material3.HorizontalDivider
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.IconButtonDefaults
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.ModalBottomSheet
import androidx.compose.material3.Slider
import androidx.compose.material3.SliderDefaults
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.material3.rememberModalBottomSheetState
import androidx.compose.runtime.Composable
import androidx.compose.runtime.Immutable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.getValue
```

```
import androidx.compose.runtime.mutableFloatStateOf
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.draw.clip
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.graphicsLayer
import androidx.compose.ui.hapticfeedback.HapticFeedbackType
import androidx.compose.ui.input.pointer.pointerInput
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.layout.onSizeChanged
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.platform.LocalDensity
import androidx.compose.ui.platform.LocalHapticFeedback
import androidx.compose.ui.res.pluralStringResource
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.text.style.TextOverflow
import androidx.compose.ui.unit.IntSize
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.lifecycle.compose.collectAsStateWithLifecycle
import androidx.media3.common.Player
import androidx.media3.common.util.UnstableApi
import androidx.navigation.NavHostController
import coil.compose.AsyncImage
import coil.request.ImageRequest
import com.example.holodex.R
import com.example.holodex.playback.domain.model.PlaybackItem
import com.example.holodex.playback.util.formatDurationSecondsToString
import com.example.holodex.ui.navigation.AppDestinations
import com.example.holodex.util.ThumbnailQuality
import com.example.holodex.util.generateArtworkUrlList
import com.example.holodex.viewmodel.FavoritesViewModel
import com.example.holodex.viewmodel.FullPlayerViewModel
import com.example.holodex.viewmodel.PlaybackViewModel
import com.example.holodex.viewmodel.rememberFullPlayerArtworkState
import com.example.holodex.viewmodel.rememberFullPlayerCurrentItemState
import com.example.holodex.viewmodel.rememberFullPlayerLoadingState
import com.example.holodex.viewmodel.rememberFullPlayerProgressState
import com.example.holodex.viewmodel.rememberFullPlayerQueueInfoState
import kotlinx.coroutines.delay
import kotlinx.coroutines.launch
import org.orbitmvi.orbit.compose.collectAsState
import sh.calvin.reorderable.ReorderableItem
import sh.calvin.reorderable.rememberReorderableLazyListState
import kotlin.math.absoluteValue
import kotlin.math.roundToInt
```

```
private object FullPlayerDefaults {
    const val VOLUME_SWIPE_SENSITIVITY_FACTOR = 0.005f
    const val HORIZONTAL_SWIPE_THRESHOLD_PX = 50f
    val ArtworkShape = RoundedCornerShape(16.dp)
    const val ARTWORK_ANIMATION_DURATION_MS = 250
    const val VOLUME_SLIDER_AUTO_HIDE_DELAY_MS = 3000L
    const val QUEUE_SHEET_MAX_HEIGHT_FACTOR = 0.7f
}

@Immutable
data class FullPlayerActions(
    val onNavigateUp: () -> Unit,
    val onTogglePlayPause: () -> Unit,
    val onSeekTo: (Long) -> Unit,
    val onSkipToNext: () -> Unit,
    val onSkipToPrevious: () -> Unit,
    val onToggleRepeatMode: () -> Unit,
    val onToggleShuffleMode: () -> Unit,
    val onPlayQueueItemAtIndex: (Int) -> Unit,
    val onReorderQueueItem: (from: Int, to: Int) -> Unit,
    val onRemoveQueueItem: (index: Int) -> Unit,
    val onClearQueue: () -> Unit,
    val onToggleLike: (PlaybackItem) -> Unit,
    val onFindArtist: (channelId: String) -> Unit,
    val onOpenAudioSettings: (audioSessionId: Int) -> Unit,
    val onAddToPlaylist: (PlaybackItem) -> Unit,
)

@UnstableApi
@OptIn(ExperimentalMaterial3Api::class, ExperimentalFoundationApi::class)
@Composable
internal fun FullPlayerScreenContent(
    navController: NavHostController,
    player: Player?,
    actions: FullPlayerActions,
    modifier: Modifier = Modifier
) {
    val coroutineScope = rememberCoroutineScope()
    val context = LocalContext.current

    // ViewModels
    val fullPlayerViewModel: FullPlayerViewModel = hiltViewModel()
    val playbackViewModel: PlaybackViewModel = hiltViewModel()
    val favoritesViewModel: FavoritesViewModel = hiltViewModel()

    // State from ViewModels
    val isRadioMode by playbackViewModel.isRadioModeActive.collectAsStateWithLifecycle()
    val artworkUri by rememberFullPlayerArtworkState(playbackViewModel.uiState)
    val currentItem by rememberFullPlayerCurrentItemState(playbackViewModel.uiState)
    val isLoading by rememberFullPlayerLoadingState(playbackViewModel.uiState)
    val queueInfo by rememberFullPlayerQueueInfoState(playbackViewModel.uiState)
    val (queueItems, currentIndexInQueue, isQueueNotEmpty) = queueInfo
    val uiState by playbackViewModel.uiState.collectAsStateWithLifecycle()
    val repeatMode = uiState.repeatMode
    val shuffleMode = uiState.shuffleMode
```

```
val progress by rememberFullPlayerProgressState(playbackViewModel.uiState)
val favoritesState by favoritesViewModel.collectAsState()
val dynamicTheme by fullPlayerViewModel.dynamicTheme.collectAsStateWithLifecycle()

// Local state
var showVolumeSlider by remember { mutableStateOf(false) }
var volumeSliderValue by remember { mutableFloatStateOf(0.7f) }
var showLyricsView by remember { mutableStateOf(false) }
var showQueueSheet by remember { mutableStateOf(false) }

// Animation state
val artworkScale = remember { Animatable(1f) }
val artworkAlpha = remember { Animatable(1f) }

// Animated colors based on dynamic theme
val animatedPrimaryColor by animateColorAsState(
    dynamicTheme.primary,
    label = "animated_primary_color",
    animationSpec = tween(1200)
)
val animatedOnPrimaryColor by animateColorAsState(
    dynamicTheme.onPrimary,
    label = "animated_on_primary_color",
    animationSpec = tween(500)
)

// Audio and haptic feedback
val audioManager = remember { context.getSystemService(Context.AUDIO_SERVICE) as AudioMana
val maxVolume = remember { audioManager.getStreamMaxVolume(AudioManager.STREAM_MUSIC) }
val haptic = LocalHapticFeedback.current

// Computed states
val isCurrentItemLiked = remember(currentItem, favoritesState.likedItemsMap) {
    currentItem?.let { pbItem ->
        val likeId = pbItem.id
        favoritesState.likedItemsMap.containsKey(likeId)
    } == true
}

val queueSheetState = rememberModalBottomSheetState(skipPartiallyExpanded = true)

// Update theme when artwork changes
LaunchedEffect(artworkUri) {
    fullPlayerViewModel.updateThemeFromArtwork(artworkUri)
}

// Artwork transition animations
LaunchedEffect(currentItem?.id) {
    if (currentItem != null) {
        coroutineScope.launch {
            artworkAlpha.animateTo(0.5f, tween(150))
            artworkAlpha.animateTo(1f, tween(FullPlayerDefaults.ARTWORK_ANIMATION_DURATION
        }
        coroutineScope.launch {
            artworkScale.animateTo(0.95f, tween(150))
```

```
            artworkScale.animateTo(1f, tween(FullPlayerDefaults.ARTWORK_ANIMATION_DURATION
        }
    }
}

// Auto-hide volume slider
LaunchedEffect(showVolumeSlider) {
    if (showVolumeSlider) {
        delay(FullPlayerDefaults.VOLUME_SLIDER_AUTO_HIDE_DELAY_MS)
        showVolumeSlider = false
    }
}

Box(modifier = modifier.fillMaxSize()) {
    // Background with dynamic theme
    SimpleProcessedBackground(
        artworkUri = artworkUri,
        dynamicColor = dynamicTheme.primary
    )
    Surface(
        modifier = Modifier.fillMaxSize(),
        color = animatedPrimaryColor.copy(alpha = 0.45f)
    ) {}

    Column(
        modifier = Modifier
            .fillMaxSize()
            .systemBarsPadding()
    ) {
        PlayerTopBar(
            isLiked = isCurrentItemLiked,
            isShowingLyrics = showLyricsView,
            queueNotEmpty = isQueueNotEmpty,
            isItemLoaded = currentItem != null,
            iconTint = animatedOnPrimaryColor,
            onNavigateUp = actions.onNavigateUp,
            onLikeToggle = { currentItem?.let { actions.onToggleLike(it) } },
            onQueueClick = { coroutineScope.launch { showQueueSheet = true } },
            onToggleLyrics = { showLyricsView = !showLyricsView },
            actions = actions,
            currentItem = currentItem,
            navController = navController,
            playbackViewModel = playbackViewModel
        )

        Box(
            modifier = Modifier
                .weight(1f)
                .fillMaxWidth()
        ) {
            if (showLyricsView) {
                LyricsView(
                    currentItem = currentItem,
                    textColor = animatedOnPrimaryColor,
                    modifier = Modifier.fillMaxSize().padding(16.dp)
```

```
            )
        } else {
            PlayerContent(
                currentItem = currentItem,
                trackInfoColor = animatedOnPrimaryColor,
                isLoading = isLoading,
                artworkScale = artworkScale.value,
                artworkAlpha = artworkAlpha.value,
                onHorizontalSwipe = { dragAmount ->
                    if (dragAmount < -FullPlayerDefaults.HORIZONTAL_SWIPE_THRESHOLD_PX
                        haptic.performHapticFeedback(HapticFeedbackType.TextHandleMove
                        actions.onSkipToNext()
                    } else if (dragAmount > FullPlayerDefaults.HORIZONTAL_SWIPE_THRESH
                        haptic.performHapticFeedback(HapticFeedbackType.TextHandleMove
                        actions.onSkipToPrevious()
                    }
                },
                onVerticalSwipe = { deltaY ->
                    val currentVolume = audioManager.getStreamVolume(AudioManager.STRE
                    val newVolume = (currentVolume - deltaY * FullPlayerDefaults.VOLUM
                        .roundToInt().coerceIn(0, maxVolume)
                    if (newVolume != currentVolume) {
                        audioManager.setStreamVolume(AudioManager.STREAM_MUSIC, newVol
                        volumeSliderValue = newVolume.toFloat() / maxVolume
                        if (!showVolumeSlider) showVolumeSlider = true
                        haptic.performHapticFeedback(HapticFeedbackType.TextHandleMove
                    }
                },
                onDoubleTap = {
                    haptic.performHapticFeedback(HapticFeedbackType.LongPress)
                    actions.onTogglePlayPause()
                },
                modifier = Modifier.fillMaxSize()
            )
        }
    }

    player?.let { validPlayer ->
        AnimatedVisibility(visible = showVolumeSlider && !showLyricsView) {
            VolumeSlider(
                value = volumeSliderValue,
                onValueChange = {
                    volumeSliderValue = it
                    val newVolumeInt = (it * maxVolume).roundToInt()
                    audioManager.setStreamVolume(AudioManager.STREAM_MUSIC, newVolumeI
                },
                thumbColor = animatedPrimaryColor,
                activeTrackColor = animatedOnPrimaryColor
            )
        }

        Media3PlayerControls(
            player = validPlayer,
            progress = progress,
            shuffleMode = shuffleMode,
```

```
                    repeatMode = repeatMode,
                    onSeek = { positionSec -> actions.onSeekTo(positionSec) },
                    onScrubbingChange = { isScrubbing -> playbackViewModel.setScrubbing(isScru
                    primaryColor = animatedPrimaryColor,
                    onPrimaryColor = animatedOnPrimaryColor,
                    onPlayPause = actions.onTogglePlayPause,
                    onSkipPrevious = actions.onSkipToPrevious,
                    onSkipNext = actions.onSkipToNext,
                    onToggleShuffle = actions.onToggleShuffleMode,
                    isRadioMode = isRadioMode,
                    onToggleRepeat = actions.onToggleRepeatMode
                )
            }
        }

        if (showQueueSheet) {
            ModalBottomSheet(
                onDismissRequest = { showQueueSheet = false },
                sheetState = queueSheetState
            ) {
                Box(modifier = Modifier.fillMaxHeight(FullPlayerDefaults.QUEUE_SHEET_MAX_HEIGH
                    QueueSheetContent(
                        queueItems = queueItems,
                        currentIndex = currentIndexInQueue,
                        isRadioMode = isRadioMode,
                        onPlayQueueItem = actions.onPlayQueueItemAtIndex,
                        onClearQueue = {
                            actions.onClearQueue()
                            coroutineScope.launch { queueSheetState.hide(); showQueueSheet = f
                        },
                        onMoveItem = actions.onReorderQueueItem,
                        onRemoveItem = actions.onRemoveQueueItem
                    )
                }
            }
        }
    }
}

@Composable
private fun PlayerTopBar(
    isLiked: Boolean,
    isShowingLyrics: Boolean,
    queueNotEmpty: Boolean,
    isItemLoaded: Boolean,
    onNavigateUp: () -> Unit,
    onLikeToggle: () -> Unit,
    onQueueClick: () -> Unit,
    onToggleLyrics: () -> Unit,
    iconTint: Color,
    actions: FullPlayerActions,
    currentItem: PlaybackItem?,
    navController: NavHostController,
    playbackViewModel: PlaybackViewModel
) {
```

```
    var showMoreOptionsDropdown by remember { mutableStateOf(false) }

    Row(
        modifier = Modifier
            .fillMaxWidth()
            .padding(horizontal = 4.dp, vertical = 8.dp),
        verticalAlignment = Alignment.CenterVertically,
    ) {
        IconButton(onClick = onNavigateUp) {
            Icon(Icons.AutoMirrored.Filled.ArrowBack, stringResource(R.string.content_desc_nav
        }
        Spacer(Modifier.weight(1f))

        IconButton(onClick = onToggleLyrics) {
            Icon(
                imageVector = if (isShowingLyrics) Icons.Filled.MusicNote else Icons.Filled.Te
                contentDescription = stringResource(if (isShowingLyrics) R.string.action_hide_
                tint = iconTint
            )
        }

        IconButton(onClick = onLikeToggle, enabled = isItemLoaded) {
            Icon(
                imageVector = if (isLiked) Icons.Filled.Favorite else Icons.Filled.FavoriteBor
                contentDescription = stringResource(if (isLiked) R.string.content_desc_unlike_
                tint = iconTint
            )
        }

        IconButton(onClick = onQueueClick, enabled = queueNotEmpty) {
            Icon(Icons.AutoMirrored.Filled.PlaylistPlay, stringResource(R.string.content_desc_
        }

        Box {
            IconButton(onClick = { showMoreOptionsDropdown = true }, enabled = isItemLoaded) {
                Icon(Icons.Filled.MoreVert, stringResource(R.string.content_desc_more_options)
            }
            PlayerOverflowMenu(
                expanded = showMoreOptionsDropdown,
                onDismissRequest = { showMoreOptionsDropdown = false },
                currentItem = currentItem,
                actions = actions,
                navController = navController,
                playbackViewModel = playbackViewModel
            )
        }
    }
}

@Composable
private fun PlayerOverflowMenu(
    expanded: Boolean,
    onDismissRequest: () -> Unit,
    currentItem: PlaybackItem?,
    actions: FullPlayerActions,
```

```
    navController: NavHostController,
    playbackViewModel: PlaybackViewModel
) {
    val context = LocalContext.current
    DropdownMenu(
        expanded = expanded,
        onDismissRequest = onDismissRequest
    ) {
        DropdownMenuItem(
            text = { Text(stringResource(R.string.action_add_to_playlist_menu)) },
            onClick = {
                currentItem?.let { actions.onAddToPlaylist(it) }
                onDismissRequest()
            },
            leadingIcon = { Icon(Icons.AutoMirrored.Filled.PlaylistAdd, null) },
            enabled = currentItem != null
        )
        val artistChannelId = currentItem?.channelId
        DropdownMenuItem(
            text = { Text(stringResource(R.string.action_view_artist)) },
            onClick = {
                if (!artistChannelId.isNullOrBlank()) {
                    actions.onFindArtist(artistChannelId)
                    navController.navigate(AppDestinations.HOME_ROUTE) {
                        popUpTo(navController.graph.startDestinationRoute ?: AppDestinations.H
                        launchSingleTop = true; restoreState = true
                    }
                }
                onDismissRequest()
            },
            leadingIcon = { Icon(Icons.Filled.PersonSearch, null) },
            enabled = !artistChannelId.isNullOrBlank()
        )
        DropdownMenuItem(
            text = { Text(stringResource(R.string.action_audio_settings)) },
            onClick = {
                playbackViewModel.getAudioSessionId()?.let {
                    if (it != 0) actions.onOpenAudioSettings(it)
                    else Toast.makeText(context, R.string.error_no_audio_session, Toast.LENGTH
                }
                onDismissRequest()
            },
            leadingIcon = { Icon(Icons.Filled.Equalizer, null) }
        )
    }
}

@OptIn(ExperimentalFoundationApi::class)
@Composable
private fun PlayerContent(
    modifier: Modifier = Modifier,
    currentItem: PlaybackItem?,
    trackInfoColor: Color,
    isLoading: Boolean,
    artworkScale: Float,
```

```
    artworkAlpha: Float,
    onHorizontalSwipe: (dragAmount: Float) -> Unit,
    onVerticalSwipe: (deltaY: Float) -> Unit,
    onDoubleTap: () -> Unit
) {
    val context = LocalContext.current

    // Use onSizeChanged to determine artwork size based on parent container
    var parentSize by remember { mutableStateOf(IntSize.Zero) }
    val artworkSize = with(LocalDensity.current) { (parentSize.height * 0.4f).toDp() }

    val artworkUrls = remember(currentItem) {
        generateArtworkUrlList(currentItem, ThumbnailQuality.MAX)
    }
    var currentUrlIndex by remember(artworkUrls) { mutableIntStateOf(0) }

    Column(
        modifier = modifier
            .fillMaxWidth()
            .padding(horizontal = 24.dp, vertical = 16.dp)
            .onSizeChanged { parentSize = it },
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.SpaceAround
    ) {
        Spacer(modifier = Modifier.height(artworkSize * 0.1f))
        Box(
            modifier = Modifier
                .size(artworkSize)
                .aspectRatio(1f)
                .clip(FullPlayerDefaults.ArtworkShape)
                .background(MaterialTheme.colorScheme.surfaceVariant.copy(alpha = 0.3f))
                .graphicsLayer { scaleX = artworkScale; scaleY = artworkScale; alpha = artwork
                .combinedClickable(
                    interactionSource = remember { MutableInteractionSource() },
                    indication = null,
                    onDoubleClick = onDoubleTap,
                    onClick = {}
                )
                .pointerInput(Unit) {
                    detectDragGestures { change, dragAmount ->
                        change.consume()
                        if (dragAmount.y.absoluteValue > dragAmount.x.absoluteValue * 1.5) {
                            onVerticalSwipe(dragAmount.y)
                        } else {
                            onHorizontalSwipe(dragAmount.x)
                        }
                    }
                },
            contentAlignment = Alignment.Center
        ) {
            if (isLoading && currentItem == null) {
                CircularProgressIndicator()
            } else {
                AsyncImage(
                    model = ImageRequest.Builder(context)
```

```
                        .data(artworkUrls.getOrNull(currentUrlIndex))
                        .placeholder(R.drawable.ic_default_album_art_placeholder)
                        .error(R.drawable.ic_error_image)
                        .crossfade(true).build(),
                    contentDescription = stringResource(R.string.content_desc_album_art),
                    contentScale = ContentScale.Crop,
                    modifier = Modifier.fillMaxSize(),
                    onError = { if (currentUrlIndex < artworkUrls.lastIndex) { currentUrlIndex
                )
            }
        }
        Spacer(modifier = Modifier.height(artworkSize * 0.15f))
        TrackInfo(currentItem, trackInfoColor)
        Spacer(modifier = Modifier.weight(1f))
    }
}

@Composable
private fun VolumeSlider(
    value: Float,
    onValueChange: (Float) -> Unit,
    thumbColor: Color,
    activeTrackColor: Color,
    modifier: Modifier = Modifier
) {
    Row(
        modifier = modifier
            .fillMaxWidth()
            .padding(horizontal = 24.dp, vertical = 8.dp),
        verticalAlignment = Alignment.CenterVertically
    ) {
        val icon = when {
            value < 0.01f -> Icons.AutoMirrored.Filled.VolumeMute
            value < 0.5f -> Icons.AutoMirrored.Filled.VolumeDown
            else -> Icons.AutoMirrored.Filled.VolumeUp
        }
        Icon(
            imageVector = icon,
            contentDescription = stringResource(R.string.content_desc_volume),
            modifier = Modifier.size(24.dp),
            tint = activeTrackColor
        )
        Slider(
            value = value,
            onValueChange = onValueChange,
            modifier = Modifier.weight(1f).padding(horizontal = 8.dp),
            valueRange = 0f..1f,
            colors = SliderDefaults.colors(
                thumbColor = thumbColor,
                activeTrackColor = activeTrackColor,
                inactiveTrackColor = activeTrackColor.copy(alpha = 0.3f)
            )
        )
    }
}
```

```
@Composable
private fun TrackInfo(currentItem: PlaybackItem?, textColor: Color) {
    AnimatedContent(
        targetState = currentItem?.id ?: "loading",
        transitionSpec = { fadeIn(tween(220, 90)) togetherWith fadeOut(tween(90)) },
        label = "trackInfoAnimation"
    ) { targetId ->
        Column(
            horizontalAlignment = Alignment.CenterHorizontally,
            modifier = Modifier.fillMaxWidth().padding(horizontal = 16.dp)
        ) {
            Text(
                text = if (targetId == "loading" || currentItem == null) stringResource(R.stri
                color = textColor,
                style = MaterialTheme.typography.headlineSmall.copy(fontSize = 22.sp),
                fontWeight = FontWeight.SemiBold,
                textAlign = TextAlign.Center,
                maxLines = 2,
                overflow = TextOverflow.Ellipsis
            )
            Spacer(Modifier.height(4.dp))
            Text(
                text = if (targetId == "loading" || currentItem == null) "" else currentItem.a
                color = textColor.copy(alpha = 0.8f),
                style = MaterialTheme.typography.titleMedium.copy(fontSize = 17.sp, lineHeight
                textAlign = TextAlign.Center,
                maxLines = 2,
                overflow = TextOverflow.Ellipsis
            )
        }
    }
}

@Composable
private fun LyricsView(currentItem: PlaybackItem?, modifier: Modifier = Modifier, textColor: C
    Box(modifier = modifier.fillMaxSize(), contentAlignment = Alignment.Center) {
        if (currentItem?.description.isNullOrBlank()) {
            Text(
                text = stringResource(R.string.lyrics_not_available),
                style = MaterialTheme.typography.bodyLarge,
                textAlign = TextAlign.Center,
                modifier = Modifier.padding(16.dp),
                color = MaterialTheme.colorScheme.onSurfaceVariant
            )
        } else {
            Text(
                text = currentItem.description,
                style = MaterialTheme.typography.bodyLarge,
                color = textColor,
                modifier = Modifier.fillMaxSize().verticalScroll(rememberScrollState()).paddin
            )
        }
    }
}
```

```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
private fun QueueSheetContent(
    queueItems: List<PlaybackItem>,
    currentIndex: Int,
    isRadioMode: Boolean,
    onPlayQueueItem: (Int) -> Unit,
    onClearQueue: () -> Unit,
    onMoveItem: (from: Int, to: Int) -> Unit,
    onRemoveItem: (index: Int) -> Unit
) {
    val listState = rememberLazyListState()
    val reorderableState = rememberReorderableLazyListState(
        lazyListState = listState,
        onMove = { from, to -> onMoveItem(from.index, to.index) }
    )

    LaunchedEffect(currentIndex) {
        if (currentIndex in 0..queueItems.lastIndex) {
            listState.animateScrollToItem(currentIndex)
        }
    }

    Column(modifier = Modifier.fillMaxSize().navigationBarsPadding()) {
        Surface(
            modifier = Modifier.fillMaxWidth(),
            color = MaterialTheme.colorScheme.surface,
            shadowElevation = 4.dp
        ) {
            Column {
                Box(
                    modifier = Modifier.fillMaxWidth().padding(vertical = 8.dp),
                    contentAlignment = Alignment.Center
                ) {
                    Surface(
                        modifier = Modifier.width(32.dp).height(4.dp),
                        shape = RoundedCornerShape(2.dp),
                        color = MaterialTheme.colorScheme.onSurfaceVariant.copy(alpha = 0.4f)
                    ) {}
                }
                Row(
                    modifier = Modifier.fillMaxWidth().padding(horizontal = 16.dp, vertical =
                    verticalAlignment = Alignment.CenterVertically,
                    horizontalArrangement = Arrangement.SpaceBetween
                ) {
                    Column {
                        Text(
                            text = stringResource(R.string.up_next_queue_title),
                            style = MaterialTheme.typography.titleLarge,
                            color = MaterialTheme.colorScheme.onSurface
                        )
                        if (queueItems.isNotEmpty()) {
                            Text(
                                text = pluralStringResource(
```

```
                            R.plurals.queue_items_count,
                            queueItems.size,
                            queueItems.size
                        ),
                        style = MaterialTheme.typography.bodyMedium,
                        color = MaterialTheme.colorScheme.onSurfaceVariant
                    )
                }
            }
            AnimatedVisibility(
                visible = queueItems.isNotEmpty(),
                enter = fadeIn() + scaleIn(),
                exit = fadeOut() + scaleOut()
            ) {
                FilledTonalIconButton(
                    onClick = onClearQueue,
                    colors = IconButtonDefaults.filledTonalIconButtonColors(
                        containerColor = MaterialTheme.colorScheme.errorContainer,
                        contentColor = MaterialTheme.colorScheme.onErrorContainer
                    )
                ) {
                    Icon(
                        Icons.Filled.PlaylistRemove,
                        contentDescription = stringResource(R.string.action_clear_queu
                    )
                }
            }
        }
        HorizontalDivider(color = MaterialTheme.colorScheme.outlineVariant)
    }
}

AnimatedContent(
    targetState = queueItems.isEmpty(),
    transitionSpec = {
        fadeIn(animationSpec = tween(300)) togetherWith fadeOut(
            animationSpec = tween(300)
        )
    },
    label = "queue_content"
) { isEmpty ->
    if (isEmpty) {
        Box(
            modifier = Modifier.fillMaxSize().padding(32.dp),
            contentAlignment = Alignment.Center
        ) {
            Column(
                horizontalAlignment = Alignment.CenterHorizontally,
                verticalArrangement = Arrangement.spacedBy(16.dp)
            ) {
                Icon(
                    Icons.AutoMirrored.Outlined.QueueMusic,
                    null,
                    modifier = Modifier.size(64.dp),
                    tint = MaterialTheme.colorScheme.onSurfaceVariant.copy(alpha = 0.6
```

```
                        )
                        Text(
                            text = stringResource(R.string.empty_queue),
                            style = MaterialTheme.typography.titleMedium,
                            color = MaterialTheme.colorScheme.onSurfaceVariant,
                            textAlign = TextAlign.Center
                        )
                        Text(
                            text = stringResource(R.string.empty_queue_description),
                            style = MaterialTheme.typography.bodyMedium,
                            color = MaterialTheme.colorScheme.onSurfaceVariant.copy(alpha = 0.
                            textAlign = TextAlign.Center
                        )
                    }
                }
            } else {
                LazyColumn(
                    state = listState,
                    modifier = Modifier.fillMaxSize(),
                    contentPadding = PaddingValues(
                        start = 8.dp,
                        end = 8.dp,
                        top = 8.dp,
                        bottom = 16.dp
                    ),
                    verticalArrangement = Arrangement.spacedBy(4.dp)
                ) {
                    itemsIndexed(
                        items = queueItems,
                        key = { index, item -> "${item.id}_$index" }
                    ) { index, itemData ->
                        ReorderableItem(
                            state = reorderableState,
                            key = "${itemData.id}_$index",
                            enabled = !isRadioMode
                        ) { isDragging ->
                            QueueItemRow(
                                item = itemData,
                                index = index + 1,
                                isCurrentlyPlaying = index == currentIndex,
                                isDragging = isDragging,
                                isRadioMode = isRadioMode,
                                onItemClick = { onPlayQueueItem(index) },
                                onRemoveClick = { onRemoveItem(index) },
                                modifier = Modifier.longPressDraggableHandle()
                            )
                        }
                    }
                }
            }
        }
    }
}

@Composable
```

```
private fun QueueItemRow(
    item: PlaybackItem,
    index: Int,
    isCurrentlyPlaying: Boolean,
    isDragging: Boolean,
    isRadioMode: Boolean,
    onItemClick: () -> Unit,
    onRemoveClick: () -> Unit,
    modifier: Modifier = Modifier
) {
    val elevation by animateDpAsState(
        targetValue = if (isDragging) 8.dp else 0.dp,
        animationSpec = tween(200),
        label = "drag_elevation"
    )
    val containerColor by animateColorAsState(
        targetValue = when {
            isCurrentlyPlaying -> MaterialTheme.colorScheme.primaryContainer
            isDragging -> MaterialTheme.colorScheme.surfaceVariant
            else -> MaterialTheme.colorScheme.surface
        },
        animationSpec = tween(200),
        label = "container_color"
    )

    Surface(
        onClick = onItemClick,
        modifier = modifier.fillMaxWidth(),
        shape = RoundedCornerShape(12.dp),
        color = containerColor,
        shadowElevation = elevation,
        border = if (isCurrentlyPlaying) BorderStroke(2.dp, MaterialTheme.colorScheme.primary.
    ) {
        Row(
            modifier = Modifier.fillMaxWidth().padding(horizontal = 12.dp, vertical = 8.dp),
            verticalAlignment = Alignment.CenterVertically,
            horizontalArrangement = Arrangement.spacedBy(12.dp)
        ) {
            // Index/Play indicator
            Surface(
                shape = CircleShape,
                color = if (isCurrentlyPlaying) MaterialTheme.colorScheme.primary else Materia
                modifier = Modifier.size(32.dp)
            ) {
                Box(contentAlignment = Alignment.Center) {
                    if (isCurrentlyPlaying) {
                        Icon(
                            Icons.Filled.PlayArrow,
                            null,
                            tint = MaterialTheme.colorScheme.onPrimary,
                            modifier = Modifier.size(16.dp)
                        )
                    } else {
                        Text(
                            text = index.toString(),
```

```
                style = MaterialTheme.typography.labelMedium,
                color = MaterialTheme.colorScheme.onSurfaceVariant
            )
        }
    }
}

// Artwork
AsyncImage(
    model = ImageRequest.Builder(LocalContext.current)
        .data(item.artworkUri)
        .placeholder(R.drawable.ic_default_album_art_placeholder)
        .error(R.drawable.ic_error_image)
        .crossfade(true)
        .build(),
    contentDescription = null,
    modifier = Modifier.size(48.dp).clip(RoundedCornerShape(8.dp)),
    contentScale = ContentScale.Crop
)

// Title and artist
Column(
    modifier = Modifier.weight(1f),
    verticalArrangement = Arrangement.spacedBy(2.dp)
) {
    Text(
        text = item.title,
        style = MaterialTheme.typography.bodyLarge,
        color = if (isCurrentlyPlaying) MaterialTheme.colorScheme.onPrimaryContain
        maxLines = 1,
        overflow = TextOverflow.Ellipsis,
        fontWeight = if (isCurrentlyPlaying) FontWeight.Medium else FontWeight.Nor
    )
    Text(
        text = item.artistText,
        style = MaterialTheme.typography.bodyMedium,
        color = if (isCurrentlyPlaying) MaterialTheme.colorScheme.onPrimaryContain
        maxLines = 1,
        overflow = TextOverflow.Ellipsis
    )
}

// Duration
Text(
    text = formatDurationSecondsToString(item.durationSec),
    style = MaterialTheme.typography.bodySmall,
    color = MaterialTheme.colorScheme.onSurfaceVariant
)

// Remove button
IconButton(
    onClick = onRemoveClick,
    modifier = Modifier.size(40.dp)
) {
    Icon(
```

```
                    Icons.Default.Delete,
                    contentDescription = stringResource(R.string.action_remove_from_queue),
                    tint = MaterialTheme.colorScheme.onSurfaceVariant.copy(alpha = 0.7f),
                    modifier = Modifier.size(20.dp)
                )
            }

            // Drag handle (only show in non-radio mode)
            if (!isRadioMode) {
                Icon(
                    Icons.Filled.DragHandle,
                    contentDescription = stringResource(R.string.drag_to_reorder),
                    tint = MaterialTheme.colorScheme.onSurfaceVariant.copy(alpha = if (isDragg
                    modifier = Modifier.size(20.dp)
                )
            }
        }
    }
}

// File: java\com\example\holodex\ui\composables\HeroCard.kt
// File: java/com/example/holodex/ui/composables/HeroCard.kt
// (Create this new file)

package com.example.holodex.ui.composables

import androidx.compose.foundation.background
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.Spacer
import androidx.compose.foundation.layout.aspectRatio
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.Brush
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextOverflow
import androidx.compose.ui.unit.dp
import coil.compose.AsyncImage
import coil.request.ImageRequest
import com.example.holodex.R
import com.example.holodex.data.model.discovery.SingingStreamShelfItem
import com.example.holodex.util.ThumbnailQuality
import com.example.holodex.util.getYouTubeThumbnailUrl
```

```
@Composable
fun HeroCard(
    item: SingingStreamShelfItem,
    onClick: () -> Unit,
    modifier: Modifier = Modifier
) {
    val video = item.video
    val thumbnailUrl = getYouTubeThumbnailUrl(video.id, ThumbnailQuality.MAX).firstOrNull()

    Box(
        modifier = modifier
            .fillMaxWidth()
            .aspectRatio(16f / 9f)
            .clip(MaterialTheme.shapes.large)
            .clickable(onClick = onClick)
    ) {
        // Background Image
        AsyncImage(
            model = ImageRequest.Builder(LocalContext.current)
                .data(thumbnailUrl)
                .placeholder(R.drawable.ic_placeholder_image)
                .error(R.drawable.ic_error_image)
                .crossfade(true)
                .build(),
            contentDescription = video.title,
            contentScale = ContentScale.Crop,
            modifier = Modifier.fillMaxSize()
        )

        // Gradient overlay for text readability
        Box(
            modifier = Modifier
                .fillMaxSize()
                .background(
                    Brush.verticalGradient(
                        colors = listOf(
                            Color.Transparent,
                            Color.Black.copy(alpha = 0.2f),
                            Color.Black.copy(alpha = 0.8f)
                        ),
                        startY = 300f
                    )
                )
        )

        // Text Content
        Column(
            modifier = Modifier
                .fillMaxSize()
                .padding(16.dp),
            verticalArrangement = Arrangement.Bottom
        ) {
            Text(
                text = video.title,
```

```
                style = MaterialTheme.typography.titleLarge,
                color = Color.White,
                fontWeight = FontWeight.Bold,
                maxLines = 2,
                overflow = TextOverflow.Ellipsis
            )
            Spacer(modifier = Modifier.height(4.dp))
            Text(
                text = video.channel.name,
                style = MaterialTheme.typography.bodyMedium,
                color = Color.White.copy(alpha = 0.9f),
                maxLines = 1,
                overflow = TextOverflow.Ellipsis
            )
        }
    }
}

// File: java\com\example\holodex\ui\composables\HeroCarousel.kt
// File: java/com/example/holodex/ui/composables/HeroCarousel.kt
// (Create this new file)

package com.example.holodex.ui.composables

import androidx.compose.foundation.ExperimentalFoundationApi
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.aspectRatio
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.pager.HorizontalPager
import androidx.compose.foundation.pager.rememberPagerState
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.unit.dp
import com.example.holodex.data.model.discovery.SingingStreamShelfItem
import com.example.holodex.viewmodel.state.UiState

@OptIn(ExperimentalFoundationApi::class)
@Composable
fun HeroCarousel(
    title: String,
    uiState: UiState<List<SingingStreamShelfItem>>,
```

```
    onItemClicked: (SingingStreamShelfItem) -> Unit,
    modifier: Modifier = Modifier,
) {
    Column(modifier = modifier) {
        // Title (no "Show More" button)
        Text(
            text = title,
            style = MaterialTheme.typography.titleLarge,
            modifier = Modifier.padding(horizontal = 16.dp)
        )
        Spacer(Modifier.height(12.dp))

        when (uiState) {
            is UiState.Loading -> {
                // Show a single large skeleton
                Box(
                    modifier = Modifier
                        .fillMaxWidth()
                        .padding(horizontal = 16.dp)
                        .aspectRatio(16f / 9f)
                        .clip(MaterialTheme.shapes.large)
                        .background(MaterialTheme.colorScheme.surfaceVariant)
                )
            }
            is UiState.Success -> {
                if (uiState.data.isNotEmpty()) {
                    val pagerState = rememberPagerState(pageCount = { uiState.data.size })

                    HorizontalPager(
                        state = pagerState,
                        contentPadding = PaddingValues(horizontal = 16.dp),
                        pageSpacing = 12.dp,
                    ) { pageIndex ->
                        HeroCard(
                            item = uiState.data[pageIndex],
                            onClick = { onItemClicked(uiState.data[pageIndex]) }
                        )
                    }

                    // Pager Indicators
                    Row(
                        Modifier
                            .height(24.dp)
                            .fillMaxWidth(),
                        horizontalArrangement = Arrangement.Center,
                        verticalAlignment = Alignment.Bottom
                    ) {
                        repeat(pagerState.pageCount) { iteration ->
                            val color = if (pagerState.currentPage == iteration) MaterialTheme
                            Box(
                                modifier = Modifier
                                    .padding(4.dp)
                                    .clip(CircleShape)
                                    .background(color)
                                    .size(8.dp)
```

```
                            )
                        }
                    }
                }
            }
            is UiState.Error -> {
                // You can reuse the error component from CarouselShelf if you extract it
                Text(
                    text = uiState.message,
                    color = MaterialTheme.colorScheme.error,
                    modifier = Modifier.padding(horizontal = 16.dp)
                )
            }
        }
    }
}

// File: java\com\example\holodex\ui\composables\ItemOptionsMenu.kt
package com.example.holodex.ui.composables

import android.content.Intent
import androidx.compose.foundation.layout.padding
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.automirrored.filled.PlaylistAdd
import androidx.compose.material.icons.automirrored.filled.QueueMusic
import androidx.compose.material.icons.filled.Delete
import androidx.compose.material.icons.filled.Download
import androidx.compose.material.icons.filled.Movie
import androidx.compose.material.icons.filled.Person
import androidx.compose.material.icons.filled.Share
import androidx.compose.material3.DropdownMenu
import androidx.compose.material3.DropdownMenuItem
import androidx.compose.material3.HorizontalDivider
import androidx.compose.material3.Icon
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.Immutable
import androidx.compose.ui.Modifier
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.unit.dp
import com.example.holodex.R

@Composable
fun ItemOptionsMenu(
    state: ItemMenuState,
    actions: ItemMenuActions,
    expanded: Boolean,
    onDismissRequest: () -> Unit
) {
    val context = LocalContext.current
    val onShare = { textToShare: String ->
        val sendIntent: Intent = Intent().apply {
            action = Intent.ACTION_SEND
            putExtra(Intent.EXTRA_TEXT, textToShare)
```

```
            type = "text/plain"
        }
        val shareIntent = Intent.createChooser(sendIntent, null)
        context.startActivity(shareIntent)
        onDismissRequest()
    }

    DropdownMenu(expanded = expanded, onDismissRequest = onDismissRequest) {
        DropdownMenuItem(
            text = { Text(stringResource(R.string.action_add_to_queue)) },
            onClick = {
                actions.onAddToQueue()
                onDismissRequest()
            },
            leadingIcon = { Icon(Icons.AutoMirrored.Filled.QueueMusic, null) }
        )

        DropdownMenuItem(
            text = { Text(stringResource(R.string.action_add_to_playlist_menu)) },
            onClick = {
                actions.onAddToPlaylist()
                onDismissRequest()
            },
            leadingIcon = { Icon(Icons.AutoMirrored.Filled.PlaylistAdd, null) }
        )

        DropdownMenuItem(
            text = { Text(stringResource(R.string.action_share)) },
            onClick = { onShare(state.shareUrl) },
            leadingIcon = { Icon(Icons.Filled.Share, null) }
        )

        if (state.canBeDownloaded) {
            DropdownMenuItem(
                text = { Text(stringResource(R.string.action_download)) },
                onClick = {
                    actions.onDownload()
                    onDismissRequest()
                },
                leadingIcon = { Icon(Icons.Filled.Download, null) }
            )
        }

        if (state.isDownloaded) {
            DropdownMenuItem(
                text = { Text(stringResource(R.string.action_delete)) },
                onClick = {
                    actions.onDelete()
                    onDismissRequest()
                },
                leadingIcon = { Icon(Icons.Filled.Delete, null) }
            )
        }

        HorizontalDivider(modifier = Modifier.padding(vertical = 4.dp))
```

```
        if (state.isSegment) {
            DropdownMenuItem(
                text = { Text(stringResource(R.string.action_view_video)) },
                onClick = {
                    actions.onGoToVideo(state.videoId)
                    onDismissRequest()
                },
                leadingIcon = { Icon(Icons.Filled.Movie, null) }
            )
        }

        DropdownMenuItem(
            text = { Text(stringResource(R.string.action_view_artist)) },
            onClick = {
                actions.onGoToArtist(state.channelId)
                onDismissRequest()
            },
            leadingIcon = { Icon(Icons.Filled.Person, null) },
            enabled = state.channelId.isNotBlank()
        )
    }
}
/**
 * A state holder for the ItemOptionsMenu. It contains all the necessary
 * data to determine the visibility and enabled status of menu items.
 */
@Immutable
data class ItemMenuState(
    val isDownloaded: Boolean,
    val isSegment: Boolean,
    val canBeDownloaded: Boolean,
    val shareUrl: String,
    val videoId: String,
    val channelId: String
)

/**
 * A holder for all the possible actions a user can take from the ItemOptionsMenu.
 * The parent composable is responsible for providing the implementations for these actions.
 */
@Immutable
data class ItemMenuActions(
    val onAddToQueue: () -> Unit,
    val onAddToPlaylist: () -> Unit,
    val onShare: (String) -> Unit,
    val onDownload: () -> Unit,
    val onDelete: () -> Unit,
    val onGoToVideo: (String) -> Unit,
    val onGoToArtist: (String) -> Unit,
)

// File: java\com\example\holodex\ui\composables\MainScreenLayout.kt
package com.example.holodex.ui.composables
```

```
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.WindowInsets
import androidx.compose.foundation.layout.asPaddingValues
import androidx.compose.foundation.layout.systemBars
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.layout.SubcomposeLayout

/**
 * A custom layout that stacks the [content] behind the [bottomBar].
 * It measures the [bottomBar] height and passes it as [PaddingValues] to the [content].
 */
@Composable
fun MainScreenLayout(
    modifier: Modifier = Modifier,
    bottomBar: @Composable () -> Unit,
    content: @Composable (PaddingValues) -> Unit
) {
    WindowInsets.systemBars.asPaddingValues()

    SubcomposeLayout(modifier = modifier) { constraints ->
        val layoutWidth = constraints.maxWidth
        val layoutHeight = constraints.maxHeight

        // 1. Measure Bottom Bar first to know its height
        val bottomBarPlaceables = subcompose("bottomBar", bottomBar).map {
            it.measure(constraints.copy(minHeight = 0))
        }

        val bottomBarHeight = bottomBarPlaceables.maxOfOrNull { it.height } ?: 0
        val bottomBarHeightDp = bottomBarHeight.toDp()

        // 2. Prepare Content Padding
        // The content padding bottom is exactly the height of the bottom bar (Nav + Player)
        // plus any system navigation bar insets if handled internally.
        val contentPadding = PaddingValues(
            bottom = bottomBarHeightDp
        )

        // 3. Measure Content with the full screen constraints (it draws behind)
        val contentPlaceables = subcompose("content") {
            content(contentPadding)
        }.map {
            it.measure(constraints)
        }

        // 4. Place them
        layout(layoutWidth, layoutHeight) {
            // Place Content at (0,0) - full screen
            contentPlaceables.forEach { it.place(0, 0) }

            // Place Bottom Bar at the bottom
            bottomBarPlaceables.forEach {
                it.place(0, layoutHeight - bottomBarHeight)
            }
```

```
        }
    }
}

// File: java\com\example\holodex\ui\composables\Media3PlayerControls.kt
@file:UnstableApi

package com.example.holodex.ui.composables

import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.PauseCircleFilled
import androidx.compose.material.icons.filled.PlayCircleFilled
import androidx.compose.material.icons.filled.SkipNext
import androidx.compose.material.icons.filled.SkipPrevious
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Slider
import androidx.compose.material3.SliderDefaults
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableFloatStateOf
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.graphics.Color
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.unit.dp
import androidx.media3.common.Player
import androidx.media3.common.util.UnstableApi
import com.example.holodex.R
import com.example.holodex.playback.domain.model.DomainPlaybackProgress
import com.example.holodex.playback.domain.model.DomainRepeatMode
import com.example.holodex.playback.domain.model.DomainShuffleMode
import com.example.holodex.playback.util.formatDurationSecondsToString

private const val CONTROLS_TAG = "Media3PlayerControls"

/**
 * A self-contained composable that displays a full set of player controls,
 * powered by Media3's Compose state holders.
 *
 * @param player The Media3 Player instance.
 * @param progress The current playback progress, passed from the ViewModel to display on the
```

```
 * @param onSeek A lambda to be invoked when the user interacts with the seek bar.
 * @param primaryColor The primary theme color, used for prominent elements like the play butt
 * @param onPrimaryColor The color for icons and text that appear on the primary color, used f
 */
@Composable
fun Media3PlayerControls(
    // The player is now ONLY for reading state for button enabled/disabled status
    player: Player,
    shuffleMode: DomainShuffleMode,
    repeatMode: DomainRepeatMode,
    progress: DomainPlaybackProgress,
    isRadioMode: Boolean,
    onPlayPause: () -> Unit,
    onSkipPrevious: () -> Unit,
    onSkipNext: () -> Unit,
    onToggleShuffle: () -> Unit,
    onToggleRepeat: () -> Unit,
    onSeek: (Long) -> Unit,
    onScrubbingChange: (Boolean) -> Unit,
    primaryColor: Color,
    onPrimaryColor: Color,
    modifier: Modifier = Modifier
) {
    Column(
        modifier = modifier
            .fillMaxWidth()
            .padding(horizontal = 16.dp, vertical = 8.dp)
    ) {
        PlayerSeekBar(
            progress = progress,
            onSeek = onSeek,
            onScrubbingChange = onScrubbingChange,
            thumbColor = primaryColor,
            activeTrackColor = onPrimaryColor,
            inactiveTrackColor = onPrimaryColor.copy(alpha = 0.3f),
            timeTextColor = onPrimaryColor.copy(alpha = 0.7f)
        )

        Row(
            modifier = Modifier.fillMaxWidth(),
            horizontalArrangement = Arrangement.SpaceEvenly,
            verticalAlignment = Alignment.CenterVertically
        ) {
            // --- CUSTOM SHUFFLE BUTTON ---
            val isShuffleOn = shuffleMode == DomainShuffleMode.ON // Use the provided state
            IconButton(
                onClick = onToggleShuffle,
                enabled = player.isCommandAvailable(Player.COMMAND_SET_SHUFFLE_MODE) && !isRad
            ) {
                Icon(
                    painter = painterResource(id = if (isShuffleOn) R.drawable.ic_shuffle_on_2
                    contentDescription = stringResource(R.string.action_shuffle),
                    modifier = Modifier.size(28.dp),
                    tint = if (isShuffleOn) primaryColor else onPrimaryColor.copy(alpha = 0.6f
                )
```

```
}

// --- CUSTOM PREVIOUS BUTTON ---
IconButton(
    onClick = onSkipPrevious,
    enabled = player.isCommandAvailable(Player.COMMAND_SEEK_TO_PREVIOUS_MEDIA_ITEM
) {
    Icon(
        imageVector = Icons.Filled.SkipPrevious,
        contentDescription = stringResource(R.string.action_previous),
        modifier = Modifier.size(36.dp),
        tint = onPrimaryColor
    )
}

// --- CUSTOM PLAY/PAUSE BUTTON ---
IconButton(
    onClick = onPlayPause,
    enabled = player.isCommandAvailable(Player.COMMAND_PLAY_PAUSE)
) {
    Icon(
        imageVector = if (player.isPlaying) Icons.Filled.PauseCircleFilled else Ic
        contentDescription = if (player.isPlaying) stringResource(R.string.action_
            R.string.action_play
        ),
        modifier = Modifier.size(64.dp),
        tint = primaryColor
    )
}

// --- CUSTOM NEXT BUTTON ---
IconButton(
    onClick = onSkipNext,
    enabled = player.isCommandAvailable(Player.COMMAND_SEEK_TO_NEXT_MEDIA_ITEM)
) {
    Icon(
        imageVector = Icons.Filled.SkipNext,
        contentDescription = stringResource(R.string.action_next),
        modifier = Modifier.size(36.dp),
        tint = onPrimaryColor
    )
}

// --- CUSTOM REPEAT BUTTON ---
IconButton(
    onClick = onToggleRepeat,
    enabled = player.isCommandAvailable(Player.COMMAND_SET_REPEAT_MODE) && !isRadi
) {
    val iconRes = when (repeatMode) { // Use the provided state
        DomainRepeatMode.ONE -> R.drawable.ic_repeat_one_24
        DomainRepeatMode.ALL -> R.drawable.ic_repeat_on_24
        else -> R.drawable.ic_repeat_off_24
    }
    Icon(
        painter = painterResource(id = iconRes),
```

```
                    contentDescription = stringResource(R.string.action_repeat),
                    modifier = Modifier.size(28.dp),
                    tint = if (repeatMode != DomainRepeatMode.NONE) primaryColor else onPrimar
                        alpha = 0.6f
                    )
                )
            }
        }
    }
}


// ================================================================
// PRIVATE, INTERNAL BUILDING BLOCKS FOR THE CONTROLS
// ================================================================

@Composable
private fun PlayerSeekBar(
    progress: DomainPlaybackProgress,
    onSeek: (Long) -> Unit,
    onScrubbingChange: (Boolean) -> Unit,
    thumbColor: Color,
    activeTrackColor: Color,
    inactiveTrackColor: Color,
    timeTextColor: Color
) {
    var sliderPosition by remember(progress.positionSec) { mutableFloatStateOf(progress.positi
    var isUserScrubbing by remember { mutableStateOf(false) }

    LaunchedEffect(isUserScrubbing) {
        onScrubbingChange(isUserScrubbing)
    }

    Column(Modifier.fillMaxWidth()) {
        Slider(
            value = if (isUserScrubbing) sliderPosition else progress.positionSec.toFloat(),
            onValueChange = {
                isUserScrubbing = true
                sliderPosition = it
            },
            onValueChangeFinished = {
                onSeek(sliderPosition.toLong())
                isUserScrubbing = false
            },
            valueRange = 0f..(progress.durationSec.toFloat().coerceAtLeast(1f)),
            modifier = Modifier.fillMaxWidth(),
            colors = SliderDefaults.colors(
                thumbColor = thumbColor,
                activeTrackColor = activeTrackColor,
                inactiveTrackColor = inactiveTrackColor
            )
        )
        Row(
            modifier = Modifier
                .fillMaxWidth()
```

```
                .padding(horizontal = 4.dp),
            horizontalArrangement = Arrangement.SpaceBetween
        ) {
            Text(
                text = formatDurationSecondsToString(if (isUserScrubbing) sliderPosition.toLon
                style = MaterialTheme.typography.bodySmall,
                color = timeTextColor
            )
            Text(
                text = formatDurationSecondsToString(progress.durationSec),
                style = MaterialTheme.typography.bodySmall,
                color = timeTextColor
            )
        }
    }
}



// File: java\com\example\holodex\ui\composables\MiniPlayerWithProgressBar.kt
// File: java/com/example/holodex/ui/composables/MiniPlayerWithProgressBar.kt

package com.example.holodex.ui.composables

import androidx.compose.animation.AnimatedVisibility
import androidx.compose.animation.fadeOut
import androidx.compose.animation.shrinkVertically
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.IntrinsicSize
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.layout.width
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Pause
import androidx.compose.material.icons.filled.PlayArrow
import androidx.compose.material.icons.filled.SkipNext
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.LinearProgressIndicator
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Surface
import androidx.compose.material3.SwipeToDismissBox
import androidx.compose.material3.SwipeToDismissBoxValue
import androidx.compose.material3.Text
import androidx.compose.material3.rememberSwipeToDismissBoxState
import androidx.compose.material3.surfaceColorAtElevation
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.draw.clip
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.style.TextOverflow
import androidx.compose.ui.unit.dp
import androidx.lifecycle.compose.collectAsStateWithLifecycle
import coil.compose.AsyncImage
import coil.request.ImageRequest
import com.example.holodex.R
import com.example.holodex.util.ThumbnailQuality
import com.example.holodex.util.generateArtworkUrlList
import com.example.holodex.viewmodel.PlaybackViewModel
import com.example.holodex.viewmodel.rememberFullPlayerLoadingState
import com.example.holodex.viewmodel.rememberIsPlayingState
import com.example.holodex.viewmodel.rememberMiniPlayerArtistState
import com.example.holodex.viewmodel.rememberMiniPlayerProgressState
import com.example.holodex.viewmodel.rememberMiniPlayerQueueStateForButton
import com.example.holodex.viewmodel.rememberMiniPlayerTitleState
import kotlinx.coroutines.delay

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun MiniPlayerWithProgressBar(
    playbackViewModel: PlaybackViewModel,
    modifier: Modifier = Modifier,
    onTapped: () -> Unit = {}
) {
    val uiState by playbackViewModel
        .uiState
        .collectAsStateWithLifecycle()
    val currentItem = uiState.currentItem

    val title by rememberMiniPlayerTitleState(playbackViewModel.uiState)
    val artist by rememberMiniPlayerArtistState(playbackViewModel.uiState)
    val isPlaying by rememberIsPlayingState(playbackViewModel.uiState)
    val progressFraction by rememberMiniPlayerProgressState(playbackViewModel.uiState)
    val queueStatePair by rememberMiniPlayerQueueStateForButton(playbackViewModel.uiState)
    // FIX 2: Deconstruct to ignore the unused variable
    val (_, canSkipNext) = queueStatePair
    val isLoading by rememberFullPlayerLoadingState(playbackViewModel.uiState)

    // FIX 3: Simplify this check. If title is not null, an item exists.
    val currentItemExists = title != null

    var showPlayer by remember { mutableStateOf(true) }

    val dismissState = rememberSwipeToDismissBoxState(
```

```
    confirmValueChange = {
        if (it == SwipeToDismissBoxValue.StartToEnd || it == SwipeToDismissBoxValue.EndToS
            showPlayer = false
            true
        } else {
            false
        }
    }
)

LaunchedEffect(showPlayer) {
    if (!showPlayer) {
        delay(300)
        playbackViewModel.clearCurrentQueue()
    }
}

LaunchedEffect(currentItem?.id) {
    if (currentItem != null && !showPlayer) {
        showPlayer = true
        dismissState.reset()
    }
}

if (!currentItemExists && !isLoading) {
    Spacer(modifier = modifier.height(0.dp))
    return
}

AnimatedVisibility(
    visible = showPlayer,
    exit = shrinkVertically() + fadeOut()
) {
    SwipeToDismissBox(
        state = dismissState,
        backgroundContent = {},
        modifier = modifier
    ) {
        Surface(
            modifier = Modifier
                .fillMaxWidth()
                .height(IntrinsicSize.Min)
                .clickable(onClick = onTapped, enabled = currentItemExists),
            color = MaterialTheme.colorScheme.surfaceColorAtElevation(3.dp),
            tonalElevation = 3.dp,
            shadowElevation = 3.dp
        ) {
            Column {
                Row(
                    modifier = Modifier
                        .fillMaxWidth()
                        .padding(
                            start = 8.dp,
                            end = 4.dp,
                            top = 8.dp,
```

```
                bottom = 8.dp
            ),
        verticalAlignment = Alignment.CenterVertically
    ) {
        val miniPlayerArtworkUrls = remember(currentItem) {
            generateArtworkUrlList(currentItem, ThumbnailQuality.HIGH)
        }

        AsyncImage(
            model = ImageRequest.Builder(LocalContext.current)
                .data(miniPlayerArtworkUrls.firstOrNull())
                .placeholder(R.drawable.ic_default_album_art_placeholder)
                .error(R.drawable.ic_error_image)
                .crossfade(true)
                .build(),
            contentDescription = stringResource(R.string.content_desc_album_ar
            contentScale = ContentScale.Crop,
            modifier = Modifier
                .size(48.dp)
                .clip(MaterialTheme.shapes.small)
                .background(MaterialTheme.colorScheme.surfaceVariant)
        )

        Spacer(modifier = Modifier.width(12.dp))

        Column(
            modifier = Modifier.weight(1f),
            verticalArrangement = Arrangement.Center
        ) {
            Text(
                text = title ?: stringResource(R.string.loading_track),
                style = MaterialTheme.typography.titleMedium,
                maxLines = 1,
                overflow = TextOverflow.Ellipsis,
                color = MaterialTheme.colorScheme.onSurface
            )
            // FIX 1: Use `let` to create a local, smart-casted variable
            artist?.let { artistText ->
                Text(
                    text = artistText,
                    style = MaterialTheme.typography.bodyMedium,
                    maxLines = 1,
                    overflow = TextOverflow.Ellipsis,
                    color = MaterialTheme.colorScheme.onSurfaceVariant
                )
            }
        }

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

        IconButton(
            onClick = { playbackViewModel.togglePlayPause() },
            enabled = currentItemExists && !isLoading
        ) {
            Icon(
```

```
                                imageVector = if (isPlaying) Icons.Filled.Pause else Icons.Fil
                                contentDescription = if (isPlaying) stringResource(R.string.ac
                                modifier = Modifier.size(36.dp),
                                tint = MaterialTheme.colorScheme.primary
                            )
                        }

                        IconButton(
                            onClick = { playbackViewModel.skipToNext() },
                            enabled = canSkipNext && !isLoading
                        ) {
                            Icon(
                                imageVector = Icons.Filled.SkipNext,
                                contentDescription = stringResource(R.string.action_next),
                                modifier = Modifier.size(36.dp),
                                tint = MaterialTheme.colorScheme.onSurfaceVariant
                            )
                        }
                    }

                    if (isLoading && !currentItemExists) {
                        LinearProgressIndicator(modifier = Modifier.fillMaxWidth().height(2.dp
                    } else if (currentItemExists) {
                        LinearProgressIndicator(
                            progress = { progressFraction },
                            modifier = Modifier.fillMaxWidth().height(2.dp),
                            color = MaterialTheme.colorScheme.primary,
                            trackColor = MaterialTheme.colorScheme.surfaceVariant.copy(alpha =
                        )
                    }
                }
            }
        }
    }
}

// File: java\com\example\holodex\ui\composables\PlayerBackground.kt
package com.example.holodex.ui.composables

import androidx.compose.animation.animateColorAsState
import androidx.compose.animation.core.tween
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.blur
import androidx.compose.ui.graphics.Brush
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.ColorFilter
import androidx.compose.ui.graphics.ColorMatrix
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.unit.dp
```

```
import coil.compose.AsyncImage

/**
 * A highly optimized, reusable composable that displays a blurred and themed background
 * based on album artwork. It's designed for performance by minimizing overdraw and
 * caching expensive graphics objects.
 *
 * @param artworkUri The URL of the artwork to display in the background.
 * @param dynamicColor The dominant color extracted from the artwork, used for the gradient ov
 * @param modifier The modifier to be applied to this composable.
 */
@Composable
fun SimpleProcessedBackground(
    artworkUri: String?,
    dynamicColor: Color,
    modifier: Modifier = Modifier,
    blurRadius: Int = 80,
    saturation: Float = 0.5f,
    darkenFactor: Float = 0.7f
) {
    val animatedPrimaryColor by animateColorAsState(
        targetValue = dynamicColor,
        label = "animated_primary_color_background",
        animationSpec = tween(1200)
    )

    val colorFilter = remember(saturation, darkenFactor) {
        ColorFilter.colorMatrix(
            ColorMatrix().apply {
                setToSaturation(saturation)
                val values = floatArrayOf(
                    darkenFactor, 0f, 0f, 0f, 0f,
                    0f, darkenFactor, 0f, 0f, 0f,
                    0f, 0f, darkenFactor, 0f, 0f,
                    0f, 0f, 0f, 1f, 0f
                )
                timesAssign(ColorMatrix(values))
            }
        )
    }

    val gradientBrush = remember(animatedPrimaryColor) {
        Brush.verticalGradient(
            colors = listOf(
                animatedPrimaryColor.copy(alpha = 0.2f),
                animatedPrimaryColor.copy(alpha = 0.4f),
                Color.Black.copy(alpha = 0.7f)
            )
        )
    }

    Box(modifier = modifier.fillMaxSize()) {
        AsyncImage(
            model = artworkUri,
            contentDescription = "Background artwork",
```

```
            contentScale = ContentScale.Crop,
            modifier = Modifier
                .fillMaxSize()
                .blur(radius = blurRadius.dp),
            colorFilter = colorFilter
        )
        Box(
            modifier = Modifier
                .fillMaxSize()
                .background(gradientBrush)
        )
    }
}

// File: java\com\example\holodex\ui\composables\PlaylistArtwork.kt
package com.example.holodex.ui.composables

import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.aspectRatio
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Podcasts
import androidx.compose.material3.Icon
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.remember
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.draw.shadow
import androidx.compose.ui.graphics.Brush
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.text.style.TextOverflow
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import coil.compose.AsyncImage
import coil.request.ImageRequest
import com.example.holodex.R
import com.example.holodex.data.model.discovery.PlaylistStub
import com.example.holodex.util.ArtworkResolver
import com.example.holodex.util.PlaylistFormatter
import java.util.Locale
```

```
import kotlin.math.max

@Composable
fun PlaylistArtwork(
    playlist: PlaylistStub,
    modifier: Modifier = Modifier
) {
    val context = LocalContext.current

    val artworkUrl = remember(playlist.id) {
        ArtworkResolver.getPlaylistArtworkUrl(playlist)
    }
    val (type, title) = remember(playlist.id, playlist.title) {
        val formattedTitle = PlaylistFormatter.getDisplayTitle(playlist, context) { englishNam
            japaneseName?.takeIf { it.isNotBlank() } ?: englishName
        }
        val playlistType = if (playlist.id.startsWith(":")) {
            playlist.id.substringBefore('[')
        } else {
            "ugp"
        }
        playlistType to formattedTitle
    }

    Surface(
        modifier = modifier
            .fillMaxWidth()
            .aspectRatio(1f)
            .clip(MaterialTheme.shapes.medium),
        shadowElevation = 2.dp
    ) {
        when (type) {
            ":artist", ":hot" -> RadioTextArt(
                titleText = title,
                imageUrl = artworkUrl,
            )
            ":dailyrandom", ":weekly", ":mv", ":latest" -> {
                val lastSpaceIndex = title.lastIndexOf(' ')
                val (typeText, titleText) = if (lastSpaceIndex > 0 && title.length > lastSpace
                    title.substring(0, lastSpaceIndex) to title.substring(lastSpaceIndex + 1)
                } else {
                    title.substringBefore(": ") to title.substringAfter(": ", title)
                }
                StackedTextArt(
                    typeText = typeText,
                    titleText = titleText,
                    imageUrl = artworkUrl
                )
            }
            else -> OverlayTextArt(
                titleText = title,
                imageUrl = artworkUrl
            )
        }
    }
```

```
}

@Composable
private fun OverlayTextArt(
    titleText: String,
    imageUrl: String?,
    modifier: Modifier = Modifier
) {
    Box(modifier = modifier.fillMaxSize(), contentAlignment = Alignment.BottomStart) {
        AsyncImage(
            model = ImageRequest.Builder(LocalContext.current)
                .data(imageUrl)
                .placeholder(R.drawable.ic_placeholder_image)
                .error(R.drawable.ic_error_image)
                .crossfade(true)
                .build(),
            contentDescription = titleText,
            contentScale = ContentScale.Crop,
            modifier = Modifier.fillMaxSize()
        )
        Box(
            modifier = Modifier
                .fillMaxSize()
                .background(
                    Brush.verticalGradient(
                        colors = listOf(Color.Transparent, Color.Black.copy(alpha = 0.8f)),
                        startY = 150f
                    )
                )
        )
        Text(
            text = titleText,
            style = MaterialTheme.typography.bodyLarge,
            fontWeight = FontWeight.Bold,
            color = Color.White,
            maxLines = 3,
            overflow = TextOverflow.Ellipsis,
            modifier = Modifier.padding(12.dp)
        )
    }
}

@Composable
private fun RadioTextArt(
    titleText: String,
    imageUrl: String?,
    modifier: Modifier = Modifier
) {
    Box(
        modifier = modifier
            .fillMaxSize()
            .background(
                Brush.radialGradient(
                    colors = listOf(
                        MaterialTheme.colorScheme.onSurface.copy(alpha = 0.2f),
```

```
                        MaterialTheme.colorScheme.surfaceVariant.copy(alpha = 0.1f),
                    ),
                    radius = 250f
                )
            )
            .background(MaterialTheme.colorScheme.surfaceVariant),
        contentAlignment = Alignment.Center
    ) {
        Icon(
            imageVector = Icons.Default.Podcasts,
            contentDescription = null,
            modifier = Modifier
                .fillMaxSize(0.9f)
                .align(Alignment.Center),
            tint = MaterialTheme.colorScheme.onSurface.copy(alpha = 0.1f)
        )
        Column(
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.spacedBy(4.dp),
            modifier = Modifier.padding(8.dp)
        ) {
            AsyncImage(
                model = ImageRequest.Builder(LocalContext.current)
                    .data(imageUrl)
                    .placeholder(R.drawable.ic_placeholder_image)
                    .error(R.drawable.ic_error_image)
                    .crossfade(true)
                    .build(),
                contentDescription = titleText,
                contentScale = ContentScale.Crop,
                modifier = Modifier
                    .fillMaxWidth(0.6f)
                    .aspectRatio(1f)
                    .shadow(elevation = 8.dp, shape = CircleShape)
                    .clip(CircleShape)
            )
            Text(
                text = titleText,
                style = MaterialTheme.typography.titleSmall,
                textAlign = TextAlign.Center,
                maxLines = 1,
                overflow = TextOverflow.Ellipsis,
                modifier = Modifier.padding(top = 4.dp)
            )
            Text(
                text = stringResource(id = R.string.sgp_radio_type),
                style = MaterialTheme.typography.bodySmall,
                color = MaterialTheme.colorScheme.onSurfaceVariant,
                textAlign = TextAlign.Center
            )
        }
    }
}

@Composable
```

```
private fun StackedTextArt(
    typeText: String,
    titleText: String,
    imageUrl: String?,
    modifier: Modifier = Modifier
) {
    val adjFontSize = max(14, (18 - (titleText.length / 15))).sp

    Column(modifier = modifier.fillMaxSize()) {
        Column(
            modifier = Modifier
                .fillMaxWidth()
                .weight(0.4f)
                .background(MaterialTheme.colorScheme.primaryContainer)
                .padding(horizontal = 12.dp, vertical = 4.dp),
            verticalArrangement = Arrangement.Center
        ) {
            Text(
                text = typeText.toUpperCase(Locale.getDefault()),
                style = MaterialTheme.typography.labelMedium,
                color = MaterialTheme.colorScheme.onPrimaryContainer.copy(alpha = 0.7f),
            )
            Text(
                text = titleText,
                style = MaterialTheme.typography.titleMedium.copy(fontSize = adjFontSize),
                color = MaterialTheme.colorScheme.onPrimaryContainer,
                maxLines = 2,
                overflow = TextOverflow.Ellipsis,
                fontWeight = FontWeight.Bold
            )
        }
        AsyncImage(
            model = ImageRequest.Builder(LocalContext.current)
                .data(imageUrl)
                .placeholder(R.drawable.ic_placeholder_image)
                .error(R.drawable.ic_error_image)
                .crossfade(true)
                .build(),
            contentDescription = titleText,
            contentScale = ContentScale.Crop,
            modifier = Modifier
                .fillMaxWidth()
                .weight(0.6f)
        )
    }
}

// File: java\com\example\holodex\ui\composables\PlaylistCard.kt
package com.example.holodex.ui.composables

import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.width
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.remember
import androidx.compose.ui.Modifier
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.text.style.TextOverflow
import androidx.compose.ui.unit.dp
import com.example.holodex.data.model.discovery.PlaylistStub
import com.example.holodex.util.PlaylistFormatter

@Composable
fun PlaylistCard(
    playlist: PlaylistStub,
    onPlaylistClicked: (PlaylistStub) -> Unit,
    modifier: Modifier = Modifier
) {
    val context = LocalContext.current

    // --- START OF IMPLEMENTATION ---
    val (type, displayTitle, displayDescription) = remember(playlist) {
        val playlistType = if (playlist.id.startsWith(":")) playlist.id.substringBefore('[') e
        val title = PlaylistFormatter.getDisplayTitle(playlist, context) { en, jp -> jp?.takeI
        val description = PlaylistFormatter.getDisplayDescription(playlist, context) { en, jp
        Triple(playlistType, title, description)
    }

    val textToShow = when (type) {
        ":artist", ":hot" -> displayDescription ?: displayTitle
        ":dailyrandom", ":weekly", ":mv", ":latest" -> displayDescription ?: displayTitle
        else -> displayTitle // For UGP and others, show the title
    }
    // --- END OF IMPLEMENTATION ---

    Card(
        modifier = modifier
            .width(140.dp)
            .clickable { onPlaylistClicked(playlist) },
        colors = CardDefaults.cardColors(containerColor = MaterialTheme.colorScheme.surface)
    ) {
        Column {
            PlaylistArtwork(
                playlist = playlist,
                modifier = Modifier
            )
            Text(
                text = textToShow,
                style = MaterialTheme.typography.bodyMedium,
                maxLines = 2,
                overflow = TextOverflow.Ellipsis,
                modifier = Modifier.padding(8.dp)
            )
        }
    }
```

```
}

// File: java\com\example\holodex\ui\composables\PlaylistManagementDialogs.kt
package com.example.holodex.ui.composables

import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.lifecycle.compose.collectAsStateWithLifecycle
import com.example.holodex.ui.dialogs.CreatePlaylistDialog
import com.example.holodex.ui.dialogs.SelectPlaylistDialog
import com.example.holodex.viewmodel.PlaylistManagementViewModel

@Composable
fun PlaylistManagementDialogs(
    playlistManagementViewModel: PlaylistManagementViewModel
) {
    // Using standard lifecycle collection for these booleans
    val showSelectPlaylistDialog by playlistManagementViewModel.showSelectPlaylistDialog.colle
    val userPlaylistsForDialog by playlistManagementViewModel.userPlaylists.collectAsStateWith
    val showCreatePlaylistDialog by playlistManagementViewModel.showCreatePlaylistDialog.colle

    if (showSelectPlaylistDialog) {
        SelectPlaylistDialog(
            playlists = userPlaylistsForDialog,
            onDismissRequest = { playlistManagementViewModel.cancelAddToPlaylistFlow() },
            onPlaylistSelected = { playlist -> playlistManagementViewModel.addItemToExistingPl
            onCreateNewPlaylistClicked = { playlistManagementViewModel.handleCreateNewPlaylist
        )
    }
    if (showCreatePlaylistDialog) {
        CreatePlaylistDialog(
            onDismissRequest = { playlistManagementViewModel.cancelAddToPlaylistFlow() },
            onCreatePlaylist = { name, desc -> playlistManagementViewModel.confirmCreatePlayli
        )
    }
}

// File: java\com\example\holodex\ui\composables\StateDisplayComposables.kt
package com.example.holodex.ui.composables

import androidx.compose.foundation.background
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.height
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.shape.RoundedCornerShape
import androidx.compose.material.icons.Icons
```

```
import androidx.compose.material.icons.filled.ErrorOutline
import androidx.compose.material.icons.filled.MusicOff
import androidx.compose.material.icons.filled.Refresh
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.CircularProgressIndicator
import androidx.compose.material3.Icon
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import com.example.holodex.R

@Composable
fun LoadingState(message: String, modifier: Modifier = Modifier) {
    Box(modifier = modifier.fillMaxSize(), contentAlignment = Alignment.Center) {
        Column(
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.Center
        ) {
            CircularProgressIndicator()
            Spacer(modifier = Modifier.height(16.dp))
            Text(
                text = message,
                style = MaterialTheme.typography.bodyLarge,
                textAlign = TextAlign.Center
            )
        }
    }
}

@Composable
fun LoadingSkeleton(modifier: Modifier = Modifier, itemCount: Int = 10) {
    LazyColumn(modifier = modifier) {
        items(itemCount) {
            Row(
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(horizontal = 16.dp, vertical = 8.dp),
                verticalAlignment = Alignment.CenterVertically
            ) {
                Box(
                    modifier = Modifier
                        .size(56.dp)
                        .clip(RoundedCornerShape(8.dp))
                        .background(MaterialTheme.colorScheme.surfaceVariant.copy(alpha = 0.3f
                )
                Spacer(modifier = Modifier.width(12.dp))
                Column(verticalArrangement = Arrangement.spacedBy(6.dp)) {
                    Box(
```

```
                        modifier = Modifier
                            .fillMaxWidth(0.7f)
                            .height(18.dp)
                            .clip(RoundedCornerShape(4.dp))
                            .background(MaterialTheme.colorScheme.surfaceVariant.copy(alpha =
                    )
                    Box(
                        modifier = Modifier
                            .fillMaxWidth(0.5f)
                            .height(14.dp)
                            .clip(RoundedCornerShape(4.dp))
                            .background(MaterialTheme.colorScheme.surfaceVariant.copy(alpha =
                    )
                }
            }
        }
    }
}

@Composable
fun EmptyState(message: String, onRefresh: () -> Unit, modifier: Modifier = Modifier) {
    Box(
        modifier = modifier.fillMaxSize().padding(16.dp),
        contentAlignment = Alignment.Center
    ) {
        Column(
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.Center
        ) {
            Icon(
                imageVector = Icons.Filled.MusicOff,
                contentDescription = null,
                modifier = Modifier.size(64.dp),
                tint = MaterialTheme.colorScheme.onSurfaceVariant.copy(alpha = 0.4f)
            )
            Spacer(modifier = Modifier.height(16.dp))
            Text(
                text = message,
                color = MaterialTheme.colorScheme.onSurfaceVariant,
                textAlign = TextAlign.Center,
                style = MaterialTheme.typography.bodyLarge
            )
            Spacer(modifier = Modifier.height(16.dp))
            Button(onClick = onRefresh) {
                Icon(Icons.Filled.Refresh, contentDescription = stringResource(R.string.action
                Spacer(modifier = Modifier.size(ButtonDefaults.IconSpacing))
                Text(stringResource(R.string.action_refresh))
            }
        }
    }
}

@Composable
fun ErrorStateWithRetry(message: String, onRetry: () -> Unit, modifier: Modifier = Modifier) {
    Box(
```

```
        modifier = modifier.fillMaxSize().padding(16.dp),
        contentAlignment = Alignment.Center
    ) {
        Column(
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.Center
        ) {
            Icon(
                imageVector = Icons.Filled.ErrorOutline,
                contentDescription = null,
                modifier = Modifier.size(64.dp),
                tint = MaterialTheme.colorScheme.error
            )
            Spacer(modifier = Modifier.height(16.dp))
            Text(
                text = message,
                color = MaterialTheme.colorScheme.error,
                textAlign = TextAlign.Center,
                style = MaterialTheme.typography.bodyLarge
            )
            Spacer(modifier = Modifier.height(16.dp))
            Button(onClick = onRetry) {
                Icon(Icons.Filled.Refresh, contentDescription = stringResource(R.string.action
                Spacer(modifier = Modifier.size(ButtonDefaults.IconSpacing))
                Text(stringResource(R.string.action_retry))
            }
        }
    }
}

// File: java\com\example\holodex\ui\composables\UnifiedGridItem.kt
package com.example.holodex.ui.composables

import androidx.compose.foundation.background
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.aspectRatio
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.shape.RoundedCornerShape
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.CloudDone
import androidx.compose.material3.Card
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableIntStateOf
import androidx.compose.runtime.remember
```

```
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.Brush
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextOverflow
import androidx.compose.ui.unit.dp
import coil.compose.AsyncImage
import coil.request.ImageRequest
import com.example.holodex.R
import com.example.holodex.viewmodel.UnifiedDisplayItem

/**
 * A universal, reusable composable for displaying any music-related item in a grid or carouse
 * It intelligently displays badges and context based on the properties of the UnifiedDisplayI
 *
 * @param item The canonical UnifiedDisplayItem containing all necessary data for display.
 * @param onClick The action to perform when the card is clicked.
 * @param modifier The modifier to be applied to the Card.
 */
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun UnifiedGridItem(
    item: UnifiedDisplayItem,
    onClick: () -> Unit,
    modifier: Modifier = Modifier,
) {
    var currentUrlIndex by remember(item.artworkUrls) { mutableIntStateOf(0) }

    Card(
        onClick = onClick,
        modifier = modifier.width(140.dp)
    ) {
        Column {
            Box(contentAlignment = Alignment.BottomStart) {
                // Main Artwork Image
                AsyncImage(
                    model = ImageRequest.Builder(LocalContext.current)
                        .data(item.artworkUrls.getOrNull(currentUrlIndex))
                        .placeholder(R.drawable.ic_placeholder_image)
                        .error(R.drawable.ic_error_image)
                        .crossfade(true)
                        .build(),
                    onError = {
                        // If the primary URL fails, try the next one in the prioritized list
                        if (currentUrlIndex < item.artworkUrls.lastIndex) {
                            currentUrlIndex++
                        }
                    },
                    contentDescription = stringResource(R.string.content_desc_album_art_for, i
```

```
            contentScale = ContentScale.Crop,
            modifier = Modifier
                .fillMaxWidth()
                .aspectRatio(1f)
                .clip(MaterialTheme.shapes.medium)
        )

        // Gradient scrim for text readability
        Box(
            modifier = Modifier
                .matchParentSize()
                .background(
                    Brush.verticalGradient(
                        colors = listOf(Color.Transparent, Color.Black.copy(alpha = 0.
                        startY = 100f // Start gradient lower down
                    )
                )
        )

        // Badges and Duration, overlaid on the artwork
        Row(
            modifier = Modifier
                .fillMaxWidth()
                .padding(6.dp),
            horizontalArrangement = Arrangement.End,
            verticalAlignment = Alignment.CenterVertically
        ) {
            if (item.isDownloaded) {
                Icon(
                    imageVector = Icons.Filled.CloudDone,
                    contentDescription = stringResource(R.string.content_description_d
                    tint = Color.White,
                    modifier = Modifier
                        .size(16.dp)
                        .padding(end = 4.dp)
                )
            }
            Text(
                text = item.durationText,
                style = MaterialTheme.typography.labelSmall,
                color = Color.White,
                fontWeight = FontWeight.Bold,
                modifier = Modifier
                    .background(Color.Black.copy(alpha = 0.5f), RoundedCornerShape(4.d
                    .padding(horizontal = 4.dp, vertical = 2.dp)
            )
        }
    }

    // Text content below the artwork
    Column(
        modifier = Modifier.padding(8.dp),
        verticalArrangement = Arrangement.spacedBy(2.dp)
    ) {
        Text(
```

```
                    text = item.title,
                    style = MaterialTheme.typography.titleSmall,
                    maxLines = 2,
                    overflow = TextOverflow.Ellipsis
                )
                Text(
                    text = item.artistText,
                    style = MaterialTheme.typography.bodySmall,
                    maxLines = 1,
                    overflow = TextOverflow.Ellipsis,
                    color = MaterialTheme.colorScheme.onSurfaceVariant
                )
            }
        }
    }
}

// File: java\com\example\holodex\ui\composables\UnifiedListItem.kt
// File: java/com/example/holodex/ui/composables/UnifiedListItem.kt
package com.example.holodex.ui.composables

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.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.layout.width
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.CloudDone
import androidx.compose.material.icons.filled.Delete
import androidx.compose.material.icons.filled.DragHandle
import androidx.compose.material.icons.filled.ErrorOutline
import androidx.compose.material.icons.filled.Favorite
import androidx.compose.material.icons.filled.FavoriteBorder
import androidx.compose.material.icons.filled.MoreVert
import androidx.compose.material3.Card
import androidx.compose.material3.CardDefaults
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableIntStateOf
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.draw.clip
```

```
import androidx.compose.ui.hapticfeedback.HapticFeedbackType
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.platform.LocalHapticFeedback
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextOverflow
import androidx.compose.ui.unit.dp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.media3.common.util.UnstableApi
import androidx.navigation.NavController
import coil.compose.AsyncImage
import coil.request.ImageRequest
import com.example.holodex.R
import com.example.holodex.ui.navigation.AppDestinations
import com.example.holodex.viewmodel.DownloadsViewModel
import com.example.holodex.viewmodel.FavoritesViewModel
import com.example.holodex.viewmodel.PlaylistManagementViewModel
import com.example.holodex.viewmodel.UnifiedDisplayItem
import com.example.holodex.viewmodel.VideoDetailsViewModel
import com.example.holodex.viewmodel.VideoListViewModel
import com.example.holodex.viewmodel.mappers.toPlaybackItem
import org.orbitmvi.orbit.compose.collectAsState

@UnstableApi
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun UnifiedListItem(
    item: UnifiedDisplayItem,
    onItemClicked: () -> Unit,
    navController: NavController,
    videoListViewModel: VideoListViewModel,
    favoritesViewModel: FavoritesViewModel,
    playlistManagementViewModel: PlaylistManagementViewModel,
    modifier: Modifier = Modifier,
    isEditing: Boolean = false,
    onRemoveClicked: () -> Unit = {},
    dragHandleModifier: Modifier = Modifier,
    isExternal: Boolean = false
) {
    val haptic = LocalHapticFeedback.current
    var currentUrlIndex by remember(item.artworkUrls) { mutableIntStateOf(0) }
    var showOptionsMenu by remember { mutableStateOf(false) }
    val favoritesState by favoritesViewModel.collectAsState()


    val isItemLiked = remember(item.playbackItemId, favoritesState.likedItemsMap) {
        favoritesState.likedItemsMap.containsKey(item.playbackItemId)
    }
    val videoDetailsViewModel: VideoDetailsViewModel = hiltViewModel()
    val downloadsViewModel: DownloadsViewModel = hiltViewModel()

    val menuState = remember(item) {
        ItemMenuState(
            isDownloaded = item.isDownloaded,
```

```
        isSegment = item.isSegment,
        canBeDownloaded = item.isSegment && !item.isDownloaded,
        shareUrl = if (item.isSegment && item.songStartSec != null) {
            "https://music.holodex.net/watch/${item.videoId}/${item.songStartSec}"
        } else {
            "https://music.holodex.net/watch/${item.videoId}"
        },
        videoId = item.videoId,
        channelId = item.channelId
    )
}

val menuActions = remember(item) {
    ItemMenuActions(
        onAddToQueue = { videoListViewModel.addVideoOrItsSegmentsToQueue(item.toPlaybackIt
        onAddToPlaylist = {
            playlistManagementViewModel.prepareItemForPlaylistAddition(item)
        },
        onShare = { /* Handled internally by the menu now */ },
        onDownload = { videoDetailsViewModel.requestDownloadForSongFromPlaybackItem(item.t
        onDelete = { downloadsViewModel.deleteDownload(item.playbackItemId) },
        onGoToVideo = { videoId -> navController.navigate(AppDestinations.videoDetailRoute
        onGoToArtist = { channelId -> navController.navigate("channel_details/$channelId")

    )
}

Card(
    onClick = {
        haptic.performHapticFeedback(HapticFeedbackType.TextHandleMove)
        onItemClicked()
    },
    modifier = modifier
        .fillMaxWidth()
        .padding(horizontal = 16.dp, vertical = 6.dp),
    elevation = CardDefaults.cardElevation(defaultElevation = 2.dp),
    shape = MaterialTheme.shapes.medium
) {
    Row(
        modifier = Modifier
            .fillMaxWidth()
            .padding(start = 12.dp, top = 8.dp, bottom = 8.dp, end = 4.dp),
        verticalAlignment = Alignment.CenterVertically
    ) {
        AsyncImage(
            model = ImageRequest.Builder(LocalContext.current)
                .data(item.artworkUrls.getOrNull(currentUrlIndex))
                .placeholder(R.drawable.ic_placeholder_image)
                .error(R.drawable.ic_error_image)
                .crossfade(true)
                .build(),
            onError = {
                if (currentUrlIndex < item.artworkUrls.lastIndex) {
                    currentUrlIndex++
                }
```

```
        },
        contentDescription = stringResource(R.string.content_desc_channel_thumbnail),
        contentScale = ContentScale.Crop,
        modifier = Modifier
            .size(56.dp)
            .clip(RoundedCornerShape(8.dp))
    )

    Spacer(modifier = Modifier.width(12.dp))

    Column(
        modifier = Modifier
            .weight(1f)
            .padding(end = 4.dp),
        verticalArrangement = Arrangement.spacedBy(3.dp)
    ) {
        Row(verticalAlignment = Alignment.CenterVertically) {
            Text(
                text = item.title,
                style = MaterialTheme.typography.titleSmall,
                fontWeight = FontWeight.Medium,
                maxLines = 2,
                overflow = TextOverflow.Ellipsis,
                color = MaterialTheme.colorScheme.onSurface,
                modifier = Modifier.weight(1f, fill = false)
            )

            // --- CHANGED SECTION ---
            when (item.downloadStatus) {
                "COMPLETED" -> {
                    Spacer(Modifier.width(6.dp))
                    Icon(
                        Icons.Filled.CloudDone,
                        contentDescription = "Downloaded",
                        tint = MaterialTheme.colorScheme.primary,
                        modifier = Modifier.size(16.dp)
                    )
                }
                "FAILED", "EXPORT_FAILED" -> {
                    Spacer(Modifier.width(6.dp))
                    Icon(
                        Icons.Filled.ErrorOutline, // Make sure to import this
                        contentDescription = "Download Failed",
                        tint = MaterialTheme.colorScheme.error,
                        modifier = Modifier.size(16.dp)
                    )
                }
                "DOWNLOADING", "PROCESSING", "ENQUEUED" -> {
                    Spacer(Modifier.width(6.dp))
                    androidx.compose.material3.CircularProgressIndicator(
                        modifier = Modifier.size(12.dp),
                        strokeWidth = 2.dp,
                        color = MaterialTheme.colorScheme.secondary
                    )
                }
```

```
            }
            // ----------------------
        }

        Text(
            text = item.artistText,
            style = MaterialTheme.typography.bodyMedium,
            maxLines = 1,
            overflow = TextOverflow.Ellipsis,
            color = MaterialTheme.colorScheme.onSurfaceVariant
        )

        Row(verticalAlignment = Alignment.CenterVertically) {
            if (item.durationText.isNotEmpty()) {
                Text(
                    text = item.durationText,
                    style = MaterialTheme.typography.bodySmall,
                    color = MaterialTheme.colorScheme.onSurfaceVariant
                )
            }
            if (!item.isSegment) {
                item.songCount?.let { count ->
                    if (count > 0) {
                        if (item.durationText.isNotEmpty()) {
                            Text(
                                " ? ",
                                style = MaterialTheme.typography.bodySmall,
                                color = MaterialTheme.colorScheme.onSurfaceVariant
                            )
                        }
                        Text(
                            stringResource(R.string.song_count_format, count),
                            style = MaterialTheme.typography.bodySmall,
                            color = MaterialTheme.colorScheme.onSurfaceVariant
                        )
                    }
                }
            }
        }
    }

    if (isEditing) {
        IconButton(onClick = onRemoveClicked) {
            Icon(
                Icons.Default.Delete,
                contentDescription = stringResource(R.string.action_remove_from_playli
                tint = MaterialTheme.colorScheme.error
            )
        }
        Box(modifier = dragHandleModifier) {
            Icon(
                Icons.Default.DragHandle,
                contentDescription = stringResource(R.string.drag_to_reorder)
            )
        }
```

```
            } else {
                IconButton(
                    onClick = {
                        val playbackItem = item.toPlaybackItem().copy(isExternal = isExternal)
                        favoritesViewModel.toggleLike(playbackItem)
                        haptic.performHapticFeedback(HapticFeedbackType.LongPress)
                    },
                    modifier = Modifier.size(40.dp)
                ) {
                    Icon(
                        imageVector = if (isItemLiked) Icons.Filled.Favorite else Icons.Filled
                        contentDescription = if (isItemLiked) stringResource(R.string.action_u
                        tint = if (isItemLiked) MaterialTheme.colorScheme.primary else Materia
                        modifier = Modifier.size(24.dp)
                    )
                }

                Box {
                    IconButton(onClick = { showOptionsMenu = true }, modifier = Modifier.size(
                        Icon(Icons.Filled.MoreVert, stringResource(R.string.action_more_option
                    }
                    ItemOptionsMenu(
                        state = menuState,
                        actions = menuActions,
                        expanded = showOptionsMenu,
                        onDismissRequest = { showOptionsMenu = false }
                    )
                }
            }
        }
    }
}

// File: java\com\example\holodex\ui\composables\sheets\BrowseFiltersSheet.kt
@file:OptIn(androidx.media3.common.util.UnstableApi::class)
package com.example.holodex.ui.composables.sheets

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.height
import androidx.compose.foundation.layout.navigationBarsPadding
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.width
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.verticalScroll
import androidx.compose.material3.Button
import androidx.compose.material3.DropdownMenuItem
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.ExposedDropdownMenuBox
import androidx.compose.material3.ExposedDropdownMenuDefaults
import androidx.compose.material3.MaterialTheme
```

```
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.RadioButton
import androidx.compose.material3.Text
import androidx.compose.material3.TextButton
import androidx.compose.runtime.Composable
import androidx.compose.runtime.derivedStateOf
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.res.stringResource
import androidx.compose.ui.unit.dp
import androidx.hilt.navigation.compose.hiltViewModel
import com.example.holodex.R
import com.example.holodex.viewmodel.VideoListViewModel
import com.example.holodex.viewmodel.state.BrowseFilterState
import com.example.holodex.viewmodel.state.SortOrder
import com.example.holodex.viewmodel.state.VideoSortField
import com.example.holodex.viewmodel.state.ViewTypePreset
import org.orbitmvi.orbit.compose.collectAsState

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun BrowseFiltersSheet(
    initialFilters: BrowseFilterState,
    onFiltersApplied: (BrowseFilterState) -> Unit,
    onDismiss: () -> Unit,
    videoListViewModel: VideoListViewModel = hiltViewModel()
) {
    var tempFilters by remember { mutableStateOf(initialFilters) }

    val videoListState by videoListViewModel.collectAsState()
    val organizationsForDropdown = videoListState.availableOrganizations

    val viewTypePresetOptions = videoListViewModel.browseScreenCategories

    var viewPresetExpanded by remember { mutableStateOf(false) }
    var orgExpanded by remember { mutableStateOf(false) }
    var sortFieldExpanded by remember { mutableStateOf(false) }

    Column(
        modifier = Modifier
            .fillMaxWidth()
            .navigationBarsPadding()
            .padding(16.dp)
            .verticalScroll(rememberScrollState())
    ) {
        Text(
            text = "Filter & Sort Music Streams",
            style = MaterialTheme.typography.titleLarge,
            modifier = Modifier.padding(bottom = 16.dp)
        )
```

```
// --- VIEW TYPE PRESET ---
FilterSectionHeader("View As")
val currentViewPresetDisplay by remember(tempFilters, organizationsForDropdown) {
    derivedStateOf {
        val orgDisplayPart = if (tempFilters.selectedOrganization != null) {
            val orgName = organizationsForDropdown.find { it.second == tempFilters.sel
            if (orgName != "All Vtubers") " - $orgName" else ""
        } else ""

        "${tempFilters.selectedViewPreset.defaultDisplayName}$orgDisplayPart"
    }
}

ExposedDropdownMenuBox(
    expanded = viewPresetExpanded,
    onExpandedChange = { viewPresetExpanded = it },
    modifier = Modifier.fillMaxWidth()
) {
    OutlinedTextField(
        value = currentViewPresetDisplay,
        onValueChange = {}, readOnly = true, label = { Text("View Type") },
        trailingIcon = { ExposedDropdownMenuDefaults.TrailingIcon(expanded = viewPrese
        modifier = Modifier.menuAnchor().fillMaxWidth()
    )
    ExposedDropdownMenu(expanded = viewPresetExpanded, onDismissRequest = { viewPreset
        viewTypePresetOptions.forEach { (displayName, presetBrowseFilterStateFromVM) -
            DropdownMenuItem(
                text = { Text(displayName) },
                onClick = {
                    val currentOrgApiVal = tempFilters.selectedOrganization
                    val currentPrimaryTopic = tempFilters.selectedPrimaryTopic
                    val currentSortField = tempFilters.sortField
                    val currentSortOrder = tempFilters.sortOrder

                    var newFilterState = BrowseFilterState.create(
                        preset = presetBrowseFilterStateFromVM.selectedViewPreset,
                        organization = currentOrgApiVal,
                        primaryTopic = currentPrimaryTopic,
                        sortFieldOverride = if (presetBrowseFilterStateFromVM.selected
                        sortOrderOverride = if (presetBrowseFilterStateFromVM.selected
                    )

                    // Logic to ensure valid sort for preset
                    if (newFilterState.selectedViewPreset == ViewTypePreset.LATEST_STR
                        if (newFilterState.sortField == VideoSortField.START_SCHEDULED
                            newFilterState = newFilterState.copy(
                                sortField = VideoSortField.AVAILABLE_AT,
                                sortOrder = SortOrder.DESC
                            )
                        }
                    } else if (newFilterState.selectedViewPreset == ViewTypePreset.UPC
                        if (newFilterState.sortField != VideoSortField.START_SCHEDULED
                            newFilterState = newFilterState.copy(
                                sortField = VideoSortField.START_SCHEDULED,
                                sortOrder = SortOrder.ASC
```

```
                            )
                        }
                    }

                    tempFilters = newFilterState
                    viewPresetExpanded = false
                }
            )
        }
    }
}
Spacer(Modifier.height(16.dp))

// --- ORGANIZATION ---
FilterSectionHeader("Organization")
val currentOrgDisplay = organizationsForDropdown.find { it.second == tempFilters.selec
ExposedDropdownMenuBox(expanded = orgExpanded, onExpandedChange = { orgExpanded = it }
    OutlinedTextField(
        value = currentOrgDisplay, onValueChange = {}, readOnly = true, label = { Text
        trailingIcon = { ExposedDropdownMenuDefaults.TrailingIcon(expanded = orgExpand
        modifier = Modifier.menuAnchor().fillMaxWidth()
    )
    ExposedDropdownMenu(expanded = orgExpanded, onDismissRequest = { orgExpanded = fal
        organizationsForDropdown.forEach { (name, value) ->
            DropdownMenuItem(text = { Text(name) }, onClick = {
                tempFilters = BrowseFilterState.create(
                    preset = tempFilters.selectedViewPreset,
                    organization = value,
                    primaryTopic = tempFilters.selectedPrimaryTopic,
                    sortFieldOverride = tempFilters.sortField,
                    sortOrderOverride = tempFilters.sortOrder,
                )
                orgExpanded = false
            })
        }
    }
}
Spacer(Modifier.height(16.dp))

// --- SORT FIELD ---
FilterSectionHeader("Sort By")
ExposedDropdownMenuBox(expanded = sortFieldExpanded, onExpandedChange = { sortFieldExp
    OutlinedTextField(
        value = tempFilters.sortField.displayName, onValueChange = {}, readOnly = true
        trailingIcon = { ExposedDropdownMenuDefaults.TrailingIcon(expanded = sortField
        modifier = Modifier.menuAnchor().fillMaxWidth()
    )
    ExposedDropdownMenu(expanded = sortFieldExpanded, onDismissRequest = { sortFieldEx
        VideoSortField.entries.forEach { field ->
            val isApplicable = when (tempFilters.selectedViewPreset) {
                ViewTypePreset.UPCOMING_STREAMS -> {
                    field == VideoSortField.START_SCHEDULED ||
                            field == VideoSortField.LIVE_VIEWERS ||
                            field == VideoSortField.TITLE ||
                            field == VideoSortField.PUBLISHED_AT
```

```
                        }
                        ViewTypePreset.LATEST_STREAMS -> {
                            field != VideoSortField.START_SCHEDULED && field != VideoSortField
                        }
                    }

                    if (isApplicable) {
                        DropdownMenuItem(text = { Text(field.displayName) }, onClick = {
                            var newSortOrder = tempFilters.sortOrder
                            // Automatically set default sort order based on field
                            if (field == VideoSortField.AVAILABLE_AT || field == VideoSortFiel
                                newSortOrder = SortOrder.DESC
                            } else if (field == VideoSortField.START_SCHEDULED) {
                                newSortOrder = SortOrder.ASC
                            } else if (field == VideoSortField.LIVE_VIEWERS) {
                                newSortOrder = SortOrder.DESC
                            }

                            tempFilters = tempFilters.copy(sortField = field, sortOrder = newS
                            sortFieldExpanded = false
                        })
                    }
                }
            }
        }
        Spacer(Modifier.height(8.dp))

        // --- SORT ORDER ---
        Row(verticalAlignment = Alignment.CenterVertically, modifier = Modifier.fillMaxWidth()
            SortOrder.entries.forEach { order ->
                Row(Modifier.clickable { tempFilters = tempFilters.copy(sortOrder = order) }.p
                    RadioButton(selected = tempFilters.sortOrder == order, onClick = { tempFil
                    Text(order.displayName, style = MaterialTheme.typography.bodyLarge, modifi
                }
            }
        }
        Spacer(Modifier.height(24.dp))

        // --- ACTION BUTTONS ---
        Row(modifier = Modifier.fillMaxWidth(), horizontalArrangement = Arrangement.End) {
            TextButton(onClick = onDismiss) { Text(stringResource(id = R.string.cancel)) }
            Spacer(Modifier.width(8.dp))
            Button(onClick = {
                onFiltersApplied(tempFilters)
            }) { Text(stringResource(id = R.string.apply_filters_button)) }
        }
    }
}

@Composable
fun FilterSectionHeader(title: String) {
    Text(
        text = title,
        style = MaterialTheme.typography.titleSmall,
        modifier = Modifier.padding(top = 12.dp, bottom = 8.dp)
```

```
    )
}

// File: java\com\example\holodex\ui\dialogs\AddExternalChannelDialog.kt
// File: java/com/example/holodex/ui/dialogs/AddExternalChannelDialog.kt
package com.example.holodex.ui.dialogs

import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.heightIn
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.foundation.text.KeyboardActions
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Add
import androidx.compose.material.icons.filled.Clear
import androidx.compose.material.icons.filled.Search
import androidx.compose.material3.Button
import androidx.compose.material3.Card
import androidx.compose.material3.CircularProgressIndicator
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.HorizontalDivider
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.ListItem
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.Text
import androidx.compose.material3.TextButton
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.platform.LocalFocusManager
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.input.ImeAction
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.compose.ui.window.Dialog
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.lifecycle.compose.collectAsStateWithLifecycle
import coil.compose.AsyncImage
import com.example.holodex.R
import com.example.holodex.data.model.ChannelSearchResult
import com.example.holodex.viewmodel.AddChannelViewModel
```

```
import com.example.holodex.viewmodel.state.UiState

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun AddExternalChannelDialog(
    onDismissRequest: () -> Unit,
    viewModel: AddChannelViewModel = hiltViewModel()
) {
    val searchQuery by viewModel.searchQuery.collectAsStateWithLifecycle()
    val searchState by viewModel.searchState.collectAsStateWithLifecycle()
    val isAdding by viewModel.isAdding.collectAsStateWithLifecycle()
    val focusManager = LocalFocusManager.current

    Dialog(onDismissRequest = onDismissRequest) {
        Card(
            modifier = Modifier.fillMaxWidth().heightIn(max = 600.dp),
            shape = MaterialTheme.shapes.large
        ) {
            Column {
                Text(
                    text = "Add External Channel",
                    style = MaterialTheme.typography.titleLarge,
                    modifier = Modifier.padding(16.dp)
                )
                OutlinedTextField(
                    value = searchQuery,
                    onValueChange = { viewModel.onSearchQueryChanged(it) },
                    modifier = Modifier.fillMaxWidth().padding(horizontal = 16.dp),
                    placeholder = { Text("Search YouTube for a channel...") },
                    leadingIcon = { Icon(Icons.Default.Search, null) },
                    trailingIcon = {
                        if (searchQuery.isNotEmpty()) {
                            IconButton(onClick = { viewModel.onSearchQueryChanged("") }) {
                                Icon(Icons.Default.Clear, "Clear search")
                            }
                        }
                    },
                    singleLine = true,
                    keyboardOptions = KeyboardOptions(imeAction = ImeAction.Search),
                    keyboardActions = KeyboardActions(onSearch = { focusManager.clearFocus() }
                )

                Box(
                    modifier = Modifier.weight(1f).fillMaxWidth(),
                    contentAlignment = Alignment.Center
                ) {
                    when (val state = searchState) {
                        is UiState.Loading -> CircularProgressIndicator()
                        is UiState.Error -> Text(state.message, color = MaterialTheme.colorSch
                        is UiState.Success -> {
                            if (state.data.isEmpty() && searchQuery.length > 2) {
                                Text("No channels found.", modifier = Modifier.padding(16.dp))
                            } else {
                                LazyColumn(contentPadding = PaddingValues(16.dp)) {
                                    items(state.data, key = { it.channelId }) { channel ->
```

```
                                    ChannelSearchResultItem(
                                        channel = channel,
                                        isAdding = isAdding.contains(channel.channelId),
                                        onAddClicked = { viewModel.addChannel(channel) }
                                    )
                                }
                            }
                        }
                    }
                }

                HorizontalDivider()
                Row(
                    modifier = Modifier.fillMaxWidth().padding(horizontal = 16.dp, vertical =
                    horizontalArrangement = Arrangement.End
                ) {
                    TextButton(onClick = onDismissRequest) {
                        Text(stringResource(R.string.cancel))
                    }
                }
            }
        }
    }
}

@Composable
private fun ChannelSearchResultItem(
    channel: ChannelSearchResult,
    isAdding: Boolean,
    onAddClicked: () -> Unit
) {
    ListItem(
        headlineContent = { Text(channel.name, maxLines = 1, overflow = androidx.compose.ui.te
        supportingContent = { channel.subscriberCount?.let { Text(it) } },
        leadingContent = {
            AsyncImage(
                model = channel.thumbnailUrl,
                contentDescription = channel.name,
                modifier = Modifier.size(40.dp).clip(CircleShape),
                contentScale = ContentScale.Crop
            )
        },
        trailingContent = {
            Button(onClick = onAddClicked, enabled = !isAdding) {
                if (isAdding) {
                    CircularProgressIndicator(modifier = Modifier.size(18.dp), strokeWidth = 2
                } else {
                    Icon(Icons.Default.Add, null)
                }
            }
        }
    )
}
```

```
// File: java\com\example\holodex\ui\dialogs\CreatePlaylistDialog.kt
package com.example.holodex.ui.dialogs

import androidx.compose.foundation.layout.*
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.unit.dp
import androidx.compose.ui.window.Dialog
import com.example.holodex.R // Assuming strings are in R.string

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun CreatePlaylistDialog(
    onDismissRequest: () -> Unit,
    onCreatePlaylist: (name: String, description: String?) -> Unit
) {
    var playlistName by remember { mutableStateOf("") }
    var playlistDescription by remember { mutableStateOf("") }

    Dialog(onDismissRequest = onDismissRequest) {
        Card(
            modifier = Modifier
                .fillMaxWidth()
                .padding(16.dp),
            shape = MaterialTheme.shapes.large
        ) {
            Column(
                modifier = Modifier.padding(16.dp),
                horizontalAlignment = Alignment.CenterHorizontally
            ) {
                Text(
                    text = stringResource(R.string.dialog_title_create_playlist),
                    style = MaterialTheme.typography.titleLarge,
                    modifier = Modifier.padding(bottom = 16.dp)
                )
                OutlinedTextField(
                    value = playlistName,
                    onValueChange = { playlistName = it },
                    label = { Text(stringResource(R.string.hint_playlist_name)) },
                    singleLine = true,
                    modifier = Modifier.fillMaxWidth()
                )
                Spacer(Modifier.height(8.dp))
                OutlinedTextField(
                    value = playlistDescription,
                    onValueChange = { playlistDescription = it },
                    label = { Text(stringResource(R.string.hint_playlist_description_optional)
                    modifier = Modifier.fillMaxWidth(),
                    maxLines = 3
                )
                Spacer(Modifier.height(24.dp))
                Row(
```

```
                    modifier = Modifier.fillMaxWidth(),
                    horizontalArrangement = Arrangement.End
                ) {
                    TextButton(onClick = onDismissRequest) {
                        Text(stringResource(R.string.cancel))
                    }
                    Spacer(Modifier.width(8.dp))
                    Button(
                        onClick = {
                            onCreatePlaylist(playlistName, playlistDescription.ifBlank { null
                        },
                        enabled = playlistName.isNotBlank()
                    ) {
                        Text(stringResource(R.string.create))
                    }
                }
            }
        }
    }
}

// File: java\com\example\holodex\ui\dialogs\SelectPlaylistDialog.kt
package com.example.holodex.ui.dialogs

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.fillMaxWidth
import androidx.compose.foundation.layout.heightIn
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Add
import androidx.compose.material3.Card
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.HorizontalDivider
import androidx.compose.material3.Icon
import androidx.compose.material3.ListItem
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.material3.TextButton
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.unit.dp
import androidx.compose.ui.window.Dialog
import com.example.holodex.R
import com.example.holodex.data.db.PlaylistEntity

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun SelectPlaylistDialog(
    playlists: List<PlaylistEntity>,
```

```
    onDismissRequest: () -> Unit,
    onPlaylistSelected: (PlaylistEntity) -> Unit,
    onCreateNewPlaylistClicked: () -> Unit
) {
    Dialog(onDismissRequest = onDismissRequest) {
        Card(
            modifier = Modifier.fillMaxWidth().padding(16.dp),
            shape = MaterialTheme.shapes.large
        ) {
            Column {
                Text(
                    text = stringResource(R.string.dialog_title_add_to_playlist),
                    style = MaterialTheme.typography.titleLarge,
                    modifier = Modifier.padding(16.dp)
                )
                HorizontalDivider()
                LazyColumn(modifier = Modifier.heightIn(max = 240.dp)) { // Limit height
                    items(playlists, key = { it.playlistId }) { playlist ->
                        ListItem(
                            headlineContent = { Text(playlist.name ?: "Untitled Playlist") },
                            modifier = Modifier.clickable { onPlaylistSelected(playlist) }
                        )
                        HorizontalDivider()
                    }
                    item {
                        ListItem(
                            headlineContent = { Text(stringResource(R.string.action_create_new
                            leadingContent = { Icon(Icons.Filled.Add, contentDescription = nul
                            modifier = Modifier.clickable(onClick = onCreateNewPlaylistClicked
                        )
                    }
                }
                HorizontalDivider()
                Row(
                    modifier = Modifier.fillMaxWidth().padding(horizontal = 16.dp, vertical =
                    horizontalArrangement = Arrangement.End
                ) {
                    TextButton(onClick = onDismissRequest) {
                        Text(stringResource(R.string.cancel))
                    }
                }
            }
        }
    }
}

// File: java\com\example\holodex\ui\screens\ChannelDetailsScreen.kt
package com.example.holodex.ui.screens

import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.PaddingValues
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.foundation.lazy.LazyColumn
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.automirrored.filled.ArrowBack
import androidx.compose.material.icons.filled.Favorite
import androidx.compose.material.icons.filled.FavoriteBorder
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.OutlinedButton
import androidx.compose.material3.Scaffold
import androidx.compose.material3.Text
import androidx.compose.material3.TextButton
import androidx.compose.material3.TopAppBar
import androidx.compose.material3.TopAppBarDefaults
import androidx.compose.runtime.Composable
import androidx.compose.runtime.derivedStateOf
import androidx.compose.runtime.getValue
import androidx.compose.runtime.remember
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.platform.LocalUriHandler
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.media3.common.util.UnstableApi
import androidx.navigation.NavController
import coil.compose.AsyncImage
import coil.request.ImageRequest
import com.example.holodex.R
import com.example.holodex.data.model.discovery.ChannelDetails
import com.example.holodex.data.model.discovery.DiscoveryChannel
import com.example.holodex.data.model.discovery.SingingStreamShelfItem
import com.example.holodex.ui.composables.CarouselShelf
import com.example.holodex.ui.composables.ChannelCard
import com.example.holodex.ui.composables.ErrorStateWithRetry
import com.example.holodex.ui.composables.LoadingState
import com.example.holodex.ui.composables.SimpleProcessedBackground
import com.example.holodex.ui.composables.UnifiedGridItem
import com.example.holodex.ui.navigation.AppDestinations
import com.example.holodex.util.ThumbnailQuality
import com.example.holodex.util.findActivity
```

```
import com.example.holodex.util.getYouTubeThumbnailUrl
import com.example.holodex.viewmodel.ChannelDetailsViewModel
import com.example.holodex.viewmodel.DiscoveryViewModel
import com.example.holodex.viewmodel.FavoritesViewModel
import com.example.holodex.viewmodel.MusicCategoryType
import com.example.holodex.viewmodel.UnifiedDisplayItem
import com.example.holodex.viewmodel.VideoListViewModel
import com.example.holodex.viewmodel.mappers.toUnifiedDisplayItem
import com.example.holodex.viewmodel.state.UiState
import org.orbitmvi.orbit.compose.collectAsState

@UnstableApi
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun ChannelDetailsScreen(
    navController: NavController,
    onNavigateUp: () -> Unit
) {
    val channelViewModel: ChannelDetailsViewModel = hiltViewModel()
    val favoritesViewModel: FavoritesViewModel = hiltViewModel()
    val discoveryViewModel: DiscoveryViewModel = hiltViewModel()
    val videoListViewModel: VideoListViewModel = hiltViewModel(findActivity())

    val state by channelViewModel.collectAsState()
    val favoritesState by favoritesViewModel.collectAsState()

    val backgroundImageUrl by remember(state) {
        derivedStateOf {
            state.channelDetails?.bannerUrl?.takeIf { it.isNotBlank() }
                ?: state.discoveryContent?.recentSingingStreams?.firstOrNull()?.video?.id?.let
                    getYouTubeThumbnailUrl(it, ThumbnailQuality.MAX).firstOrNull()
                }
        }
    }

    Box(modifier = Modifier.fillMaxSize()) {
        SimpleProcessedBackground(
            artworkUri = backgroundImageUrl,
            dynamicColor = state.dynamicTheme.primary
        )

        Scaffold(
            topBar = {
                TopAppBar(
                    title = {},
                    navigationIcon = {
                        IconButton(onClick = onNavigateUp) {
                            Icon(
                                Icons.AutoMirrored.Filled.ArrowBack,
                                "Back"
                            )
                        }
                    },
                    colors = TopAppBarDefaults.topAppBarColors(containerColor = Color.Transpar
                )
```

```
        },
        containerColor = Color.Transparent
    ) { paddingValues ->
        Box(modifier = Modifier
            .padding(paddingValues)
            .fillMaxSize()) {
            if (state.isLoading) {
                LoadingState(message = "Loading channel...")
            } else if (state.error != null) {
                ErrorStateWithRetry(
                    message = state.error!!,
                    onRetry = { /* TODO: Add retry intent to VM */ })
            } else {
                LazyColumn(
                    modifier = Modifier.fillMaxSize(),
                    contentPadding = PaddingValues(bottom = 80.dp),
                    verticalArrangement = Arrangement.spacedBy(24.dp)
                ) {
                    // 1. Header
                    item {
                        state.channelDetails?.let { details ->
                            ChannelHeader(
                                details = details,
                                isFavorited = favoritesState.likedItemsMap.containsKey(det
                                onFavoriteClicked = {
                                    favoritesViewModel.toggleFavoriteChannel(details)
                                }
                            )
                        }
                    }

                    // 2. Popular Songs
                    if (state.popularSongs.isNotEmpty()) {
                        item {
                            CarouselShelf<UnifiedDisplayItem>(
                                title = "Popular",
                                uiState = UiState.Success(state.popularSongs),
                                actionContent = {
                                    TextButton(onClick = {
                                        navController.navigate(
                                            AppDestinations.fullListViewRoute(
                                                MusicCategoryType.TRENDING,
                                                channelViewModel.channelId
                                            )
                                        )
                                    }) {
                                        Text(stringResource(id = R.string.action_show_more
                                    }
                                },
                                itemContent = { item ->
                                    UnifiedGridItem(
                                        item = item,
                                        onClick = { discoveryViewModel.playUnifiedItem(ite
                                    }
                                )
```

```
    }
}

// 3. Latest Streams
val recentStreams =
    state.discoveryContent?.recentSingingStreams ?: emptyList()
if (recentStreams.isNotEmpty()) {
    item {
        CarouselShelf<SingingStreamShelfItem>(
            title = "Latest Streams",
            uiState = UiState.Success(recentStreams),
            actionContent = {
                TextButton(onClick = {
                    videoListViewModel.setBrowseContextAndNavigate(cha
                    navController.navigate(AppDestinations.HOME_ROUTE)
                }) {
                    Text(stringResource(id = R.string.action_show_more
                }
            },
            itemContent = { item ->
                val shell =
                    item.video.toUnifiedDisplayItem(false, emptySet())
                UnifiedGridItem(
                    item = shell,
                    onClick = {
                        navController.navigate(
                            AppDestinations.videoDetailRoute(
                                item.video.id
                            )
                        )
                    })
            }
        )
    }
}

// 4. Other Channels (Sub-org)
val otherChannels = state.discoveryContent?.channels ?: emptyList()
if (otherChannels.isNotEmpty()) {
    item {
        val orgName = state.channelDetails?.org ?: "Organization"
        CarouselShelf<DiscoveryChannel>(
            title = "Discover More from $orgName",
            uiState = UiState.Success(otherChannels),
            actionContent = {
                TextButton(onClick = { /* TODO */ }) {
                    Text(stringResource(id = R.string.action_show_more
                }
            },
            itemContent = { channel ->
                ChannelCard(
                    channel = channel,
                    onChannelClicked = { navController.navigate("chann
                }
        )
```

```
                            }
                        }
                    }
                }
            }
        }
    }
}

@Composable
private fun ChannelHeader(
    details: ChannelDetails,
    isFavorited: Boolean,
    onFavoriteClicked: () -> Unit
) {
    val uriHandler = LocalUriHandler.current

    Column(
        modifier = Modifier
            .fillMaxWidth()
            .padding(horizontal = 16.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.spacedBy(12.dp)
    ) {
        AsyncImage(
            model = ImageRequest.Builder(LocalContext.current).data(details.photoUrl)
                .crossfade(true).build(),
            contentDescription = "Channel Avatar",
            modifier = Modifier
                .size(96.dp)
                .clip(CircleShape),
            contentScale = ContentScale.Crop
        )
        Column(horizontalAlignment = Alignment.CenterHorizontally) {
            Text(
                details.englishName ?: details.name,
                style = MaterialTheme.typography.headlineMedium,
                fontWeight = FontWeight.Bold
            )
            details.org?.let {
                Text(
                    it,
                    style = MaterialTheme.typography.titleMedium,
                    color = MaterialTheme.colorScheme.onSurfaceVariant
                )
            }
        }
        Row(horizontalArrangement = Arrangement.spacedBy(8.dp)) {
            Button(onClick = onFavoriteClicked) {
                Icon(
                    if (isFavorited) Icons.Default.Favorite else Icons.Default.FavoriteBorder,
                    null,
                    Modifier.size(ButtonDefaults.IconSize)
                )
                Spacer(Modifier.size(ButtonDefaults.IconSpacing))
```

```
                Text(if (isFavorited) "Favorited" else "Favorite")
            }
            OutlinedButton(onClick = { uriHandler.openUri("https://youtube.com/channel/${detai
                Icon(painterResource(R.drawable.youtube), null)
            }
            details.twitter?.let {
                OutlinedButton(onClick = { uriHandler.openUri("https://twitter.com/${it}") })
                    Icon(painterResource(R.drawable.twitter), null)
                }
            }
        }
    }
}

// File: java\com\example\holodex\ui\screens\DiscoveryScreen.kt
// File: java/com/example/holodex/ui/screens/DiscoveryScreen.kt

package com.example.holodex.ui.screens

import android.widget.Toast
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.calculateEndPadding
import androidx.compose.foundation.layout.calculateStartPadding
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.automirrored.filled.QueueMusic
import androidx.compose.material.icons.filled.ArrowDropDown
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.DropdownMenu
import androidx.compose.material3.DropdownMenuItem
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.Scaffold
import androidx.compose.material3.Text
import androidx.compose.material3.TextButton
import androidx.compose.material3.TopAppBar
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.Modifier
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.platform.LocalLayoutDirection
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.unit.dp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.lifecycle.compose.collectAsStateWithLifecycle
```

```
import androidx.media3.common.util.UnstableApi
import androidx.navigation.NavController
import com.example.holodex.R
import com.example.holodex.auth.AuthViewModel
import com.example.holodex.data.model.discovery.DiscoveryChannel
import com.example.holodex.data.model.discovery.PlaylistStub
import com.example.holodex.data.model.discovery.SingingStreamShelfItem
import com.example.holodex.ui.composables.CarouselShelf
import com.example.holodex.ui.composables.ChannelCard
import com.example.holodex.ui.composables.HeroCarousel
import com.example.holodex.ui.composables.PlaylistCard
import com.example.holodex.ui.composables.UnifiedGridItem
import com.example.holodex.ui.navigation.AppDestinations
import com.example.holodex.util.findActivity
import com.example.holodex.viewmodel.DiscoveryViewModel
import com.example.holodex.viewmodel.MusicCategoryType
import com.example.holodex.viewmodel.ShelfType
import com.example.holodex.viewmodel.UnifiedDisplayItem
import com.example.holodex.viewmodel.VideoListViewModel
import com.example.holodex.viewmodel.mappers.toUnifiedDisplayItem
import com.example.holodex.viewmodel.state.UiState
import kotlinx.coroutines.flow.collectLatest
import org.orbitmvi.orbit.compose.collectAsState

@androidx.annotation.OptIn(UnstableApi::class)
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun DiscoveryScreen(
    navController: NavController,
    contentPadding: PaddingValues = PaddingValues(0.dp) // Added
) {
    val discoveryViewModel: DiscoveryViewModel = hiltViewModel()
    val authViewModel: AuthViewModel = hiltViewModel()
    val videoListViewModel: VideoListViewModel = hiltViewModel(findActivity())

    val uiState by discoveryViewModel.uiState.collectAsStateWithLifecycle()
    val authState by authViewModel.authState.collectAsStateWithLifecycle()
    val videoListState by videoListViewModel.collectAsState()
    val selectedOrg = videoListState.selectedOrganization
    val availableOrganizations = videoListState.availableOrganizations
    val context = LocalContext.current

    var showOrgMenu by remember { mutableStateOf(false) }

    LaunchedEffect(authState, selectedOrg) {
        discoveryViewModel.loadDiscoveryContent(selectedOrg, authState)
    }

    LaunchedEffect(Unit) {
        discoveryViewModel.transientMessage.collectLatest { message ->
            Toast.makeText(context, message, Toast.LENGTH_SHORT).show()
        }
    }

    Scaffold(
```

```
    topBar = {
        TopAppBar(
            title = { Text(stringResource(R.string.bottom_nav_discover)) },
            actions = {
                Box {
                    TextButton(onClick = { showOrgMenu = true }) {
                        Text(selectedOrg)
                        Icon(
                            Icons.Default.ArrowDropDown,
                            contentDescription = "Select Organization"
                        )
                    }
                    DropdownMenu(
                        expanded = showOrgMenu,
                        onDismissRequest = { showOrgMenu = false }
                    ) {
                        availableOrganizations.forEach { (name, value) ->
                            if (value != null) {
                                DropdownMenuItem(
                                    text = { Text(name) },
                                    onClick = {
                                        videoListViewModel.setOrganization(value)
                                        showOrgMenu = false
                                    }
                                )
                            }
                        }
                    }
                }
            }
        )
    }
) { paddingValues ->

    // Merge the Scaffold padding (top bar) with the content padding (bottom bar)
    val layoutDirection = LocalLayoutDirection.current
    val mergedPadding = PaddingValues(
        top = paddingValues.calculateTopPadding() + 16.dp,
        bottom = contentPadding.calculateBottomPadding() + 16.dp,
        start = paddingValues.calculateStartPadding(layoutDirection),
        end = paddingValues.calculateEndPadding(layoutDirection)
    )

    LazyColumn(
        modifier = Modifier.fillMaxSize(),
        contentPadding = mergedPadding, // Use merged padding
        verticalArrangement = Arrangement.spacedBy(24.dp)
    ) {
        items(uiState.shelfOrder, key = { it.name }) { shelfType ->
            val shelfState = uiState.shelves[shelfType] ?: UiState.Loading

            when (shelfType) {
                ShelfType.RECENT_STREAMS -> {
                    @Suppress("UNCHECKED_CAST")
                    HeroCarousel(
```

```
                        title = shelfType.toTitle(selectedOrg),
                        uiState = shelfState as UiState<List<SingingStreamShelfItem>>,
                        onItemClicked = { item ->
                            navController.navigate(AppDestinations.videoDetailRoute(item.v
                        }
                    )
                }

                else -> {
                    CarouselShelf<Any>(
                        title = shelfType.toTitle(selectedOrg),
                        uiState = shelfState,
                        itemContent = { item ->
                            ShelfItemContent(
                                item = item,
                                discoveryViewModel = discoveryViewModel,
                                navController = navController
                            )
                        },
                        actionContent = {
                            if (shelfType == ShelfType.TRENDING_SONGS) {
                                TextButton(onClick = {
                                    (shelfState as? UiState.Success)?.data?.let {
                                        discoveryViewModel.addAllToQueue(it.filterIsInstan
                                    }
                                }) {
                                    Icon(
                                        Icons.AutoMirrored.Filled.QueueMusic,
                                        null,
                                        modifier = Modifier.size(ButtonDefaults.IconSize)
                                    )
                                    Spacer(Modifier.size(ButtonDefaults.IconSpacing))
                                    Text("Queue")
                                }
                            } else {
                                TextButton(onClick = {
                                    navController.navigate(
                                        AppDestinations.fullListViewRoute(
                                            shelfType.toMusicCategoryType(),
                                            selectedOrg
                                        )
                                    )
                                }) {
                                    Text(stringResource(R.string.action_show_more))
                                }
                            }
                        }
                    )
                }
            }
        }
    }
}
```

```
    @Composable
private fun ShelfItemContent(
    item: Any,
    discoveryViewModel: DiscoveryViewModel,
    navController: NavController
) {
    when (item) {
        is UnifiedDisplayItem -> UnifiedGridItem(item = item, onClick = { discoveryViewModel.p
        is SingingStreamShelfItem -> {
            val displayShell = item.video.toUnifiedDisplayItem(isLiked = false, downloadedSegm
            UnifiedGridItem(
                item = displayShell,
                onClick = {
                    navController.navigate(AppDestinations.videoDetailRoute(item.video.id))
                }
            )
        }
        is PlaylistStub -> PlaylistCard(
            playlist = item,
            onPlaylistClicked = { playlistStub ->
                // Decide what to do based on the type
                if (playlistStub.type.startsWith("radio")) {
                    discoveryViewModel.playRadioPlaylist(playlistStub)
                } else {
                    navController.navigate(AppDestinations.playlistDetailsRoute(playlistStub.i
                }
            }
        )
        is DiscoveryChannel -> ChannelCard(
            channel = item,
            onChannelClicked = { channelId -> navController.navigate("channel_details/$channel
        )
    }
}


private fun ShelfType.toTitle(orgName: String): String {
    val displayOrg = if (orgName == "All Vtubers") "All" else orgName
    return when (this) {
        ShelfType.RECENT_STREAMS -> "Recent Singing Streams"
        ShelfType.SYSTEM_PLAYLISTS -> "$displayOrg Playlists"
        ShelfType.ARTIST_RADIOS -> "$displayOrg Radios"
        ShelfType.FAN_PLAYLISTS -> "$displayOrg Community Playlists"
        ShelfType.TRENDING_SONGS -> "Trending Songs"
        ShelfType.DISCOVER_CHANNELS -> "Discover $displayOrg"
        ShelfType.FOR_YOU -> "For You"
    }
}

@androidx.annotation.OptIn(UnstableApi::class)
private fun ShelfType.toMusicCategoryType(): MusicCategoryType {
    return when (this) {
        ShelfType.TRENDING_SONGS -> MusicCategoryType.TRENDING
        ShelfType.RECENT_STREAMS -> MusicCategoryType.RECENT_STREAMS
        ShelfType.FAN_PLAYLISTS -> MusicCategoryType.COMMUNITY_PLAYLISTS
```

```
        ShelfType.ARTIST_RADIOS -> MusicCategoryType.ARTIST_RADIOS
        // --- START OF MODIFICATION ---
        ShelfType.SYSTEM_PLAYLISTS -> MusicCategoryType.SYSTEM_PLAYLISTS
        ShelfType.DISCOVER_CHANNELS -> MusicCategoryType.DISCOVER_CHANNELS
        // --- END OF MODIFICATION ---
        ShelfType.FOR_YOU -> MusicCategoryType.FAVORITES // For You is a special case of favor
    }
}

// File: java\com\example\holodex\ui\screens\DownloadsScreen.kt
package com.example.holodex.ui.screens

import android.widget.Toast
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.calculateEndPadding
import androidx.compose.foundation.layout.calculateStartPadding
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.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.grid.GridCells
import androidx.compose.foundation.lazy.grid.LazyVerticalGrid
import androidx.compose.foundation.lazy.grid.items
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.text.KeyboardActions
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.automirrored.filled.PlaylistPlay
import androidx.compose.material.icons.automirrored.filled.ViewList
import androidx.compose.material.icons.filled.Clear
import androidx.compose.material.icons.filled.GridView
import androidx.compose.material.icons.filled.Search
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.Scaffold
import androidx.compose.material3.Text
import androidx.compose.material3.TextButton
import androidx.compose.material3.TopAppBar
import androidx.compose.material3.TopAppBarDefaults
import androidx.compose.material3.pulltorefresh.PullToRefreshBox
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Modifier
import androidx.compose.ui.input.nestedscroll.nestedScroll
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.platform.LocalFocusManager
```

```
import androidx.compose.ui.platform.LocalLayoutDirection
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.input.ImeAction
import androidx.compose.ui.unit.dp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.media3.common.util.UnstableApi
import androidx.navigation.NavController
import com.example.holodex.R
import com.example.holodex.ui.composables.EmptyState
import com.example.holodex.ui.composables.UnifiedGridItem
import com.example.holodex.ui.composables.UnifiedListItem
import com.example.holodex.viewmodel.DownloadsSideEffect
import com.example.holodex.viewmodel.DownloadsViewModel
import com.example.holodex.viewmodel.FavoritesViewModel
import com.example.holodex.viewmodel.PlaylistManagementViewModel
import com.example.holodex.viewmodel.VideoListViewModel
import org.orbitmvi.orbit.compose.collectAsState
import org.orbitmvi.orbit.compose.collectSideEffect

@UnstableApi
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun DownloadsScreen(
    navController: NavController,
    downloadsViewModel: DownloadsViewModel = hiltViewModel(),
    playlistManagementViewModel: PlaylistManagementViewModel,
    contentPadding: PaddingValues // Added
) {
    val state by downloadsViewModel.collectAsState()
    val context = LocalContext.current
    val videoListViewModel: VideoListViewModel = hiltViewModel()
    val favoritesViewModel: FavoritesViewModel = hiltViewModel()

    downloadsViewModel.collectSideEffect { effect ->
        when (effect) {
            is DownloadsSideEffect.ShowToast -> Toast.makeText(context, effect.message, Toast.
        }
    }

    var isGridView by remember { mutableStateOf(false) }
    val focusManager = LocalFocusManager.current
    val scrollBehavior = TopAppBarDefaults.enterAlwaysScrollBehavior()

    Scaffold(
        modifier = Modifier.nestedScroll(scrollBehavior.nestedScrollConnection),
        topBar = {
            TopAppBar(
                title = { Text(stringResource(R.string.bottom_nav_downloads)) },
                actions = {
                    if (state.items.isNotEmpty()) {
                        TextButton(onClick = { downloadsViewModel.playAllDownloadsShuffled() }
                            Icon(Icons.AutoMirrored.Filled.PlaylistPlay, contentDescription =
                            androidx.compose.foundation.layout.Spacer(Modifier.size(ButtonDefa
                            Text(stringResource(R.string.action_play_all))
                        }
```

```
                }
                IconButton(onClick = { isGridView = !isGridView }) {
                    Icon(
                        imageVector = if (isGridView) Icons.AutoMirrored.Filled.ViewList e
                        contentDescription = null
                    )
                }
            },
            scrollBehavior = scrollBehavior
        )
    }
) { innerPadding ->

    // Merge padding
    val layoutDirection = LocalLayoutDirection.current

    val unifiedPadding = PaddingValues(
        top = innerPadding.calculateTopPadding(),
        bottom = contentPadding.calculateBottomPadding() + 16.dp,
        start = innerPadding.calculateStartPadding(layoutDirection),
        end = innerPadding.calculateEndPadding(layoutDirection)
    )

    PullToRefreshBox(
        isRefreshing = false, // Unified Repo updates automatically
        onRefresh = { /* No-op, list updates via Flow */ },
        modifier = Modifier.padding(top = unifiedPadding.calculateTopPadding()).fillMaxSiz
    ) {
        Column(modifier = Modifier.fillMaxSize()) {
            OutlinedTextField(
                value = state.searchQuery,
                onValueChange = { downloadsViewModel.onSearchQueryChanged(it) },
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(horizontal = 16.dp, vertical = 8.dp),
                placeholder = { Text(stringResource(R.string.search_your_downloads_hint))
                leadingIcon = { Icon(Icons.Filled.Search, null) },
                trailingIcon = {
                    if (state.searchQuery.isNotEmpty()) {
                        IconButton(onClick = { downloadsViewModel.onSearchQueryChanged("")
                            Icon(Icons.Filled.Clear, stringResource(R.string.action_clear_
                        }
                    }
                },
                singleLine = true,
                keyboardOptions = KeyboardOptions(imeAction = ImeAction.Search),
                keyboardActions = KeyboardActions(onSearch = { focusManager.clearFocus() }
            )

            if (state.items.isEmpty()) {
                // Reusing EmptyState for consistency
                EmptyState(
                    message = if (state.searchQuery.isNotEmpty()) stringResource(R.string.
                    else stringResource(R.string.message_no_downloads),
                    onRefresh = {}
```

```
                    )
                } else {
                    if (isGridView) {
                        LazyVerticalGrid(
                            columns = GridCells.Adaptive(minSize = 140.dp),
                            contentPadding = PaddingValues(start = 16.dp, end = 16.dp, bottom
                            horizontalArrangement = Arrangement.spacedBy(16.dp),
                            verticalArrangement = Arrangement.spacedBy(16.dp)
                        ) {
                            items(state.items, key = { it.stableId }) { item ->
                                UnifiedGridItem(
                                    item = item,
                                    onClick = { downloadsViewModel.playDownloads(item) }
                                )
                            }
                        }
                    } else {
                        LazyColumn(
                            contentPadding = PaddingValues(bottom = unifiedPadding.calculateBo
                        ) {
                            items(state.items, key = { it.stableId }) { item ->
                                UnifiedListItem(
                                    item = item,
                                    onItemClicked = { downloadsViewModel.playDownloads(item) }
                                    navController = navController,
                                    videoListViewModel = videoListViewModel,
                                    favoritesViewModel = favoritesViewModel,
                                    playlistManagementViewModel = playlistManagementViewModel,
                                    isEditing = false, // View mode
                                    onRemoveClicked = { /* Not in edit mode */ }
                                )
                            }
                        }
                    }
                }
            }
        }
    }
}

// File: java\com\example\holodex\ui\screens\EditablePlaylistHeader.kt
package com.example.holodex.ui.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.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.setValue
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import com.example.holodex.R
import com.example.holodex.data.db.PlaylistEntity
import kotlinx.coroutines.FlowPreview
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.debounce

@OptIn(FlowPreview::class)
@Composable
fun EditablePlaylistHeader(
    playlist: PlaylistEntity,
    onNameChange: (String) -> Unit,
    onDescriptionChange: (String) -> Unit,
    modifier: Modifier = Modifier
) {
    var name by remember(playlist.name) { mutableStateOf(playlist.name ?: "") }
    var description by remember(playlist.description) { mutableStateOf(playlist.description ?:

    // Use StateFlows with debounce to avoid excessive recompositions on every keystroke
    val nameFlow = remember { MutableStateFlow(name) }
    val descriptionFlow = remember { MutableStateFlow(description) }

    LaunchedEffect(Unit) {
        nameFlow.debounce(300).collect { onNameChange(it) }
    }
    LaunchedEffect(Unit) {
        descriptionFlow.debounce(300).collect { onDescriptionChange(it) }
    }

    Column(
        modifier = modifier
            .fillMaxWidth()
            .padding(16.dp),
        verticalArrangement = Arrangement.spacedBy(8.dp)
    ) {
        OutlinedTextField(
            value = name,
            onValueChange = {
                name = it
                nameFlow.value = it
            },
            label = { Text(stringResource(R.string.hint_playlist_name)) },
            modifier = Modifier.fillMaxWidth(),
            textStyle = MaterialTheme.typography.headlineSmall.copy(fontWeight = FontWeight.Bo
            singleLine = true
        )
        OutlinedTextField(
            value = description,
            onValueChange = {
                description = it
                descriptionFlow.value = it
```

```
            },
            label = { Text(stringResource(R.string.hint_playlist_description_optional)) },
            modifier = Modifier.fillMaxWidth(),
            textStyle = MaterialTheme.typography.bodyMedium,
            maxLines = 3
        )
    }
}

// File: java\com\example\holodex\ui\screens\FavoritesScreen.kt
package com.example.holodex.ui.screens

import android.widget.Toast
import androidx.compose.animation.core.animateFloatAsState
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.aspectRatio
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.heightIn
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.LazyRow
import androidx.compose.foundation.lazy.grid.GridCells
import androidx.compose.foundation.lazy.grid.LazyVerticalGrid
import androidx.compose.foundation.lazy.grid.items
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.ExpandMore
import androidx.compose.material3.Card
import androidx.compose.material3.HorizontalDivider
import androidx.compose.material3.Icon
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
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.draw.clip
import androidx.compose.ui.draw.rotate
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.style.TextAlign
```

```
import androidx.compose.ui.text.style.TextOverflow
import androidx.compose.ui.unit.dp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.media3.common.util.UnstableApi
import androidx.navigation.NavController
import coil.compose.AsyncImage
import coil.request.ImageRequest
import com.example.holodex.R
import com.example.holodex.ui.composables.LoadingSkeleton
import com.example.holodex.ui.composables.UnifiedListItem
import com.example.holodex.ui.navigation.AppDestinations
import com.example.holodex.viewmodel.FavoritesSideEffect
import com.example.holodex.viewmodel.FavoritesViewModel
import com.example.holodex.viewmodel.PlaylistManagementViewModel
import com.example.holodex.viewmodel.UnifiedDisplayItem
import com.example.holodex.viewmodel.VideoListViewModel
import com.example.holodex.viewmodel.mappers.toPlaybackItem
import org.orbitmvi.orbit.compose.collectAsState
import org.orbitmvi.orbit.compose.collectSideEffect

@OptIn(UnstableApi::class)
@Composable
fun FavoritesScreen(
    modifier: Modifier = Modifier,
    isGridView: Boolean,
    videoListViewModel: VideoListViewModel,
    playlistManagementViewModel: PlaylistManagementViewModel,
    navController: NavController,
    favoritesViewModel: FavoritesViewModel = hiltViewModel(),
    contentPadding: PaddingValues = PaddingValues(0.dp) // NEW PARAMETER
) {
    val state by favoritesViewModel.collectAsState()
    val context = LocalContext.current

    favoritesViewModel.collectSideEffect { sideEffect ->
        when (sideEffect) {
            is FavoritesSideEffect.ShowToast -> {
                Toast.makeText(context, sideEffect.message, Toast.LENGTH_SHORT).show()
            }
        }
    }

    var isChannelsExpanded by remember { mutableStateOf(true) }
    var isFavoritesExpanded by remember { mutableStateOf(true) }
    var isSegmentsExpanded by remember { mutableStateOf(true) }

    if (state.isLoading) {
        LoadingSkeleton(itemCount = 8, modifier = modifier.padding(16.dp))
        return
    }

    LazyColumn(
        modifier = modifier.fillMaxSize(),
        // *** FIX: Use the dynamic content padding ***
        contentPadding = contentPadding
```

```
) {
    if (state.favoriteChannels.isNotEmpty()) {
        item {
            ExpandableSectionHeader(
                title = stringResource(R.string.category_favorite_channels),
                itemCount = state.favoriteChannels.size,
                isExpanded = isChannelsExpanded,
                onToggle = { isChannelsExpanded = !isChannelsExpanded }
            )
        }
        if (isChannelsExpanded) {
            item {
                FavoriteChannelsRow(
                    channels = state.favoriteChannels,
                    onChannelClick = { channelItem ->
                        navController.navigate("channel_details/${channelItem.channelId}")
                    }
                )
            }
        }
        item { HorizontalDivider(modifier = Modifier.padding(vertical = 8.dp)) }
    }

    item {
        ExpandableSectionHeader(
            title = stringResource(R.string.category_favorites),
            itemCount = state.unifiedFavoritedVideos.size,
            isExpanded = isFavoritesExpanded,
            onToggle = { isFavoritesExpanded = !isFavoritesExpanded }
        )
    }

    if (isFavoritesExpanded && state.unifiedFavoritedVideos.isNotEmpty()) {
        if (isGridView) {
            item {
                FavoritesGrid(
                    items = state.unifiedFavoritedVideos,
                    onItemClicked = { item ->
                        navController.navigate(AppDestinations.videoDetailRoute(item.video
                    }
                )
            }
        } else {
            items(items = state.unifiedFavoritedVideos, key = { it.stableId }) { item ->
                UnifiedListItem(
                    item = item,
                    onItemClicked = {
                        navController.navigate(AppDestinations.videoDetailRoute(item.video
                    },
                    videoListViewModel = videoListViewModel,
                    favoritesViewModel = favoritesViewModel,
                    playlistManagementViewModel = playlistManagementViewModel,
                    navController = navController
                )
            }
```

```
            }
        }

        item {
            HorizontalDivider(modifier = Modifier.padding(vertical = 8.dp))
            ExpandableSectionHeader(
                title = stringResource(R.string.category_liked_segments),
                itemCount = state.unifiedLikedSegments.size,
                isExpanded = isSegmentsExpanded,
                onToggle = { isSegmentsExpanded = !isSegmentsExpanded }
            )
        }

        if (isSegmentsExpanded && state.unifiedLikedSegments.isNotEmpty()) {
            if (isGridView) {
                item {
                    FavoritesGrid(
                        items = state.unifiedLikedSegments,
                        onItemClicked = { item ->
                            videoListViewModel.playFavoriteOrLikedSegmentItem(item.toPlaybackI
                        }
                    )
                }
            } else {
                items(items = state.unifiedLikedSegments, key = { it.stableId }) { item ->
                    UnifiedListItem(
                        item = item,
                        onItemClicked = { videoListViewModel.playFavoriteOrLikedSegmentItem(it
                        videoListViewModel = videoListViewModel,
                        navController = navController,
                        playlistManagementViewModel = playlistManagementViewModel,
                        favoritesViewModel = favoritesViewModel
                    )
                }
            }
        }
    }
}

@Composable
private fun UnifiedGridItem(
    item: UnifiedDisplayItem,
    onClick: () -> Unit,
    modifier: Modifier = Modifier
) {
    Card(onClick = onClick, modifier = modifier) {
        Column {
            AsyncImage(
                model = ImageRequest.Builder(LocalContext.current)
                    .data(item.artworkUrls.firstOrNull())
                    .placeholder(R.drawable.ic_placeholder_image)
                    .error(R.drawable.ic_error_image)
                    .crossfade(true)
                    .build(),
                contentDescription = "Artwork for ${item.title}",
```

```
                contentScale = ContentScale.Crop,
                modifier = Modifier
                    .fillMaxWidth()
                    .aspectRatio(1f)
            )
            Column(Modifier.padding(8.dp)) {
                Text(
                    text = item.title,
                    style = MaterialTheme.typography.titleSmall,
                    maxLines = 2,
                    overflow = TextOverflow.Ellipsis
                )
                Text(
                    text = item.artistText,
                    style = MaterialTheme.typography.bodySmall,
                    maxLines = 1,
                    overflow = TextOverflow.Ellipsis,
                    color = MaterialTheme.colorScheme.onSurfaceVariant
                )
            }
        }
    }
}

@Composable
private fun FavoritesGrid(
    items: List<UnifiedDisplayItem>,
    onItemClicked: (UnifiedDisplayItem) -> Unit,
) {
    LazyVerticalGrid(
        columns = GridCells.Adaptive(minSize = 128.dp),
        contentPadding = PaddingValues(16.dp),
        horizontalArrangement = Arrangement.spacedBy(16.dp),
        verticalArrangement = Arrangement.spacedBy(16.dp),
        modifier = Modifier.heightIn(min = 1.dp),
        userScrollEnabled = false
    ) {
        items(items, key = { it.stableId }) { item ->
            UnifiedGridItem(item = item, onClick = { onItemClicked(item) })
        }
    }
}

@Composable
private fun ExpandableSectionHeader(
    title: String,
    itemCount: Int,
    isExpanded: Boolean,
    onToggle: () -> Unit
) {
    val rotationAngle by animateFloatAsState(
        targetValue = if (isExpanded) 0f else -90f,
        label = "expansion_arrow"
    )
    Row(
```

```
        modifier = Modifier
            .fillMaxWidth()
            .clickable { onToggle() }
            .padding(horizontal = 16.dp, vertical = 12.dp),
        verticalAlignment = Alignment.CenterVertically
    ) {
        Text(
            text = "$title ($itemCount)",
            style = MaterialTheme.typography.headlineSmall,
            modifier = Modifier.weight(1f)
        )
        Icon(
            imageVector = Icons.Default.ExpandMore,
            contentDescription = if (isExpanded) "Collapse" else "Expand",
            modifier = Modifier.rotate(rotationAngle)
        )
    }
}

@Composable
private fun FavoriteChannelsRow(
    channels: List<UnifiedDisplayItem>,
    onChannelClick: (UnifiedDisplayItem) -> Unit
) {
    LazyRow(
        contentPadding = PaddingValues(horizontal = 16.dp, vertical = 8.dp),
        horizontalArrangement = Arrangement.spacedBy(16.dp)
    ) {
        items(channels, key = { it.stableId }) { channelItem ->
            Column(
                horizontalAlignment = Alignment.CenterHorizontally,
                modifier = Modifier
                    .width(80.dp)
                    .clickable { onChannelClick(channelItem) }
            ) {
                AsyncImage(
                    model = ImageRequest.Builder(LocalContext.current)
                        .data(channelItem.artworkUrls.firstOrNull())
                        .placeholder(R.drawable.ic_placeholder_image)
                        .error(R.drawable.ic_error_image)
                        .crossfade(true)
                        .build(),
                    contentDescription = channelItem.title,
                    modifier = Modifier
                        .size(64.dp)
                        .clip(CircleShape),
                    contentScale = ContentScale.Crop
                )
                Spacer(Modifier.height(4.dp))
                Text(
                    text = channelItem.title,
                    style = MaterialTheme.typography.bodySmall,
                    maxLines = 2,
                    overflow = TextOverflow.Ellipsis,
                    textAlign = TextAlign.Center
```

```
                )
            }
        }
    }
}

// File: java\com\example\holodex\ui\screens\ForYouScreen.kt
// File: java/com/example/holodex/ui/screens/ForYouScreen.kt

package com.example.holodex.ui.screens

import android.widget.Toast
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.automirrored.filled.ArrowBack
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.Scaffold
import androidx.compose.material3.Text
import androidx.compose.material3.TextButton
import androidx.compose.material3.TopAppBar
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.getValue
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.unit.dp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.lifecycle.compose.collectAsStateWithLifecycle
import androidx.media3.common.util.UnstableApi
import androidx.navigation.NavController
import com.example.holodex.R
import com.example.holodex.data.model.discovery.DiscoveryChannel
import com.example.holodex.data.model.discovery.PlaylistStub
import com.example.holodex.data.model.discovery.SingingStreamShelfItem
import com.example.holodex.ui.composables.CarouselShelf
import com.example.holodex.ui.composables.ChannelCard
import com.example.holodex.ui.composables.ErrorStateWithRetry
import com.example.holodex.ui.composables.LoadingState
import com.example.holodex.ui.composables.PlaylistCard
import com.example.holodex.ui.composables.UnifiedGridItem
import com.example.holodex.ui.navigation.AppDestinations
import com.example.holodex.viewmodel.DiscoveryViewModel
import com.example.holodex.viewmodel.VideoListViewModel
import com.example.holodex.viewmodel.mappers.toUnifiedDisplayItem
import com.example.holodex.viewmodel.state.UiState
import kotlinx.coroutines.flow.collectLatest
```

```
@androidx.annotation.OptIn(UnstableApi::class)
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun ForYouScreen(
    navController: NavController
) {
    val discoveryViewModel: DiscoveryViewModel = hiltViewModel()
    val videoListViewModel: VideoListViewModel = hiltViewModel()

    val forYouState by discoveryViewModel.forYouState.collectAsStateWithLifecycle()
    val context = LocalContext.current

    LaunchedEffect(Unit) {
        discoveryViewModel.loadForYouContent()
        discoveryViewModel.transientMessage.collectLatest { message ->
            Toast.makeText(context, message, Toast.LENGTH_SHORT).show()
        }
    }

    Scaffold(
        topBar = {
            TopAppBar(
                title = { Text(stringResource(R.string.shelf_title_for_you)) },
                navigationIcon = {
                    IconButton(onClick = { navController.popBackStack() }) {
                        Icon(Icons.AutoMirrored.Filled.ArrowBack, "Back")
                    }
                }
            )
        }
    ) { paddingValues ->
        Box(modifier = Modifier
            .padding(paddingValues)
            .fillMaxSize()) {
            when (val state = forYouState) {
                is UiState.Loading -> LoadingState(message = "Loading your personalized conten
                is UiState.Error -> ErrorStateWithRetry(
                    message = state.message,
                    onRetry = { discoveryViewModel.loadForYouContent() })

                is UiState.Success -> {
                    val data = state.data
                    LazyColumn(
                        modifier = Modifier.fillMaxSize(),
                        contentPadding = PaddingValues(vertical = 16.dp),
                        verticalArrangement = Arrangement.spacedBy(24.dp)
                    ) {
                        item {
                            val uiState = UiState.Success(data.recentSingingStreams ?: emptyLi
                            // --- START OF FIX ---
                            CarouselShelf<SingingStreamShelfItem>(
                                title = stringResource(R.string.shelf_title_recent_streams_fav
                                uiState = uiState,
                                actionContent = {
```

```
                TextButton(onClick = {
                    videoListViewModel.setBrowseContextAndNavigate(org = "
                    navController.navigate(AppDestinations.HOME_ROUTE)
                }) {
                    Text(stringResource(R.string.action_show_more))
                }
            },
            itemContent = { item ->
                val shell = item.video.toUnifiedDisplayItem(false, emptySe
                UnifiedGridItem(
                    item = shell,
                    onClick = { navController.navigate(AppDestinations.vid
                )
            }
        )
        // --- END OF FIX ---
    }

    item {
        val radios = remember {
            data.recommended?.playlists?.filter {
                it.type.startsWith("radio")
            } ?: emptyList()
        }
        val uiState = UiState.Success(radios)
        // --- START OF FIX ---
        CarouselShelf<PlaylistStub>(
            title = "Favorite Artist Radios",
            uiState = uiState,
            actionContent = {
                TextButton(onClick = { /* TODO: Navigate to full list of f
                    Text(stringResource(R.string.action_show_more))
                }
            },
            itemContent = { item ->
                PlaylistCard(
                    playlist = item,
                    onPlaylistClicked = { navController.navigate(AppDestin
                )
            }
        )
        // --- END OF FIX ---
    }

    item {
        val recommendedChannels = remember { data.channels ?: emptyList()
        val uiState = UiState.Success(recommendedChannels)
        // --- START OF FIX ---
        CarouselShelf<DiscoveryChannel>(
            title = "Discover More Channels",
            uiState = uiState,
            actionContent = {
                TextButton(onClick = { /* TODO: Navigate to full list of f
                    Text(stringResource(R.string.action_show_more))
                }
```

```
                                },
                                itemContent = { channel ->
                                    ChannelCard(
                                        channel = channel,
                                        onChannelClicked = { channelId -> navController.naviga
                                    )
                                }
                            )
                            // --- END OF FIX ---
                        }
                    }
                }
            }
        }
    }
}

// File: java\com\example\holodex\ui\screens\FullListViewScreen.kt
package com.example.holodex.ui.screens

import android.widget.Toast
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.PaddingValues
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.foundation.lazy.grid.GridCells
import androidx.compose.foundation.lazy.grid.GridItemSpan
import androidx.compose.foundation.lazy.grid.LazyVerticalGrid
import androidx.compose.foundation.lazy.grid.rememberLazyGridState
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.automirrored.filled.ArrowBack
import androidx.compose.material3.CircularProgressIndicator
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Scaffold
import androidx.compose.material3.Text
import androidx.compose.material3.TopAppBar
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.derivedStateOf
import androidx.compose.runtime.getValue
import androidx.compose.runtime.remember
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.media3.common.util.UnstableApi
```

```
import androidx.navigation.NavController
import com.example.holodex.R
import com.example.holodex.data.model.discovery.DiscoveryChannel
import com.example.holodex.data.model.discovery.PlaylistStub
import com.example.holodex.ui.composables.ChannelCard
import com.example.holodex.ui.composables.PlaylistCard
import com.example.holodex.ui.composables.UnifiedGridItem
import com.example.holodex.ui.navigation.AppDestinations
import com.example.holodex.viewmodel.DiscoveryViewModel
import com.example.holodex.viewmodel.FullListSideEffect
import com.example.holodex.viewmodel.FullListViewModel
import com.example.holodex.viewmodel.MusicCategoryType
import com.example.holodex.viewmodel.SubOrgHeader
import com.example.holodex.viewmodel.UnifiedDisplayItem
import org.orbitmvi.orbit.compose.collectAsState
import org.orbitmvi.orbit.compose.collectSideEffect

@androidx.annotation.OptIn(UnstableApi::class)
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun FullListViewScreen(
    navController: NavController,
    categoryType: MusicCategoryType
) {
    val fullListViewModel: FullListViewModel = hiltViewModel()
    val discoveryViewModel: DiscoveryViewModel = hiltViewModel()
    val context = LocalContext.current

    // Collect State
    val state by fullListViewModel.collectAsState()

    // Handle Side Effects
    fullListViewModel.collectSideEffect { effect ->
        when (effect) {
            is FullListSideEffect.ShowToast -> {
                Toast.makeText(context, effect.message, Toast.LENGTH_SHORT).show()
            }
        }
    }

    val gridState = rememberLazyGridState()

    // Pagination Logic
    val shouldLoadMore by remember {
        derivedStateOf {
            val layoutInfo = gridState.layoutInfo
            val totalItems = layoutInfo.totalItemsCount
            if (totalItems == 0) return@derivedStateOf false
            val lastVisibleItem =
                layoutInfo.visibleItemsInfo.lastOrNull() ?: return@derivedStateOf false
            lastVisibleItem.index >= totalItems - 10
        }
    }

    LaunchedEffect(shouldLoadMore) {
```

```
    if (shouldLoadMore && !state.isLoadingMore && !state.endOfList) {
        fullListViewModel.loadMore()
    }
}

Scaffold(
    topBar = {
        TopAppBar(
            title = {
                Text(
                    categoryType.name.replace('_', ' ').lowercase()
                        .replaceFirstChar { it.uppercase() })
            },
            navigationIcon = {
                IconButton(onClick = { navController.popBackStack() }) {
                    Icon(
                        Icons.AutoMirrored.Filled.ArrowBack,
                        contentDescription = stringResource(R.string.action_back)
                    )
                }
            }
        )
    }
) { paddingValues ->
    LazyVerticalGrid(
        state = gridState,
        columns = GridCells.Adaptive(140.dp),
        modifier = Modifier
            .padding(paddingValues)
            .fillMaxSize(),
        contentPadding = PaddingValues(16.dp),
        horizontalArrangement = Arrangement.spacedBy(12.dp),
        verticalArrangement = Arrangement.spacedBy(12.dp)
    ) {
        val items = state.items
        items(
            count = items.size,
            key = { index ->
                val item = items[index]
                when (item) {
                    is UnifiedDisplayItem -> item.stableId
                    is PlaylistStub -> item.id
                    is DiscoveryChannel -> item.id
                    is SubOrgHeader -> item.name
                    else -> item.hashCode()
                }
            },
            span = { index ->
                if (items[index] is SubOrgHeader) GridItemSpan(maxLineSpan) else GridItemS
            }
        ) { index ->
            when (val item = items[index]) {
                is UnifiedDisplayItem -> UnifiedGridItem(
                    item = item,
                    onClick = { discoveryViewModel.playUnifiedItem(item) })
```

```
                    is PlaylistStub -> PlaylistCard(
                        playlist = item,
                        onPlaylistClicked = { playlistStub ->
                            navController.navigate(AppDestinations.playlistDetailsRoute(playli
                        })

                    is DiscoveryChannel -> ChannelCard(
                        channel = item,
                        onChannelClicked = { channelId ->
                            navController.navigate("channel_details/$channelId")
                        })

                    is SubOrgHeader -> Text(
                        text = item.name,
                        style = MaterialTheme.typography.titleMedium,
                        fontWeight = FontWeight.Bold,
                        modifier = Modifier
                            .fillMaxWidth()
                            .padding(bottom = 8.dp, top = if (index == 0) 0.dp else 16.dp)
                    )
                }
            }

            if (state.isLoadingMore) {
                item(span = { GridItemSpan(maxLineSpan) }) {
                    Box(
                        modifier = Modifier
                            .fillMaxWidth()
                            .padding(vertical = 16.dp),
                        contentAlignment = Alignment.Center
                    ) {
                        CircularProgressIndicator(modifier = Modifier.size(36.dp))
                    }
                }
            }
        }
    }
}

// File: java\com\example\holodex\ui\screens\HistoryScreen.kt
package com.example.holodex.ui.screens

import android.widget.Toast
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.PaddingValues
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.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
```

```
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.automirrored.filled.PlaylistAdd
import androidx.compose.material.icons.filled.PlayArrow
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.HorizontalDivider
import androidx.compose.material3.Icon
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.OutlinedButton
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.ui.Modifier
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.res.pluralStringResource
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.navigation.NavController
import com.example.holodex.R
import com.example.holodex.ui.composables.EmptyState
import com.example.holodex.ui.composables.UnifiedListItem
import com.example.holodex.viewmodel.FavoritesViewModel
import com.example.holodex.viewmodel.HistorySideEffect
import com.example.holodex.viewmodel.HistoryViewModel
import com.example.holodex.viewmodel.PlaylistManagementViewModel
import com.example.holodex.viewmodel.VideoListViewModel
import org.orbitmvi.orbit.compose.collectAsState
import org.orbitmvi.orbit.compose.collectSideEffect

@Composable
fun HistoryScreen(
    modifier: Modifier = Modifier,
    navController: NavController,
    videoListViewModel: VideoListViewModel,
    favoritesViewModel: FavoritesViewModel,
    playlistManagementViewModel: PlaylistManagementViewModel,
    contentPadding: PaddingValues = PaddingValues(0.dp)
) {
    val historyViewModel: HistoryViewModel = hiltViewModel()

    // --- MIGRATED: Use Orbit collection ---
    val state by historyViewModel.collectAsState()
    val historyItems = state.items
    // ------------------------------------

    val context = LocalContext.current

    // --- MIGRATED: Use Orbit Side Effects ---
    historyViewModel.collectSideEffect { effect ->
        when(effect) {
            is HistorySideEffect.ShowToast -> {
                Toast.makeText(context, effect.message, Toast.LENGTH_SHORT).show()
            }
```

```
        }
    }

    Column(modifier = modifier.fillMaxSize()) {
        if (historyItems.isEmpty() && !state.isLoading) {
            EmptyState(
                message = stringResource(R.string.message_no_history),
                onRefresh = {}
            )
        } else {
            HistoryHeader(
                songCount = historyItems.size,
                onPlayAll = { historyViewModel.playAllHistory() },
                onAddAllToQueue = { historyViewModel.addAllHistoryToQueue() }
            )
            HorizontalDivider()

            LazyColumn(
                modifier = Modifier.fillMaxSize(),
                contentPadding = contentPadding
            ) {
                items(
                    items = historyItems,
                    key = { item -> item.stableId }
                ) { item ->
                    UnifiedListItem(
                        item = item,
                        onItemClicked = { historyViewModel.playFromHistoryItem(item) },
                        videoListViewModel = videoListViewModel,
                        favoritesViewModel = favoritesViewModel,
                        playlistManagementViewModel = playlistManagementViewModel,
                        navController = navController
                    )
                }
            }
        }
    }
}

// ... HistoryHeader remains the same ...
@Composable
private fun HistoryHeader(
    songCount: Int,
    onPlayAll: () -> Unit,
    onAddAllToQueue: () -> Unit
) {
    Column(
        modifier = Modifier
            .fillMaxWidth()
            .padding(16.dp),
        verticalArrangement = Arrangement.spacedBy(16.dp)
    ) {
        Column {
            Text(
                text = stringResource(id = R.string.recently_played_songs),
```

```
                style = MaterialTheme.typography.titleLarge,
                fontWeight = FontWeight.Bold
            )
            Text(
                text = pluralStringResource(
                    id = R.plurals.song_count_label,
                    count = songCount,
                    songCount
                ),
                style = MaterialTheme.typography.bodyMedium,
                color = MaterialTheme.colorScheme.onSurfaceVariant
            )
        }

        Row(
            modifier = Modifier.fillMaxWidth(),
            horizontalArrangement = Arrangement.spacedBy(8.dp)
        ) {
            Button(
                onClick = onPlayAll,
                modifier = Modifier.weight(1f)
            ) {
                Icon(Icons.Default.PlayArrow, contentDescription = null)
                Spacer(Modifier.size(ButtonDefaults.IconSpacing))
                Text(stringResource(id = R.string.action_play))
            }
            OutlinedButton(
                onClick = onAddAllToQueue,
                modifier = Modifier.weight(1f)
            ) {
                Icon(Icons.AutoMirrored.Filled.PlaylistAdd, contentDescription = null)
                Spacer(Modifier.size(ButtonDefaults.IconSpacing))
                Text(stringResource(id = R.string.action_add_to_queue))
            }
        }
    }
}

// File: java\com\example\holodex\ui\screens\HomeScreen.kt
package com.example.holodex.ui.screens

import android.widget.Toast
import androidx.activity.compose.BackHandler
import androidx.compose.animation.AnimatedVisibility
import androidx.compose.animation.fadeIn
import androidx.compose.animation.fadeOut
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.calculateEndPadding
import androidx.compose.foundation.layout.calculateStartPadding
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
```

```
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.lazy.rememberLazyListState
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.automirrored.filled.ArrowBack
import androidx.compose.material.icons.filled.Clear
import androidx.compose.material.icons.filled.History
import androidx.compose.material.icons.filled.KeyboardArrowUp
import androidx.compose.material.icons.filled.Search
import androidx.compose.material.icons.filled.Settings
import androidx.compose.material.icons.filled.Tune
import androidx.compose.material3.Badge
import androidx.compose.material3.BadgedBox
import androidx.compose.material3.DockedSearchBar
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.FloatingActionButton
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.ModalBottomSheet
import androidx.compose.material3.Scaffold
import androidx.compose.material3.SnackbarHost
import androidx.compose.material3.SnackbarHostState
import androidx.compose.material3.Text
import androidx.compose.material3.rememberModalBottomSheetState
import androidx.compose.runtime.Composable
import androidx.compose.runtime.derivedStateOf
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.platform.LocalLayoutDirection
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.style.TextOverflow
import androidx.compose.ui.unit.dp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.media3.common.util.UnstableApi
import androidx.navigation.NavController
import com.example.holodex.R
import com.example.holodex.ui.composables.CustomPagedUnifiedList
import com.example.holodex.ui.composables.EmptyState
import com.example.holodex.ui.composables.LoadingSkeleton
import com.example.holodex.ui.composables.sheets.BrowseFiltersSheet
import com.example.holodex.ui.navigation.AppDestinations
import com.example.holodex.viewmodel.FavoritesViewModel
import com.example.holodex.viewmodel.MusicCategoryType
import com.example.holodex.viewmodel.PlaylistManagementViewModel
import com.example.holodex.viewmodel.VideoListSideEffect
import com.example.holodex.viewmodel.VideoListViewModel
import kotlinx.coroutines.launch
import org.orbitmvi.orbit.compose.collectAsState
```

```
import org.orbitmvi.orbit.compose.collectSideEffect
import timber.log.Timber

@UnstableApi
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun HomeScreen(
    navController: NavController,
    videoListViewModel: VideoListViewModel,
    playlistManagementViewModel: PlaylistManagementViewModel,
    contentPadding: PaddingValues // NEW PARAMETER
) {
    val favoritesViewModel: FavoritesViewModel = hiltViewModel()
    val context = LocalContext.current
    val coroutineScope = rememberCoroutineScope()
    val snackbarHostState = remember { SnackbarHostState() }
    val listState = rememberLazyListState()
    var showFilterSheet by remember { mutableStateOf(false) }

    // --- ORBIT STATE COLLECTION ---
    val state by videoListViewModel.collectAsState()

    // --- ORBIT SIDE EFFECTS ---
    videoListViewModel.collectSideEffect { effect ->
        when (effect) {
            is VideoListSideEffect.ShowToast -> {
                Toast.makeText(context, effect.message, Toast.LENGTH_SHORT).show()
            }
            is VideoListSideEffect.NavigateTo -> {
                when (val destination = effect.destination) {
                    is VideoListViewModel.NavigationDestination.VideoDetails -> {
                        navController.navigate(AppDestinations.videoDetailRoute(destination.vi
                    }
                    is VideoListViewModel.NavigationDestination.HomeScreenWithSearch -> {
                        // Already on Home Screen
                    }
                }
            }
        }
    }

    val searchHistory = state.searchHistory

    BackHandler(enabled = state.isSearchActive) {
        videoListViewModel.clearSearchAndReturnToBrowse()
    }

    Box(Modifier.fillMaxSize()) {
        Scaffold(
            snackbarHost = { SnackbarHost(snackbarHostState) },
            floatingActionButton = {
                val showFab by remember { derivedStateOf { listState.firstVisibleItemIndex > 5
                AnimatedVisibility(
                    visible = showFab && !state.isSearchActive,
                    enter = fadeIn(),
```

```
                exit = fadeOut()
            ) {
                // Adjust FAB padding to be above the bottom bar
                FloatingActionButton(
                    onClick = { coroutineScope.launch { listState.animateScrollToItem(0) }
                    modifier = Modifier.padding(bottom = contentPadding.calculateBottomPad
                ) {
                    Icon(Icons.Filled.KeyboardArrowUp, stringResource(R.string.scroll_to_t
                }
            }
        }
    ) { innerPadding ->

        // Combine the padding from this Scaffold (Search Bar offset) with the dynamic bot
        val unifiedPadding = PaddingValues(
            top = innerPadding.calculateTopPadding() + 80.dp, // Search bar height offset
            bottom = contentPadding.calculateBottomPadding() + 16.dp,
            start = innerPadding.calculateStartPadding(LocalLayoutDirection.current),
            end = innerPadding.calculateEndPadding(LocalLayoutDirection.current)
        )

        Box(modifier = Modifier.fillMaxSize()) {
            if (state.activeContextType == MusicCategoryType.SEARCH) {
                // --- SEARCH CONTENT ---
                if (state.searchIsLoadingInitial && state.searchItems.isEmpty()) {
                    LoadingSkeleton(modifier = Modifier.fillMaxSize().padding(top = 80.dp)
                } else if (state.searchItems.isEmpty() && state.searchEndOfList && !state.
                    EmptyState(
                        message = stringResource(R.string.status_search_no_results, state.
                        onRefresh = { videoListViewModel.refreshCurrentListViaPull() },
                        modifier = Modifier.padding(top = 80.dp)
                    )
                } else {
                    CustomPagedUnifiedList(
                        listKeyPrefix = "home_search",
                        items = state.searchItems,
                        listState = listState,
                        onItemClicked = { item ->
                            Timber.d("HomeScreen (Search): Click ${item.title}")
                            videoListViewModel.onVideoClicked(item)
                        },
                        videoListViewModel = videoListViewModel,
                        playlistManagementViewModel = playlistManagementViewModel,
                        navController = navController,
                        favoritesViewModel = favoritesViewModel,
                        isLoadingMore = state.searchIsLoadingInitial,
                        endOfList = state.searchEndOfList,
                        onLoadMore = { videoListViewModel.loadMore(MusicCategoryType.SEARC
                        isRefreshing = false,
                        onRefresh = {},
                        contentPadding = unifiedPadding // PASS THE PADDING HERE
                    )
                }
            } else {
                // --- BROWSE CONTENT ---
```

```
            if (state.browseIsLoadingInitial && state.browseItems.isEmpty()) {
                LoadingSkeleton(modifier = Modifier.fillMaxSize().padding(top = 80.dp)
            } else {
                CustomPagedUnifiedList(
                    listKeyPrefix = "home_browse",
                    items = state.browseItems,
                    listState = listState,
                    onItemClicked = { item ->
                        videoListViewModel.onVideoClicked(item)
                    },
                    videoListViewModel = videoListViewModel,
                    favoritesViewModel = favoritesViewModel,
                    playlistManagementViewModel = playlistManagementViewModel,
                    isLoadingMore = state.browseIsLoadingMore,
                    endOfList = state.browseEndOfList,
                    navController = navController,
                    onLoadMore = { videoListViewModel.loadMore(state.activeContextType
                    isRefreshing = state.browseIsRefreshing,
                    onRefresh = { videoListViewModel.refreshCurrentListViaPull() },
                    contentPadding = unifiedPadding // PASS THE PADDING HERE
                )
            }
        }
    }
}

// Search Bar (Overlay)
DockedSearchBar(
    modifier = Modifier
        .align(Alignment.TopCenter)
        .padding(top = 8.dp, start = 16.dp, end = 16.dp)
        .fillMaxWidth(),
    query = state.currentSearchQuery,
    onQueryChange = { query -> videoListViewModel.onSearchQueryChange(query) },
    onSearch = { query -> videoListViewModel.performSearch(query) },
    active = state.isSearchActive,
    onActiveChange = { active -> videoListViewModel.setSearchActive(active) },
    placeholder = {
        Text(
            stringResource(R.string.search_your_music_hint),
            maxLines = 1,
            overflow = TextOverflow.Ellipsis
        )
    },
    leadingIcon = {
        if (state.isSearchActive) {
            IconButton(onClick = { videoListViewModel.clearSearchAndReturnToBrowse() }
                Icon(Icons.AutoMirrored.Filled.ArrowBack, "Back")
            }
        } else {
            Icon(Icons.Filled.Search, "Search Icon")
        }
    },
    trailingIcon = {
        Row {
```

```
                    if (state.isSearchActive && state.currentSearchQuery.isNotEmpty()) {
                        IconButton(onClick = { videoListViewModel.onSearchQueryChange("") }) {
                            Icon(Icons.Filled.Clear, stringResource(R.string.action_clear_sear
                        }
                    }
                    if (!state.isSearchActive) {
                        IconButton(onClick = { showFilterSheet = true }) {
                            BadgedBox(badge = {
                                if (state.browseFilterState.hasActiveFilters) {
                                    Badge { Text(state.browseFilterState.activeFilterCount.toS
                                }
                            }) {
                                Icon(Icons.Filled.Tune, stringResource(R.string.action_filter_
                            }
                        }
                        IconButton(onClick = { navController.navigate(AppDestinations.SETTINGS
                            Icon(Icons.Filled.Settings, stringResource(R.string.settings_title
                        }
                    }
                }
            }
        ) {
            SearchHistoryList(
                searchHistory = searchHistory,
                onHistoryItemClick = { query ->
                    videoListViewModel.onSearchQueryChange(query)
                    videoListViewModel.performSearch(query)
                }
            )
        }
    }

    if (showFilterSheet) {
        ModalBottomSheet(
            onDismissRequest = { showFilterSheet = false },
            sheetState = rememberModalBottomSheetState(skipPartiallyExpanded = true)
        ) {
            BrowseFiltersSheet(
                initialFilters = state.browseFilterState,
                onFiltersApplied = { newFilters ->
                    showFilterSheet = false
                    videoListViewModel.updateBrowseFilters(newFilters)
                },
                onDismiss = { showFilterSheet = false },
                videoListViewModel = videoListViewModel
            )
        }
    }
}


@Composable
private fun SearchHistoryList(
    searchHistory: List<String>,
    onHistoryItemClick: (String) -> Unit
```

```
) {
    if (searchHistory.isEmpty()) {
        Box(
            modifier = Modifier
                .fillMaxSize()
                .padding(16.dp), contentAlignment = Alignment.Center
        ) {
            Text(
                "No recent searches",
                style = MaterialTheme.typography.bodyLarge,
                color = MaterialTheme.colorScheme.onSurfaceVariant
            )
        }
        return
    }

    LazyColumn(modifier = Modifier.fillMaxSize()) {
        item {
            Text(
                text = stringResource(R.string.search_history_title),
                style = MaterialTheme.typography.titleSmall,
                modifier = Modifier.padding(horizontal = 16.dp, vertical = 8.dp)
            )
        }
        items(searchHistory) { query ->
            Row(
                modifier = Modifier
                    .fillMaxWidth()
                    .clickable { onHistoryItemClick(query) }
                    .padding(horizontal = 16.dp, vertical = 12.dp),
                verticalAlignment = Alignment.CenterVertically
            ) {
                Icon(Icons.Filled.History, null, modifier = Modifier.padding(end = 16.dp))
                Text(query, style = MaterialTheme.typography.bodyLarge)
            }
        }
    }
}

// File: java\com\example\holodex\ui\screens\LibraryScreen.kt
// File: java/com/example/holodex/ui/screens/LibraryScreen.kt
package com.example.holodex.ui.screens

import androidx.compose.animation.animateColorAsState
import androidx.compose.animation.core.FastOutSlowInEasing
import androidx.compose.animation.core.animateDpAsState
import androidx.compose.animation.core.tween
import androidx.compose.foundation.ExperimentalFoundationApi
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.BoxWithConstraints
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.Row
```

```
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.offset
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.width
import androidx.compose.foundation.pager.HorizontalPager
import androidx.compose.foundation.pager.PagerState
import androidx.compose.foundation.pager.rememberPagerState
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.automirrored.filled.ViewList
import androidx.compose.material.icons.filled.Add
import androidx.compose.material.icons.filled.GridView
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.FloatingActionButton
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Scaffold
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.material3.TopAppBar
import androidx.compose.material3.TopAppBarDefaults
import androidx.compose.runtime.Composable
import androidx.compose.runtime.derivedStateOf
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.draw.clip
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.lifecycle.compose.collectAsStateWithLifecycle
import androidx.navigation.NavController
import com.example.holodex.R
import com.example.holodex.ui.dialogs.AddExternalChannelDialog
import com.example.holodex.ui.navigation.AppDestinations
import com.example.holodex.viewmodel.AddChannelViewModel
import com.example.holodex.viewmodel.PlaylistManagementViewModel
import kotlinx.coroutines.launch

private enum class LibraryTab(val titleRes: Int) {
    PLAYLISTS(R.string.bottom_nav_playlists),
    FAVORITES(R.string.bottom_nav_favorites),
    HISTORY(R.string.screen_title_history)
}

@OptIn(ExperimentalMaterial3Api::class, ExperimentalFoundationApi::class)
```

```
@Composable
fun LibraryScreen(
    navController: NavController,
    playlistManagementViewModel: PlaylistManagementViewModel,
    contentPadding: PaddingValues // NEW PARAMETER
) {
    val pagerState = rememberPagerState(initialPage = 1, pageCount = { LibraryTab.entries.size
    val coroutineScope = rememberCoroutineScope()
    var isGridView by remember { mutableStateOf(false) }

    val addChannelViewModel: AddChannelViewModel = hiltViewModel()
    val showAddChannelDialog by addChannelViewModel.showDialog.collectAsStateWithLifecycle()

    if (showAddChannelDialog) {
        AddExternalChannelDialog(onDismissRequest = { addChannelViewModel.closeDialog() })
    }

    Scaffold(
        topBar = {
            LibraryTopAppBar(
                isGridView = isGridView,
                onViewToggle = { isGridView = !isGridView }
            )
        },
        floatingActionButton = {
            FloatingActionButton(
                onClick = { addChannelViewModel.openDialog() },
                // Adjust FAB padding
                modifier = Modifier.padding(bottom = contentPadding.calculateBottomPadding())
            ) {
                Icon(Icons.Default.Add, contentDescription = "Add External Channel")
            }
        }
    ) { innerPadding ->

        // Combine padding
        val unifiedPadding = PaddingValues(
            top = innerPadding.calculateTopPadding(),
            bottom = contentPadding.calculateBottomPadding() + 16.dp
        )

        Column(
            modifier = Modifier
                .fillMaxSize()
                .padding(top = unifiedPadding.calculateTopPadding()) // Only top here
        ) {
            AnimatedCustomTabRow(
                selectedTabIndex = pagerState.currentPage,
                tabs = LibraryTab.entries.map { stringResource(it.titleRes) },
                pagerState = pagerState,
                onTabSelected = { index ->
                    coroutineScope.launch { pagerState.animateScrollToPage(index) }
                }
            )
```

```
            HorizontalPager(state = pagerState, modifier = Modifier.fillMaxSize()) { page ->
                when (LibraryTab.entries[page]) {
                    LibraryTab.FAVORITES -> FavoritesTab(
                        isGridView = isGridView,
                        navController = navController,
                        playlistManagementViewModel = playlistManagementViewModel,
                        contentPadding = PaddingValues(bottom = unifiedPadding.calculateBottom
                    )

                    LibraryTab.PLAYLISTS -> PlaylistsTab(
                        onPlaylistClicked = { playlist ->
                            val idToNavigate = when {
                                playlist.playlistId < 0 && playlist.serverId == null -> playli
                                playlist.playlistId > 0 -> playlist.playlistId.toString()
                                else -> playlist.serverId
                            }
                            if (!idToNavigate.isNullOrBlank()) {
                                navController.navigate(
                                    AppDestinations.playlistDetailsRoute(
                                        idToNavigate
                                    )
                                )
                            }
                        },
                        contentPadding = PaddingValues(bottom = unifiedPadding.calculateBottom
                    )

                    LibraryTab.HISTORY -> HistoryTab(
                        navController = navController,
                        playlistManagementViewModel = playlistManagementViewModel,
                        contentPadding = PaddingValues(bottom = unifiedPadding.calculateBottom
                    )
                }
            }
        }
    }
}


@OptIn(ExperimentalMaterial3Api::class)
@Composable
private fun LibraryTopAppBar(
    isGridView: Boolean,
    onViewToggle: () -> Unit
) {
    TopAppBar(
        title = {
            Text(
                text = stringResource(R.string.bottom_nav_library),
                style = MaterialTheme.typography.headlineSmall,
                fontWeight = FontWeight.Bold
            )
        },
        actions = {
            IconButton(onClick = onViewToggle) {
```

```
                Icon(
                    imageVector = if (isGridView) Icons.AutoMirrored.Filled.ViewList else Icon
                    contentDescription = stringResource(if (isGridView) R.string.action_view_a
                )
            }
        },
        colors = TopAppBarDefaults.topAppBarColors(
            containerColor = MaterialTheme.colorScheme.surface,
            titleContentColor = MaterialTheme.colorScheme.onSurface
        )
    )
}

@OptIn(ExperimentalFoundationApi::class)
@Composable
private fun AnimatedCustomTabRow(
    selectedTabIndex: Int,
    tabs: List<String>,
    pagerState: PagerState,
    onTabSelected: (Int) -> Unit,
    modifier: Modifier = Modifier,
    containerColor: Color = MaterialTheme.colorScheme.surface,
    indicatorColor: Color = MaterialTheme.colorScheme.primary
) {
    BoxWithConstraints(
        modifier = modifier
            .fillMaxWidth()
            .background(containerColor)
            .padding(horizontal = 16.dp, vertical = 8.dp)
    ) {
        val tabWidth = this@BoxWithConstraints.maxWidth / tabs.size
        Box(
            modifier = Modifier
                .fillMaxWidth()
                .height(48.dp)
                .background(
                    color = MaterialTheme.colorScheme.surfaceVariant.copy(alpha = 0.3f),
                    shape = RoundedCornerShape(24.dp)
                )
        )
        val targetOffset = tabWidth * selectedTabIndex
        val pagerOffset = tabWidth * pagerState.currentPageOffsetFraction
        val indicatorOffset = targetOffset + pagerOffset
        val animatedIndicatorOffset by animateDpAsState(
            targetValue = indicatorOffset,
            animationSpec = tween(durationMillis = 250, easing = FastOutSlowInEasing),
            label = "indicator_offset"
        )
        Box(
            modifier = Modifier
                .offset(x = animatedIndicatorOffset)
                .width(tabWidth)
                .height(48.dp)
                .padding(4.dp)
                .background(color = indicatorColor, shape = RoundedCornerShape(20.dp))
```

```
        )
        Row(modifier = Modifier.fillMaxWidth(), horizontalArrangement = Arrangement.SpaceEvenl
            tabs.forEachIndexed { index, title ->
                AnimatedTab(
                    title = title,
                    selected = selectedTabIndex == index,
                    onClick = { onTabSelected(index) },
                    modifier = Modifier.weight(1f)
                )
            }
        }
    }
}

@Composable
private fun AnimatedTab(
    title: String,
    selected: Boolean,
    onClick: () -> Unit,
    modifier: Modifier = Modifier
) {
    val animatedColor by animateColorAsState(
        targetValue = if (selected) MaterialTheme.colorScheme.onPrimary else MaterialTheme.col
        animationSpec = tween(durationMillis = 150, easing = FastOutSlowInEasing),
        label = "tab_color"
    )
    val animatedFontWeight by remember { derivedStateOf { if (selected) FontWeight.Bold else F
    Surface(
        modifier = modifier
            .height(48.dp)
            .clip(RoundedCornerShape(24.dp)),
        onClick = onClick,
        color = Color.Transparent,
        contentColor = animatedColor,
        shape = RoundedCornerShape(24.dp)
    ) {
        Box(contentAlignment = Alignment.Center, modifier = Modifier.fillMaxSize()) {
            Text(
                text = title,
                color = animatedColor,
                style = MaterialTheme.typography.titleSmall,
                fontWeight = animatedFontWeight
            )
        }
    }
}

@Composable
private fun PlaylistsTab(
    onPlaylistClicked: (com.example.holodex.data.db.PlaylistEntity) -> Unit,
    contentPadding: PaddingValues
) {
    PlaylistsScreen(
        modifier = Modifier.fillMaxSize(),
        playlistManagementViewModel = hiltViewModel(),
```

```
        onPlaylistClicked = onPlaylistClicked,
        contentPadding = contentPadding
    )
}

@Composable
private fun FavoritesTab(
    isGridView: Boolean,
    navController: NavController,
    playlistManagementViewModel: PlaylistManagementViewModel,
    contentPadding: PaddingValues
) {
    FavoritesScreen(
        isGridView = isGridView,
        modifier = Modifier.fillMaxSize(),
        videoListViewModel = hiltViewModel(),
        favoritesViewModel = hiltViewModel(),
        playlistManagementViewModel = playlistManagementViewModel,
        navController = navController,
        contentPadding = contentPadding // PASS IT
    )
}

@Composable
private fun HistoryTab(
    navController: NavController,
    playlistManagementViewModel: PlaylistManagementViewModel,
    contentPadding: PaddingValues
) {
    HistoryScreen(
        modifier = Modifier.fillMaxSize(),
        navController = navController,
        videoListViewModel = hiltViewModel(),
        favoritesViewModel = hiltViewModel(),
        playlistManagementViewModel = playlistManagementViewModel,
        contentPadding = contentPadding // PASS IT
    )
}

// File: java\com\example\holodex\ui\screens\PlaylistDetailsScreen.kt
@file:OptIn(ExperimentalMaterial3Api::class, ExperimentalFoundationApi::class)

package com.example.holodex.ui.screens

import android.widget.Toast
import androidx.compose.foundation.BorderStroke
import androidx.compose.foundation.ExperimentalFoundationApi
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.calculateEndPadding
import androidx.compose.foundation.layout.calculateStartPadding
```

```
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.itemsIndexed
import androidx.compose.foundation.lazy.rememberLazyListState
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.automirrored.filled.ArrowBack
import androidx.compose.material.icons.automirrored.filled.PlaylistAdd
import androidx.compose.material.icons.automirrored.filled.PlaylistPlay
import androidx.compose.material.icons.filled.Cancel
import androidx.compose.material.icons.filled.Check
import androidx.compose.material.icons.filled.Edit
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.HorizontalDivider
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.LocalContentColor
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.OutlinedButton
import androidx.compose.material3.Scaffold
import androidx.compose.material3.SnackbarDuration
import androidx.compose.material3.SnackbarHost
import androidx.compose.material3.SnackbarHostState
import androidx.compose.material3.Text
import androidx.compose.material3.TopAppBar
import androidx.compose.material3.TopAppBarDefaults
import androidx.compose.runtime.Composable
import androidx.compose.runtime.CompositionLocalProvider
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.derivedStateOf
import androidx.compose.runtime.getValue
import androidx.compose.runtime.remember
import androidx.compose.runtime.rememberCoroutineScope
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.hapticfeedback.HapticFeedbackType
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.platform.LocalHapticFeedback
import androidx.compose.ui.platform.LocalLayoutDirection
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.res.pluralStringResource
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.text.style.TextOverflow
import androidx.compose.ui.unit.dp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.media3.common.util.UnstableApi
import androidx.navigation.NavController
```

```
import com.example.holodex.R
import com.example.holodex.data.db.PlaylistEntity
import com.example.holodex.data.model.discovery.PlaylistStub
import com.example.holodex.ui.composables.EmptyState
import com.example.holodex.ui.composables.ErrorStateWithRetry
import com.example.holodex.ui.composables.LoadingState
import com.example.holodex.ui.composables.SimpleProcessedBackground
import com.example.holodex.ui.composables.UnifiedListItem
import com.example.holodex.util.ArtworkResolver
import com.example.holodex.util.DynamicTheme
import com.example.holodex.util.findActivity
import com.example.holodex.viewmodel.FavoritesViewModel
import com.example.holodex.viewmodel.PlaylistDetailsSideEffect
import com.example.holodex.viewmodel.PlaylistDetailsViewModel
import com.example.holodex.viewmodel.PlaylistManagementViewModel
import com.example.holodex.viewmodel.UnifiedDisplayItem
import com.example.holodex.viewmodel.VideoListViewModel
import kotlinx.coroutines.launch
import org.orbitmvi.orbit.compose.collectAsState
import org.orbitmvi.orbit.compose.collectSideEffect
import sh.calvin.reorderable.ReorderableItem
import sh.calvin.reorderable.rememberReorderableLazyListState

@OptIn(UnstableApi::class)
@Composable
fun PlaylistDetailsScreen(
    navController: NavController,
    onNavigateUp: () -> Unit,
    playlistManagementViewModel: PlaylistManagementViewModel,
    contentPadding: PaddingValues = PaddingValues(0.dp) // NEW PARAMETER
) {
    val playlistDetailsViewModel: PlaylistDetailsViewModel = hiltViewModel()
    val state by playlistDetailsViewModel.collectAsState()

    // ... (Derived variables remain same) ...
    val playlistDetails = state.playlist
    val items = state.items
    val isEditMode = state.isEditMode
    val editablePlaylist = state.editablePlaylist
    val dynamicTheme = state.dynamicTheme
    val isPlaylistOwned = state.isPlaylistOwned
    val isShuffleActive = state.isShuffleActive

    val snackbarHostState = remember { SnackbarHostState() }
    val coroutineScope = rememberCoroutineScope()
    val context = LocalContext.current

    val videoListViewModel: VideoListViewModel = hiltViewModel(findActivity())
    val favoritesViewModel: FavoritesViewModel = hiltViewModel()

    val backgroundImageUrl by remember(items, playlistDetails) {
        derivedStateOf {
            items.firstOrNull()?.artworkUrls?.firstOrNull()
                ?: playlistDetails?.let {
                    val stub = PlaylistStub(it.serverId ?: it.playlistId.toString(), it.name ?
```

```
                ArtworkResolver.getPlaylistArtworkUrl(stub)
            }
        }
    }

    playlistDetailsViewModel.collectSideEffect { effect ->
        when (effect) {
            is PlaylistDetailsSideEffect.ShowToast -> {
                Toast.makeText(context, effect.message, Toast.LENGTH_SHORT).show()
            }
        }
    }

    LaunchedEffect(state.error) {
        state.error?.let {
            coroutineScope.launch {
                snackbarHostState.showSnackbar(message = it, duration = SnackbarDuration.Long)
                playlistDetailsViewModel.clearError()
            }
        }
    }

    Scaffold(
        snackbarHost = { SnackbarHost(snackbarHostState) },
        topBar = {
            // ... (TopAppBar remains same) ...
            TopAppBar(
                title = {
                    if (!isEditMode) {
                        Text(text = playlistDetails?.name ?: "", maxLines = 1, overflow = Text
                    }
                },
                navigationIcon = {
                    IconButton(onClick = onNavigateUp) {
                        Icon(Icons.AutoMirrored.Filled.ArrowBack, contentDescription = stringR
                    }
                },
                actions = {
                    if (isEditMode) {
                        IconButton(onClick = { playlistDetailsViewModel.cancelEditMode() }) {
                        IconButton(onClick = { playlistDetailsViewModel.saveChanges() }) { Ico
                    } else {
                        if (isPlaylistOwned) {
                            IconButton(onClick = { playlistDetailsViewModel.enterEditMode() })
                        }
                        IconButton(onClick = { playlistDetailsViewModel.togglePlaylistShuffleM
                            Icon(
                                painter = painterResource(id = if (isShuffleActive) R.drawable
                                contentDescription = stringResource(R.string.action_shuffle),
                                tint = if (isShuffleActive) dynamicTheme.primary else dynamicT
                            )
                        }
                    }
                },
                colors = TopAppBarDefaults.topAppBarColors(
```

```
                    containerColor = Color.Transparent,
                    titleContentColor = dynamicTheme.onPrimary,
                    navigationIconContentColor = dynamicTheme.onPrimary,
                    actionIconContentColor = dynamicTheme.onPrimary
                )
            )
        }
    ) { paddingValues ->
        Box(modifier = Modifier.fillMaxSize()) {
            SimpleProcessedBackground(artworkUri = backgroundImageUrl, dynamicColor = dynamicT
            CompositionLocalProvider(LocalContentColor provides dynamicTheme.onPrimary) {
                // Apply top padding from Scaffold (TopAppBar)
                Box(modifier = Modifier.padding(paddingValues).fillMaxSize()) {
                    when {
                        state.isLoading && items.isEmpty() -> {
                            LoadingState(message = stringResource(R.string.loading_content_mes
                        }
                        state.error != null && items.isEmpty() -> {
                            ErrorStateWithRetry(message = state.error!!, onRetry = { playlistD
                        }
                        items.isEmpty() && !state.isLoading -> {
                            EmptyState(message = stringResource(R.string.message_playlist_is_e
                        }
                        else -> {
                            PlaylistContent(
                                items = items,
                                playlistDetails = if (isEditMode) editablePlaylist else playli
                                isEditMode = isEditMode,
                                navController = navController,
                                videoListViewModel = videoListViewModel,
                                favoritesViewModel = favoritesViewModel,
                                playlistManagementViewModel = playlistManagementViewModel,
                                playlistDetailsViewModel = playlistDetailsViewModel,
                                dynamicTheme = dynamicTheme,
                                // PASS THE PADDING HERE
                                contentPadding = contentPadding
                            )
                        }
                    }
                }
            }
        }
    }
}

@UnstableApi
@Composable
private fun PlaylistContent(
    items: List<UnifiedDisplayItem>,
    playlistDetails: PlaylistEntity?,
    isEditMode: Boolean,
    navController: NavController,
    videoListViewModel: VideoListViewModel,
    favoritesViewModel: FavoritesViewModel,
    playlistManagementViewModel: PlaylistManagementViewModel,
```

```
    playlistDetailsViewModel: PlaylistDetailsViewModel,
    dynamicTheme: DynamicTheme,
    contentPadding: PaddingValues // NEW PARAMETER
) {
    val lazyListState = rememberLazyListState()
    val reorderableState = rememberReorderableLazyListState(
        lazyListState = lazyListState,
        onMove = { from, to -> playlistDetailsViewModel.reorderItemInEditMode(from.index, to.i
    )

    // Calculate actual bottom padding: passed padding + extra space
    val layoutDirection = LocalLayoutDirection.current
    val effectivePadding = PaddingValues(
        bottom = contentPadding.calculateBottomPadding() + 16.dp, // Add extra space at bottom
        top = 0.dp,
        start = contentPadding.calculateStartPadding(layoutDirection),
        end = contentPadding.calculateEndPadding(layoutDirection)
    )

    LazyColumn(
        state = lazyListState,
        modifier = Modifier.fillMaxSize(),
        contentPadding = effectivePadding // Use dynamic padding
    ) {
        item {
            if (isEditMode && playlistDetails != null) {
                EditablePlaylistHeader(
                    playlist = playlistDetails,
                    onNameChange = { playlistDetailsViewModel.updateDraftName(it) },
                    onDescriptionChange = { playlistDetailsViewModel.updateDraftDescription(it
                )
            } else {
                PlaylistHeader(
                    playlist = playlistDetails,
                    itemCount = items.size,
                    onPlayAll = { playlistDetailsViewModel.playAllItemsInPlaylist() },
                    onAddAllToQueue = { playlistDetailsViewModel.addAllToQueue() },
                    dynamicTheme = dynamicTheme
                )
            }
            HorizontalDivider(color = LocalContentColor.current.copy(alpha = 0.2f))
        }

        itemsIndexed(
            items = items,
            key = { _, item -> item.stableId }
        ) { index, item ->
            ReorderableItem(state = reorderableState, key = item.stableId, enabled = isEditMod
                UnifiedListItem(
                    item = item,
                    onItemClicked = { if (!isEditMode) playlistDetailsViewModel.playFromItem(i
                    navController = navController,
                    videoListViewModel = videoListViewModel,
                    favoritesViewModel = favoritesViewModel,
                    playlistManagementViewModel = playlistManagementViewModel,
```

```
                    isEditing = isEditMode,
                    onRemoveClicked = { playlistDetailsViewModel.removeItemInEditMode(item) },
                    dragHandleModifier = Modifier.longPressDraggableHandle()
                )
            }
        }
    }
}

@Composable
private fun PlaylistHeader(
    playlist: PlaylistEntity?,
    itemCount: Int,
    onPlayAll: () -> Unit,
    onAddAllToQueue: () -> Unit,
    dynamicTheme: DynamicTheme
) {
    val haptic = LocalHapticFeedback.current

    Column(
        modifier = Modifier
            .fillMaxWidth()
            .padding(16.dp),
        verticalArrangement = Arrangement.spacedBy(8.dp),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        playlist?.let {
            Text(
                text = it.name ?: "Untitled Playlist",
                style = MaterialTheme.typography.headlineSmall,
                fontWeight = FontWeight.Bold,
                textAlign = TextAlign.Center
            )
            val description = it.description
            if (!description.isNullOrBlank()) {
                Text(
                    text = description,
                    style = MaterialTheme.typography.bodyMedium,
                    textAlign = TextAlign.Center
                )
            }
        }
        Text(
            text = pluralStringResource(R.plurals.item_count, itemCount, itemCount),
            style = MaterialTheme.typography.labelMedium
        )

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

        if (itemCount > 0) {
            ActionButtons(
                onPlayAll = {
                    onPlayAll()
                    haptic.performHapticFeedback(HapticFeedbackType.LongPress)
                },
```

```
                onAddAllToQueue = {
                    onAddAllToQueue()
                    haptic.performHapticFeedback(HapticFeedbackType.LongPress)
                },
                dynamicTheme = dynamicTheme
            )
        }
    }
}

@Composable
private fun ActionButtons(
    onPlayAll: () -> Unit,
    onAddAllToQueue: () -> Unit,
    dynamicTheme: DynamicTheme
) {
    Row(modifier = Modifier.fillMaxWidth(), horizontalArrangement = Arrangement.spacedBy(8.dp)
        Button(
            onClick = onPlayAll,
            modifier = Modifier.weight(1f),
            colors = ButtonDefaults.buttonColors(
                containerColor = dynamicTheme.onPrimary,
                contentColor = dynamicTheme.primary
            )
        ) {
            Icon(Icons.AutoMirrored.Filled.PlaylistPlay, contentDescription = null, modifier =
            Spacer(Modifier.size(ButtonDefaults.IconSpacing))
            Text(stringResource(R.string.action_play_all))
        }

        OutlinedButton(
            onClick = onAddAllToQueue,
            modifier = Modifier.weight(1f),
            colors = ButtonDefaults.outlinedButtonColors(contentColor = dynamicTheme.onPrimary
            border = BorderStroke(1.dp, dynamicTheme.onPrimary.copy(alpha = 0.5f))
        ) {
            Icon(Icons.AutoMirrored.Filled.PlaylistAdd, contentDescription = null, modifier =
            Spacer(Modifier.size(ButtonDefaults.IconSpacing))
            Text(stringResource(id = R.string.action_add_to_queue))
        }
    }
}

// File: java\com\example\holodex\ui\screens\PlaylistsScreen.kt
package com.example.holodex.ui.screens

import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.lazy.LazyColumn
```

```
import androidx.compose.foundation.lazy.items
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.automirrored.filled.PlaylistPlay
import androidx.compose.material.icons.filled.Delete
import androidx.compose.material3.Button
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.HorizontalDivider
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.ListItem
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.style.TextOverflow
import androidx.compose.ui.unit.dp
import androidx.lifecycle.compose.collectAsStateWithLifecycle
import com.example.holodex.R
import com.example.holodex.data.db.PlaylistEntity
import com.example.holodex.ui.dialogs.CreatePlaylistDialog
import com.example.holodex.viewmodel.PlaylistManagementViewModel

@Composable
fun PlaylistsScreen(
    modifier: Modifier = Modifier,
    playlistManagementViewModel: PlaylistManagementViewModel,
    onPlaylistClicked: (PlaylistEntity) -> Unit,
    contentPadding: PaddingValues = PaddingValues(0.dp) // NEW PARAMETER
) {
    val playlists: List<PlaylistEntity> by playlistManagementViewModel.allDisplayablePlaylists
    val showCreateDialog by playlistManagementViewModel.showCreatePlaylistDialog.collectAsStat

    if (showCreateDialog) {
        CreatePlaylistDialog(
            onDismissRequest = { playlistManagementViewModel.closeCreatePlaylistDialog() },
            onCreatePlaylist = { name, description ->
                playlistManagementViewModel.confirmCreatePlaylist(name, description)
            }
        )
    }

    Box(modifier = modifier.fillMaxSize()) {
        if (playlists.isEmpty()) {
            Box(modifier = Modifier.fillMaxSize(), contentAlignment = Alignment.Center) {
                Column(horizontalAlignment = Alignment.CenterHorizontally) {
                    Text(stringResource(R.string.message_no_playlists_yet))
                    Spacer(Modifier.height(8.dp))
                    Button(onClick = { playlistManagementViewModel.openCreatePlaylistDialog()
                        Text(stringResource(R.string.action_create_playlist))
                    }
                }
            }
```

```
        } else {
            LazyColumn(
                // *** FIX: Use the dynamic content padding ***
                contentPadding = contentPadding
            ) {
                items(playlists, key = { it.playlistId }) { playlist ->
                    PlaylistItemRow(
                        playlist = playlist,
                        onPlaylistClicked = { onPlaylistClicked(playlist) },
                        onDeleteClicked = { playlistManagementViewModel.deletePlaylist(playlis
                    )
                    HorizontalDivider()
                }
            }
        }
    }
}


@OptIn(ExperimentalMaterial3Api::class)
@Composable
private fun PlaylistItemRow(
    playlist: PlaylistEntity,
    onPlaylistClicked: () -> Unit,
    onDeleteClicked: () -> Unit
) {
    ListItem(
        headlineContent = { Text(playlist.name ?: "Untitled Playlist", maxLines = 1, overflow
        supportingContent = {
            playlist.description?.takeIf { it.isNotBlank() }?.let {
                Text(it, maxLines = 2, overflow = TextOverflow.Ellipsis, style = MaterialTheme
            }
        },
        leadingContent = {
            Icon(
                Icons.AutoMirrored.Filled.PlaylistPlay,
                contentDescription = "Playlist icon",
                modifier = Modifier.size(24.dp)
            )
        },
        trailingContent = {
            IconButton(onClick = onDeleteClicked) {
                Icon(Icons.Filled.Delete, contentDescription = stringResource(R.string.action_
            }
        },
        modifier = Modifier.clickable(onClick = onPlaylistClicked)
    )
}

// File: java\com\example\holodex\ui\screens\SettingsScreen.kt
package com.example.holodex.ui.screens

import android.Manifest
import android.content.pm.PackageManager
import android.os.Build
```

```
import android.widget.Toast
import androidx.activity.compose.rememberLauncherForActivityResult
import androidx.activity.result.contract.ActivityResultContracts
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.ColumnScope
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.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.selection.selectable
import androidx.compose.foundation.selection.selectableGroup
import androidx.compose.foundation.verticalScroll
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.automirrored.filled.ArrowBack
import androidx.compose.material.icons.automirrored.filled.Login
import androidx.compose.material.icons.filled.Add
import androidx.compose.material.icons.filled.Build
import androidx.compose.material.icons.filled.Clear
import androidx.compose.material.icons.filled.CloudSync
import androidx.compose.material.icons.filled.DocumentScanner
import androidx.compose.material.icons.filled.FolderOpen
import androidx.compose.material.icons.filled.Link
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.CircularProgressIndicator
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.HorizontalDivider
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.ListItem
import androidx.compose.material3.ListItemDefaults
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.RadioButton
import androidx.compose.material3.Scaffold
import androidx.compose.material3.Switch
import androidx.compose.material3.Text
import androidx.compose.material3.TextButton
import androidx.compose.material3.TopAppBar
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.graphics.Color
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.platform.LocalUriHandler
```

```
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.semantics.Role
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextOverflow
import androidx.compose.ui.unit.dp
import androidx.core.content.ContextCompat
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.lifecycle.compose.collectAsStateWithLifecycle
import androidx.media3.common.util.UnstableApi
import androidx.navigation.NavController
import androidx.work.OneTimeWorkRequestBuilder
import com.example.holodex.BuildConfig
import com.example.holodex.R
import com.example.holodex.auth.AuthState
import com.example.holodex.auth.AuthViewModel
import com.example.holodex.background.ChannelRepairWorker
import com.example.holodex.data.AppPreferenceConstants
import com.example.holodex.data.ThemePreference
import com.example.holodex.ui.composables.ApiKeyInputScreen
import com.example.holodex.ui.dialogs.AddExternalChannelDialog
import com.example.holodex.ui.navigation.AppDestinations
import com.example.holodex.viewmodel.ScanStatus
import com.example.holodex.viewmodel.SettingsSideEffect
import com.example.holodex.viewmodel.SettingsViewModel
import org.orbitmvi.orbit.compose.collectAsState
import org.orbitmvi.orbit.compose.collectSideEffect

@OptIn(ExperimentalMaterial3Api::class)
@UnstableApi
@Composable
fun SettingsScreen(
    navController: NavController,
    onNavigateUp: () -> Unit,
    onApiKeySavedRestartNeeded: () -> Unit
) {
    val authViewModel: AuthViewModel = hiltViewModel()
    val settingsViewModel: SettingsViewModel = hiltViewModel()

    val context = LocalContext.current
    val authState by authViewModel.authState.collectAsStateWithLifecycle()
    val state by settingsViewModel.collectAsState()

    // Local state for the dialog visibility
    // We keep this local because opening/closing a dialog is UI state, not business logic
    var showAddChannelDialog by remember { mutableStateOf(false) }

    // Dialog Composable
    if (showAddChannelDialog) {
        // The dialog uses its own Hilt ViewModel (AddChannelViewModel) internally
        // which is fully migrated to the Unified System.
        AddExternalChannelDialog(
            onDismissRequest = { showAddChannelDialog = false }
        )
    }
```

```
val permissionLauncher = rememberLauncherForActivityResult(
    ActivityResultContracts.RequestPermission()
) { isGranted: Boolean ->
    if (isGranted) {
        Toast.makeText(context, "Permission granted. Starting scan...", Toast.LENGTH_SHORT
        settingsViewModel.runLegacyFileScan()
    } else {
        Toast.makeText(context, "Storage permission is required.", Toast.LENGTH_LONG).show
    }
}

var isClearingCache by remember { mutableStateOf(false) }

val folderPickerLauncher = rememberLauncherForActivityResult(
    contract = ActivityResultContracts.OpenDocumentTree(),
    onResult = { uri ->
        if (uri != null) {
            settingsViewModel.saveDownloadLocation(uri)
        }
    }
)
var showRestartMessageForDataSettings by remember { mutableStateOf(false) }

settingsViewModel.collectSideEffect { effect ->
    when(effect) {
        is SettingsSideEffect.ShowToast -> Toast.makeText(context, effect.message, Toast.L
    }
}

LaunchedEffect(state.cacheClearStatus) {
    state.cacheClearStatus?.let { status ->
        Toast.makeText(context, status, Toast.LENGTH_LONG).show()
        settingsViewModel.resetCacheClearStatus()
        isClearingCache = false
    }
}

LaunchedEffect(state.scanStatus) {
    when (val status = state.scanStatus) {
        is ScanStatus.Complete -> {
            val message = if (status.importedCount > 0) "Successfully imported ${status.im
            Toast.makeText(context, message, Toast.LENGTH_LONG).show()
            settingsViewModel.resetScanStatus()
        }
        is ScanStatus.Error -> {
            Toast.makeText(context, status.message, Toast.LENGTH_LONG).show()
            settingsViewModel.resetScanStatus()
        }
        else -> {}
    }
}

if (showRestartMessageForDataSettings) {
    LaunchedEffect(Unit) {
        Toast.makeText(context, "Settings apply after app restart.", Toast.LENGTH_LONG).sh
```

```
        showRestartMessageForDataSettings = false
    }
}

Scaffold(
    topBar = {
        TopAppBar(
            title = { Text(stringResource(R.string.settings_title)) },
            navigationIcon = {
                IconButton(onClick = onNavigateUp) {
                    Icon(Icons.AutoMirrored.Filled.ArrowBack, stringResource(R.string.acti
                }
            }
        )
    }
) { paddingValues ->
    Column(
        modifier = Modifier
            .padding(paddingValues)
            .fillMaxSize()
            .verticalScroll(rememberScrollState())
            .padding(horizontal = 16.dp),
        verticalArrangement = Arrangement.spacedBy(0.dp)
    ) {
        // ... (API Key Section - Keep as is) ...
        SettingsSectionTitle(stringResource(R.string.settings_section_api_key))
        ApiKeyInputScreen(
            settingsViewModel = settingsViewModel,
            onApiKeySavedSuccessfully = { onApiKeySavedRestartNeeded() },
            modifier = Modifier.padding(bottom = 16.dp)
        )

        HorizontalDivider()

        // --- NEW SECTION: CONTENT SOURCES ---
        SettingsSectionTitle("Content Sources")

        ListItem(
            headlineContent = { Text("Add YouTube Channel") },
            supportingContent = { Text("Import music from external YouTube channels.", sty
            leadingContent = { Icon(Icons.Default.Add, contentDescription = null) },
            modifier = Modifier.clickable {
                showAddChannelDialog = true // Open Dialog
            },
            colors = ListItemDefaults.colors(containerColor = Color.Transparent)
        )

        HorizontalDivider()
        // ------------------------------------
        SettingsSectionTitle("Debug & Maintenance")

        ListItem(
            headlineContent = { Text("Fix Channel Metadata") },
            supportingContent = { Text("Detects and fixes incorrectly labeled channels (Ex
            leadingContent = { Icon(Icons.Default.Build, null) },
```

```
        modifier = Modifier.clickable {
            val request = OneTimeWorkRequestBuilder<ChannelRepairWorker>().build()
            settingsViewModel.enqueueWork(request) // You might need to expose WorkMan
            Toast.makeText(context, "Migration started. Check logs.", Toast.LENGTH_SHO
        }
    )
    SettingsSectionTitle(stringResource(R.string.settings_section_account))
    // ... (Account Section - Keep as is) ...
    when (val s = authState) {
        is AuthState.LoggedIn -> {
            ListItem(
                headlineContent = { Text("Logged In") },
                supportingContent = { Text("Your data is being synchronized.") },
                leadingContent = { Icon(Icons.Default.CloudSync, null) },
                trailingContent = { TextButton(onClick = { authViewModel.logout() }) {
                colors = ListItemDefaults.colors(containerColor = Color.Transparent)
            )
            Button(
                onClick = { settingsViewModel.triggerManualSync() },
                modifier = Modifier.fillMaxWidth().padding(horizontal = 16.dp, vertica
            ) {
                Icon(Icons.Default.CloudSync, null, modifier = Modifier.size(ButtonDef
                Spacer(modifier = Modifier.size(ButtonDefaults.IconSpacing))
                Text("Sync Now")
            }
        }
        is AuthState.LoggedOut, is AuthState.Error -> {
            ListItem(
                headlineContent = { Text(stringResource(R.string.action_login)) },
                supportingContent = { Text(stringResource(R.string.settings_desc_login
                leadingContent = { Icon(Icons.AutoMirrored.Filled.Login, null) },
                modifier = Modifier.clickable { navController.navigate(AppDestinations
                colors = ListItemDefaults.colors(containerColor = Color.Transparent)
            )
            if (s is AuthState.Error) {
                Text("Login failed: ${s.message}", color = MaterialTheme.colorScheme.e
            }
        }
        is AuthState.InProgress -> {
            ListItem(headlineContent = { Text("Logging in...") }, leadingContent = { C
        }
    }

    HorizontalDivider()
    SettingsSectionTitle(stringResource(R.string.settings_section_playback))
    // ... (Playback Section - Keep as is) ...
    ListItem(
        headlineContent = { Text(stringResource(R.string.settings_label_autoplay_next_
        supportingContent = { Text(stringResource(R.string.settings_desc_autoplay_next
        trailingContent = {
            Switch(
                checked = state.autoplayEnabled,
                onCheckedChange = { settingsViewModel.setAutoplayNextVideoEnabled(it)
            )
        },
```

```
    modifier = Modifier.fillMaxWidth().clickable { settingsViewModel.setAutoplayNe
    colors = ListItemDefaults.colors(containerColor = Color.Transparent)
)
ListItem(
    headlineContent = { Text(stringResource(R.string.settings_label_shuffle_on_pla
    supportingContent = { Text(stringResource(R.string.settings_desc_shuffle_on_pl
    trailingContent = {
        Switch(
            checked = state.shuffleOnPlayStartEnabled,
            onCheckedChange = { settingsViewModel.setShuffleOnPlayStartEnabled(it)
        )
    },
    modifier = Modifier.fillMaxWidth().clickable { settingsViewModel.setShuffleOnP
    colors = ListItemDefaults.colors(containerColor = Color.Transparent)
)

Spacer(modifier = Modifier.height(16.dp))
HorizontalDivider()

SettingsSectionTitle(stringResource(R.string.settings_section_data_performance))

// ... (Data & Performance - Keep Migrate Button and Legacy Import) ...

ListItem(
    headlineContent = { Text("Import Legacy Downloads") },
    supportingContent = { Text("Scan the HolodexMusic folder for any downloads not
    leadingContent = {
        if (state.scanStatus is ScanStatus.Scanning) CircularProgressIndicator(mod
        else Icon(Icons.Default.DocumentScanner, null)
    },
    modifier = Modifier.clickable(enabled = state.scanStatus !is ScanStatus.Scanni
        val permission = if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU
        if (ContextCompat.checkSelfPermission(context, permission) == PackageManag
            settingsViewModel.runLegacyFileScan()
        } else {
            permissionLauncher.launch(permission)
        }
    },
    colors = ListItemDefaults.colors(containerColor = Color.Transparent)
)

PreferenceGroupTitle(stringResource(R.string.settings_label_download_location))
// ... (Location, Image Quality, Audio Quality, etc. - Keep as is) ...
ListItem(
    headlineContent = {
        Text(if (state.downloadLocation.isEmpty()) stringResource(R.string.setting
    },
    supportingContent = { Text(stringResource(R.string.settings_desc_download_loca
    leadingContent = { Icon(Icons.Default.FolderOpen, null) },
    modifier = Modifier.clickable { try { folderPickerLauncher.launch(null) } catc
    trailingContent = {
        if (state.downloadLocation.isNotEmpty()) {
            IconButton(onClick = { settingsViewModel.clearDownloadLocation() }) {
        }
    },
```

```
        colors = ListItemDefaults.colors(containerColor = Color.Transparent)
    )

    PreferenceGroupTitle(stringResource(R.string.settings_label_image_quality))
    PreferenceRadioGroup {
        ImageQualityOptions.entries.forEach { quality ->
            PreferenceRadioButton(
                text = quality.displayName,
                selected = state.currentImageQuality == quality.key,
                onClick = {
                    settingsViewModel.setImageQualityPreference(quality.key)
                    if (quality.key != AppPreferenceConstants.IMAGE_QUALITY_AUTO) show
                }
            )
        }
    }
    PreferenceDescription(stringResource(R.string.settings_desc_image_quality))

    PreferenceGroupTitle(stringResource(R.string.settings_label_audio_quality))
    PreferenceRadioGroup {
        AudioQualityOptions.entries.forEach { quality ->
            PreferenceRadioButton(
                text = quality.displayName,
                selected = state.currentAudioQuality == quality.key,
                onClick = { settingsViewModel.setAudioQualityPreference(quality.key) }
            )
        }
    }
    PreferenceDescription(stringResource(R.string.settings_desc_audio_quality))

    PreferenceGroupTitle(stringResource(R.string.settings_label_list_loading_config))
    PreferenceRadioGroup {
        ListLoadingConfigOptions.entries.forEach { config ->
            PreferenceRadioButton(
                text = config.displayName,
                selected = state.currentListLoadingConfig == config.key,
                onClick = {
                    settingsViewModel.setListLoadingConfigPreference(config.key)
                    showRestartMessageForDataSettings = true
                }
            )
        }
    }
    PreferenceDescription(stringResource(R.string.settings_desc_list_loading_config))

    PreferenceGroupTitle(stringResource(R.string.settings_label_buffering_strategy))
    PreferenceRadioGroup {
        BufferingStrategyOptions.entries.forEach { strategy ->
            PreferenceRadioButton(
                text = strategy.displayName,
                selected = state.currentBufferingStrategy == strategy.key,
                onClick = {
                    settingsViewModel.setBufferingStrategyPreference(strategy.key)
                    showRestartMessageForDataSettings = true
                }
```

```
                    )
                }
            }
            PreferenceDescription(stringResource(R.string.settings_desc_buffering_strategy))

            Spacer(modifier = Modifier.height(16.dp))
            HorizontalDivider()

            SettingsSectionTitle(stringResource(R.string.settings_section_cache))
            Row(verticalAlignment = Alignment.CenterVertically, modifier = Modifier.fillMaxWid
                Button(
                    onClick = { isClearingCache = true; settingsViewModel.clearAllApplicationC
                    enabled = !isClearingCache,
                    colors = ButtonDefaults.buttonColors(containerColor = MaterialTheme.colorS
                    modifier = Modifier.weight(1f)
                ) {
                    Text(stringResource(R.string.settings_button_clear_cache))
                }
                if (isClearingCache) CircularProgressIndicator(modifier = Modifier.size(24.dp)
            }
            Text(stringResource(R.string.settings_desc_clear_cache), style = MaterialTheme.typ
            HorizontalDivider()

            SettingsSectionTitle(stringResource(R.string.settings_section_theme))
            PreferenceRadioGroup {
                ThemePreferenceOptions.entries.forEach { themeOpt ->
                    PreferenceRadioButton(
                        text = themeOpt.displayName,
                        selected = state.currentTheme == themeOpt.key,
                        onClick = { settingsViewModel.setThemePreference(themeOpt.key) }
                    )
                }
            }
            Spacer(modifier = Modifier.height(16.dp))
            HorizontalDivider()

            SettingsSectionTitle(stringResource(R.string.settings_section_about))
            InfoRow(label = stringResource(R.string.settings_label_version), value = BuildConf
            Spacer(Modifier.height(8.dp))
            Text(stringResource(R.string.settings_label_powered_by), style = MaterialTheme.typ
            LinkItem(text = stringResource(R.string.settings_link_holodex), url = "https://hol
            LinkItem(text = stringResource(R.string.settings_link_newpipe), url = "https://new
            Spacer(modifier = Modifier.height(24.dp))
        }
    }
}

// ... (Keep all helper Composables and Enums at the bottom of the file) ...
@Composable
private fun SettingsSectionTitle(title: String) {
    Text(text = title, style = MaterialTheme.typography.titleLarge, modifier = Modifier.paddin
}
@Composable
private fun PreferenceGroupTitle(title: String) {
    Text(text = title, style = MaterialTheme.typography.titleMedium, modifier = Modifier.paddi
```

```
}
@Composable
private fun PreferenceDescription(text: String) {
    Text(text = text, style = MaterialTheme.typography.bodySmall, color = MaterialTheme.colorS
}
@Composable
private fun PreferenceRadioGroup(content: @Composable ColumnScope.() -> Unit) {
    Column(Modifier.selectableGroup()) { content() }
}
@Composable
private fun PreferenceRadioButton(text: String, selected: Boolean, onClick: () -> Unit, enable
    ListItem(
        headlineContent = { Text(text, style = MaterialTheme.typography.bodyLarge) },
        leadingContent = { RadioButton(selected = selected, onClick = null, enabled = enabled)
        modifier = Modifier.fillMaxWidth().selectable(selected = selected, onClick = if (enabl
        colors = ListItemDefaults.colors(containerColor = Color.Transparent)
    )
}
@Composable
private fun InfoRow(label: String, value: String) {
    Row(modifier = Modifier.fillMaxWidth().padding(vertical = 4.dp), verticalAlignment = Align
        Text(text = label, style = MaterialTheme.typography.bodyLarge, fontWeight = FontWeight
        Text(text = value, style = MaterialTheme.typography.bodyLarge, color = MaterialTheme.c
    }
}
@OptIn(ExperimentalMaterial3Api::class)
@Composable
private fun LinkItem(text: String, url: String) {
    val uriHandler = LocalUriHandler.current
    val context = LocalContext.current
    ListItem(
        headlineContent = { Text(text, style = MaterialTheme.typography.bodyLarge) },
        trailingContent = { Icon(Icons.Filled.Link, stringResource(R.string.content_desc_exter
        modifier = Modifier.clickable { try { uriHandler.openUri(url) } catch (e: Exception) {
        colors = ListItemDefaults.colors(containerColor = Color.Transparent)
    )
}

enum class ImageQualityOptions(val key: String, val displayName: String) {
    AUTO(AppPreferenceConstants.IMAGE_QUALITY_AUTO, "Auto (Recommended)"),
    MEDIUM(AppPreferenceConstants.IMAGE_QUALITY_MEDIUM, "Medium (Faster loading)"),
    LOW(AppPreferenceConstants.IMAGE_QUALITY_LOW, "Low (Data saver)")
}
enum class AudioQualityOptions(val key: String, val displayName: String) {
    BEST(AppPreferenceConstants.AUDIO_QUALITY_BEST, "Best Available"),
    STANDARD(AppPreferenceConstants.AUDIO_QUALITY_STANDARD, "Standard (~128kbps)"),
    SAVER(AppPreferenceConstants.AUDIO_QUALITY_SAVER, "Data Saver (~64kbps)")
}
enum class ListLoadingConfigOptions(val key: String, val displayName: String) {
    NORMAL(AppPreferenceConstants.LIST_LOADING_NORMAL, "Normal (Smooth scrolling)"),
    REDUCED(AppPreferenceConstants.LIST_LOADING_REDUCED, "Reduced (Less data, faster initial)"
    MINIMAL(AppPreferenceConstants.LIST_LOADING_MINIMAL, "Minimal (Data saver, slowest scroll)
}
enum class BufferingStrategyOptions(val key: String, val displayName: String) {
    AGGRESSIVE(AppPreferenceConstants.BUFFERING_STRATEGY_AGGRESSIVE, "Quick Start (Default)"),
```

```
    BALANCED(AppPreferenceConstants.BUFFERING_STRATEGY_BALANCED, "Balanced"),
    STABLE(AppPreferenceConstants.BUFFERING_STRATEGY_STABLE, "Stable Playback (More buffering)
}
enum class ThemePreferenceOptions(val key: String, val displayName: String) {
    LIGHT(ThemePreference.LIGHT, "Light"),
    DARK(ThemePreference.DARK, "Dark"),
    SYSTEM(ThemePreference.SYSTEM, "Follow System")
}

// File: java\com\example\holodex\ui\screens\VideoDetailsScreen.kt
// File: java/com/example/holodex/ui/screens/VideoDetailsScreen.kt
// File: java/com/example/holodex/ui/screens/VideoDetailsScreen.kt
@file:OptIn(ExperimentalMaterial3Api::class, ExperimentalFoundationApi::class)

package com.example.holodex.ui.screens

import android.widget.Toast
import androidx.compose.foundation.BorderStroke
import androidx.compose.foundation.ExperimentalFoundationApi
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.aspectRatio
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.itemsIndexed
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.automirrored.filled.ArrowBack
import androidx.compose.material.icons.automirrored.filled.QueueMusic
import androidx.compose.material.icons.filled.Download
import androidx.compose.material.icons.filled.Favorite
import androidx.compose.material.icons.filled.FavoriteBorder
import androidx.compose.material.icons.filled.MusicNote
import androidx.compose.material.icons.filled.PlayArrow
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.HorizontalDivider
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.IconButtonDefaults
import androidx.compose.material3.LocalContentColor
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.OutlinedButton
import androidx.compose.material3.Scaffold
import androidx.compose.material3.SnackbarDuration
import androidx.compose.material3.SnackbarHost
import androidx.compose.material3.SnackbarHostState
```

```
import androidx.compose.material3.Text
import androidx.compose.material3.TopAppBar
import androidx.compose.material3.TopAppBarDefaults
import androidx.compose.runtime.Composable
import androidx.compose.runtime.CompositionLocalProvider
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.derivedStateOf
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableIntStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.res.pluralStringResource
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextOverflow
import androidx.compose.ui.unit.dp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.lifecycle.compose.collectAsStateWithLifecycle
import androidx.media3.common.util.UnstableApi
import androidx.navigation.NavController
import coil.compose.AsyncImage
import coil.request.ImageRequest
import com.example.holodex.R
import com.example.holodex.data.model.HolodexVideoItem
import com.example.holodex.ui.composables.ErrorStateWithRetry
import com.example.holodex.ui.composables.LoadingState
import com.example.holodex.ui.composables.SimpleProcessedBackground
import com.example.holodex.ui.composables.UnifiedListItem
import com.example.holodex.util.DynamicTheme
import com.example.holodex.util.ThumbnailQuality
import com.example.holodex.util.findActivity
import com.example.holodex.util.getYouTubeThumbnailUrl
import com.example.holodex.viewmodel.FavoritesViewModel
import com.example.holodex.viewmodel.PlaylistManagementViewModel
import com.example.holodex.viewmodel.VideoDetailsViewModel
import com.example.holodex.viewmodel.VideoListViewModel
import org.orbitmvi.orbit.compose.collectAsState

@UnstableApi
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun VideoDetailsScreen(
    navController: NavController,
    onNavigateUp: () -> Unit,
) {
    val videoDetailsViewModel: VideoDetailsViewModel = hiltViewModel()
    // *** THE FIX: Get the activity-scoped ViewModel here ***
    val videoListViewModel: VideoListViewModel = hiltViewModel(findActivity())
    val favoritesViewModel: FavoritesViewModel = hiltViewModel()
```

```
val playlistManagementViewModel: PlaylistManagementViewModel = hiltViewModel(findActivity(

// Call the initialize function once when the screen is first composed.
LaunchedEffect(Unit) {
    videoDetailsViewModel.initialize(videoListViewModel)
}

val videoDetails by videoDetailsViewModel.videoDetails.collectAsStateWithLifecycle()
val favoritesState by favoritesViewModel.collectAsState()
val isLoading by videoDetailsViewModel.isLoading.collectAsStateWithLifecycle()
val error by videoDetailsViewModel.error.collectAsStateWithLifecycle()
val transientMessage by videoDetailsViewModel.transientMessage.collectAsStateWithLifecycle
val snackbarHostState = remember { SnackbarHostState() }
val context = LocalContext.current
val dynamicTheme by videoDetailsViewModel.dynamicTheme.collectAsStateWithLifecycle()

val backgroundImageUrl by remember(videoDetails) {
    derivedStateOf { videoDetails?.id?.let { getYouTubeThumbnailUrl(it, ThumbnailQuality.M
}

LaunchedEffect(error) {
    error?.let {
        snackbarHostState.showSnackbar(message = it, duration = SnackbarDuration.Long)
        videoDetailsViewModel.clearError()
    }
}
LaunchedEffect(transientMessage) {
    transientMessage?.let {
        Toast.makeText(context, it, Toast.LENGTH_SHORT).show()
        videoDetailsViewModel.clearTransientMessage()
    }
}

Scaffold(
    snackbarHost = { SnackbarHost(snackbarHostState) },
    topBar = {
        TopAppBar(
            title = { Text(text = videoDetails?.title ?: "", maxLines = 1, overflow = Text
            navigationIcon = {
                IconButton(onClick = onNavigateUp) {
                    Icon(Icons.AutoMirrored.Filled.ArrowBack, contentDescription = stringR
                }
            },
            actions = {
                videoDetails?.let { video ->
                    // *** FIX 1: Check Unified Favorites Map ***
                    // The channel ID is the key in the map.
                    val channelId = video.channel.id ?: ""
                    val isFavorited = favoritesState.likedItemsMap.containsKey(channelId)

                    IconButton(onClick = {
                        // *** FIX 2: Call the overload that accepts HolodexVideoItem ***
                        // (We already added this overload in FavoritesViewModel previousl
                        favoritesViewModel.toggleFavoriteChannel(video)
                    }) {
```

```
                            Icon(
                                imageVector = if (isFavorited) Icons.Filled.Favorite else Icon
                                contentDescription = "Favorite Channel",
                                tint = if (isFavorited) dynamicTheme.primary else dynamicTheme
                            )
                        }
                    }
                },
                colors = TopAppBarDefaults.topAppBarColors(
                    containerColor = Color.Transparent,
                    titleContentColor = dynamicTheme.onPrimary,
                    navigationIconContentColor = dynamicTheme.onPrimary,
                    actionIconContentColor = dynamicTheme.onPrimary
                )
            )
        }
    ) { paddingValues ->
        Box(modifier = Modifier.fillMaxSize()) {
            SimpleProcessedBackground(artworkUri = backgroundImageUrl, dynamicColor = dynamicT
            CompositionLocalProvider(LocalContentColor provides dynamicTheme.onPrimary) {
                Box(modifier = Modifier.padding(paddingValues).fillMaxSize()) {
                    when {
                        isLoading && videoDetails == null -> LoadingState(message = stringReso
                        error != null && videoDetails == null -> ErrorStateWithRetry(
                            message = error!!,
                            onRetry = { videoDetailsViewModel.initialize(videoListViewModel) }
                        )
                        videoDetails != null -> {
                            VideoDetailsContent(
                                videoDetailsViewModel = videoDetailsViewModel,
                                navController = navController,
                                playlistManagementViewModel = playlistManagementViewModel,
                                dynamicTheme = dynamicTheme
                            )
                        }
                    }
                }
            }
        }
    }
}

@UnstableApi
@Composable
private fun VideoDetailsContent(
    navController: NavController,
    videoDetailsViewModel: VideoDetailsViewModel,
    playlistManagementViewModel: PlaylistManagementViewModel,
    dynamicTheme: DynamicTheme
) {
    val videoItem by videoDetailsViewModel.videoDetails.collectAsStateWithLifecycle()
    val songItems by videoDetailsViewModel.unifiedSongItems.collectAsStateWithLifecycle()

    val videoListViewModel: VideoListViewModel = hiltViewModel(findActivity())
    val favoritesViewModel: FavoritesViewModel = hiltViewModel()
```

```
    LazyColumn(modifier = Modifier.fillMaxSize(), contentPadding = PaddingValues(bottom = 80.d
        item {
            videoItem?.let {
                VideoDetailsHeader(
                    videoItem = it,
                    songCount = songItems.size,
                    onPlayAll = { videoDetailsViewModel.playAllSegments() },
                    onAddToQueue = { videoDetailsViewModel.addAllSegmentsToQueue() },
                    onDownloadAll = { videoDetailsViewModel.downloadAllSegments() },
                    dynamicTheme = dynamicTheme
                )
            }
            HorizontalDivider(color = LocalContentColor.current.copy(alpha = 0.2f))
        }

        if (songItems.isNotEmpty()) {
            itemsIndexed(
                songItems,
                key = { _, song -> song.stableId }
            ) { index, songItem ->
                UnifiedListItem(
                    item = songItem,
                    onItemClicked = { videoDetailsViewModel.playSegment(index) },
                    navController = navController,
                    videoListViewModel = videoListViewModel,
                    favoritesViewModel = favoritesViewModel,
                    playlistManagementViewModel = playlistManagementViewModel
                )
            }
        } else {
            // Display empty state only if the parent video has finished loading
            if (videoItem != null && !videoDetailsViewModel.isLoading.value) {
                item { EmptyStateMessage() }
            }
        }
    }
}

@Composable
private fun VideoDetailsHeader(
    videoItem: HolodexVideoItem,
    songCount: Int,
    onPlayAll: () -> Unit,
    onAddToQueue: () -> Unit,
    onDownloadAll: () -> Unit,
    dynamicTheme: DynamicTheme
) {
    val thumbnailUrls = remember(videoItem.id) {
        getYouTubeThumbnailUrl(videoItem.id, ThumbnailQuality.MAX)
    }
    var currentUrlIndex by remember(thumbnailUrls) { mutableIntStateOf(0) }

    Column(modifier = Modifier.padding(16.dp)) {
        AsyncImage(
```

```
            model = ImageRequest.Builder(LocalContext.current)
                .data(thumbnailUrls.getOrNull(currentUrlIndex))
                .placeholder(R.drawable.ic_placeholder_image)
                .error(R.drawable.ic_error_image)
                .crossfade(true).build(),
            onError = { if (currentUrlIndex < thumbnailUrls.lastIndex) currentUrlIndex++ },
            contentDescription = stringResource(R.string.video_thumbnail_description),
            contentScale = ContentScale.Crop,
            modifier = Modifier
                .fillMaxWidth()
                .aspectRatio(16f / 9f)
                .clip(MaterialTheme.shapes.medium)
        )
        Spacer(Modifier.height(16.dp))
        Text(videoItem.title, style = MaterialTheme.typography.headlineSmall, fontWeight = Fon
        Spacer(Modifier.height(4.dp))
        Text(videoItem.channel.name, style = MaterialTheme.typography.titleMedium)
        if (songCount > 0) {
            Spacer(Modifier.height(4.dp))
            Text(
                text = pluralStringResource(R.plurals.song_count, songCount, songCount),
                style = MaterialTheme.typography.bodyMedium
            )
        }
        Spacer(Modifier.height(16.dp))
        if (songCount > 0) {
            ActionButtons(
                onPlayAll = onPlayAll,
                onAddToQueue = onAddToQueue,
                onDownloadAll = onDownloadAll,
                dynamicTheme = dynamicTheme
            )
        }
        Spacer(Modifier.height(16.dp))
    }
}

@Composable
private fun ActionButtons(
    onPlayAll: () -> Unit,
    onAddToQueue: () -> Unit,
    onDownloadAll: () -> Unit,
    dynamicTheme: DynamicTheme
) {
    Row(modifier = Modifier.fillMaxWidth(), horizontalArrangement = Arrangement.spacedBy(8.dp)
        Button(
            onClick = onPlayAll,
            modifier = Modifier.weight(1f),
            colors = ButtonDefaults.buttonColors(
                containerColor = dynamicTheme.onPrimary,
                contentColor = dynamicTheme.primary
            )
        ) {
            Icon(Icons.Filled.PlayArrow, null, modifier = Modifier.size(ButtonDefaults.IconSiz
            Spacer(Modifier.size(ButtonDefaults.IconSpacing))
```

```
            Text(stringResource(R.string.action_play_all))
        }
        OutlinedButton(
            onClick = onAddToQueue,
            modifier = Modifier.weight(1f),
            colors = ButtonDefaults.outlinedButtonColors(contentColor = dynamicTheme.onPrimary
            border = BorderStroke(1.dp, dynamicTheme.onPrimary.copy(alpha = 0.5f))
        ) {
            Icon(Icons.AutoMirrored.Filled.QueueMusic, null, modifier = Modifier.size(ButtonDe
            Spacer(Modifier.size(ButtonDefaults.IconSpacing))
            Text(stringResource(R.string.action_add_to_queue_short))
        }
        IconButton(
            onClick = onDownloadAll,
            colors = IconButtonDefaults.iconButtonColors(contentColor = dynamicTheme.onPrimary
        ) {
            Icon(Icons.Filled.Download, stringResource(R.string.action_download_all))
        }
    }
}


@Composable
private fun EmptyStateMessage() {
    Box(
        modifier = Modifier
            .fillMaxWidth()
            .padding(32.dp), contentAlignment = Alignment.Center
    ) {
        Column(
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.spacedBy(8.dp)
        ) {
            Icon(
                Icons.Filled.MusicNote,
                contentDescription = null,
                tint = MaterialTheme.colorScheme.onSurfaceVariant,
                modifier = Modifier.size(48.dp)
            )
            Text(
                text = stringResource(R.string.no_song_segments_available),
                style = MaterialTheme.typography.bodyLarge,
                color = MaterialTheme.colorScheme.onSurfaceVariant
            )
        }
    }
}

// File: java\com\example\holodex\ui\screens\navigation\AppDestinations.kt
// File: java/com/example/holodex/ui/navigation/AppDestinations.kt
package com.example.holodex.ui.navigation

import com.example.holodex.viewmodel.FullListViewModel
import com.example.holodex.viewmodel.MusicCategoryType // FIX: Correct import
import com.example.holodex.viewmodel.PlaylistDetailsViewModel
```

```
import com.example.holodex.viewmodel.VideoDetailsViewModel
import java.net.URLEncoder
import java.nio.charset.StandardCharsets

object AppDestinations {
    const val HOME_ROUTE = "home"
    const val DISCOVERY_ROUTE = "discover"
    const val LIBRARY_ROUTE = "library"
    const val DOWNLOADS_ROUTE = "downloads"
    const val SETTINGS_ROUTE = "settings"
    const val LOGIN_ROUTE = "login"
    const val FOR_YOU_ROUTE = "for_you"

    const val VIDEO_DETAILS_ROUTE_TEMPLATE = "video_details/{${VideoDetailsViewModel.VIDEO_ID_
    fun videoDetailRoute(videoId: String) = "video_details/$videoId"

    const val FULL_LIST_VIEW_ROUTE_TEMPLATE =
        "full_list/{${FullListViewModel.CATEGORY_TYPE_ARG}}/{${FullListViewModel.ORG_ARG}}"

    // FIX: Use MusicCategoryType directly
    fun fullListViewRoute(category: MusicCategoryType, org: String): String {
        val encodedOrg = URLEncoder.encode(org, StandardCharsets.UTF_8.toString())
        return "full_list/${category.name}/$encodedOrg"
    }

    const val PLAYLIST_DETAILS_ROUTE_TEMPLATE =
        "playlist_details/{${PlaylistDetailsViewModel.PLAYLIST_ID_ARG}}"

    fun playlistDetailsRoute(playlistId: String): String {
        val encodedId = URLEncoder.encode(playlistId, StandardCharsets.UTF_8.toString())
        return "playlist_details/$encodedId"
    }
}

// File: java\com\example\holodex\ui\screens\navigation\BottomNavItem.kt
// --- FULL REPLACEMENT of the file content ---
package com.example.holodex.ui.screens.navigation

import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Download
import androidx.compose.material.icons.filled.Explore
import androidx.compose.material.icons.filled.LibraryMusic
import androidx.compose.material.icons.filled.Search
import androidx.compose.ui.graphics.vector.ImageVector
import com.example.holodex.R
import com.example.holodex.ui.navigation.AppDestinations

sealed class BottomNavItem(
    val route: String,
    val titleResId: Int,
    val icon: ImageVector
) {
    object Discover : BottomNavItem(
        route = AppDestinations.DISCOVERY_ROUTE,
        titleResId = R.string.bottom_nav_discover, // Add string
```

```
        icon = Icons.Filled.Explore
    )

    object Browse : BottomNavItem(
        route = AppDestinations.HOME_ROUTE,
        titleResId = R.string.bottom_nav_browse,
        icon = Icons.Filled.Search
    )

    object Library : BottomNavItem(
        route = AppDestinations.LIBRARY_ROUTE,
        titleResId = R.string.bottom_nav_library,
        icon = Icons.Filled.LibraryMusic
    )

    object Downloads : BottomNavItem(
        route = AppDestinations.DOWNLOADS_ROUTE,
        titleResId = R.string.bottom_nav_downloads,
        icon = Icons.Filled.Download
    )
}

// File: java\com\example\holodex\ui\screens\navigation\HolodexNavHost.kt
package com.example.holodex.ui.navigation

import android.annotation.SuppressLint
import androidx.activity.ComponentActivity
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.Button
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.navigation.NavHostController
import androidx.navigation.NavType
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
import androidx.navigation.navArgument
import com.example.holodex.auth.LoginScreen
import com.example.holodex.ui.screens.ChannelDetailsScreen
import com.example.holodex.ui.screens.DiscoveryScreen
import com.example.holodex.ui.screens.DownloadsScreen
import com.example.holodex.ui.screens.ForYouScreen
import com.example.holodex.ui.screens.FullListViewScreen
import com.example.holodex.ui.screens.HomeScreen
import com.example.holodex.ui.screens.LibraryScreen
import com.example.holodex.ui.screens.PlaylistDetailsScreen
import com.example.holodex.ui.screens.SettingsScreen
```

```
import com.example.holodex.ui.screens.VideoDetailsScreen
import com.example.holodex.viewmodel.ChannelDetailsViewModel
import com.example.holodex.viewmodel.FullListViewModel
import com.example.holodex.viewmodel.MusicCategoryType
import com.example.holodex.viewmodel.PlaylistDetailsViewModel
import com.example.holodex.viewmodel.PlaylistManagementViewModel
import com.example.holodex.viewmodel.SettingsViewModel
import com.example.holodex.viewmodel.VideoDetailsViewModel
import com.example.holodex.viewmodel.VideoListViewModel
import org.orbitmvi.orbit.compose.collectAsState

@SuppressLint("UnstableApi")
@Composable
fun HolodexNavHost(
    navController: NavHostController,
    videoListViewModel: VideoListViewModel,
    playlistManagementViewModel: PlaylistManagementViewModel,
    activity: ComponentActivity,
    contentPadding: PaddingValues, // NEW PARAMETER
    modifier: Modifier = Modifier
) {
    NavHost(
        navController = navController,
        startDestination = AppDestinations.LIBRARY_ROUTE,
        modifier = modifier.fillMaxSize()
    ) {
        // --- Discovery Tab (Uses contentPadding) ---
        composable(AppDestinations.DISCOVERY_ROUTE) {
            DiscoveryScreen(
                navController = navController,
                contentPadding = contentPadding
            )
        }

        composable(AppDestinations.FOR_YOU_ROUTE) {
            ForYouScreen(navController = navController)
        }

        // --- Home / Browse Tab (Uses contentPadding) ---
        composable(AppDestinations.HOME_ROUTE) {
            val settingsViewModel: SettingsViewModel = hiltViewModel()

            val state by settingsViewModel.collectAsState()
            val currentApiKey = state.currentApiKey

            if (currentApiKey.isBlank()) {
                ApiKeyMissingContent(navController = navController)
            } else {
                HomeScreen(
                    navController = navController,
                    videoListViewModel = videoListViewModel,
                    playlistManagementViewModel = playlistManagementViewModel,
                    contentPadding = contentPadding
                )
            }
```

```
}

// --- Library Tab (Uses contentPadding) ---
composable(AppDestinations.LIBRARY_ROUTE) {
    LibraryScreen(
        navController = navController,
        playlistManagementViewModel = playlistManagementViewModel,
        contentPadding = contentPadding
    )
}

// --- Downloads Tab (Uses contentPadding) ---
composable(AppDestinations.DOWNLOADS_ROUTE) {
    DownloadsScreen(
        navController = navController,
        playlistManagementViewModel = playlistManagementViewModel,
        contentPadding = contentPadding
    )
}

composable(AppDestinations.SETTINGS_ROUTE) {
    val vListVm: VideoListViewModel = hiltViewModel(activity)
    SettingsScreen(
        navController = navController,
        onNavigateUp = { navController.popBackStack() },
        onApiKeySavedRestartNeeded = { vListVm.refreshCurrentListViaPull() }
    )
}

composable(AppDestinations.LOGIN_ROUTE) {
    LoginScreen(onLoginSuccess = { navController.popBackStack() })
}

// --- Detail Screens (Usually draw over bottom bar, so might ignore contentPadding or

composable(
    route = "channel_details/{${ChannelDetailsViewModel.CHANNEL_ID_ARG}}",
    arguments = listOf(navArgument(ChannelDetailsViewModel.CHANNEL_ID_ARG) { type = Na
) {
    ChannelDetailsScreen(navController = navController, onNavigateUp = { navController
}

composable(
    route = AppDestinations.FULL_LIST_VIEW_ROUTE_TEMPLATE,
    arguments = listOf(
        navArgument(FullListViewModel.CATEGORY_TYPE_ARG) { type = NavType.StringType }
        navArgument(FullListViewModel.ORG_ARG) { type = NavType.StringType }
    )
) { backStackEntry ->
    val categoryName = backStackEntry.arguments?.getString(FullListViewModel.CATEGORY_
    val category = try {
        MusicCategoryType.valueOf(categoryName)
    } catch (e: IllegalArgumentException) {
        MusicCategoryType.TRENDING
    }
```

```
            FullListViewScreen(navController = navController, categoryType = category)
        }

        composable(
            AppDestinations.PLAYLIST_DETAILS_ROUTE_TEMPLATE,
            arguments = listOf(navArgument(PlaylistDetailsViewModel.PLAYLIST_ID_ARG) { type =
        ) {
            PlaylistDetailsScreen(
                navController = navController,
                playlistManagementViewModel = playlistManagementViewModel,
                onNavigateUp = { navController.popBackStack() },
                contentPadding = contentPadding
            )
        }

        composable(
            AppDestinations.VIDEO_DETAILS_ROUTE_TEMPLATE,
            arguments = listOf(navArgument(VideoDetailsViewModel.VIDEO_ID_ARG) { type = NavTyp
        ) {
            VideoDetailsScreen(navController = navController, onNavigateUp = { navController.p
        }
    }
}

@Composable
private fun ApiKeyMissingContent(navController: NavHostController) {
    Box(Modifier.fillMaxSize(), contentAlignment = Alignment.Center) {
        androidx.compose.foundation.layout.Column(horizontalAlignment = Alignment.CenterHorizo
            Text("API Key Required", style = MaterialTheme.typography.headlineSmall)
            Button(
                onClick = { navController.navigate(AppDestinations.SETTINGS_ROUTE) },
                modifier = Modifier.padding(top = 16.dp)
            ) {
                Text("Go to Settings")
            }
        }
    }
}

// File: java\com\example\holodex\ui\theme\Color.kt
package com.example.holodex.ui.theme

import androidx.compose.ui.graphics.Color

val md_theme_light_primary = Color(0xFF6750A4)
val md_theme_light_onPrimary = Color(0xFFFFFFFF)
val md_theme_light_primaryContainer = Color(0xFFEADDFF)
val md_theme_light_onPrimaryContainer = Color(0xFF21005D)
val md_theme_light_secondary = Color(0xFF625B71)
val md_theme_light_onSecondary = Color(0xFFFFFFFF)
val md_theme_light_secondaryContainer = Color(0xFFE8DEF8)
val md_theme_light_onSecondaryContainer = Color(0xFF1D192B)
val md_theme_light_tertiary = Color(0xFF7D5260)
val md_theme_light_onTertiary = Color(0xFFFFFFFF)
val md_theme_light_tertiaryContainer = Color(0xFFFFD8E4)
```

```
val md_theme_light_onTertiaryContainer = Color(0xFF31111D)
val md_theme_light_error = Color(0xFFB3261E)
val md_theme_light_onError = Color(0xFFFFFFFF)
val md_theme_light_errorContainer = Color(0xFFF9DEDC)
val md_theme_light_onErrorContainer = Color(0xFF410E0B)
val md_theme_light_background = Color(0xFFFFFBFE)
val md_theme_light_onBackground = Color(0xFF1C1B1F)
val md_theme_light_surface = Color(0xFFFFFBFE)
val md_theme_light_onSurface = Color(0xFF1C1B1F)
val md_theme_light_surfaceVariant = Color(0xFFE7E0EC)
val md_theme_light_onSurfaceVariant = Color(0xFF49454F)
val md_theme_light_outline = Color(0xFF79747E)
val md_theme_light_inverseOnSurface = Color(0xFFF4EFF4)
val md_theme_light_inverseSurface = Color(0xFF313033)
val md_theme_light_inversePrimary = Color(0xFFD0BCFF)
val md_theme_light_surfaceTint = Color(0xFF6750A4)
val md_theme_light_outlineVariant = Color(0xFFCAC4D0)
val md_theme_light_scrim = Color(0xFF000000)

val md_theme_dark_primary = Color(0xFFD0BCFF)
val md_theme_dark_onPrimary = Color(0xFF381E72)
val md_theme_dark_primaryContainer = Color(0xFF4F378B)
val md_theme_dark_onPrimaryContainer = Color(0xFFEADDFF)
val md_theme_dark_secondary = Color(0xFFCCC2DC)
val md_theme_dark_onSecondary = Color(0xFF332D41)
val md_theme_dark_secondaryContainer = Color(0xFF4A4458)
val md_theme_dark_onSecondaryContainer = Color(0xFFE8DEF8)
val md_theme_dark_tertiary = Color(0xFFEFB8C8)
val md_theme_dark_onTertiary = Color(0xFF492532)
val md_theme_dark_tertiaryContainer = Color(0xFF633B48)
val md_theme_dark_onTertiaryContainer = Color(0xFFFFD8E4)
val md_theme_dark_error = Color(0xFFF2B8B5)
val md_theme_dark_onError = Color(0xFF601410)
val md_theme_dark_errorContainer = Color(0xFF8C1D18)
val md_theme_dark_onErrorContainer = Color(0xFFF9DEDC)
val md_theme_dark_background = Color(0xFF1C1B1F)
val md_theme_dark_onBackground = Color(0xFFE6E1E5)
val md_theme_dark_surface = Color(0xFF1C1B1F)
val md_theme_dark_onSurface = Color(0xFFE6E1E5)
val md_theme_dark_surfaceVariant = Color(0xFF49454F)
val md_theme_dark_onSurfaceVariant = Color(0xFFCAC4D0)
val md_theme_dark_outline = Color(0xFF938F99)
val md_theme_dark_inverseOnSurface = Color(0xFF1C1B1F)
val md_theme_dark_inverseSurface = Color(0xFFE6E1E5)
val md_theme_dark_inversePrimary = Color(0xFF6750A4)
val md_theme_dark_surfaceTint = Color(0xFFD0BCFF)
val md_theme_dark_outlineVariant = Color(0xFF49454F)
val md_theme_dark_scrim = Color(0xFF000000)

// File: java\com\example\holodex\ui\theme\Shape.kt
package com.example.holodex.ui.theme

import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material3.Shapes
import androidx.compose.ui.unit.dp
```

```
val Shapes = Shapes(
    small = RoundedCornerShape(4.dp),
    medium = RoundedCornerShape(8.dp),
    large = RoundedCornerShape(12.dp)
)

// File: java\com\example\holodex\ui\theme\Theme.kt
package com.example.holodex.ui.theme

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.runtime.SideEffect
import androidx.compose.runtime.getValue
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.platform.LocalContext
import androidx.hilt.navigation.compose.hiltViewModel
import com.example.holodex.data.ThemePreference // <--- Import this
import com.example.holodex.viewmodel.SettingsViewModel
import com.google.accompanist.systemuicontroller.rememberSystemUiController
import org.orbitmvi.orbit.compose.collectAsState // <--- Import this

// ... (Keep AppDarkColorScheme and AppLightColorScheme definitions as they are) ...
private val AppDarkColorScheme = darkColorScheme(
    primary = md_theme_dark_primary,
    onPrimary = md_theme_dark_onPrimary,
    primaryContainer = md_theme_dark_primaryContainer,
    onPrimaryContainer = md_theme_dark_onPrimaryContainer,
    secondary = md_theme_dark_secondary,
    onSecondary = md_theme_dark_onSecondary,
    secondaryContainer = md_theme_dark_secondaryContainer,
    onSecondaryContainer = md_theme_dark_onSecondaryContainer,
    tertiary = md_theme_dark_tertiary,
    onTertiary = md_theme_dark_onTertiary,
    tertiaryContainer = md_theme_dark_tertiaryContainer,
    onTertiaryContainer = md_theme_dark_onTertiaryContainer,
    error = md_theme_dark_error,
    onError = md_theme_dark_onError,
    errorContainer = md_theme_dark_errorContainer,
    onErrorContainer = md_theme_dark_onErrorContainer,
    background = md_theme_dark_background,
    onBackground = md_theme_dark_onBackground,
    surface = md_theme_dark_surface,
    onSurface = md_theme_dark_onSurface,
    surfaceVariant = md_theme_dark_surfaceVariant,
    onSurfaceVariant = md_theme_dark_onSurfaceVariant,
    outline = md_theme_dark_outline,
    inverseOnSurface = md_theme_dark_inverseOnSurface,
    inverseSurface = md_theme_dark_inverseSurface,
```

```
    inversePrimary = md_theme_dark_inversePrimary,
    surfaceTint = md_theme_dark_surfaceTint,
    outlineVariant = md_theme_dark_outlineVariant,
    scrim = md_theme_dark_scrim,
)

private val AppLightColorScheme = lightColorScheme(
    primary = md_theme_light_primary,
    onPrimary = md_theme_light_onPrimary,
    primaryContainer = md_theme_light_primaryContainer,
    onPrimaryContainer = md_theme_light_onPrimaryContainer,
    secondary = md_theme_light_secondary,
    onSecondary = md_theme_light_onSecondary,
    secondaryContainer = md_theme_light_secondaryContainer,
    onSecondaryContainer = md_theme_light_onSecondaryContainer,
    tertiary = md_theme_light_tertiary,
    onTertiary = md_theme_light_onTertiary,
    tertiaryContainer = md_theme_light_tertiaryContainer,
    onTertiaryContainer = md_theme_light_onTertiaryContainer,
    error = md_theme_light_error,
    onError = md_theme_light_onError,
    errorContainer = md_theme_light_errorContainer,
    onErrorContainer = md_theme_light_onErrorContainer,
    background = md_theme_light_background,
    onBackground = md_theme_light_onBackground,
    surface = md_theme_light_surface,
    onSurface = md_theme_light_onSurface,
    surfaceVariant = md_theme_light_surfaceVariant,
    onSurfaceVariant = md_theme_light_onSurfaceVariant,
    outline = md_theme_light_outline,
    inverseOnSurface = md_theme_light_inverseOnSurface,
    inverseSurface = md_theme_light_inverseSurface,
    inversePrimary = md_theme_light_inversePrimary,
    surfaceTint = md_theme_light_surfaceTint,
    outlineVariant = md_theme_light_outlineVariant,
    scrim = md_theme_light_scrim,
)

@Composable
fun HolodexMusicTheme(
    settingsViewModel: SettingsViewModel = hiltViewModel(),
    dynamicColor: Boolean = true,
    content: @Composable () -> Unit
) {
    // FIX: Collect state from Orbit
    val state by settingsViewModel.collectAsState()

    // FIX: Access property from state (renamed to currentTheme)
    val useDarkTheme = when (state.currentTheme) {
        ThemePreference.LIGHT -> false
        ThemePreference.DARK -> true
        else -> isSystemInDarkTheme()
    }

    val colorScheme = when {
```

```
        dynamicColor && Build.VERSION.SDK_INT >= Build.VERSION_CODES.S -> {
            val context = LocalContext.current
            if (useDarkTheme) dynamicDarkColorScheme(context) else dynamicLightColorScheme(con
        }
        useDarkTheme -> AppDarkColorScheme
        else -> AppLightColorScheme
    }

    val systemUiController = rememberSystemUiController()
    SideEffect {
        systemUiController.setStatusBarColor(
            color = Color.Transparent,
            darkIcons = !useDarkTheme
        )
        systemUiController.setNavigationBarColor(
            color = Color.Transparent,
            darkIcons = !useDarkTheme,
            navigationBarContrastEnforced = false
        )
    }

    MaterialTheme(
        colorScheme = colorScheme,
        typography = Typography,
        shapes = Shapes,
        content = content
    )
}

// File: java\com\example\holodex\ui\theme\Type.kt
package com.example.holodex.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

// Replace with your own font families if desired
val Typography = Typography(
    bodyLarge = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Normal,
        fontSize = 16.sp,
        lineHeight = 24.sp,
        letterSpacing = 0.5.sp
    ),
    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
    )
    // Add other text styles as needed
)

// File: java\com\example\holodex\util\ArtworkResolver.kt
package com.example.holodex.util

import com.example.holodex.data.model.discovery.PlaylistStub
import timber.log.Timber
import java.util.regex.Pattern

/**
 * A utility object to resolve the best possible artwork URL for different data models.
 */
object ArtworkResolver {

    // Regex to extract channel ID from playlist IDs like ":dailyrandom[ch=...]" or ":artist[c
    private val CHANNEL_ID_PATTERN: Pattern = Pattern.compile("ch=([a-zA-Z0-9_-]{24})")

    // --- START OF IMPLEMENTATION ---
    /**
     * Constructs the standard URL for a channel's photo based on its ID.
     * This is used as a fallback when the API does not provide a direct photo URL.
     *
     * @param channelId The unique ID of the channel.
     * @return The fully-formed URL to the channel's 200px profile picture.
     */
    fun getChannelPhotoUrl(channelId: String): String {
        return "https://holodex.net/statics/channelImg/$channelId/200.png"
    }
    // --- END OF IMPLEMENTATION ---

    /**
     * The main function to resolve playlist artwork. It follows the fallback logic
     * discovered from the Musicdex frontend.
     *
     * @param playlist The PlaylistStub object from the API.
     * @return The best available URL string for the playlist's artwork, or null if none can b
     */
    fun getPlaylistArtworkUrl(playlist: PlaylistStub): String? {
        Timber.d("Resolving artwork for playlist: ${playlist.title} (Type: ${playlist.type})")

        // Method 1: Use the pre-defined channel image for specific types
        if (playlist.type.startsWith(":dailyrandom") || playlist.type.startsWith(":artist")) {
            val matcher = CHANNEL_ID_PATTERN.matcher(playlist.id)
            if (matcher.find()) {
                val channelId = matcher.group(1)
                if (!channelId.isNullOrBlank()) {
                    Timber.d("PlaylistArtwork: Using Method 1 (Channel ID from playlist ID)")
                    return "https://holodex.net/statics/channelImg/$channelId/200.png"
```

```
                }
            }
        }

        // Method 2: Use the art_context field
        val artContext = playlist.artContext
        if (artContext != null) {
            // Rule from Musicdex: If it seems channel-focused, use the channel image.
            val isChannelFocused = (artContext.channels?.size ?: 0) < 3 && (artContext.videos?
            if (isChannelFocused && !artContext.channels.isNullOrEmpty()) {
                Timber.d("PlaylistArtwork: Using Method 2 (Channel-focused art_context)")
                return "https://holodex.net/statics/channelImg/${artContext.channels.first()}/
            }
            // Otherwise, assume it's video-focused.
            if (!artContext.videos.isNullOrEmpty()) {
                Timber.d("PlaylistArtwork: Using Method 2 (Video-focused art_context)")
                return getYouTubeThumbnailUrl(artContext.videos.first(), ThumbnailQuality.HIGH
            }
            // Fallback within art_context to channel photo if no videos
            if (!artContext.channelPhotoUrl.isNullOrBlank()) {
                Timber.d("PlaylistArtwork: Using Method 2 (Fallback channel photo from art_con
                return artContext.channelPhotoUrl
            }
        }

        Timber.w("PlaylistArtwork: No suitable URL found for playlist '${playlist.title}' usin
        return null
    }
}

// File: java\com\example\holodex\util\ComposableUtils.kt
package com.example.holodex.util

import android.content.ContextWrapper
import androidx.activity.ComponentActivity
import androidx.compose.runtime.Composable
import androidx.compose.runtime.remember
import androidx.compose.ui.platform.LocalContext

@Composable
fun findActivity(): ComponentActivity {
    val context = LocalContext.current
    return remember(context) {
        var a = context
        while (a is ContextWrapper) {
            if (a is ComponentActivity) {
                return@remember a
            }
            a = a.baseContext
        }
        // This should not happen in a normal app setup
        error("Could not find activity context")
    }
}
```

```
// File: java\com\example\holodex\util\Extract_util.kt
// File: java/com/example/holodex/util/Extract_util.kt
package com.example.holodex.util

import timber.log.Timber

// Made into a top-level function as it was called that way
// Alternatively, make Extract_util an object and call Extract_util.extractVideoIdFromQuery
fun extractVideoIdFromQuery(query: String): String? {
    val trimmedQuery = query.trim()
    val videoIdRegex = Regex("^[a-zA-Z0-9_-]{11}$")
    if (videoIdRegex.matches(trimmedQuery)) {
        Timber.Forest.d("extractVideoIdFromQuery: Matched direct video ID: $trimmedQuery")
        return trimmedQuery
    }

    // Regex for Holodex Music URL (music.holodex.net/video/ID or holodex.net/watch/ID)
    // Allows for optional trailing slashes or query params after ID
    val holodexUrlRegex = Regex("""^https://(music\.)?holodex\.net/(video|watch)/([a-zA-Z0-9_-
    var matcher = holodexUrlRegex.find(trimmedQuery)
    if (matcher != null) {
        val videoId = matcher.groupValues[3] // Group 3 is the ID
        Timber.Forest.d("extractVideoIdFromQuery: Matched Holodex URL, extracted ID: $videoId"
        return videoId
    }

    // Regex for YouTube URLs (youtube.com/watch?v=ID or youtu.be/ID)
    // More comprehensive regex to catch various YouTube URL formats
    val youtubeUrlRegex = Regex("""(?:youtube\.com/(?:[^/]+/.+/|(?:v|e(?:mbed)?)/|.*[?&]v=)|yo
    matcher = youtubeUrlRegex.find(trimmedQuery)
    if (matcher != null) {
        val videoId = matcher.groupValues[1] // Group 1 is the ID
        Timber.Forest.d("extractVideoIdFromQuery: Matched YouTube URL, extracted ID: $videoId"
        return videoId
    }

    Timber.Forest.d("extractVideoIdFromQuery: No video ID extracted from query: $trimmedQuery"
    return null
}


// File: java\com\example\holodex\util\ImageUtils.kt
// File: java/com/example/holodex/util/ImageUtils.kt
package com.example.holodex.util // Or your preferred util package

import com.example.holodex.data.AppPreferenceConstants
import com.example.holodex.playback.domain.model.PlaybackItem
import timber.log.Timber

// Regex to find the size part of an iTunes/mzstatic image URL
private val ITUNES_ARTWORK_SIZE_REGEX = Regex("""/(\d+x\d+)(bb)?\.jpg$""")

// Default sizes remain, but will be overridden by preference
private const val DEFAULT_HIGH_RES_SIZE = "600x600"
private const val MEDIUM_RES_SIZE = "300x300"
```

```
private const val LOW_RES_SIZE = "150x150"

enum class ThumbnailQuality {
    LOW,    // For tiny notifications or widgets (default.jpg - 120x90)
    MEDIUM, // For list items (mqdefault.jpg - 320x180)
    HIGH,   // For larger cards or mini-player (hqdefault.jpg - 480x360)
    MAX     // For full-screen displays (maxresdefault.jpg - 1280x720)
}

/**
 * NEW: Generates a prioritized list of YouTube thumbnail URLs for a given video ID.
 * Coil will attempt to load them in the order provided.
 */
fun getYouTubeThumbnailUrl(videoId: String, quality: ThumbnailQuality): List<String> {
    val baseUrl = "https://i.ytimg.com/vi/$videoId"
    return when (quality) {
        ThumbnailQuality.MAX -> listOf(
            "$baseUrl/maxresdefault.jpg",
            "$baseUrl/sddefault.jpg",
            "$baseUrl/hqdefault.jpg"
        )
        ThumbnailQuality.HIGH -> listOf(
            "$baseUrl/hqdefault.jpg",
            "$baseUrl/mqdefault.jpg",
            "$baseUrl/sddefault.jpg" // sddefault is often better than mqdefault
        )
        ThumbnailQuality.MEDIUM -> listOf(
            "$baseUrl/mqdefault.jpg",
            "$baseUrl/default.jpg"
        )
        ThumbnailQuality.LOW -> listOf(
            "$baseUrl/default.jpg"
        )
    }
}


// Updated function to take imageQualityKey as a parameter
fun getHighResArtworkUrl(
    originalUrl: String?,
    imageQualityKey: String = AppPreferenceConstants.IMAGE_QUALITY_AUTO, // Default to AUTO
    preferredSizeOverride: String? = null // Allows specific components (like FullPlayer) to s
): String? {
    if (originalUrl.isNullOrBlank()) {
        return null
    }

    if (!originalUrl.contains("mzstatic.com")) {
        return originalUrl // Return original if not an mzstatic URL
    }

    val targetSize = preferredSizeOverride ?: when (imageQualityKey) {
        AppPreferenceConstants.IMAGE_QUALITY_LOW -> LOW_RES_SIZE
        AppPreferenceConstants.IMAGE_QUALITY_MEDIUM -> MEDIUM_RES_SIZE
        AppPreferenceConstants.IMAGE_QUALITY_AUTO -> DEFAULT_HIGH_RES_SIZE // "Auto" here mean
```

```
        else -> DEFAULT_HIGH_RES_SIZE // Fallback
    }

    return ITUNES_ARTWORK_SIZE_REGEX.replace(originalUrl) { matchResult ->
        val bbSuffix = matchResult.groupValues.getOrNull(2) ?: ""
        Timber.d("getHighResArtworkUrl: URL: '$originalUrl', Quality: '$imageQualityKey', Targ
        "/${targetSize}${bbSuffix}.jpg"
    }.ifEmpty { originalUrl }
}
/**
 * Generates a prioritized, context-aware list of artwork URLs for a given PlaybackItem.
 * Coil will attempt to load from this list in order, using the first one that succeeds.
 *
 * @param item The PlaybackItem to get artwork for.
 * @param quality The desired quality for the *fallback* YouTube thumbnail. This is key
 * for requesting a high-res image for the player and a medium-res one for list items.
 * @return A list of URL strings in order of priority.
 */
fun generateArtworkUrlList(item: PlaybackItem?, quality: ThumbnailQuality): List<String> {
    if (item == null) return emptyList()

    val urls = mutableListOf<String>()

    // PRIORITY 1: The song's specific artwork URI from mzstatic.
    // We will use getHighResArtworkUrl to ensure we get a decently sized version.
    if (!item.artworkUri.isNullOrBlank() && item.artworkUri.contains("mzstatic.com")) {
        // For song-specific art, we usually want a high quality version regardless of context
        // We can use the existing utility for this.
        val highResSongArt = getHighResArtworkUrl(item.artworkUri, AppPreferenceConstants.IMAG
        if(highResSongArt != null) {
            urls.add(highResSongArt)
        }
    }

    // PRIORITY 2 (FALLBACK): YouTube thumbnails for the parent video, at the requested qualit
    urls.addAll(getYouTubeThumbnailUrl(item.videoId, quality))

    // As a final fallback, you could add the channel photo, but for now, we'll stick to
    // the user's request: song art -> video thumbnail.
    // if (!item.artworkUri.isNullOrBlank()) { urls.add(item.artworkUri) } // Fallback to chan

    return urls.distinct()
}

// File: java\com\example\holodex\util\PaletteExtractor.kt
package com.example.holodex.util

import android.content.Context
import android.graphics.Bitmap
import android.graphics.drawable.BitmapDrawable
import androidx.annotation.ColorInt
import androidx.collection.LruCache
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.toArgb
import androidx.core.graphics.ColorUtils
```

```
import androidx.palette.graphics.Palette
import coil.imageLoader
import coil.request.ImageRequest
import coil.request.SuccessResult
import coil.size.Size
import dagger.hilt.android.qualifiers.ApplicationContext
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.withContext
import timber.log.Timber
import javax.inject.Inject

// Data class to hold the extracted theme colors - keeping original structure
data class DynamicTheme(
    val primary: Color,
    val onPrimary: Color,
) {
    companion object {
        fun default(defaultPrimary: Color, defaultOnPrimary: Color) = DynamicTheme(
            primary = defaultPrimary,
            onPrimary = defaultOnPrimary
        )
    }
}

/**
 * Extracts a color palette from a given image URL with enhanced blur processing and in-memory
 */
class PaletteExtractor @Inject constructor(
    @ApplicationContext private val context: Context
) {
    private val cache = LruCache<String, DynamicTheme>(20) // Keep original cache size

    suspend fun extractThemeFromUrl(
        imageUrl: String?,
        defaultTheme: DynamicTheme
    ): DynamicTheme = withContext(Dispatchers.Default) {
        if (imageUrl.isNullOrBlank()) return@withContext defaultTheme

        // Return from cache if available
        cache.get(imageUrl)?.let {
            Timber.d("PaletteExtractor: Cache HIT for $imageUrl")
            return@withContext it
        }

        Timber.d("PaletteExtractor: Cache MISS for $imageUrl. Processing.")
        try {
            val request = ImageRequest.Builder(context)
                .data(imageUrl)
                .size(Size(256, 256)) // Slightly larger for better blur quality while keeping
                .allowHardware(false) // Palette requires software bitmaps
                .memoryCacheKey("${imageUrl}_palette_enhanced")
                .build()

            val result = (context.imageLoader.execute(request) as? SuccessResult)?.drawable
            val bitmap = (result as? BitmapDrawable)?.bitmap ?: return@withContext defaultThem
```

```
        // Create a more beautiful blurred version of the bitmap for better color extracti
        val enhancedBitmap = createEnhancedBitmap(bitmap)

        val palette = Palette.from(enhancedBitmap).generate()

        val swatch = palette.vibrantSwatch
            ?: palette.lightVibrantSwatch
            ?: palette.darkVibrantSwatch
            ?: palette.dominantSwatch
            ?: palette.mutedSwatch
            ?: palette.lightMutedSwatch
            ?: palette.darkMutedSwatch
            ?: return@withContext defaultTheme

        val primaryColor = Color(swatch.rgb)
        val onPrimaryColor = Color(getBestTextColorForBackground(swatch.rgb, Color.White.t

        val newTheme = DynamicTheme(primary = primaryColor, onPrimary = onPrimaryColor)
        cache.put(imageUrl, newTheme) // Store in cache
        return@withContext newTheme

    } catch (e: Exception) {
        Timber.e(e, "Failed to extract palette from URL: $imageUrl")
        return@withContext defaultTheme
    }
}

/**
 * Creates an enhanced version of the bitmap with better color saturation and slight blur
 * for more beautiful color extraction without affecting the original API
 */
private fun createEnhancedBitmap(originalBitmap: Bitmap): Bitmap {
    return try {
        val config = originalBitmap.config ?: Bitmap.Config.ARGB_8888
        val enhancedBitmap = originalBitmap.copy(config, false)

        // Apply a gentle blur to smooth out harsh details and create more cohesive colors
        val blurredBitmap = applyGaussianBlur(enhancedBitmap, 3f)

        // Enhance color saturation for more vibrant palette extraction
        enhanceSaturation(blurredBitmap, 1.2f)
    } catch (e: Exception) {
        Timber.w(e, "Failed to enhance bitmap, using original")
        originalBitmap
    }
}

/**
 * Apply Gaussian blur using a modern, efficient algorithm
 */
private fun applyGaussianBlur(bitmap: Bitmap, radius: Float): Bitmap {
    if (radius <= 0) return bitmap

    val config = bitmap.config ?: Bitmap.Config.ARGB_8888
```

```
    val blurred = bitmap.copy(config, true)

    val width = blurred.width
    val height = blurred.height
    val pixels = IntArray(width * height)
    blurred.getPixels(pixels, 0, width, 0, 0, width, height)

    // Apply horizontal blur
    blurPixels(pixels, width, height, radius.toInt(), true)
    // Apply vertical blur
    blurPixels(pixels, width, height, radius.toInt(), false)

    blurred.setPixels(pixels, 0, width, 0, 0, width, height)
    return blurred
}

/**
 * Efficient box blur implementation for horizontal and vertical passes
 */
private fun blurPixels(pixels: IntArray, width: Int, height: Int, radius: Int, horizontal:
    val blur = IntArray(pixels.size)
    val kernel = createGaussianKernel(radius)
    val kernelSize = kernel.size
    val kernelRadius = kernelSize / 2

    for (y in 0 until height) {
        for (x in 0 until width) {
            var r = 0f
            var g = 0f
            var b = 0f
            var a = 0f

            for (k in 0 until kernelSize) {
                val weight = kernel[k]
                val sampleX = if (horizontal) {
                    (x + k - kernelRadius).coerceIn(0, width - 1)
                } else x
                val sampleY = if (horizontal) y else {
                    (y + k - kernelRadius).coerceIn(0, height - 1)
                }

                val pixel = pixels[sampleY * width + sampleX]
                r += ((pixel shr 16) and 0xFF) * weight
                g += ((pixel shr 8) and 0xFF) * weight
                b += (pixel and 0xFF) * weight
                a += ((pixel shr 24) and 0xFF) * weight
            }

            val blurredPixel = (a.toInt() shl 24) or
                    (r.toInt() shl 16) or
                    (g.toInt() shl 8) or
                    b.toInt()
            blur[y * width + x] = blurredPixel
        }
    }
```

```
    System.arraycopy(blur, 0, pixels, 0, pixels.size)
}

/**
 * Create a Gaussian kernel for blur
 */
private fun createGaussianKernel(radius: Int): FloatArray {
    val size = radius * 2 + 1
    val kernel = FloatArray(size)
    val sigma = radius / 3f
    var sum = 0f

    for (i in kernel.indices) {
        val x = i - radius
        kernel[i] = kotlin.math.exp(-(x * x) / (2 * sigma * sigma))
        sum += kernel[i]
    }

    // Normalize
    for (i in kernel.indices) {
        kernel[i] /= sum
    }

    return kernel
}

/**
 * Enhance color saturation for more vibrant palette extraction
 */
private fun enhanceSaturation(bitmap: Bitmap, factor: Float): Bitmap {
    val width = bitmap.width
    val height = bitmap.height
    val pixels = IntArray(width * height)
    bitmap.getPixels(pixels, 0, width, 0, 0, width, height)

    for (i in pixels.indices) {
        val pixel = pixels[i]
        val hsv = FloatArray(3)
        val r = (pixel shr 16) and 0xFF
        val g = (pixel shr 8) and 0xFF
        val b = pixel and 0xFF

        android.graphics.Color.RGBToHSV(r, g, b, hsv)
        hsv[1] = (hsv[1] * factor).coerceIn(0f, 1f) // Enhance saturation

        val enhancedColor = android.graphics.Color.HSVToColor(
            (pixel shr 24) and 0xFF, hsv
        )
        pixels[i] = enhancedColor
    }

    bitmap.setPixels(pixels, 0, width, 0, 0, width, height)
    return bitmap
}
```

```
    /**
     * Determines whether light or dark text is more readable on a given background color.
     */
    @ColorInt
    private fun getBestTextColorForBackground(@ColorInt backgroundColor: Int, @ColorInt lightC
        val contrastWithLight = ColorUtils.calculateContrast(lightColor, backgroundColor)
        val contrastWithDark = ColorUtils.calculateContrast(darkColor, backgroundColor)
        return if (contrastWithLight > contrastWithDark) lightColor else darkColor
    }
}

// File: java\com\example\holodex\util\PlaylistFormatter.kt
// File: java/com/example/holodex/util/PlaylistFormatter.kt

package com.example.holodex.util

import android.content.Context
import com.example.holodex.R
import com.example.holodex.data.model.discovery.PlaylistStub
import com.google.gson.Gson
import com.google.gson.JsonSyntaxException
import timber.log.Timber

// Data classes to safely parse the JSON that is often in the description field of SGPs
private data class DescriptionContext(val channel: ChannelInfo?, val org: String?, val id: Str
private data class ChannelInfo(val name: String?, val english_name: String?)

// Base interface for all our specific formatters
private interface SgpFormatter {
    fun getTitle(
        playlist: PlaylistStub,
        params: Map<String, String>,
        descriptionJson: String?,
        context: Context,
        namePicker: (en: String?, jp: String?) -> String?
    ): String

    fun getDescription(
        playlist: PlaylistStub,
        params: Map<String, String>,
        descriptionJson: String?,
        context: Context,
        namePicker: (en: String?, jp: String?) -> String?
    ): String?
}

/**
 * A utility object to format playlist titles and descriptions, replicating
 * the logic from the Musicdex web frontend for consistency.
 */
object PlaylistFormatter {

    private val GSON = Gson()
```

```
// The central map, mirroring the web app's `formatters` object
private val formatters = mapOf<String, SgpFormatter>(
    ":artist" to ArtistFormatter,
    ":dailyrandom" to DailyRandomFormatter,
    ":video" to VideoFormatter,
    ":latest" to LatestFormatter,
    ":mv" to MvFormatter,
    ":weekly" to WeeklyFormatter,
    ":userweekly" to UserWeeklyFormatter,
    ":history" to HistoryFormatter,
    ":hot" to HotFormatter
)

/**
 * Gets the user-facing display title for any playlist.
 *
 * @param playlist The playlist object.
 * @param context Android context for string resources.
 * @param namePicker A helper lambda to choose between English and Japanese names.
 * @return The formatted title string.
 */
fun getDisplayTitle(
    playlist: PlaylistStub,
    context: Context,
    namePicker: (en: String?, jp: String?) -> String?
): String {
    if (!playlist.id.startsWith(":")) {
        return playlist.title
    }

    val (type, params) = parsePlaylistID(playlist.id)
    val formatter = formatters[type] ?: DefaultFormatter

    return formatter.getTitle(playlist, params, playlist.description, context, namePicker)
}
/**
 * Gets the user-facing display description for any playlist.
 *
 * @param playlist The playlist object.
 * @param context Android context for string resources.
 * @param namePicker A helper lambda to choose between English and Japanese names.
 * @return The formatted description string, or null if there is none.
 */
fun getDisplayDescription(
    playlist: PlaylistStub,
    context: Context,
    namePicker: (en: String?, jp: String?) -> String?
): String? {
    if (!playlist.id.startsWith(":")) {
        return playlist.description
    }

    val (type, params) = parsePlaylistID(playlist.id)
    val formatter = formatters[type] ?: DefaultFormatter
```

```
        return formatter.getDescription(playlist, params, playlist.description, context, nameP
    }
    private fun parsePlaylistID(id: String): Pair<String, Map<String, String>> {
        if (!id.startsWith(":")) return Pair(id, emptyMap())
        val typeEndIndex = id.indexOf('[')
        if (typeEndIndex == -1) return Pair(id, emptyMap())

        val type = id.substring(0, typeEndIndex)
        val paramsString = id.substring(typeEndIndex + 1, id.lastIndexOf(']'))

        val params = paramsString.split(',')
            .mapNotNull {
                val parts = it.split('=', limit = 2)
                if (parts.size == 2) parts[0] to parts[1] else null
            }
            .toMap()

        return Pair(type, params)
    }

    private fun parseDescription(json: String?): DescriptionContext? {
        if (json.isNullOrBlank()) return null
        return try {
            GSON.fromJson(json, DescriptionContext::class.java)
        } catch (e: JsonSyntaxException) {
            Timber.e(e, "Failed to parse SGP description JSON: $json")
            null
        }
    }

    // --- Formatter Implementations ---

    private object DefaultFormatter : SgpFormatter {
        override fun getTitle(p: PlaylistStub, pa: Map<String, String>, d: String?, c: Context
        override fun getDescription(p: PlaylistStub, pa: Map<String, String>, d: String?, c: C
    }


    private object ArtistFormatter : SgpFormatter {
        override fun getTitle(p: PlaylistStub, pa: Map<String, String>, d: String?, c: Context
            val channelInfo = parseDescription(d)?.channel
            val name = n(channelInfo?.english_name, channelInfo?.name) ?: "Artist"
            return c.getString(R.string.sgp_artist_radio_title, name)
        }
        override fun getDescription(p: PlaylistStub, pa: Map<String, String>, d: String?, c: C
            val channelInfo = parseDescription(d)?.channel
            val name = n(channelInfo?.english_name, channelInfo?.name) ?: "Artist"
            return c.getString(R.string.sgp_artist_radio_desc, name)
        }
    }

    private object DailyRandomFormatter : SgpFormatter {
        override fun getTitle(p: PlaylistStub, pa: Map<String, String>, d: String?, c: Context
            val channelInfo = parseDescription(d)?.channel
            val name = n(channelInfo?.english_name, channelInfo?.name) ?: "Artist"
```

```
            return c.getString(R.string.sgp_daily_mix_title, name)
        }
        override fun getDescription(p: PlaylistStub, pa: Map<String, String>, d: String?, c: C
            val channelInfo = parseDescription(d)?.channel
            val name = n(channelInfo?.english_name, channelInfo?.name) ?: "Artist"
            return c.getString(R.string.sgp_daily_mix_desc, name)
        }
    }

    private object MvFormatter : SgpFormatter {
        override fun getTitle(p: PlaylistStub, pa: Map<String, String>, d: String?, c: Context
            val org = pa["org"] ?: "Community"
            return when (pa["sort"]) {
                "random" -> c.getString(R.string.sgp_mv_random_title, org)
                "latest" -> c.getString(R.string.sgp_mv_latest_title, org)
                else -> p.title
            }
        }
        override fun getDescription(p: PlaylistStub, pa: Map<String, String>, d: String?, c: C
            val org = pa["org"] ?: "Community"
            return when (pa["sort"]) {
                "random" -> c.getString(R.string.sgp_mv_random_desc, org)
                "latest" -> c.getString(R.string.sgp_mv_latest_desc, org)
                else -> p.description
            }
        }
    }

    private object LatestFormatter : SgpFormatter {
        override fun getTitle(p: PlaylistStub, pa: Map<String, String>, d: String?, c: Context
            val org = pa["org"] ?: "Community"
            return c.getString(R.string.sgp_latest_title, org)
        }
        override fun getDescription(p: PlaylistStub, pa: Map<String, String>, d: String?, c: C
            val org = pa["org"] ?: "Community"
            return c.getString(R.string.sgp_latest_desc, org)
        }
    }

    private object WeeklyFormatter : SgpFormatter {
        override fun getTitle(p: PlaylistStub, pa: Map<String, String>, d: String?, c: Context
            val org = pa["org"] ?: "Community"
            return c.getString(R.string.sgp_weekly_mix_title, org)
        }
        override fun getDescription(p: PlaylistStub, pa: Map<String, String>, d: String?, c: C
            val org = pa["org"] ?: "Community"
            return c.getString(R.string.sgp_weekly_mix_desc, org)
        }
    }

    private object UserWeeklyFormatter : SgpFormatter {
        override fun getTitle(p: PlaylistStub, pa: Map<String, String>, d: String?, c: Context
        override fun getDescription(p: PlaylistStub, pa: Map<String, String>, d: String?, c: C
    }
```

```
    private object HistoryFormatter : SgpFormatter {
        override fun getTitle(p: PlaylistStub, pa: Map<String, String>, d: String?, c: Context
        override fun getDescription(p: PlaylistStub, pa: Map<String, String>, d: String?, c: C
    }

    private object HotFormatter : SgpFormatter {
        override fun getTitle(p: PlaylistStub, pa: Map<String, String>, d: String?, c: Context
        override fun getDescription(p: PlaylistStub, pa: Map<String, String>, d: String?, c: C
    }

    private object VideoFormatter: SgpFormatter {
        override fun getTitle(p: PlaylistStub, pa: Map<String, String>, d: String?, c: Context
            return parseDescription(d)?.title ?: p.title
        }
        override fun getDescription(p: PlaylistStub, pa: Map<String, String>, d: String?, c: C
            val desc = parseDescription(d)
            val name = n(desc?.channel?.english_name, desc?.channel?.name) ?: "Artist"
            return c.getString(R.string.sgp_video_desc, name)
        }
    }
}

// File: java\com\example\holodex\util\VideoFilteringUtil.kt
// File: java/com/example/holodex/util/VideoFilteringUtil.kt
package com.example.holodex.util

import com.example.holodex.data.model.HolodexVideoItem
import timber.log.Timber
import java.text.Normalizer

object VideoFilteringUtil {

    private val STRICT_CORE_MUSIC_TOPICS = setOf("singing", "Music_Cover", "Original_Song")

    private val musicKeywords = setOf(
        "cover", "?????", "song", "singing", "karaoke", "mv", "original song", "original", "mu
        "op/ed", "theme", "?", "??", "??????", "????????", "acoustic", "live", "concert",
        "????", "??", "medley", "arrange", "remix", "instrumental", "bgm", "soundtrack", "ost"
        "vocaloid", "???", "album", "single", "???", "guitar", "piano", "???", "??",
        "?", "???", "??????", "official audio", "music video"
    )

    private val channelMusicKeywords = setOf(
        "music", "song", "cover", "vsinger", "singer", "utaite", "archive", "records",
        "official channel", "??????", "??"
    )

    /**
     * Normalizes a title by converting all Unicode variants to their ASCII equivalents.
     * This includes:
     * - Full-width characters (Japanese/Chinese input style)
     * - Mathematical Alphanumeric Symbols (bold, italic, script, etc.)
     * - Enclosed Alphanumerics
     * - Accented characters
     * - Various stylized Unicode text
```

```
 *
 * Works for ANY word, not just hardcoded ones.
 */
private fun normalizeTitle(title: String): String {
    var normalized = title

    // Step 1: Replace common full-width punctuation
    normalized = normalized
        .replace("?", "(").replace("?", ")")
        .replace("?", "[").replace("?", "]")
        .replace("?", "{").replace("?", "}")
        .replace("?", "/").replace("?", "|")
        .replace("?", "@").replace("?", "#")
        .replace("?", "$").replace("?", "%")
        .replace("?", "&").replace("?", "*")
        .replace("?", "+").replace("?", "-")
        .replace("?", "=").replace("?", ":")
        .replace("?", ";").replace("?", "!")
        .replace("?", "?").replace("?", "~")
        .replace("?", "<").replace("?", ">")
        .replace("?", ".").replace("?", ",")
        .replace("'", "'").replace("'", "'")
        .replace("""", "\"").replace("""", "\"")
        .replace("?", " ") // Full-width space to regular space

    // Step 2: Use Unicode normalization to decompose accented characters
    // NFD = Canonical Decomposition (é becomes e + ´)
    normalized = Normalizer.normalize(normalized, Normalizer.Form.NFD)
        .replace(Regex("\\p{Mn}"), "") // Remove all diacritical marks

    // Step 3: Convert all stylized Unicode characters to ASCII
    val result = StringBuilder()
    var i = 0
    while (i < normalized.length) {
        val codePoint = normalized.codePointAt(i)
        val converted = convertUnicodeToAscii(codePoint)
        result.append(converted)
        i += Character.charCount(codePoint)
    }

    return result.toString()
}

/**
 * Converts a single Unicode code point to its ASCII equivalent string.
 * Handles multiple Unicode ranges for stylized text.
 */
private fun convertUnicodeToAscii(codePoint: Int): String {
    return when {
        // Full-width alphanumeric (?-?, ?-?, ?-?)
        codePoint in 0xFF01..0xFF5E -> {
            (codePoint - 0xFEE0).toChar().toString()
        }

        // Mathematical Bold (?-?, ?-?) U+1D400-U+1D433
```

```
codePoint in 0x1D400..0x1D419 -> ('A'.code + (codePoint - 0x1D400)).toChar().toStr
codePoint in 0x1D41A..0x1D433 -> ('a'.code + (codePoint - 0x1D41A)).toChar().toStr

// Mathematical Italic (?-?, ?-?) U+1D434-U+1D467
codePoint in 0x1D434..0x1D44D -> ('A'.code + (codePoint - 0x1D434)).toChar().toStr
codePoint in 0x1D44E..0x1D467 -> ('a'.code + (codePoint - 0x1D44E)).toChar().toStr

// Mathematical Bold Italic (?-?, ?-?) U+1D468-U+1D49B
codePoint in 0x1D468..0x1D481 -> ('A'.code + (codePoint - 0x1D468)).toChar().toStr
codePoint in 0x1D482..0x1D49B -> ('a'.code + (codePoint - 0x1D482)).toChar().toStr

// Mathematical Script (?-?, ?-?) U+1D49C-U+1D4CF
codePoint in 0x1D49C..0x1D4B5 -> ('A'.code + (codePoint - 0x1D49C)).toChar().toStr
codePoint in 0x1D4B6..0x1D4CF -> ('a'.code + (codePoint - 0x1D4B6)).toChar().toStr

// Mathematical Bold Script (?-?, ?-?) U+1D4D0-U+1D503
codePoint in 0x1D4D0..0x1D4E9 -> ('A'.code + (codePoint - 0x1D4D0)).toChar().toStr
codePoint in 0x1D4EA..0x1D503 -> ('a'.code + (codePoint - 0x1D4EA)).toChar().toStr

// Mathematical Fraktur (?-?, ?-?) U+1D504-U+1D537
codePoint in 0x1D504..0x1D51C -> ('A'.code + (codePoint - 0x1D504)).toChar().toStr
codePoint in 0x1D51E..0x1D537 -> ('a'.code + (codePoint - 0x1D51E)).toChar().toStr

// Mathematical Double-Struck (?-?, ?-?) U+1D538-U+1D56B
codePoint in 0x1D538..0x1D550 -> ('A'.code + (codePoint - 0x1D538)).toChar().toStr
codePoint in 0x1D552..0x1D56B -> ('a'.code + (codePoint - 0x1D552)).toChar().toStr

// Mathematical Bold Fraktur (?-?, ?-?) U+1D56C-U+1D59F
codePoint in 0x1D56C..0x1D585 -> ('A'.code + (codePoint - 0x1D56C)).toChar().toStr
codePoint in 0x1D586..0x1D59F -> ('a'.code + (codePoint - 0x1D586)).toChar().toStr

// Mathematical Sans-Serif (?-?, ?-?) U+1D5A0-U+1D5D3
codePoint in 0x1D5A0..0x1D5B9 -> ('A'.code + (codePoint - 0x1D5A0)).toChar().toStr
codePoint in 0x1D5BA..0x1D5D3 -> ('a'.code + (codePoint - 0x1D5BA)).toChar().toStr

// Mathematical Sans-Serif Bold (?-?, ?-?) U+1D5D4-U+1D607
codePoint in 0x1D5D4..0x1D5ED -> ('A'.code + (codePoint - 0x1D5D4)).toChar().toStr
codePoint in 0x1D5EE..0x1D607 -> ('a'.code + (codePoint - 0x1D5EE)).toChar().toStr

// Mathematical Sans-Serif Italic (?-?, ?-?) U+1D608-U+1D63B
codePoint in 0x1D608..0x1D621 -> ('A'.code + (codePoint - 0x1D608)).toChar().toStr
codePoint in 0x1D622..0x1D63B -> ('a'.code + (codePoint - 0x1D622)).toChar().toStr

// Mathematical Sans-Serif Bold Italic (?-?, ?-?) U+1D63C-U+1D66F
codePoint in 0x1D63C..0x1D655 -> ('A'.code + (codePoint - 0x1D63C)).toChar().toStr
codePoint in 0x1D656..0x1D66F -> ('a'.code + (codePoint - 0x1D656)).toChar().toStr

// Mathematical Monospace (?-?, ?-?) U+1D670-U+1D6A3
codePoint in 0x1D670..0x1D689 -> ('A'.code + (codePoint - 0x1D670)).toChar().toStr
codePoint in 0x1D68A..0x1D6A3 -> ('a'.code + (codePoint - 0x1D68A)).toChar().toStr

// Enclosed Alphanumerics (?-?, ?-?) U+24B6-U+24E9
codePoint in 0x24B6..0x24CF -> ('A'.code + (codePoint - 0x24B6)).toChar().toString
codePoint in 0x24D0..0x24E9 -> ('a'.code + (codePoint - 0x24D0)).toChar().toString
```

```
        // Parenthesized Latin (?-?) U+249C-U+24B5
        codePoint in 0x249C..0x24B5 -> ('a'.code + (codePoint - 0x249C)).toChar().toString

        // Squared Latin (?-?, ?-?) U+1F130-U+1F149, U+1F170-U+1F189
        codePoint in 0x1F130..0x1F149 -> ('A'.code + (codePoint - 0x1F130)).toChar().toStr
        codePoint in 0x1F170..0x1F189 -> ('A'.code + (codePoint - 0x1F170)).toChar().toStr

        // Regional Indicator Symbols (?-?) U+1F1E6-U+1F1FF
        codePoint in 0x1F1E6..0x1F1FF -> ('A'.code + (codePoint - 0x1F1E6)).toChar().toStr

        else -> Character.toChars(codePoint).concatToString()
    }
}

fun isMusicContent(video: HolodexVideoItem): Boolean {
    val videoLogId = "${video.id} ('${video.title.take(30)}...')"
    Timber.d("isMusicContent Checking: ID=${videoLogId}, Type=${video.type}, Topic=${video

    // 1. Strongest Indicator: Positive Song Count
    if ((video.songcount ?: 0) > 0 || !video.songs.isNullOrEmpty()) {
        Timber.d("isMusicContent [PASS] ID=${videoLogId} via songcount > 0 or non-empty so
        return true
    }

    // 2. Strict Core Music Topics
    if (STRICT_CORE_MUSIC_TOPICS.contains(video.topicId)) {
        Timber.d("isMusicContent [PASS] ID=${videoLogId} via STRICT_CORE_MUSIC_TOPICS: ${v
        return true
    }

    // 3. Normalize and check title keywords
    val normalizedTitle = normalizeTitle(video.title).lowercase()

    if (musicKeywords.any { keyword -> normalizedTitle.contains(keyword) }) {
        Timber.d("isMusicContent [PASS] ID=${videoLogId} via musicKeyword in title (topic
        return true
    }

    // 4. Specific Check for Music Shorts
    if (video.topicId == "shorts" || (video.type == "clip" && video.duration > 0 && video.
        if (musicKeywords.any { keyword -> normalizedTitle.contains(keyword) }) {
            Timber.d("isMusicContent [PASS] ID=${videoLogId} via music short (type/duratio
            return true
        }
    }

    // 5. Fallback for Generic Topics
    val potentiallyMusicRelatedTopics = setOf("3D_Stream", "FreeChat", "??", "misc", "unkn
    if (potentiallyMusicRelatedTopics.contains(video.topicId)) {
        val channelNameLower = video.channel.name.lowercase()
        if (channelMusicKeywords.any { keyword -> channelNameLower.contains(keyword) }) {
            if (musicKeywords.any { keyword -> normalizedTitle.contains(keyword) }) {
                Timber.d("isMusicContent [PASS] ID=${videoLogId} via generic topic, channe
                return true
            }
```

```
            }
        }

        Timber.d("isMusicContent [FAIL] ID=${videoLogId}. No conditions met.")
        return false
    }
}

// File: java\com\example\holodex\viewmodel\AddChannelViewModel.kt
package com.example.holodex.viewmodel

import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.example.holodex.data.model.ChannelSearchResult
import com.example.holodex.data.model.discovery.ChannelDetails
import com.example.holodex.data.repository.HolodexRepository
import com.example.holodex.data.repository.UnifiedVideoRepository
import com.example.holodex.viewmodel.state.UiState
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.FlowPreview
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.flow.debounce
import kotlinx.coroutines.launch
import javax.inject.Inject

@OptIn(FlowPreview::class)
@HiltViewModel
class AddChannelViewModel @Inject constructor(
    private val holodexRepository: HolodexRepository, // For Searching (NewPipe)
    private val unifiedRepository: UnifiedVideoRepository // For Saving (DB)
) : ViewModel() {

    // UI States
    private val _showDialog = MutableStateFlow(false)
    val showDialog: StateFlow<Boolean> = _showDialog.asStateFlow()

    private val _searchQuery = MutableStateFlow("")
    val searchQuery: StateFlow<String> = _searchQuery.asStateFlow()

    private val _searchState = MutableStateFlow<UiState<List<ChannelSearchResult>>>(UiState.Su
    val searchState: StateFlow<UiState<List<ChannelSearchResult>>> = _searchState.asStateFlow(

    private val _isAdding = MutableStateFlow<Set<String>>(emptySet())
    val isAdding: StateFlow<Set<String>> = _isAdding.asStateFlow()

    init {
        // Live Search Logic
        viewModelScope.launch {
            _searchQuery.debounce(500).collect { query ->
                if (query.length > 2) {
                    performChannelSearch(query)
                } else if (query.isEmpty()) {
                    _searchState.value = UiState.Success(emptyList())
```

```
                }
            }
        }
    }

    fun openDialog() { _showDialog.value = true }

    fun closeDialog() {
        _showDialog.value = false
        _searchQuery.value = ""
        _searchState.value = UiState.Success(emptyList())
    }

    fun onSearchQueryChanged(query: String) { _searchQuery.value = query }

    private fun performChannelSearch(query: String) {
        viewModelScope.launch {
            _searchState.value = UiState.Loading
            // Search via NewPipe (External)
            holodexRepository.searchForExternalChannels(query)
                .onSuccess { results -> _searchState.value = UiState.Success(results) }
                .onFailure { error -> _searchState.value = UiState.Error(error.localizedMessag
        }
    }

    fun addChannel(channel: ChannelSearchResult) {
        viewModelScope.launch {
            _isAdding.value += channel.channelId

            // 1. Convert Search Result to Unified Channel Format
            val details = ChannelDetails(
                id = channel.channelId,
                name = channel.name,
                englishName = channel.name, // External channels usually don't have separate E
                photoUrl = channel.thumbnailUrl,
                org = "External", // <--- CRITICAL: Marks it as non-Holodex
                description = null,
                bannerUrl = null,
                suborg = null,
                twitter = null,
                group = null
            )

            // 2. Save to Unified DB
            unifiedRepository.toggleChannelLike(details)

            // 3. UI Cleanup (The Set update triggers button state, but dialog closes anyway)
            closeDialog()
            _isAdding.value -= channel.channelId
        }
    }
}

// File: java\com\example\holodex\viewmodel\ChannelDetailsViewModel.kt
package com.example.holodex.viewmodel
```

```
import androidx.compose.ui.graphics.Color
import androidx.lifecycle.SavedStateHandle
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.example.holodex.data.model.discovery.ChannelDetails
import com.example.holodex.data.model.discovery.DiscoveryResponse
import com.example.holodex.data.repository.HolodexRepository
import com.example.holodex.data.repository.UnifiedVideoRepository
import com.example.holodex.util.DynamicTheme
import com.example.holodex.util.PaletteExtractor
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.collections.immutable.ImmutableList
import kotlinx.collections.immutable.persistentListOf
import kotlinx.collections.immutable.toImmutableList
import kotlinx.coroutines.async
import kotlinx.coroutines.coroutineScope
import kotlinx.coroutines.flow.SharingStarted
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.stateIn
import org.orbitmvi.orbit.Container
import org.orbitmvi.orbit.ContainerHost
import org.orbitmvi.orbit.viewmodel.container
import org.schabi.newpipe.extractor.Page
import javax.inject.Inject
import com.example.holodex.viewmodel.mappers.toUnifiedDisplayItem
import com.example.holodex.viewmodel.mappers.toVideoShell
// --- State & SideEffect (remain unchanged) ---
data class ChannelDetailsState(
    val isExternal: Boolean = false,
    val channelDetails: ChannelDetails? = null,
    val dynamicTheme: DynamicTheme = DynamicTheme.default(Color.Black, Color.White),
    val discoveryContent: DiscoveryResponse? = null,
    val popularSongs: ImmutableList<UnifiedDisplayItem> = persistentListOf(),
    val externalMusicItems: ImmutableList<UnifiedDisplayItem> = persistentListOf(),
    val nextPageCursor: Page? = null,
    val isLoadingMore: Boolean = false,
    val endOfList: Boolean = false,
    val isLoading: Boolean = true,
    val error: String? = null
)

sealed class ChannelDetailsSideEffect {
    data class ShowToast(val message: String) : ChannelDetailsSideEffect()
}

@HiltViewModel
class ChannelDetailsViewModel @Inject constructor(
    savedStateHandle: SavedStateHandle,
    private val holodexRepository: HolodexRepository,
    private val unifiedRepository: UnifiedVideoRepository,
    private val paletteExtractor: PaletteExtractor
) : ContainerHost<ChannelDetailsState, ChannelDetailsSideEffect>, ViewModel() {

    companion object {
```

```
    const val CHANNEL_ID_ARG = "channelId"
}

val channelId: String = savedStateHandle.get<String>(CHANNEL_ID_ARG) ?: ""

// *** FIX: isFavorited StateFlow is now correctly defined inside the class ***


override val container = container<ChannelDetailsState, ChannelDetailsSideEffect>(ChannelD
    if (channelId.isNotBlank()) {
        initializeChannel()
    } else {
        intent { reduce { state.copy(isLoading = false, error = "Invalid Channel ID") } }
    }
}

private fun initializeChannel() = intent {
    reduce { state.copy(isLoading = true, error = null) }

    val localChannel = unifiedRepository.getChannel(channelId)

    if (localChannel != null && localChannel.org == "External") {
        loadExternalChannel(localChannel)
    } else {
        loadHolodexChannel()
    }
}

private suspend fun loadExternalChannel(details: ChannelDetails) = intent {
    val theme = paletteExtractor.extractThemeFromUrl(details.photoUrl, DynamicTheme.defaul
    reduce {
        state.copy(
            isExternal = true,
            channelDetails = details,
            dynamicTheme = theme
        )
    }
    loadMoreExternalMusic(isInitial = true)
}

private suspend fun loadHolodexChannel() = intent {
    coroutineScope {
        val detailsDeferred = async { holodexRepository.getChannelDetails(channelId) }
        val discoveryDeferred = async { holodexRepository.getDiscoveryForChannel(channelId
        val popularDeferred = async { holodexRepository.getHotSongsForCarousel(channelId)

        val detailsResult = detailsDeferred.await()
        val discoveryResult = discoveryDeferred.await()
        val popularResult = popularDeferred.await()

        if (detailsResult.isSuccess) {
            val details = detailsResult.getOrThrow()
            val theme = paletteExtractor.extractThemeFromUrl(details.bannerUrl, DynamicThe
            val discovery = discoveryResult.getOrNull()
            val popular = popularResult.getOrNull()?.map { song ->
```

```
                    val videoShell = song.toVideoShell()
                    song.toUnifiedDisplayItem(parentVideo = videoShell, isLiked = false, isDow
                } ?: emptyList()


                reduce {
                    state.copy(
                        isExternal = false,
                        isLoading = false,
                        channelDetails = details,
                        dynamicTheme = theme,
                        discoveryContent = discovery,
                        popularSongs = popular.toImmutableList()
                    )
                }
            } else {
                val error = detailsResult.exceptionOrNull()?.localizedMessage ?: "Failed to lo
                reduce { state.copy(isLoading = false, error = error) }
            }
        }
    }

    fun loadMoreExternalMusic(isInitial: Boolean = false) = intent {
        if (!isInitial && (state.isLoadingMore || state.endOfList)) return@intent

        reduce { state.copy(isLoadingMore = true) }

        val cursor = if (isInitial) null else state.nextPageCursor
        val result = holodexRepository.getMusicFromExternalChannel(channelId, cursor)

        result.onSuccess { fetcherResult ->
            val newItems = fetcherResult.data.map { it.toUnifiedDisplayItem(isLiked = false, d
            val nextCursor = fetcherResult.nextPageCursor as? Page

            reduce {
                val currentList = if (isInitial) persistentListOf() else state.externalMusicIt
                state.copy(
                    externalMusicItems = (currentList + newItems).toImmutableList(),
                    nextPageCursor = nextCursor,
                    endOfList = nextCursor == null,
                    isLoading = false,
                    isLoadingMore = false
                )
            }
        }.onFailure {
            postSideEffect(ChannelDetailsSideEffect.ShowToast("Failed to load music"))
            reduce { state.copy(isLoading = false, isLoadingMore = false, error = if(isInitial
        }
    }
}

// File: java\com\example\holodex\viewmodel\DiscoveryViewModel.kt
// File: java/com/example/holodex/viewmodel/DiscoveryViewModel.kt

package com.example.holodex.viewmodel
```

```
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.example.holodex.auth.AuthState
import com.example.holodex.data.model.discovery.DiscoveryResponse
import com.example.holodex.data.model.discovery.PlaylistStub
import com.example.holodex.data.repository.HolodexRepository
import com.example.holodex.playback.domain.usecase.AddItemsToQueueUseCase
import com.example.holodex.playback.player.PlaybackController
import com.example.holodex.viewmodel.mappers.toPlaybackItem
import com.example.holodex.viewmodel.mappers.toUnifiedDisplayItem
import com.example.holodex.viewmodel.mappers.toVideoShell
import com.example.holodex.viewmodel.state.UiState
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.MutableSharedFlow
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.SharedFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asSharedFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.flow.update
import kotlinx.coroutines.launch
import timber.log.Timber
import javax.inject.Inject

enum class ShelfType {
    RECENT_STREAMS,
    SYSTEM_PLAYLISTS,
    ARTIST_RADIOS,
    FAN_PLAYLISTS,
    TRENDING_SONGS,
    DISCOVER_CHANNELS,
    FOR_YOU
}

data class DiscoveryScreenState(
    val shelves: Map<ShelfType, UiState<List<Any>>> = emptyMap(),
    val shelfOrder: List<ShelfType> = emptyList()
)

@HiltViewModel
class DiscoveryViewModel @Inject constructor(
    private val holodexRepository: HolodexRepository,
    private val playbackController: PlaybackController,
    private val addItemsToQueueUseCase: AddItemsToQueueUseCase
) : ViewModel() {

    private val _uiState = MutableStateFlow(DiscoveryScreenState())
    val uiState: StateFlow<DiscoveryScreenState> = _uiState.asStateFlow()

    private val _forYouState = MutableStateFlow<UiState<DiscoveryResponse>>(UiState.Loading)
    val forYouState: StateFlow<UiState<DiscoveryResponse>> = _forYouState.asStateFlow()

    private val _transientMessage = MutableSharedFlow<String>()
    val transientMessage: SharedFlow<String> = _transientMessage.asSharedFlow()
```

```
fun loadDiscoveryContent(organization: String, authState: AuthState) {
    val isFavoritesView = organization == "Favorites"
    val newShelfOrder = if (isFavoritesView && authState is AuthState.LoggedIn) {
        listOf(ShelfType.FOR_YOU)
    } else {
        listOf(
            ShelfType.RECENT_STREAMS,
            ShelfType.SYSTEM_PLAYLISTS,
            ShelfType.ARTIST_RADIOS,
            ShelfType.FAN_PLAYLISTS,
            ShelfType.TRENDING_SONGS,
            ShelfType.DISCOVER_CHANNELS
        )
    }

    val initialShelves = newShelfOrder.associateWith { UiState.Loading }
    _uiState.value = DiscoveryScreenState(shelves = initialShelves, shelfOrder = newShelfO

    if (isFavoritesView && authState is AuthState.LoggedIn) {
        fetchFavoritesHub()
    } else {
        val orgParam = organization.takeIf { it != "All Vtubers" }
        fetchTrendingSongs(orgParam)
        fetchDiscoveryHub(organization)
    }
}

fun loadForYouContent() {
    _forYouState.value = UiState.Loading
    viewModelScope.launch {
        holodexRepository.getFavoritesHubContent()
            .onSuccess { response ->
                _forYouState.value = UiState.Success(response)
            }
            .onFailure { error ->
                _forYouState.value = UiState.Error(error.localizedMessage ?: "Failed to lo
            }
    }
}

private fun fetchTrendingSongs(organization: String?) {
    viewModelScope.launch {
        holodexRepository.getHotSongsForCarousel(org = organization)
            .onSuccess { songs ->
                val displayItems = songs.map { song ->
                    val videoShell = song.toVideoShell()
                    song.toUnifiedDisplayItem(                        // FIX: Updated sign

                        parentVideo = videoShell,
                        isLiked = false, // We don't check likes for trending carousel for
                        isDownloaded = false
                    )
                }
                _uiState.update { s -> s.copy(shelves = s.shelves + (ShelfType.TRENDING_SO
```

```
            }.onFailure { e ->
                _uiState.update { s -> s.copy(shelves = s.shelves + (ShelfType.TRENDING_SO
            }
        }
    }

    private fun fetchDiscoveryHub(org: String) {
        viewModelScope.launch {
            holodexRepository.getDiscoveryHubContent(org)
                .onSuccess { response ->
                    val allPlaylists = response.recommended?.playlists ?: emptyList()

                    val systemPlaylists = allPlaylists.filter { it.type.startsWith("playlist/"
                    val radios = allPlaylists.filter { it.type.startsWith("radio/") }
                    val communityPlaylists = allPlaylists.filter { it.type == "ugp" }
                    val recentStreams = response.recentSingingStreams?.filter {
                        it.playlist?.content?.isNotEmpty() == true
                    } ?: emptyList()
                    val discoverChannels = response.channels ?: emptyList()

                    _uiState.update { s ->
                        s.copy(
                            shelves = s.shelves +
                                    (ShelfType.RECENT_STREAMS to UiState.Success(recentStreams
                                    (ShelfType.SYSTEM_PLAYLISTS to UiState.Success(systemPlayl
                                    (ShelfType.ARTIST_RADIOS to UiState.Success(radios)) +
                                    (ShelfType.FAN_PLAYLISTS to UiState.Success(communityPlayl
                                    (ShelfType.DISCOVER_CHANNELS to UiState.Success(discoverCh
                        )
                    }
                }.onFailure { e ->
                    val errorState = UiState.Error(e.localizedMessage ?: "Error")
                    _uiState.update { s ->
                        s.copy(
                            shelves = s.shelves +
                                    (ShelfType.RECENT_STREAMS to errorState) +
                                    (ShelfType.SYSTEM_PLAYLISTS to errorState) +
                                    (ShelfType.ARTIST_RADIOS to errorState) +
                                    (ShelfType.FAN_PLAYLISTS to errorState) +
                                    (ShelfType.DISCOVER_CHANNELS to errorState)
                        )
                    }
                }
        }
    }

    private fun fetchFavoritesHub() {
        viewModelScope.launch {
            holodexRepository.getFavoritesHubContent()
                .onSuccess { response ->
                    val recentStreams = response.recentSingingStreams?.filter {
                        it.playlist?.content?.isNotEmpty() == true
                    } ?: emptyList()
                    _uiState.update { s -> s.copy(shelves = s.shelves + (ShelfType.FOR_YOU to
                }.onFailure { e ->
```

```
                    _uiState.update { s -> s.copy(shelves = s.shelves + (ShelfType.FOR_YOU to
                }
            }
        }

        fun playUnifiedItem(item: UnifiedDisplayItem) {
            viewModelScope.launch {
                // New Call:
                playbackController.loadAndPlay(listOf(item.toPlaybackItem()))
            }
        }

        fun playRadioPlaylist(playlist: PlaylistStub) {
            viewModelScope.launch {
                if (playlist.type.startsWith("radio")) {
                    Timber.d("Playing playlist as Radio: ${playlist.id}")

                    playbackController.loadRadio(playlist.id)
                } else {
                    val result = holodexRepository.getFullPlaylistContent(playlist.id)
                    result.onSuccess { fullPlaylist ->
                        val playbackItems = fullPlaylist.content?.mapNotNull { song ->
                            if (song.channel.id == null) null
                            else {
                                val videoShell = song.toVideoShell(fullPlaylist.title)
                                song.toPlaybackItem(videoShell)
                            }
                        } ?: emptyList()

                        if (playbackItems.isNotEmpty()) {
                            // New Call:
                            playbackController.loadAndPlay(playbackItems)
                        } else {
                            _transientMessage.emit("This playlist appears to be empty.")
                        }
                    }.onFailure { error ->
                        _transientMessage.emit("Error: Could not load playlist.")
                    }
                }
            }
        }

        fun addAllToQueue(items: List<UnifiedDisplayItem>) {
            viewModelScope.launch {
                if (items.isNotEmpty()) {
                    val playbackItems = items.map { it.toPlaybackItem() }
                    // New Call (via UseCase wrapper or direct):
                    addItemsToQueueUseCase(playbackItems)
                    _transientMessage.emit("Added ${items.size} songs to queue.")
                }
            }
        }
    }

// File: java\com\example\holodex\viewmodel\DownloadsViewModel.kt
```

```
package com.example.holodex.viewmodel

import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import androidx.media3.common.util.UnstableApi
import com.example.holodex.data.repository.DownloadRepository
import com.example.holodex.data.repository.HolodexRepository
import com.example.holodex.data.repository.UnifiedVideoRepository
import com.example.holodex.playback.player.PlaybackController
import com.example.holodex.viewmodel.mappers.toPlaybackItem
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.collections.immutable.ImmutableList
import kotlinx.collections.immutable.persistentListOf
import kotlinx.collections.immutable.toImmutableList
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.combine
import kotlinx.coroutines.launch
import org.orbitmvi.orbit.ContainerHost
import org.orbitmvi.orbit.viewmodel.container
import timber.log.Timber
import javax.inject.Inject

data class DownloadsState(
    val items: ImmutableList<UnifiedDisplayItem> = persistentListOf(),
    val searchQuery: String = "",
    val isLoading: Boolean = true
)

sealed class DownloadsSideEffect {
    data class ShowToast(val message: String) : DownloadsSideEffect()
}

@UnstableApi
@HiltViewModel
class DownloadsViewModel @Inject constructor(
    private val unifiedRepository: UnifiedVideoRepository,
    private val downloadRepository: DownloadRepository,
    private val holodexRepository: HolodexRepository,
    private val playbackController: PlaybackController
) : ContainerHost<DownloadsState, DownloadsSideEffect>, ViewModel() {

    companion object {
        private const val TAG = "DownloadsViewModel"
    }

    private val queryFlow = MutableStateFlow("")

    override val container = container<DownloadsState, DownloadsSideEffect>(DownloadsState())
        observeDownloads()
    }

    private fun observeDownloads() = intent {
        combine(
            unifiedRepository.getDownloads(),
            queryFlow
```

```
    ) { downloads: List<UnifiedDisplayItem>, query: String ->
        if (query.isBlank()) {
            downloads
        } else {
            downloads.filter {
                it.title.contains(query, ignoreCase = true) ||
                        it.artistText.contains(query, ignoreCase = true)
            }
        }
    }.collect { filteredList ->
        reduce {
            state.copy(
                items = filteredList.toImmutableList(),
                isLoading = false,
                searchQuery = queryFlow.value
            )
        }
    }
}

fun onSearchQueryChanged(query: String) {
    queryFlow.value = query
}

fun playDownloads(tappedItem: UnifiedDisplayItem) = intent {
    // 1. ERROR STATE: Prioritize Retry
    if (tappedItem.downloadStatus == "FAILED" || tappedItem.downloadStatus == "EXPORT_FAIL
        postSideEffect(DownloadsSideEffect.ShowToast("Retrying download..."))
        // If it failed during export, retry export. Otherwise, full retry.
        if (tappedItem.downloadStatus == "EXPORT_FAILED") {
            retryExport(tappedItem)
        } else {
            retryDownload(tappedItem)
        }
        return@intent
    }

    // 2. PROCESSING STATE: Prevent Playback
    if (tappedItem.downloadStatus == "DOWNLOADING" ||
        tappedItem.downloadStatus == "PROCESSING" ||
        tappedItem.downloadStatus == "ENQUEUED") {
        postSideEffect(DownloadsSideEffect.ShowToast("Please wait, download in progress...
        return@intent
    }

    // 3. SUCCESS STATE: Play
    if (tappedItem.isDownloaded) {
        // Filter current list for only downloaded items to play in queue
        val playableItems = state.items.filter { it.isDownloaded }

        if (playableItems.isNotEmpty()) {
            val playbackItems = playableItems.map { it.toPlaybackItem() }
            // Use playbackItemId to find index, robust against composite IDs
            val startIndex = playbackItems.indexOfFirst { it.id == tappedItem.playbackItem
```

```
            playbackController.loadAndPlay(playbackItems, startIndex)

        }
        return@intent
    }

    // 4. Fallback (Zombie state)
    postSideEffect(DownloadsSideEffect.ShowToast("Status unknown. Deleting..."))
    deleteDownload(tappedItem.playbackItemId)
}


fun playAllDownloadsShuffled() = intent {
    val playableItems = state.items.filter { it.isDownloaded }
    if (playableItems.isNotEmpty()) {
        val playbackItems = playableItems.map { it.toPlaybackItem() }

        playbackItems.shuffled()

        playbackController.loadAndPlay(playbackItems.shuffled(), 0)
       playbackController.toggleShuffle()
    } else {
        postSideEffect(DownloadsSideEffect.ShowToast("No playable downloads found."))
    }
}


fun deleteDownload(itemId: String) = intent {
    viewModelScope.launch {
        try {
            downloadRepository.deleteDownloadById(itemId)
            postSideEffect(DownloadsSideEffect.ShowToast("Download deleted"))
        } catch (e: Exception) {
            postSideEffect(DownloadsSideEffect.ShowToast("Failed to delete download"))
        }
    }
}

fun cancelDownload(itemId: String) = intent {
    viewModelScope.launch {
        try {
            downloadRepository.cancelDownload(itemId)
            postSideEffect(DownloadsSideEffect.ShowToast("Download cancelled"))
        } catch (e: Exception) {
            Timber.e(e, "Failed to cancel $itemId")
        }
    }
}

fun resumeDownload(itemId: String) = intent {
    viewModelScope.launch {
        try {
            downloadRepository.resumeDownload(itemId)
            postSideEffect(DownloadsSideEffect.ShowToast("Resuming download..."))
        } catch (e: Exception) {
```

```
                Timber.e(e, "Failed to resume $itemId")
            }
        }
    }

    fun retryExport(item: UnifiedDisplayItem) = intent {
        // Retry logic using UnifiedDisplayItem ID
        viewModelScope.launch {
            try {
                downloadRepository.retryExport(item.playbackItemId)
                postSideEffect(DownloadsSideEffect.ShowToast("Retrying export..."))
            } catch (e: Exception) {
                Timber.e(e, "Failed to retry export")
            }
        }
    }

    fun retryDownload(item: UnifiedDisplayItem) = intent {
        viewModelScope.launch {
            val result = holodexRepository.getVideoWithSongs(item.videoId, forceRefresh = true
            result.onSuccess { videoWithSongs ->
                val songToRetry = videoWithSongs.songs?.find { it.start == (item.songStartSec
                if (songToRetry != null) {
                    downloadRepository.startDownload(videoWithSongs, songToRetry)
                    postSideEffect(DownloadsSideEffect.ShowToast("Retrying download..."))
                }
            }
        }
    }

}

// File: java\com\example\holodex\viewmodel\FavoritesViewModel.kt
package com.example.holodex.viewmodel

import androidx.lifecycle.ViewModel
import androidx.media3.common.util.UnstableApi
import com.example.holodex.data.model.HolodexVideoItem
import com.example.holodex.data.model.discovery.ChannelDetails
import com.example.holodex.data.repository.UnifiedVideoRepository
import com.example.holodex.playback.domain.model.PlaybackItem
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.collections.immutable.ImmutableList
import kotlinx.collections.immutable.persistentListOf
import kotlinx.collections.immutable.toImmutableList
import kotlinx.coroutines.flow.combine
import org.orbitmvi.orbit.ContainerHost
import org.orbitmvi.orbit.viewmodel.container
import javax.inject.Inject

enum class StorageLocation { LIKED_ITEMS, LOCAL_FAVORITES }

data class FavoritesState(
    val likedItemsMap: Map<String, StorageLocation> = emptyMap(),
    val unifiedLikedSegments: ImmutableList<UnifiedDisplayItem> = persistentListOf(),
```

```
    val unifiedFavoritedVideos: ImmutableList<UnifiedDisplayItem> = persistentListOf(),
    val favoriteChannels: ImmutableList<UnifiedDisplayItem> = persistentListOf(),
    val isLoading: Boolean = true
)

sealed class FavoritesSideEffect {
    data class ShowToast(val message: String) : FavoritesSideEffect()
}

@UnstableApi
@HiltViewModel
class FavoritesViewModel @Inject constructor(
    private val unifiedRepository: UnifiedVideoRepository
) : ViewModel(), ContainerHost<FavoritesState, FavoritesSideEffect> {

    override val container = container<FavoritesState, FavoritesSideEffect>(FavoritesState())
        intent {
            combine(
                unifiedRepository.getFavorites(),
                unifiedRepository.getFavoriteChannels(),
                unifiedRepository.observeLikedItemIds()
            ) { favorites, channels, likedIds ->

                // *** FIX: Add Channel IDs to the map ***
                val channelIds = channels.map { it.playbackItemId }.toSet()
                val allLikedIds = likedIds + channelIds

                val map = allLikedIds.associateWith { StorageLocation.LIKED_ITEMS }

                val videos = favorites.filter { !it.isSegment }.toImmutableList()
                val segments = favorites.filter { it.isSegment }.toImmutableList()
                val channelList = channels.toImmutableList()

                FavoritesState(
                    likedItemsMap = map,
                    unifiedFavoritedVideos = videos,
                    unifiedLikedSegments = segments,
                    favoriteChannels = channelList,
                    isLoading = false
                )
            }.collect { newState ->
                reduce { newState }
            }
        }
    }

    fun toggleLike(item: PlaybackItem) = intent {
        try {
            unifiedRepository.toggleLike(item)
        } catch (e: Exception) {
            postSideEffect(FavoritesSideEffect.ShowToast("Error updating favorites"))
        }
    }

    fun toggleFavoriteChannel(video: HolodexVideoItem) = intent {
```

```
        val details = ChannelDetails(
            id = video.channel.id ?: "",
            name = video.channel.name,
            englishName = video.channel.englishName,
            photoUrl = video.channel.photoUrl,
            org = video.channel.org,
            // Fill defaults for fields not present in VideoItem
            description = null,
            bannerUrl = null,
            suborg = null,
            twitter = null,
            group = null
        )

        try {
            unifiedRepository.toggleChannelLike(details)
            postSideEffect(FavoritesSideEffect.ShowToast("Channel updated"))
        } catch (e: Exception) {
            postSideEffect(FavoritesSideEffect.ShowToast("Failed to update channel"))
        }
    }

    // Keep this overload for the Channel Screen
    fun toggleFavoriteChannel(details: ChannelDetails) = intent {
        try {
            unifiedRepository.toggleChannelLike(details)
            postSideEffect(FavoritesSideEffect.ShowToast("Channel updated"))
        } catch (e: Exception) {
            postSideEffect(FavoritesSideEffect.ShowToast("Failed to update channel"))
        }
    }
}

// File: java\com\example\holodex\viewmodel\FullListViewModel.kt
package com.example.holodex.viewmodel

import androidx.lifecycle.SavedStateHandle
import androidx.lifecycle.ViewModel
import androidx.media3.common.util.UnstableApi
import com.example.holodex.data.model.discovery.DiscoveryChannel
import com.example.holodex.data.repository.HolodexRepository
import com.example.holodex.data.repository.UnifiedVideoRepository
import com.example.holodex.viewmodel.mappers.toUnifiedDisplayItem
import com.example.holodex.viewmodel.mappers.toVideoShell
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.first
import org.orbitmvi.orbit.Container
import org.orbitmvi.orbit.ContainerHost
import org.orbitmvi.orbit.viewmodel.container
import timber.log.Timber
import java.net.URLDecoder
import java.nio.charset.StandardCharsets
import javax.inject.Inject

// --- STATE & SIDE EFFECT DEFINITIONS ---
```

```
data class FullListState(
    val items: List<Any> = emptyList(),
    val isLoadingInitial: Boolean = true,
    val isLoadingMore: Boolean = false,
    val endOfList: Boolean = false,
    val currentOffset: Int = 0
)

sealed class FullListSideEffect {
    data class ShowToast(val message: String) : FullListSideEffect()
}

// ----------------------------------------

@UnstableApi
@HiltViewModel
class FullListViewModel @Inject constructor(
    private val savedStateHandle: SavedStateHandle,
    private val holodexRepository: HolodexRepository,
    private val unifiedRepository: UnifiedVideoRepository,
    ) : ContainerHost<FullListState, FullListSideEffect>, ViewModel() {

    companion object {
        const val CATEGORY_TYPE_ARG = "category"
        const val ORG_ARG = "org"
        private const val PAGE_SIZE = 50
    }

    val categoryType: MusicCategoryType = MusicCategoryType.valueOf(
        savedStateHandle.get<String>(CATEGORY_TYPE_ARG) ?: MusicCategoryType.TRENDING.name
    )
    private val organization: String = URLDecoder.decode(
        savedStateHandle.get<String>(ORG_ARG) ?: "All Vtubers",
        StandardCharsets.UTF_8.toString()
    )

    override val container: Container<FullListState, FullListSideEffect> = container(FullListS
        loadMore(isInitialLoad = true)
    }

    fun loadMore(isInitialLoad: Boolean = false) = intent {
        if (!isInitialLoad && (state.isLoadingMore || state.endOfList)) return@intent

        reduce {
            state.copy(
                isLoadingInitial = isInitialLoad,
                isLoadingMore = !isInitialLoad
            )
        }

        val offset = if (isInitialLoad) 0 else state.currentOffset

        val result: Result<Any> = runCatching {
            when (categoryType) {
```

```
            MusicCategoryType.TRENDING -> holodexRepository.getHotSongsForCarousel(organiz
            MusicCategoryType.UPCOMING_MUSIC -> holodexRepository.getUpcomingMusicPaginate
                org = organization.takeIf { it != "All Vtubers" },
                offset = offset
            ).getOrThrow()
            MusicCategoryType.RECENT_STREAMS -> holodexRepository.getLatestSongsPaginated(
            MusicCategoryType.COMMUNITY_PLAYLISTS -> holodexRepository.getOrgPlaylistsPagi
                org = organization, type = "ugp", offset = offset, limit = PAGE_SIZE
            ).getOrThrow()
            MusicCategoryType.ARTIST_RADIOS -> holodexRepository.getOrgPlaylistsPaginated(
                org = organization, type = "radio", offset = offset, limit = PAGE_SIZE
            ).getOrThrow()
            MusicCategoryType.SYSTEM_PLAYLISTS -> holodexRepository.getOrgPlaylistsPaginat
                org = organization, type = "sgp", offset = offset, limit = PAGE_SIZE
            ).getOrThrow()
            MusicCategoryType.DISCOVER_CHANNELS -> holodexRepository.getOrgChannelsPaginat
                org = organization, offset = offset, limit = PAGE_SIZE
            ).getOrThrow()
            else -> throw NotImplementedError("Category $categoryType not implemented")
        }
    }

    result.onSuccess { response ->
        val likedIds = holodexRepository.likedItemIds.first()

        // FIX: Use Unified Repository for downloads
        val downloadedIds = unifiedRepository.getDownloads().first().map { it.playbackItem

        val newItems: List<Any> = when (response) {
            is List<*> -> {
                @Suppress("UNCHECKED_CAST")
                (response as List<com.example.holodex.data.model.discovery.MusicdexSong>).
                    // FIX: Ensure toVideoShell is imported
                    val videoShell = song.toVideoShell()
                    song.toUnifiedDisplayItem(
                        parentVideo = videoShell,
                        isLiked = likedIds.contains("${song.videoId}_${song.start}"),
                        isDownloaded = downloadedIds.contains("${song.videoId}_${song.star
                    )
                }
            }
            is com.example.holodex.data.cache.FetcherResult<*> -> {
                (response.data as List<com.example.holodex.data.model.HolodexVideoItem>).m
                    video.toUnifiedDisplayItem(
                        isLiked = likedIds.contains(video.id),
                        downloadedSegmentIds = downloadedIds
                    )
                }
            }
            is com.example.holodex.data.api.PaginatedSongsResponse -> {
                response.items.map { song ->
                    val videoShell = song.toVideoShell()
                    song.toUnifiedDisplayItem(
                        parentVideo = videoShell,
                        isLiked = likedIds.contains("${song.videoId}_${song.start}"),
```

```
                            isDownloaded = downloadedIds.contains("${song.videoId}_${song.star
                        )
                    }
                }
                is com.example.holodex.data.api.PlaylistListResponse -> response.items
                is com.example.holodex.data.api.PaginatedChannelsResponse -> response.items.ma
                else -> emptyList()
            }

            val newItemsCount = newItems.size
            var finalItemsList = if (isInitialLoad) emptyList() else state.items

            if (categoryType == MusicCategoryType.DISCOVER_CHANNELS) {
                val currentChannels = finalItemsList.filterIsInstance<DiscoveryChannel>()
                val incomingChannels = newItems.filterIsInstance<DiscoveryChannel>()
                val allChannels = (currentChannels + incomingChannels).distinctBy { it.id }

                val grouped = allChannels.groupBy { it.suborg?.takeIf { s -> s.isNotBlank() }

                val groupedList = mutableListOf<Any>()
                grouped.keys.sorted().forEach { header ->
                    groupedList.add(SubOrgHeader(header))
                    groupedList.addAll(grouped[header]?.sortedBy { it.name } ?: emptyList())
                }
                finalItemsList = groupedList
            } else {
                finalItemsList = finalItemsList + newItems
            }

            val isEndOfList = newItemsCount < PAGE_SIZE || categoryType == MusicCategoryType.T

            reduce {
                state.copy(
                    items = finalItemsList,
                    currentOffset = offset + newItemsCount,
                    endOfList = isEndOfList,
                    isLoadingInitial = false,
                    isLoadingMore = false
                )
            }

        }.onFailure { error ->
            Timber.e(error, "Error loading full list")
            postSideEffect(FullListSideEffect.ShowToast("Failed to load data"))
            reduce {
                state.copy(isLoadingInitial = false, isLoadingMore = false)
            }
        }
    }
}

private fun com.example.holodex.data.model.discovery.ChannelDetails.toDiscoveryChannel(): Disc
    return DiscoveryChannel(
        id = this.id,
        name = this.name,
```

```
        englishName = this.englishName,
        photoUrl = this.photoUrl,
        songCount = null,
        suborg = this.group
    )
}

// File: java\com\example\holodex\viewmodel\FullPlayerViewModel.kt
// Create this new file: java/com/example/holodex/viewmodel/FullPlayerViewModel.kt
package com.example.holodex.viewmodel

import androidx.compose.ui.graphics.Color
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.example.holodex.util.DynamicTheme
import com.example.holodex.util.PaletteExtractor
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.launch
import javax.inject.Inject

@HiltViewModel
class FullPlayerViewModel @Inject constructor(
    private val paletteExtractor: PaletteExtractor
) : ViewModel() {

    private val _dynamicTheme = MutableStateFlow(DynamicTheme.default(Color.Black, Color.White
    val dynamicTheme: StateFlow<DynamicTheme> = _dynamicTheme.asStateFlow()

    fun updateThemeFromArtwork(artworkUri: String?) {
        viewModelScope.launch {
            val defaultTheme = DynamicTheme.default(
                defaultPrimary = _dynamicTheme.value.primary,
                defaultOnPrimary = _dynamicTheme.value.onPrimary
            )
            _dynamicTheme.value = paletteExtractor.extractThemeFromUrl(artworkUri, defaultThem
        }
    }
}

// File: java\com\example\holodex\viewmodel\HistoryViewModel.kt
package com.example.holodex.viewmodel

import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.example.holodex.data.repository.UnifiedVideoRepository
import com.example.holodex.playback.domain.usecase.AddItemsToQueueUseCase
import com.example.holodex.playback.player.PlaybackController
import com.example.holodex.viewmodel.mappers.toPlaybackItem
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.collections.immutable.ImmutableList
import kotlinx.collections.immutable.persistentListOf
import kotlinx.collections.immutable.toImmutableList
```

```
import kotlinx.coroutines.flow.collectLatest
import kotlinx.coroutines.launch
import org.orbitmvi.orbit.Container
import org.orbitmvi.orbit.ContainerHost
import org.orbitmvi.orbit.viewmodel.container
import javax.inject.Inject

data class HistoryState(
    val items: ImmutableList<UnifiedDisplayItem> = persistentListOf(),
    val isLoading: Boolean = true
)

sealed class HistorySideEffect {
    data class ShowToast(val message: String) : HistorySideEffect()
}

@HiltViewModel
class HistoryViewModel @Inject constructor(
    private val unifiedRepository: UnifiedVideoRepository,
    private val playbackController: PlaybackController,
    private val addItemsToQueueUseCase: AddItemsToQueueUseCase
) : ContainerHost<HistoryState, HistorySideEffect>, ViewModel() {

    override val container: Container<HistoryState, HistorySideEffect> = container(HistoryStat
        observeHistory()
    }

    private fun observeHistory() = intent {
        // The unified repository returns items that already have liked/download status merged
        // We just need to collect it.
        unifiedRepository.getHistory().collectLatest { historyItems ->
            reduce {
                state.copy(
                    items = historyItems.toImmutableList(),
                    isLoading = false
                )
            }
        }
    }

    fun playFromHistoryItem(tappedItem: UnifiedDisplayItem) {
        // Removed 'intent' scope for simplicity, use viewModelScope if using standard MVVM no
        // or keep 'intent' if using Orbit.
        // Assuming Orbit 'intent':
        intent {
            val currentHistory = state.items
            val tappedIndex = currentHistory.indexOf(tappedItem)

            if (tappedIndex == -1) {
                postSideEffect(HistorySideEffect.ShowToast("Error: Could not find item to play
                return@intent
            }

            val playbackItems = currentHistory.map { it.toPlaybackItem() }
```

```
            // New Call:
            // Note: No need to launch a new coroutine inside intent if loadAndPlay handles sc
            // but safe to call from here.
            playbackController.loadAndPlay(items = playbackItems, startIndex = tappedIndex)
        }
    }

    fun playAllHistory() = intent {
        val playbackItems = state.items.map { it.toPlaybackItem() }
        if (playbackItems.isNotEmpty()) {
            // New Call:
            playbackController.loadAndPlay(items = playbackItems)
        } else {
            postSideEffect(HistorySideEffect.ShowToast("History is empty."))
        }
    }

    fun addAllHistoryToQueue() = intent {
        val playbackItems = state.items.map { it.toPlaybackItem() }
        if (playbackItems.isNotEmpty()) {
            viewModelScope.launch {
                addItemsToQueueUseCase(playbackItems)
            }
            postSideEffect(HistorySideEffect.ShowToast("Added ${playbackItems.size} songs to q
        } else {
            postSideEffect(HistorySideEffect.ShowToast("History is empty."))
        }
    }
}

// File: java\com\example\holodex\viewmodel\PlaybackUiStateSelectors.kt
package com.example.holodex.viewmodel

import androidx.compose.runtime.Composable
import androidx.compose.runtime.State
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.lifecycle.compose.collectAsStateWithLifecycle
import com.example.holodex.playback.domain.model.DomainPlaybackProgress
import com.example.holodex.playback.domain.model.DomainRepeatMode
import com.example.holodex.playback.domain.model.DomainShuffleMode
import com.example.holodex.playback.domain.model.PlaybackItem
import com.example.holodex.util.getHighResArtworkUrl
import kotlinx.coroutines.flow.StateFlow
import org.orbitmvi.orbit.compose.collectAsState

// --- Selectors for MiniPlayer ---

@Composable
fun rememberMiniPlayerArtworkState(
    uiStateFlow: StateFlow<PlaybackUiState>,
    settingsViewModel: SettingsViewModel = hiltViewModel()
): State<String?> {
```

```
    val uiState by uiStateFlow.collectAsStateWithLifecycle()
    // FIX: Collect Orbit state
    val settingsState by settingsViewModel.collectAsState()
    val imageQuality = settingsState.currentImageQuality

    return remember(uiState.currentItem?.artworkUri, imageQuality) {
        mutableStateOf(
            getHighResArtworkUrl(
                uiState.currentItem?.artworkUri,
                imageQualityKey = imageQuality,
                preferredSizeOverride = "200x200"
            )
        )
    }
}

// ... (rememberMiniPlayerTitleState, rememberMiniPlayerArtistState, rememberIsPlayingState, r

@Composable
fun rememberMiniPlayerTitleState(uiStateFlow: StateFlow<PlaybackUiState>): State<String?> {
    val uiState by uiStateFlow.collectAsStateWithLifecycle()
    return remember(uiState.currentItem?.id) { mutableStateOf(uiState.currentItem?.title) }
}

@Composable
fun rememberMiniPlayerArtistState(uiStateFlow: StateFlow<PlaybackUiState>): State<String?> {
    val uiState by uiStateFlow.collectAsStateWithLifecycle()
    return remember(uiState.currentItem?.id) { mutableStateOf(uiState.currentItem?.artistText)
}

@Composable
fun rememberIsPlayingState(uiStateFlow: StateFlow<PlaybackUiState>): State<Boolean> {
    val uiState by uiStateFlow.collectAsStateWithLifecycle()
    return remember(uiState.isPlaying) { mutableStateOf(uiState.isPlaying) }
}

@Composable
fun rememberMiniPlayerProgressState(uiStateFlow: StateFlow<PlaybackUiState>): State<Float> {
    val uiState by uiStateFlow.collectAsStateWithLifecycle()
    return remember(uiState.progress, uiState.currentItem?.id) {
        val progressFraction = if (uiState.currentItem != null && uiState.progress.durationSec
            (uiState.progress.positionSec.toFloat() / uiState.progress.durationSec.toFloat()).
        } else {
            0f
        }
        mutableStateOf(progressFraction)
    }
}

@Composable
fun rememberMiniPlayerQueueStateForButton(uiStateFlow: StateFlow<PlaybackUiState>): State<Pair
    val uiState by uiStateFlow.collectAsStateWithLifecycle()
    return remember(uiState.queue, uiState.currentItem, uiState.currentIndexInQueue, uiState.r
        val hasItemAndQueue = uiState.queue.isNotEmpty() && uiState.currentItem != null
        val canSkipNext = hasItemAndQueue &&
```

```
                (uiState.currentIndexInQueue < uiState.queue.size - 1 || uiState.queue.size ==
        mutableStateOf(hasItemAndQueue to canSkipNext)
    }
}

// --- Selectors for FullPlayerScreen ---

@Composable
fun rememberFullPlayerArtworkState(
    uiStateFlow: StateFlow<PlaybackUiState>,
    settingsViewModel: SettingsViewModel = hiltViewModel()
): State<String?> {
    val uiState by uiStateFlow.collectAsStateWithLifecycle()
    // FIX: Collect Orbit state
    val settingsState by settingsViewModel.collectAsState()
    val imageQuality = settingsState.currentImageQuality

    return remember(uiState.currentItem?.artworkUri, imageQuality) {
        mutableStateOf(
            getHighResArtworkUrl(
                uiState.currentItem?.artworkUri,
                imageQualityKey = imageQuality,
                preferredSizeOverride = null
            )
        )
    }
}

// ... (rememberFullPlayerCurrentItemState, etc. REMAIN UNCHANGED)

@Composable
fun rememberFullPlayerCurrentItemState(uiStateFlow: StateFlow<PlaybackUiState>): State<Playbac
    val uiState by uiStateFlow.collectAsStateWithLifecycle()
    return remember(uiState.currentItem) { mutableStateOf(uiState.currentItem) }
}

@Composable
fun rememberFullPlayerProgressState(uiStateFlow: StateFlow<PlaybackUiState>): State<DomainPlay
    val uiState by uiStateFlow.collectAsStateWithLifecycle()
    return remember(uiState.progress) { mutableStateOf(uiState.progress) }
}

@Composable
fun rememberFullPlayerQueueInfoState(uiStateFlow: StateFlow<PlaybackUiState>): State<Triple<Li
    val uiState by uiStateFlow.collectAsStateWithLifecycle()
    return remember(uiState.queue, uiState.currentIndexInQueue) {
        mutableStateOf(Triple(uiState.queue, uiState.currentIndexInQueue, uiState.queue.isNotE
    }
}

@Composable
fun rememberFullPlayerLoadingState(uiStateFlow: StateFlow<PlaybackUiState>): State<Boolean> {
    val uiState by uiStateFlow.collectAsStateWithLifecycle()
    return remember(uiState.isLoading) { mutableStateOf(uiState.isLoading) }
}
```

```
@Composable
fun rememberFullPlayerControlModesState(uiStateFlow: StateFlow<PlaybackUiState>): State<Pair<D
    val uiState by uiStateFlow.collectAsStateWithLifecycle()
    return remember(uiState.repeatMode, uiState.shuffleMode) {
        mutableStateOf(uiState.repeatMode to uiState.shuffleMode)
    }
}

// File: java\com\example\holodex\viewmodel\PlaybackViewModel.kt
package com.example.holodex.viewmodel

import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import androidx.media3.common.util.UnstableApi
import com.example.holodex.playback.domain.model.DomainPlaybackProgress
import com.example.holodex.playback.domain.model.PlaybackItem
import com.example.holodex.playback.player.PlaybackController
import com.example.holodex.viewmodel.autoplay.ContinuationManager
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.SharingStarted
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.map
import kotlinx.coroutines.flow.stateIn
import javax.inject.Inject

// We can map the Controller's State to the UI State the Views expect
// ensuring we don't break the UI layer code.
data class PlaybackUiState(
    val currentItem: PlaybackItem? = null,
    val isPlaying: Boolean = false,
    val progress: DomainPlaybackProgress = DomainPlaybackProgress.NONE,
    val queue: List<PlaybackItem> = emptyList(),
    val currentIndexInQueue: Int = -1,
    val repeatMode: com.example.holodex.playback.domain.model.DomainRepeatMode = com.example.h
    val shuffleMode: com.example.holodex.playback.domain.model.DomainShuffleMode = com.example
    val isLoading: Boolean = false
)

@UnstableApi
@HiltViewModel
class PlaybackViewModel @Inject constructor(
    private val controller: PlaybackController,
    continuationManager: ContinuationManager
) : ViewModel() {

    val isRadioModeActive = continuationManager.isRadioModeActive
        .stateIn(viewModelScope, SharingStarted.WhileSubscribed(5000), false)

    val uiState: StateFlow<PlaybackUiState> = controller.state.map { s ->
        PlaybackUiState(
            currentItem = s.activeQueue.getOrNull(s.currentIndex),
            isPlaying = s.isPlaying,
            progress = DomainPlaybackProgress(s.progressMs / 1000, s.durationMs / 1000, 0),
            queue = s.activeQueue,
```

```
            currentIndexInQueue = s.currentIndex,
            repeatMode = s.repeatMode,
            shuffleMode = s.shuffleMode,
            isLoading = s.isLoading
        )
    }.stateIn(
        viewModelScope,
        SharingStarted.WhileSubscribed(5000),
        PlaybackUiState()
    )

    // --- COMMANDS ---

    fun playItems(items: List<PlaybackItem>, startIndex: Int = 0, startPositionMs: Long = 0L)
        controller.loadAndPlay(items, startIndex, startPositionMs)
    }

    fun togglePlayPause() = controller.togglePlayPause()
    fun seekTo(positionSec: Long) = controller.seekTo(positionSec * 1000L)
    fun skipToNext() = controller.skipToNext()
    fun skipToPrevious() = controller.skipToPrevious()
    fun toggleRepeatMode() {
        // Controller logic for repeat toggle could be added or handled here
        // For now, standard ExoPlayer rotation:
        // controller.toggleRepeatMode() // You might need to add this to Controller if not pr
    }
    fun toggleShuffleMode() = controller.toggleShuffle()

    fun playQueueItemAtIndex(index: Int) = controller.exoPlayer.seekToDefaultPosition(index)
    fun removeItemFromQueue(index: Int) = controller.exoPlayer.removeMediaItem(index)
    fun reorderQueueItem(from: Int, to: Int) = controller.exoPlayer.moveMediaItem(from, to)
    fun clearCurrentQueue() {
        controller.exoPlayer.clearMediaItems()
        controller.exoPlayer.stop()
    }

    // Stub for old scrubbing logic (PlayerController handles seek efficiently now)
    fun setScrubbing(isScrubbing: Boolean) {}

    fun getAudioSessionId(): Int? = controller.exoPlayer.audioSessionId
}

// File: java\com\example\holodex\viewmodel\PlaylistDetailsViewModel.kt
package com.example.holodex.viewmodel

import android.app.Application
import androidx.compose.ui.graphics.Color
import androidx.lifecycle.SavedStateHandle
import androidx.lifecycle.ViewModel
import androidx.media3.common.util.UnstableApi
import com.example.holodex.R
import com.example.holodex.auth.TokenManager
import com.example.holodex.data.db.PlaylistEntity
import com.example.holodex.data.db.PlaylistItemEntity
import com.example.holodex.data.db.SyncStatus
```

```
import com.example.holodex.data.model.discovery.PlaylistStub
import com.example.holodex.data.repository.HolodexRepository
import com.example.holodex.data.repository.UnifiedVideoRepository
import com.example.holodex.playback.domain.usecase.AddItemsToQueueUseCase
import com.example.holodex.playback.player.PlaybackController
import com.example.holodex.util.ArtworkResolver
import com.example.holodex.util.DynamicTheme
import com.example.holodex.util.PaletteExtractor
import com.example.holodex.util.PlaylistFormatter
import com.example.holodex.viewmodel.mappers.toPlaybackItem
import com.example.holodex.viewmodel.mappers.toUnifiedDisplayItem
import com.example.holodex.viewmodel.mappers.toVideoShell
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.first
import org.orbitmvi.orbit.Container
import org.orbitmvi.orbit.ContainerHost
import org.orbitmvi.orbit.viewmodel.container
import timber.log.Timber
import java.time.Instant
import javax.inject.Inject

// --- State Definition ---
data class PlaylistDetailsState(
    val playlist: PlaylistEntity? = null,
    val items: List<UnifiedDisplayItem> = emptyList(),
    // We keep rawItems to handle editing logic for User Playlists (PlaylistItemEntity)
    val rawItems: List<Any> = emptyList(),
    val isLoading: Boolean = true,
    val error: String? = null,

    // Edit Mode State
    val isEditMode: Boolean = false,
    val editablePlaylist: PlaylistEntity? = null,
    val editableItems: List<PlaylistItemEntity> = emptyList(),

    // Context State
    val isPlaylistOwned: Boolean = false,
    val isShuffleActive: Boolean = false,
    val dynamicTheme: DynamicTheme = DynamicTheme.default(Color.Black, Color.White)
)

sealed class PlaylistDetailsSideEffect {
    data class ShowToast(val message: String) : PlaylistDetailsSideEffect()
}

@UnstableApi
@HiltViewModel
class PlaylistDetailsViewModel @Inject constructor(
    private val application: Application,
    savedStateHandle: SavedStateHandle,
    private val holodexRepository: HolodexRepository,
    private val unifiedRepository: UnifiedVideoRepository,
    private val addItemsToQueueUseCase: AddItemsToQueueUseCase,
    private val playbackController: PlaybackController,
    private val paletteExtractor: PaletteExtractor,
```

```
    private val tokenManager: TokenManager
) : ContainerHost<PlaylistDetailsState, PlaylistDetailsSideEffect>, ViewModel() {

    companion object {
        const val PLAYLIST_ID_ARG = "playlistId"
        const val LIKED_SEGMENTS_PLAYLIST_ID = "-100"
        const val DOWNLOADS_PLAYLIST_ID = "-200"
    }

    val playlistId: String = savedStateHandle.get<String>(PLAYLIST_ID_ARG) ?: ""

    override val container: Container<PlaylistDetailsState, PlaylistDetailsSideEffect> = conta
        if (playlistId.isNotBlank()) {
            loadPlaylistDetails()
        } else {
            intent { reduce { state.copy(isLoading = false, error = "Invalid Playlist ID") } }
        }
    }

    fun loadPlaylistDetails() = intent {
        reduce { state.copy(isLoading = true, error = null) }
        val now = Instant.now().toString()

        try {
            var playlistEntity: PlaylistEntity? = null
            var unifiedItems: List<UnifiedDisplayItem> = emptyList()
            var rawItemsList: List<Any> = emptyList()
            var artworkUrl: String? = null

            // 1. Fetch global states for mapping
            val likedIds = unifiedRepository.observeLikedItemIds().first()
            val downloadedIds = unifiedRepository.observeDownloadedIds().first()

            when (playlistId) {
                LIKED_SEGMENTS_PLAYLIST_ID -> {
                    // Synthetic: Liked Segments
                    // We fetch UnifiedItems directly, no mapping needed
                    val segments = unifiedRepository.getFavorites().first().filter { it.isSegm
                    unifiedItems = segments
                    rawItemsList = segments // No raw entities available/needed here

                    playlistEntity = PlaylistEntity(
                        playlistId = LIKED_SEGMENTS_PLAYLIST_ID.toLong(),
                        name = application.getString(R.string.playlist_title_liked_segments),
                        description = application.getString(R.string.playlist_desc_liked_segme
                        createdAt = now, last_modified_at = now, serverId = null, owner = null
                    )
                    artworkUrl = segments.firstOrNull()?.artworkUrls?.firstOrNull()
                }
                DOWNLOADS_PLAYLIST_ID -> {
                    // Synthetic: Downloads
                    val downloads = unifiedRepository.getDownloads().first()
                    unifiedItems = downloads
                    rawItemsList = downloads
```

```
                playlistEntity = PlaylistEntity(
                    playlistId = DOWNLOADS_PLAYLIST_ID.toLong(),
                    name = application.getString(R.string.playlist_title_downloads),
                    description = application.getString(R.string.playlist_desc_downloads),
                    createdAt = now, last_modified_at = now, serverId = null, owner = null
                )
                artworkUrl = downloads.firstOrNull()?.artworkUrls?.firstOrNull()
            }
            else -> {
                val longId = playlistId.toLongOrNull()
                // Check if it's a local numeric ID (User Playlist)
                if (longId != null && longId > 0) {
                    // User Playlist (Local DB)
                    playlistEntity = holodexRepository.getPlaylistById(longId)
                    val playlistItems = holodexRepository.getItemsForPlaylist(longId).firs
                    rawItemsList = playlistItems // Keep entities for editing

                    unifiedItems = playlistItems.map { entity ->
                        entity.toUnifiedDisplayItem(
                            isDownloaded = downloadedIds.contains(entity.itemIdInPlaylist)
                            isLiked = likedIds.contains(entity.itemIdInPlaylist)
                        )
                    }
                    artworkUrl = unifiedItems.firstOrNull()?.artworkUrls?.firstOrNull()
                } else {
                    // Remote/System Playlist (API)
                    val isRadio = playlistId.startsWith(":artist") || playlistId.startsWit
                    val result = if (isRadio) holodexRepository.getRadioContent(playlistId

                    val fullPlaylist = result.getOrThrow()

                    // Generate metadata
                    val tempStub = PlaylistStub(
                        id = fullPlaylist.id, title = fullPlaylist.title, type = fullPlayl
                        description = fullPlaylist.description, artContext = null
                    )
                    val formattedTitle = PlaylistFormatter.getDisplayTitle(tempStub, appli
                    val formattedDescription = PlaylistFormatter.getDisplayDescription(tem

                    playlistEntity = PlaylistEntity(
                        playlistId = 0L, name = formattedTitle, description = formattedDes
                        createdAt = fullPlaylist.createdAt, last_modified_at = fullPlaylis
                        serverId = fullPlaylist.id, owner = null
                    )

                    val apiSongs = fullPlaylist.content ?: emptyList()
                    rawItemsList = apiSongs

                    unifiedItems = apiSongs.map { song ->
                        val videoShell = song.toVideoShell(playlistEntity.name ?: "")
                        // Determine statuses using composite key
                        val compositeId = "${song.videoId}_${song.start}"
                        song.toUnifiedDisplayItem(
                            parentVideo = videoShell,
                            isLiked = likedIds.contains(compositeId),
```

```
                            isDownloaded = downloadedIds.contains(compositeId)
                        )
                    }
                    artworkUrl = ArtworkResolver.getPlaylistArtworkUrl(tempStub) ?: unifie
                }
            }
        }

        // Extract Theme
        val theme = paletteExtractor.extractThemeFromUrl(artworkUrl, DynamicTheme.default(

        // Check Ownership
        val isOwned = playlistEntity?.owner != null && playlistEntity.owner.toString() ==

        reduce {
            state.copy(
                playlist = playlistEntity,
                items = unifiedItems,
                rawItems = rawItemsList,
                isLoading = false,
                dynamicTheme = theme,
                isPlaylistOwned = isOwned
            )
        }

    } catch (e: Exception) {
        Timber.e(e, "Failed to load playlist details")
        reduce { state.copy(isLoading = false, error = e.localizedMessage ?: "Unknown Erro
    }
}

fun togglePlaylistShuffleMode() = intent {
    reduce { state.copy(isShuffleActive = !state.isShuffleActive) }
}

fun playAllItemsInPlaylist() = intent {
    val isRadio = playlistId.startsWith(":")
    if (isRadio) {
        playbackController.loadRadio(playlistId)
    } else {
        if (state.items.isEmpty()) {
            postSideEffect(PlaylistDetailsSideEffect.ShowToast("Playlist is empty"))
            return@intent
        }
        val playbackItems = state.items.map { it.toPlaybackItem() }
        playbackController.loadAndPlay(playbackItems, 0)
        if (state.isShuffleActive) playbackController.toggleShuffle()
    }
}

fun playFromItem(tappedItem: UnifiedDisplayItem) = intent {
    if (state.items.isEmpty()) return@intent
    val index = state.items.indexOfFirst { it.playbackItemId == tappedItem.playbackItemId
    val playbackItems = state.items.map { it.toPlaybackItem() }
```

```
    playbackController.loadAndPlay(playbackItems, index)
    if (state.isShuffleActive) playbackController.toggleShuffle()
}

fun addAllToQueue() = intent {
    val items = state.items.map { it.toPlaybackItem() }
    if (items.isNotEmpty()) {
        addItemsToQueueUseCase(items)
        postSideEffect(PlaylistDetailsSideEffect.ShowToast("Added ${items.size} songs to q
    }
}

// --- Edit Mode Logic (User Playlists Only) ---

fun enterEditMode() = intent {
    val editableList = state.rawItems.filterIsInstance<PlaylistItemEntity>()
    if (editableList.isEmpty() && state.rawItems.isNotEmpty()) {
        // Should not happen if isPlaylistOwned check is correct
        postSideEffect(PlaylistDetailsSideEffect.ShowToast("Cannot edit this type of playl
        return@intent
    }
    reduce {
        state.copy(
            isEditMode = true,
            editablePlaylist = state.playlist?.copy(),
            editableItems = editableList
        )
    }
}

fun cancelEditMode() = intent {
    reduce { state.copy(isEditMode = false, editablePlaylist = null, editableItems = empty
}

fun updateDraftName(newName: String) = intent {
    reduce { state.copy(editablePlaylist = state.editablePlaylist?.copy(name = newName)) }
}

fun updateDraftDescription(desc: String) = intent {
    reduce { state.copy(editablePlaylist = state.editablePlaylist?.copy(description = desc
}

fun reorderItemInEditMode(from: Int, to: Int) = intent {
    val list = state.editableItems.toMutableList()
    val item = list.removeAt(from)
    list.add(to, item)

    // We must re-map the editable list to UnifiedDisplayItems so the UI updates nicely
    // To avoid re-fetching status, we assume the status in the main 'items' list is still
    // or we just map naively (status usually doesn't change during reorder).
    // For simplicity/speed, we'll re-fetch the statuses from the repositories current cac
    val likedIds = unifiedRepository.observeLikedItemIds().first()
    val downloadedIds = unifiedRepository.observeDownloadedIds().first()

    val unified = list.map { it.toUnifiedDisplayItem(downloadedIds.contains(it.itemIdInPla
```

```
        reduce { state.copy(editableItems = list, items = unified) }
    }

    fun removeItemInEditMode(item: UnifiedDisplayItem) = intent {
        // Remove from the Entity list
        val list = state.editableItems.filterNot { it.itemIdInPlaylist == item.playbackItemId

        // Re-map to Unified for UI
        val likedIds = unifiedRepository.observeLikedItemIds().first()
        val downloadedIds = unifiedRepository.observeDownloadedIds().first()
        val unified = list.map { it.toUnifiedDisplayItem(downloadedIds.contains(it.itemIdInPla

        reduce { state.copy(editableItems = list, items = unified) }
    }

    fun saveChanges() = intent {
        val original = state.playlist
        val draft = state.editablePlaylist
        val draftItems = state.editableItems

        if (draft == null || draft.name.isNullOrBlank()) {
            postSideEffect(PlaylistDetailsSideEffect.ShowToast("Playlist name cannot be empty"
            return@intent
        }

        // Calculate if content changed (simple ID comparison)
        val originalSyncedIds = state.rawItems.filterIsInstance<PlaylistItemEntity>().filter {
        val newSyncedIds = draftItems.filter { !it.isLocalOnly }.map { it.itemIdInPlaylist }

        val contentChanged = originalSyncedIds != newSyncedIds
        val metaChanged = original?.name != draft.name || original?.description != draft.descr

        val finalPlaylist = if (contentChanged || metaChanged) {
            draft.copy(syncStatus = SyncStatus.DIRTY, last_modified_at = Instant.now().toStrin
        } else {
            draft
        }

        try {
            holodexRepository.savePlaylistEdits(finalPlaylist, draftItems)
            cancelEditMode()
            loadPlaylistDetails() // Refresh
            postSideEffect(PlaylistDetailsSideEffect.ShowToast("Changes saved"))
        } catch (e: Exception) {
            postSideEffect(PlaylistDetailsSideEffect.ShowToast("Failed to save: ${e.message}")
        }
    }

    fun clearError() = intent { reduce { state.copy(error = null) } }
}

// File: java\com\example\holodex\viewmodel\PlaylistManagementViewModel.kt
package com.example.holodex.viewmodel
```

```
import android.app.Application
import android.widget.Toast
import androidx.annotation.OptIn
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import androidx.media3.common.util.UnstableApi
import com.example.holodex.R
import com.example.holodex.data.db.LikedItemType
import com.example.holodex.data.db.PlaylistEntity
import com.example.holodex.data.db.PlaylistItemEntity
import com.example.holodex.data.db.StarredPlaylistEntity
import com.example.holodex.data.db.SyncStatus
import com.example.holodex.data.model.discovery.PlaylistStub
import com.example.holodex.data.repository.HolodexRepository
import com.example.holodex.data.repository.UnifiedVideoRepository
import com.example.holodex.playback.domain.model.PlaybackItem
import com.example.holodex.util.PlaylistFormatter
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.SharingStarted
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.flow.combine
import kotlinx.coroutines.flow.stateIn
import kotlinx.coroutines.launch
import timber.log.Timber
import java.time.Instant
import javax.inject.Inject
import kotlin.math.absoluteValue

data class PendingPlaylistItemDetails(
    val videoId: String,
    val itemType: LikedItemType,
    val titleForDisplay: String,
    val artistForDisplay: String,
    val artworkForDisplay: String?,
    val songStartSeconds: Int? = null,
    val songEndSeconds: Int? = null,
    val isExternal: Boolean
)

@OptIn(UnstableApi::class)
@HiltViewModel
class PlaylistManagementViewModel @Inject constructor(
    private val application: Application,
    private val holodexRepository: HolodexRepository,
    private val unifiedRepository: UnifiedVideoRepository
) : ViewModel() {

    companion object {
        const val TAG = "PlaylistMgmtVM"
        const val LIKED_SEGMENTS_PLAYLIST_ID = -100L
        const val DOWNLOADS_PLAYLIST_ID = -200L
    }
```

```
private val userCreatedPlaylists: Flow<List<PlaylistEntity>> = holodexRepository.getAllPla
private val starredPlaylists: Flow<List<StarredPlaylistEntity>> = holodexRepository.getSta

// Migration: Use Unified Display Items instead of legacy DownloadedItemEntity
private val downloadsFlow: Flow<List<UnifiedDisplayItem>> = unifiedRepository.getDownloads

val allDisplayablePlaylists: StateFlow<List<PlaylistEntity>> =
    combine(
        userCreatedPlaylists,
        starredPlaylists,
        downloadsFlow,
    ) { userPlaylists, starred, downloads ->
        val syntheticPlaylists = mutableListOf<PlaylistEntity>()
        val now = Instant.now().toString()

        // 1. Synthetic "Liked Segments" Playlist
        syntheticPlaylists.add(
            PlaylistEntity(
                playlistId = LIKED_SEGMENTS_PLAYLIST_ID,
                name = application.getString(R.string.playlist_title_liked_segments),
                description = application.getString(R.string.playlist_desc_liked_segments)
                createdAt = now, last_modified_at = now, serverId = null, owner = null
            )
        )

        // 2. Synthetic "Downloads" Playlist (Only if downloads exist)
        if (downloads.any { it.isDownloaded }) {
            syntheticPlaylists.add(
                PlaylistEntity(
                    playlistId = DOWNLOADS_PLAYLIST_ID,
                    name = application.getString(R.string.playlist_title_downloads),
                    description = application.getString(R.string.playlist_desc_downloads),
                    createdAt = now, last_modified_at = now, serverId = null, owner = null
                )
            )
        }

        // 3. Starred Playlists (Merged with User Playlists)
        val starredAsDisplayable = starred.map { starredItem ->
            val userPlaylistMatch = userPlaylists.find { it.serverId == starredItem.playli
            if (userPlaylistMatch != null) {
                userPlaylistMatch
            } else {
                // Generate a temporary negative ID for display stability
                val uniqueNegativeId = ("starred_${starredItem.playlistId}".hashCode()).ab
                val tempStub = PlaylistStub(
                    id = starredItem.playlistId,
                    title = "Starred Playlist", type = "", artContext = null,
                    description = "ID: ${starredItem.playlistId}"
                )
                val formattedTitle = PlaylistFormatter.getDisplayTitle(tempStub, applicati
                val formattedDescription = PlaylistFormatter.getDisplayDescription(tempStu
                PlaylistEntity(
                    playlistId = uniqueNegativeId, serverId = starredItem.playlistId, name
```

```
                    description = formattedDescription, createdAt = now, last_modified_at
                )
            }
        }

        val starredServerIds = starred.map { it.playlistId }.toSet()
        val uniqueUserPlaylists = userPlaylists.filter { it.serverId !in starredServerIds

        syntheticPlaylists + uniqueUserPlaylists + starredAsDisplayable
    }.stateIn(
        scope = viewModelScope,
        started = SharingStarted.WhileSubscribed(5000L),
        initialValue = emptyList()
    )

// Used for the "Select Playlist" dialog (excludes synthetic lists)
val userPlaylists: StateFlow<List<PlaylistEntity>> = holodexRepository.getAllPlaylists()
    .stateIn(
        scope = viewModelScope,
        started = SharingStarted.WhileSubscribed(5000L),
        initialValue = emptyList()
    )

private val _showCreatePlaylistDialog = MutableStateFlow(false)
val showCreatePlaylistDialog: StateFlow<Boolean> = _showCreatePlaylistDialog.asStateFlow()

private val _pendingItemForPlaylist = MutableStateFlow<PendingPlaylistItemDetails?>(null)

private val _showSelectPlaylistDialog = MutableStateFlow(false)
val showSelectPlaylistDialog: StateFlow<Boolean> = _showSelectPlaylistDialog.asStateFlow()

fun openCreatePlaylistDialog() {
    _showCreatePlaylistDialog.value = true
}

fun closeCreatePlaylistDialog() {
    _showCreatePlaylistDialog.value = false
}

fun cancelAddToPlaylistFlow() {
    _showSelectPlaylistDialog.value = false
    _showCreatePlaylistDialog.value = false
    _pendingItemForPlaylist.value = null
}

fun prepareItemForPlaylistAddition(item: UnifiedDisplayItem) {
    _pendingItemForPlaylist.value = PendingPlaylistItemDetails(
        videoId = item.videoId,
        itemType = item.itemTypeForPlaylist,
        titleForDisplay = item.title,
        artistForDisplay = item.artistText,
        artworkForDisplay = item.artworkUrls.firstOrNull(),
        songStartSeconds = item.songStartSec,
        songEndSeconds = item.songEndSec,
        isExternal = item.isExternal
```

```
    )
    _showSelectPlaylistDialog.value = true
    Timber.d("$TAG: Preparing item for playlist addition: ${_pendingItemForPlaylist.value}
}

fun addItemToExistingPlaylist(playlist: PlaylistEntity) {
    val pendingItem = _pendingItemForPlaylist.value ?: return cancelAddToPlaylistFlow()

    if (playlist.playlistId <= 0 && playlist.serverId == null) {
        Toast.makeText(application, "Cannot add items to this type of playlist.", Toast.LE
        return cancelAddToPlaylistFlow()
    }

    viewModelScope.launch {
        try {
            if (!pendingItem.isExternal) {
                val updatedPlaylist = playlist.copy(
                    syncStatus = SyncStatus.DIRTY,
                    last_modified_at = Instant.now().toString()
                )
                holodexRepository.playlistDao.updatePlaylist(updatedPlaylist)
            }

            val lastOrder = holodexRepository.getLastItemOrderInPlaylist(playlist.playlist
            val newOrder = (lastOrder ?: -1) + 1

            val playlistItemEntity = PlaylistItemEntity(
                playlistOwnerId = playlist.playlistId,
                // FIX: Construct IDs manually or move helper to PlaylistItemEntity.
                // For now, simple manual construction is safest:
                itemIdInPlaylist = if (pendingItem.itemType == LikedItemType.SONG_SEGMENT
                    "${pendingItem.videoId}_${pendingItem.songStartSeconds}"
                } else {
                    pendingItem.videoId
                },
                videoIdForItem = pendingItem.videoId,
                itemTypeInPlaylist = pendingItem.itemType,
                songStartSecondsPlaylist = pendingItem.songStartSeconds,
                songEndSecondsPlaylist = pendingItem.songEndSeconds,
                songNamePlaylist = if (pendingItem.itemType == LikedItemType.SONG_SEGMENT)
                songArtistTextPlaylist = pendingItem.artistForDisplay,
                songArtworkUrlPlaylist = pendingItem.artworkForDisplay,
                itemOrder = newOrder,
                isLocalOnly = pendingItem.isExternal,
                syncStatus = SyncStatus.DIRTY
            )

            holodexRepository.addPlaylistItem(playlistItemEntity)
            Toast.makeText(application, "'${pendingItem.titleForDisplay}' added to ${playl
        } catch (e: Exception) {
            Timber.e(e, "Failed to add item to playlist ${playlist.name}")
            Toast.makeText(application, "Failed to add to playlist: ${e.localizedMessage}"
        } finally {
            cancelAddToPlaylistFlow()
        }
```

```
    }
}

fun handleCreateNewPlaylistFromSelectionDialog() {
    _showSelectPlaylistDialog.value = false
    _showCreatePlaylistDialog.value = true
}

fun confirmCreatePlaylist(name: String, description: String?) {
    val playlistName = name.trim()
    if (playlistName.isBlank()) {
        Toast.makeText(application, "Playlist name cannot be empty.", Toast.LENGTH_SHORT).
        return
    }

    val currentPendingItem = _pendingItemForPlaylist.value
    viewModelScope.launch {
        try {
            val newPlaylistId = holodexRepository.createNewPlaylist(playlistName, descript
            Toast.makeText(application, "Playlist '$playlistName' created", Toast.LENGTH_S
            _showCreatePlaylistDialog.value = false

            if (currentPendingItem != null) {
                addItemToExistingPlaylist(
                    PlaylistEntity(
                        playlistId = newPlaylistId, name = playlistName, description = des
                        createdAt = Instant.now().toString(), last_modified_at = Instant.n
                        serverId = null, owner = null
                    )
                )
            }
        } catch (e: Exception) {
            Timber.e(e, "Failed to create playlist '$playlistName'")
            Toast.makeText(application, "Failed to create playlist: ${e.localizedMessage}"
        }
    }
}
fun prepareItemForPlaylistAdditionFromPlaybackItem(item: PlaybackItem) {
    _pendingItemForPlaylist.value = PendingPlaylistItemDetails(
        videoId = item.videoId,
        itemType = if (item.songId != null) LikedItemType.SONG_SEGMENT else LikedItemType.
        titleForDisplay = item.title,
        artistForDisplay = item.artistText,
        artworkForDisplay = item.artworkUri,
        songStartSeconds = item.clipStartSec?.toInt(),
        songEndSeconds = item.clipEndSec?.toInt(),
        isExternal = item.isExternal
    )
    _showSelectPlaylistDialog.value = true
}
fun deletePlaylist(playlist: PlaylistEntity) {
    val idToDelete = playlist.playlistId
    // This logic handles starred playlists (negative ID, has serverId) and user playlists
    if (idToDelete == 0L || (idToDelete < 0 && playlist.serverId == null)) {
        Toast.makeText(application, "Cannot delete synthetic playlists.", Toast.LENGTH_SHO
```

```
            return
        }
        viewModelScope.launch {
            try {
                holodexRepository.deletePlaylist(idToDelete)
                Toast.makeText(application, "Playlist deleted", Toast.LENGTH_SHORT).show()
            } catch (e: Exception) {
                Timber.e(e, "Failed to delete playlist ID: $idToDelete")
                Toast.makeText(application, "Failed to delete playlist: ${e.localizedMessage}"
            }
        }
    }
}

// File: java\com\example\holodex\viewmodel\SettingsViewModel.kt
package com.example.holodex.viewmodel

import android.app.Application
import android.content.Intent
import android.content.SharedPreferences
import android.net.Uri
import androidx.core.content.edit
import androidx.core.net.toUri
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import androidx.media3.common.util.UnstableApi
import androidx.work.Constraints
import androidx.work.ExistingWorkPolicy
import androidx.work.NetworkType
import androidx.work.OneTimeWorkRequestBuilder
import androidx.work.WorkManager
import com.example.holodex.MyApp
import com.example.holodex.background.SyncWorker
import com.example.holodex.data.AppPreferenceConstants
import com.example.holodex.data.ThemePreference
import com.example.holodex.data.download.LegacyDownloadScanner
import com.example.holodex.data.repository.UserPreferencesRepository
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.launch
import org.orbitmvi.orbit.Container
import org.orbitmvi.orbit.ContainerHost
import org.orbitmvi.orbit.viewmodel.container
import timber.log.Timber
import javax.inject.Inject

// --- States ---
sealed class ApiKeySaveResult {
    object Success : ApiKeySaveResult()
    object Empty : ApiKeySaveResult()
    data class Error(val message: String) : ApiKeySaveResult()
    object Idle : ApiKeySaveResult()
}

sealed class ScanStatus {
    object Idle : ScanStatus()
```

```
    object Scanning : ScanStatus()
    data class Complete(val importedCount: Int) : ScanStatus()
    data class Error(val message: String) : ScanStatus()
}

// Single Consolidated State
data class SettingsState(
    val currentApiKey: String = "",
    val apiKeySaveResult: ApiKeySaveResult = ApiKeySaveResult.Idle,
    val cacheClearStatus: String? = null,
    val scanStatus: ScanStatus = ScanStatus.Idle,
    val transientMessage: String? = null,

    // Preferences
    val currentTheme: String = ThemePreference.SYSTEM,
    val currentImageQuality: String = AppPreferenceConstants.IMAGE_QUALITY_AUTO,
    val currentAudioQuality: String = AppPreferenceConstants.AUDIO_QUALITY_BEST,
    val currentListLoadingConfig: String = AppPreferenceConstants.LIST_LOADING_NORMAL,
    val currentBufferingStrategy: String = AppPreferenceConstants.BUFFERING_STRATEGY_AGGRESSIV
    val downloadLocation: String = "",

    // DataStore values
    val autoplayEnabled: Boolean = true,
    val shuffleOnPlayStartEnabled: Boolean = false
)

// Side Effects
sealed class SettingsSideEffect {
    data class ShowToast(val message: String) : SettingsSideEffect()
}

@HiltViewModel
class SettingsViewModel @Inject constructor(
    private val application: Application,
    private val sharedPreferences: SharedPreferences,
    private val userPreferencesRepository: UserPreferencesRepository,
    private val workManager: WorkManager,
    private val legacyDownloadScanner: LegacyDownloadScanner
) : ContainerHost<SettingsState, SettingsSideEffect>, ViewModel() {

    override val container: Container<SettingsState, SettingsSideEffect> = container(SettingsS

    init {
        // Initialize State from SharedPreferences
        val apiKey = sharedPreferences.getString("API_KEY", "") ?: ""
        val theme = sharedPreferences.getString(ThemePreference.KEY, ThemePreference.SYSTEM) ?
        val imgQuality = sharedPreferences.getString(AppPreferenceConstants.PREF_IMAGE_QUALITY
        val audioQuality = sharedPreferences.getString(AppPreferenceConstants.PREF_AUDIO_QUALI
        val listLoading = sharedPreferences.getString(AppPreferenceConstants.PREF_LIST_LOADING
        val buffering = sharedPreferences.getString(AppPreferenceConstants.PREF_BUFFERING_STRA
        val downloadLoc = sharedPreferences.getString(AppPreferenceConstants.PREF_DOWNLOAD_LOC

        intent {
            reduce {
                state.copy(
```

```
                currentApiKey = apiKey,
                currentTheme = theme,
                currentImageQuality = imgQuality,
                currentAudioQuality = audioQuality,
                currentListLoadingConfig = listLoading,
                currentBufferingStrategy = buffering,
                downloadLocation = downloadLoc
            )
        }
    }

    // Observe DataStore flows
    viewModelScope.launch {
        userPreferencesRepository.autoplayEnabled.collect { enabled ->
            intent {
                reduce { state.copy(autoplayEnabled = enabled) }
            }
        }
    }
    viewModelScope.launch {
        userPreferencesRepository.shuffleOnPlayStartEnabled.collect { enabled ->
            intent {
                reduce { state.copy(shuffleOnPlayStartEnabled = enabled) }
            }
        }
    }
}

// --- Actions ---
fun enqueueWork(request: androidx.work.WorkRequest) {
    workManager.enqueue(request)
}
fun saveApiKey(key: String) = intent {
    val trimmedKey = key.trim()
    if (trimmedKey.isBlank()) {
        reduce { state.copy(apiKeySaveResult = ApiKeySaveResult.Empty) }
        return@intent
    }
    runCatching {
        sharedPreferences.edit {
            putString("API_KEY", trimmedKey)
        }
    }.onSuccess {
        reduce { state.copy(currentApiKey = trimmedKey, apiKeySaveResult = ApiKeySaveResul
    }.onFailure {
        reduce { state.copy(apiKeySaveResult = ApiKeySaveResult.Error("Failed to save API
    }
}

fun resetApiKeySaveResult() = intent {
    reduce { state.copy(apiKeySaveResult = ApiKeySaveResult.Idle) }
}

fun runLegacyFileScan() = intent {
    if (state.scanStatus is ScanStatus.Scanning) return@intent
```

```
        reduce { state.copy(scanStatus = ScanStatus.Scanning) }

        runCatching {
            legacyDownloadScanner.scanAndImportLegacyDownloads()
        }.onSuccess { count ->
            reduce { state.copy(scanStatus = ScanStatus.Complete(count)) }
        }.onFailure { e ->
            reduce { state.copy(scanStatus = ScanStatus.Error("Scan failed: ${e.localizedMessa
        }
    }

    fun resetScanStatus() = intent {
        reduce { state.copy(scanStatus = ScanStatus.Idle) }
    }

    fun triggerManualSync() = intent {
        postSideEffect(SettingsSideEffect.ShowToast("Syncing account data..."))
        val constraints = Constraints.Builder().setRequiredNetworkType(NetworkType.CONNECTED).
        val request = OneTimeWorkRequestBuilder<SyncWorker>().setConstraints(constraints).buil
        workManager.enqueueUniqueWork("ManualSync", ExistingWorkPolicy.REPLACE, request)
    }

    // --- Preferences Setters ---

    fun setThemePreference(theme: String) = intent {
        sharedPreferences.edit {
            putString(ThemePreference.KEY, theme)
        }
        reduce { state.copy(currentTheme = theme) }
    }

    fun setImageQualityPreference(quality: String) = intent {
        sharedPreferences.edit {
            putString(AppPreferenceConstants.PREF_IMAGE_QUALITY, quality)
        }
        reduce { state.copy(currentImageQuality = quality) }
    }

    fun setAudioQualityPreference(quality: String) = intent {
        sharedPreferences.edit {
            putString(AppPreferenceConstants.PREF_AUDIO_QUALITY, quality)
        }
        reduce { state.copy(currentAudioQuality = quality) }
    }

    fun setListLoadingConfigPreference(config: String) = intent {
        sharedPreferences.edit {
            putString(AppPreferenceConstants.PREF_LIST_LOADING_CONFIG, config)
        }
        reduce { state.copy(currentListLoadingConfig = config) }
    }

    fun setBufferingStrategyPreference(strategy: String) = intent {
        sharedPreferences.edit {
            putString(AppPreferenceConstants.PREF_BUFFERING_STRATEGY, strategy)
```

```
    }
    reduce { state.copy(currentBufferingStrategy = strategy) }
}

fun saveDownloadLocation(uri: Uri) = intent {
    runCatching {
        val contentResolver = application.contentResolver
        val flags = Intent.FLAG_GRANT_READ_URI_PERMISSION or Intent.FLAG_GRANT_WRITE_URI_P
        contentResolver.takePersistableUriPermission(uri, flags)

        val uriString = uri.toString()
        sharedPreferences.edit {
            putString(AppPreferenceConstants.PREF_DOWNLOAD_LOCATION, uriString)
        }
        reduce { state.copy(downloadLocation = uriString) }
    }.onFailure {
        Timber.e(it, "Failed to save download location")
        postSideEffect(SettingsSideEffect.ShowToast("Failed to save folder permission"))
    }
}

fun clearDownloadLocation() = intent {
    val currentUri = state.downloadLocation
    if (currentUri.isNotEmpty()) {
        try {
            val contentResolver = application.contentResolver
            val flags = Intent.FLAG_GRANT_READ_URI_PERMISSION or Intent.FLAG_GRANT_WRITE_U
            contentResolver.releasePersistableUriPermission(currentUri.toUri(), flags)
        } catch (e: Exception) {
            Timber.e(e, "Failed to release permission")
        }
    }
    sharedPreferences.edit {
        remove(AppPreferenceConstants.PREF_DOWNLOAD_LOCATION)
    }
    reduce { state.copy(downloadLocation = "") }
}

fun setAutoplayNextVideoEnabled(enabled: Boolean) = intent {
    userPreferencesRepository.setAutoplayEnabled(enabled)
    // UI updates via flow observation in init
}

fun setShuffleOnPlayStartEnabled(enabled: Boolean) = intent {
    userPreferencesRepository.setShuffleOnPlayStartEnabled(enabled)
    // UI updates via flow observation in init
}

@UnstableApi
fun clearAllApplicationCaches() = intent {
    reduce { state.copy(cacheClearStatus = "Clearing caches...") }
    (application as? MyApp)?.clearAllAppCachesOnDemand { success ->
        // Call intent directly - it's not a suspend function
        updateCacheStatus(success)
    }
```

```
    }

    private fun updateCacheStatus(success: Boolean) = intent {
        reduce {
            state.copy(cacheClearStatus = if (success) "Success" else "Failed")
        }
    }

    fun resetCacheClearStatus() = intent {
        reduce { state.copy(cacheClearStatus = null) }
    }
}

// File: java\com\example\holodex\viewmodel\SharedViewModelTypes.kt
package com.example.holodex.viewmodel

import androidx.compose.runtime.MutableState
import androidx.compose.runtime.mutableStateOf
import kotlinx.coroutines.Job

data class SubOrgHeader(val name: String)

enum class MusicCategoryType {
    LATEST,
    UPCOMING_MUSIC,
    SEARCH,
    FAVORITES,
    LIKED_SEGMENTS,
    TRENDING,
    RECENT_STREAMS,
    COMMUNITY_PLAYLISTS,
    ARTIST_RADIOS,
    SYSTEM_PLAYLISTS,
    DISCOVER_CHANNELS
}

// Helper for manual pagination (used by FullListViewModel)
class ListStateHolder<T> {
    val items: MutableState<List<T>> = mutableStateOf(emptyList())
    val isLoadingInitial: MutableState<Boolean> = mutableStateOf(false)
    val isLoadingMore: MutableState<Boolean> = mutableStateOf(false)
    val endOfList: MutableState<Boolean> = mutableStateOf(false)
    var currentOffset: Int = 0
    var job: Job? = null
}

// File: java\com\example\holodex\viewmodel\UnifiedDisplayItem.kt
package com.example.holodex.viewmodel

import androidx.compose.runtime.Immutable
import com.example.holodex.data.db.LikedItemType

@Immutable
data class UnifiedDisplayItem(
    val stableId: String,
```

```
    val playbackItemId: String,
    val videoId: String,
    val channelId: String,

    val title: String,
    val artistText: String,
    val artworkUrls: List<String>,
    val durationText: String,

    val songCount: Int?,
    val isDownloaded: Boolean,

    val downloadStatus: String?, // "DOWNLOADING", "PAUSED", "COMPLETED", "FAILED"

    val localFilePath: String?,

    val isSegment: Boolean,
    val isLiked: Boolean,

    val itemTypeForPlaylist: LikedItemType,
    val songStartSec: Int?,
    val songEndSec: Int?,
    val originalArtist: String?,
    val isExternal: Boolean
)

// File: java\com\example\holodex\viewmodel\VideoDetailsViewModel.kt
// File: java/com/example/holodex/viewmodel/VideoDetailsViewModel.kt
package com.example.holodex.viewmodel

import androidx.compose.ui.graphics.Color
import androidx.lifecycle.SavedStateHandle
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import androidx.media3.common.util.UnstableApi
import com.example.holodex.data.download.NoDownloadLocationException
import com.example.holodex.data.model.HolodexSong
import com.example.holodex.data.model.HolodexVideoItem
import com.example.holodex.data.repository.DownloadRepository
import com.example.holodex.data.repository.HolodexRepository
import com.example.holodex.data.repository.UnifiedVideoRepository
import com.example.holodex.playback.domain.model.PlaybackItem
import com.example.holodex.playback.domain.usecase.AddOrFetchAndAddUseCase
import com.example.holodex.playback.player.PlaybackController
import com.example.holodex.util.DynamicTheme
import com.example.holodex.util.PaletteExtractor
import com.example.holodex.util.ThumbnailQuality
import com.example.holodex.util.getYouTubeThumbnailUrl
import com.example.holodex.viewmodel.mappers.toPlaybackItem
import com.example.holodex.viewmodel.mappers.toUnifiedDisplayItem
import com.example.holodex.viewmodel.mappers.toVirtualSegmentUnifiedDisplayItem
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.collections.immutable.ImmutableList
import kotlinx.collections.immutable.persistentListOf
import kotlinx.collections.immutable.toImmutableList
```

```
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.flow.firstOrNull
import kotlinx.coroutines.launch
import timber.log.Timber
import javax.inject.Inject

@UnstableApi
@HiltViewModel
class VideoDetailsViewModel @Inject constructor(
    private val savedStateHandle: SavedStateHandle,
    private val holodexRepository: HolodexRepository,
    private val downloadRepository: DownloadRepository,
    private val unifiedRepository: UnifiedVideoRepository, // <--- ADDED THIS
    private val addOrFetchAndAddUseCase: AddOrFetchAndAddUseCase,
    private val playbackController: PlaybackController,
    private val paletteExtractor: PaletteExtractor
) : ViewModel() {

    companion object {
        const val VIDEO_ID_ARG = "videoId"
        private const val TAG = "VideoDetailsVM"
    }

    val videoId: String = savedStateHandle.get<String>(VIDEO_ID_ARG) ?: ""

    private val _videoDetails = MutableStateFlow<HolodexVideoItem?>(null)
    val videoDetails: StateFlow<HolodexVideoItem?> = _videoDetails.asStateFlow()

    private val _isLoading = MutableStateFlow(true)
    val isLoading: StateFlow<Boolean> = _isLoading.asStateFlow()

    private val _error = MutableStateFlow<String?>(null)
    val error: StateFlow<String?> = _error.asStateFlow()

    private val _transientMessage = MutableStateFlow<String?>(null)
    val transientMessage: StateFlow<String?> = _transientMessage.asStateFlow()

    private val _unifiedSongItems = MutableStateFlow<ImmutableList<UnifiedDisplayItem>>(persis
    val unifiedSongItems: StateFlow<ImmutableList<UnifiedDisplayItem>> = _unifiedSongItems.asS

    private val _dynamicTheme = MutableStateFlow(DynamicTheme.default(Color.Black, Color.White
    val dynamicTheme: StateFlow<DynamicTheme> = _dynamicTheme.asStateFlow()

    fun initialize(videoListViewModel: VideoListViewModel) {
        if (videoId.isNotBlank()) {
            viewModelScope.launch {
                _isLoading.value = true
                val prefetchedItem = videoListViewModel.videoItemForDetailScreen

                val isPrefetchedItemComplete = prefetchedItem != null &&
                        prefetchedItem.id == videoId &&
                        (prefetchedItem.channel.org == "External" || prefetchedItem.songs != n
```

```
            if (isPrefetchedItemComplete) {
                processItem(prefetchedItem!!)
            } else {
                holodexRepository.getVideoWithSongs(videoId, forceRefresh = false)
                    .onSuccess { processItem(it) }
                    .onFailure { _error.value = "Failed to load video details: ${it.locali
            }
            _isLoading.value = false
        }
    } else {
        _isLoading.value = false
        _error.value = "Video ID is missing."
    }
}

private fun processItem(videoItem: HolodexVideoItem) {
    _videoDetails.value = videoItem
    viewModelScope.launch {
        updateTheme(videoItem.id)

        val isEffectivelySegmentless = videoItem.channel.org == "External" || videoItem.so

        if (isEffectivelySegmentless) {
            // Use firstOrNull to avoid crash if flow is empty
            val likedIds = unifiedRepository.observeLikedItemIds().firstOrNull() ?: emptyS
            val isLiked = likedIds.contains(videoItem.id)

            val virtualSegment = videoItem.toVirtualSegmentUnifiedDisplayItem(isLiked, isD
            _unifiedSongItems.value = persistentListOf(virtualSegment)
        } else {
            val songs = videoItem.songs!!
            // Use firstOrNull to avoid crash
            val likedIds = unifiedRepository.observeLikedItemIds().firstOrNull() ?: emptyS

            // *** FIX: Use Unified Repository to get downloads safely ***
            val downloadedIds = unifiedRepository.observeDownloadedIds().firstOrNull() ?:

            val unifiedItems = songs.sortedBy { it.start }.map { song ->
                song.toUnifiedDisplayItem(
                    parentVideo = videoItem,
                    isLiked = likedIds.contains("${videoItem.id}_${song.start}"),
                    isDownloaded = downloadedIds.contains("${videoItem.id}_${song.start}")
                )
            }
            _unifiedSongItems.value = unifiedItems.toImmutableList()
        }
    }
}

private suspend fun updateTheme(videoId: String) {
    val artworkUrl = getYouTubeThumbnailUrl(videoId, ThumbnailQuality.MAX).firstOrNull()
    _dynamicTheme.value = paletteExtractor.extractThemeFromUrl(
        artworkUrl,
        DynamicTheme.default(Color.Black, Color.White)
    )
```

```
}

fun playAllSegments() { playSegment(0) }

fun playSegment(startIndex: Int) {
    viewModelScope.launch {
        val itemsToPlay = _unifiedSongItems.value.map { it.toPlaybackItem() }
        if (startIndex in itemsToPlay.indices) {
            playbackController.loadAndPlay(itemsToPlay, startIndex)
        } else {
            _error.value = "Invalid song index."
        }
    }
}

fun addAllSegmentsToQueue() {
    viewModelScope.launch {
        val itemsToAdd = _unifiedSongItems.value
        if (itemsToAdd.isNotEmpty()) {
            addOrFetchAndAddUseCase(itemsToAdd.first().toPlaybackItem().copy(songId = null
                .onSuccess { message -> _transientMessage.value = message }
                .onFailure { error -> _error.value = "Failed to add to queue: ${error.loca
        }
    }
}

fun downloadAllSegments() {
    val video = _videoDetails.value
    if (video == null || video.songs.isNullOrEmpty()) {
        _error.value = "No segments available to download."
        return
    }
    viewModelScope.launch {
        try {
            _transientMessage.value = "Queueing ${video.songs.size} songs for download..."
            video.songs.forEach { song ->
                downloadRepository.startDownload(video, song)
            }
        } catch (e: Exception) {
            Timber.e(e, "A general error occurred during bulk download initiation.")
            _error.value = "Could not start downloads: An unknown error occurred."
        }
    }
}

fun requestDownloadForSongFromPlaybackItem(item: PlaybackItem) {
    viewModelScope.launch {
        val videoResult = holodexRepository.getVideoWithSongs(item.videoId, false).getOrNu
        if (videoResult == null) {
            _error.value = "Could not find video details to start download."
            return@launch
        }
        val songToDownload = videoResult.songs?.find { it.start.toLong() == item.clipStart
        if (songToDownload == null) {
            _error.value = "Could not find matching song segment to download."
```

```
                return@launch
            }
            requestDownloadForSong(videoResult, songToDownload)
        }
    }

    fun requestDownloadForSong(videoItem: HolodexVideoItem, song: HolodexSong) {
        viewModelScope.launch {
            try {
                downloadRepository.startDownload(videoItem, song)
                _transientMessage.value = "Added '${song.name}' to download queue."
            } catch (e: NoDownloadLocationException) {
                _error.value = e.message
            } catch (e: Exception) {
                _error.value = "Could not start download: An unknown error occurred."
            }
        }
    }

    fun clearError() {
        _error.value = null
    }

    fun clearTransientMessage() {
        _transientMessage.value = null
    }
}

// File: java\com\example\holodex\viewmodel\VideoListViewModel.kt
package com.example.holodex.viewmodel

import android.content.SharedPreferences
import androidx.core.content.edit
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import androidx.media3.common.util.UnstableApi
import com.example.holodex.data.cache.BrowseCacheKey
import com.example.holodex.data.cache.SearchCacheKey
import com.example.holodex.data.model.HolodexVideoItem
import com.example.holodex.data.repository.DownloadRepository
import com.example.holodex.data.repository.HolodexRepository
import com.example.holodex.data.repository.SearchHistoryRepository
import com.example.holodex.data.repository.UnifiedVideoRepository
import com.example.holodex.playback.domain.model.PlaybackItem
import com.example.holodex.playback.domain.usecase.AddOrFetchAndAddUseCase
import com.example.holodex.playback.player.PlaybackController
import com.example.holodex.util.extractVideoIdFromQuery
import com.example.holodex.viewmodel.autoplay.ContinuationManager
import com.example.holodex.viewmodel.mappers.toUnifiedDisplayItem
import com.example.holodex.viewmodel.state.BrowseFilterState
import com.example.holodex.viewmodel.state.ViewTypePreset
import com.google.gson.Gson
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.collections.immutable.ImmutableList
import kotlinx.collections.immutable.persistentListOf
```

```
import kotlinx.collections.immutable.toImmutableList
import kotlinx.coroutines.flow.first
import kotlinx.coroutines.launch
import org.orbitmvi.orbit.ContainerHost
import org.orbitmvi.orbit.viewmodel.container
import javax.inject.Inject

// State & SideEffect remain unchanged...
data class VideoListState(
    val browseItems: ImmutableList<UnifiedDisplayItem> = persistentListOf(),
    val browseIsLoadingInitial: Boolean = false,
    val browseIsLoadingMore: Boolean = false,
    val browseIsRefreshing: Boolean = false,
    val browseEndOfList: Boolean = false,
    val browseCurrentOffset: Int = 0,

    val searchItems: ImmutableList<UnifiedDisplayItem> = persistentListOf(),
    val searchIsLoadingInitial: Boolean = false,
    val searchIsLoadingMore: Boolean = false,
    val searchEndOfList: Boolean = false,
    val searchCurrentOffset: Int = 0,

    val activeContextType: MusicCategoryType = MusicCategoryType.LATEST,
    val isSearchActive: Boolean = false,
    val currentSearchQuery: String = "",
    val activeSearchSource: String = "Holodex",
    val browseFilterState: BrowseFilterState,
    val selectedOrganization: String = "Nijisanji",
    val availableOrganizations: List<Pair<String, String?>> = emptyList(),
    val searchHistory: List<String> = emptyList()
)

sealed class VideoListSideEffect {
    data class ShowToast(val message: String) : VideoListSideEffect()
    data class NavigateTo(val destination: VideoListViewModel.NavigationDestination) :
        VideoListSideEffect()
}

@UnstableApi
@HiltViewModel
class VideoListViewModel @Inject constructor(
    private val holodexRepository: HolodexRepository,
    private val unifiedRepository: UnifiedVideoRepository,
    private val sharedPreferences: SharedPreferences,
    private val searchHistoryRepository: SearchHistoryRepository,
    private val continuationManager: ContinuationManager,
    private val playbackController: PlaybackController,
    private val addOrFetchAndAddUseCase: AddOrFetchAndAddUseCase
) : ContainerHost<VideoListState, VideoListSideEffect>, ViewModel() {

    companion object {
        const val TAG = "VideoListViewModel"
        const val PREF_LAST_SELECTED_ORG = "last_selected_org"
        const val PREF_LAST_CATEGORY_TYPE = "last_category_type"
        const val PREF_LAST_SEARCH_QUERY = "last_search_query"
```

```
    const val PREF_LAST_BROWSE_FILTERS = "last_browse_filters_v1"
    const val CHANNEL_ID_SEARCH_PREFIX = "channel:"
    private const val PAGE_SIZE = 50
}

sealed class NavigationDestination {
    data class VideoDetails(val videoId: String) : NavigationDestination()
    object HomeScreenWithSearch : NavigationDestination()
}

var videoItemForDetailScreen: HolodexVideoItem? = null
    private set

override val container = container<VideoListState, VideoListSideEffect>(
    VideoListState(
        browseFilterState = loadLastBrowseFilters(),
        currentSearchQuery = loadLastSearchQuery(),
        activeContextType = loadLastActiveListContextType(),
        selectedOrganization = sharedPreferences.getString(PREF_LAST_SELECTED_ORG, "Nijisa
            ?: "Nijisanji"
    )
) {
    intent {
        viewModelScope.launch {
            holodexRepository.availableOrganizations.collect { orgs ->
                intent { reduce { state.copy(availableOrganizations = orgs) } }
            }
        }
        viewModelScope.launch {
            searchHistoryRepository.loadSearchHistory()
            searchHistoryRepository.searchHistory.collect { history ->
                intent { reduce { state.copy(searchHistory = history) } }
            }
        }
    }
    initializeAndFetch()
}

fun initializeAndFetch() = intent {
    if (state.browseItems.isEmpty() && state.searchItems.isEmpty()) {
        fetchCurrentContextData(isInitial = true, isRefresh = false)
    }
}

private fun fetchCurrentContextData(isInitial: Boolean, isRefresh: Boolean) = intent {
    if (state.activeContextType == MusicCategoryType.SEARCH) {
        fetchSearchResultsInternal(isInitial, isRefresh)
    } else {
        fetchBrowseItemsInternal(isInitial, isRefresh)
    }
}

private fun fetchBrowseItemsInternal(isInitial: Boolean, isRefresh: Boolean) = intent {
    if (!isRefresh && !isInitial && (state.browseIsLoadingMore || state.browseEndOfList))
```

```
        reduce {
            state.copy(
                browseIsLoadingInitial = isInitial,
                browseIsLoadingMore = !isInitial && !isRefresh,
                browseIsRefreshing = isRefresh
            )
        }

        val offset = if (isInitial || isRefresh) 0 else state.browseCurrentOffset
        val filters = state.browseFilterState

        val result: Result<List<HolodexVideoItem>> = runCatching {
            if (filters.selectedOrganization == "Favorites") {
                val favChannels = unifiedRepository.getFavoriteChannels().first()
                if (favChannels.isEmpty()) return@runCatching emptyList<HolodexVideoItem>()

                val holodexResults = holodexRepository.getFavoritesFeed(favChannels, filters,
                holodexResults
            } else {
                val key = BrowseCacheKey(filters, offset)
                holodexRepository.fetchBrowseList(key, isRefresh).getOrThrow().data
            }
        }

        val likedIds = unifiedRepository.observeLikedItemIds().first()
        val downloadedIds = try { unifiedRepository.getDownloadedIdsSnapshot() } catch (e: Exc

        result.onSuccess { rawItems ->
            val unifiedItems =
                rawItems.map { it.toUnifiedDisplayItem(likedIds.contains(it.id), downloadedIds
            reduce {
                val currentList = if (isInitial || isRefresh) emptyList() else state.browseIte
                val newItemsUnique =
                    if (isInitial || isRefresh) unifiedItems else unifiedItems.filter { newIte

                state.copy(
                    browseItems = (currentList + newItemsUnique).toImmutableList(),
                    browseCurrentOffset = offset + rawItems.size,
                    browseEndOfList = rawItems.isEmpty() || rawItems.size < PAGE_SIZE,
                    browseIsLoadingInitial = false,
                    browseIsLoadingMore = false,
                    browseIsRefreshing = false
                )
            }

            if (unifiedItems.isNotEmpty() && state.activeContextType != MusicCategoryType.SEAR
                continuationManager.setAutoplayContext(unifiedItems)
            }
        }.onFailure {
            postSideEffect(VideoListSideEffect.ShowToast("Failed to load content"))
            reduce {
                state.copy(
                    browseIsLoadingInitial = false,
                    browseIsLoadingMore = false,
                    browseIsRefreshing = false
```

```
            )
        }
    }
}

private fun fetchSearchResultsInternal(isInitial: Boolean, isRefresh: Boolean) = intent {
    val query = state.currentSearchQuery
    if (query.isBlank()) return@intent

    if (!isRefresh && !isInitial && (state.searchIsLoadingMore || state.searchEndOfList))

    reduce {
        state.copy(
            searchIsLoadingInitial = isInitial,
            searchIsLoadingMore = !isInitial && !isRefresh
        )
    }

    val offset = if (isInitial || isRefresh) 0 else state.searchCurrentOffset

    val result = runCatching {
        if (state.activeSearchSource == "My Channels") {
            val channelIds = unifiedRepository.getFavoriteChannelIds().first()
            if (channelIds.isEmpty()) emptyList()
            else holodexRepository.searchMusicOnChannels(query, channelIds).getOrThrow()
        } else {
            val key = SearchCacheKey(query, offset)
            holodexRepository.fetchSearchList(key, isRefresh).getOrThrow().data
        }
    }

    // *** FIX 2: Use Unified Repo for Downloads ***
    val likedIds = unifiedRepository.observeLikedItemIds().first()
    val downloadedIds =
        unifiedRepository.getDownloads().first().map { it.playbackItemId }.toSet()

    result.onSuccess { rawItems ->
        val unifiedItems =
            rawItems.map { it.toUnifiedDisplayItem(likedIds.contains(it.id), downloadedIds
        reduce {
            val currentList = if (isInitial || isRefresh) emptyList() else state.searchIte
            state.copy(
                searchItems = (currentList + unifiedItems).toImmutableList(),
                searchCurrentOffset = offset + rawItems.size,
                searchEndOfList = rawItems.isEmpty(),
                searchIsLoadingInitial = false, searchIsLoadingMore = false
            )
        }
    }.onFailure {
        postSideEffect(VideoListSideEffect.ShowToast("Search failed"))
        reduce { state.copy(searchIsLoadingInitial = false, searchIsLoadingMore = false) }
    }
}

// --- PUBLIC ACTIONS ---
```

```
fun refreshCurrentListViaPull() = intent {
    fetchCurrentContextData(isInitial = false, isRefresh = true)
}

fun loadMore(contextType: MusicCategoryType) = intent {
    if (contextType == MusicCategoryType.SEARCH) fetchSearchResultsInternal(
        isInitial = false,
        isRefresh = false
    )
    else fetchBrowseItemsInternal(isInitial = false, isRefresh = false)
}

fun onVideoClicked(item: UnifiedDisplayItem) = intent {
    videoItemForDetailScreen = null
    postSideEffect(VideoListSideEffect.NavigateTo(NavigationDestination.VideoDetails(item.
}

fun setSearchActive(isActive: Boolean) = intent {
    reduce { state.copy(isSearchActive = isActive) }
    if (isActive) {
        reduce { state.copy(activeContextType = MusicCategoryType.SEARCH) }
        saveActiveListContextType(MusicCategoryType.SEARCH)
    }
}

fun onSearchQueryChange(newQuery: String) = intent {
    reduce { state.copy(currentSearchQuery = newQuery) }
}

fun performSearch(query: String) = intent {
    val trimmed = query.trim()
    if (trimmed.isBlank()) {
        postSideEffect(VideoListSideEffect.ShowToast("Please enter a search term"))
        return@intent
    }

    extractVideoIdFromQuery(trimmed)?.let { videoId ->
        postSideEffect(VideoListSideEffect.NavigateTo(NavigationDestination.VideoDetails(v
        reduce { state.copy(isSearchActive = false) }
        return@intent
    }

    viewModelScope.launch { searchHistoryRepository.addSearchQueryToHistory(trimmed) }
    saveSearchQuery(trimmed)

    reduce {
        state.copy(
            currentSearchQuery = trimmed,
            isSearchActive = false,
            activeContextType = MusicCategoryType.SEARCH
        )
    }
    fetchSearchResultsInternal(isInitial = true, isRefresh = false)
}
```

```
fun clearSearchAndReturnToBrowse() = intent {
    reduce {
        state.copy(
            currentSearchQuery = "",
            isSearchActive = false,
            activeContextType = MusicCategoryType.LATEST
        )
    }
    saveSearchQuery("")
    saveActiveListContextType(MusicCategoryType.LATEST)
    fetchBrowseItemsInternal(isInitial = true, isRefresh = false)
}

fun setOrganization(orgName: String) = intent {
    if (state.selectedOrganization != orgName) {
        sharedPreferences.edit { putString(PREF_LAST_SELECTED_ORG, orgName) }
        reduce { state.copy(selectedOrganization = orgName) }

        val newFilters = BrowseFilterState.create(
            preset = state.browseFilterState.selectedViewPreset,
            organization = orgName.takeIf { it != "All Vtubers" }
        )
        updateBrowseFilters(newFilters)
    }
}

fun updateBrowseFilters(newFilters: BrowseFilterState) = intent {
    if (state.browseFilterState != newFilters) {
        reduce { state.copy(browseFilterState = newFilters) }
        saveBrowseFilters(newFilters)

        if (state.activeContextType == MusicCategoryType.SEARCH) {
            reduce { state.copy(activeContextType = MusicCategoryType.LATEST) }
            saveActiveListContextType(MusicCategoryType.LATEST)
        }
        fetchBrowseItemsInternal(isInitial = true, isRefresh = false)
    }
}

fun setBrowseContextAndNavigate(org: String? = null, channelId: String? = null) = intent {
    if (channelId != null) {
        val newQuery = "$CHANNEL_ID_SEARCH_PREFIX$channelId"
        reduce {
            state.copy(
                currentSearchQuery = newQuery,
                isSearchActive = false,
                activeContextType = MusicCategoryType.SEARCH
            )
        }
        saveSearchQuery(newQuery)
        saveActiveListContextType(MusicCategoryType.SEARCH)
        fetchSearchResultsInternal(isInitial = true, isRefresh = false)
        postSideEffect(VideoListSideEffect.NavigateTo(NavigationDestination.HomeScreenWith
    } else {
```

```
        val newFilter = BrowseFilterState.create(
            preset = ViewTypePreset.LATEST_STREAMS,
            organization = org?.takeIf { it != "All Vtubers" },
        )
        updateBrowseFilters(newFilter)
        postSideEffect(VideoListSideEffect.NavigateTo(NavigationDestination.HomeScreenWith
    }
}

fun setActiveSearchSource(source: String) = intent {
    reduce { state.copy(activeSearchSource = source) }
}

fun addVideoOrItsSegmentsToQueue(item: PlaybackItem) = intent {
    viewModelScope.launch {
        addOrFetchAndAddUseCase(item)
            .onSuccess { postSideEffect(VideoListSideEffect.ShowToast(it)) }
            .onFailure { postSideEffect(VideoListSideEffect.ShowToast("Failed: ${it.messag
    }
}

fun playFavoriteOrLikedSegmentItem(item: PlaybackItem) = intent {
    // We need to construct the list to play.
    // Ideally, we play the list surrounding this item, but for a single item click:
    val itemsToPlay = listOf(item)
    val startIndex = 0

    // Just call it directly. No launch needed.
    playbackController.loadAndPlay(itemsToPlay, startIndex)
}

fun clearNavigationRequest() { /* Handled by side effects */
}

private fun loadLastBrowseFilters(): BrowseFilterState = try {
    Gson().fromJson(sharedPreferences.getString(PREF_LAST_BROWSE_FILTERS, null), BrowseFil
        ?: BrowseFilterState.create(ViewTypePreset.UPCOMING_STREAMS)
} catch (_: Exception) { BrowseFilterState.create(ViewTypePreset.UPCOMING_STREAMS) }

val browseScreenCategories: List<Pair<String, BrowseFilterState>> get() = listOf(
    "Upcoming & Live Music" to BrowseFilterState.create(ViewTypePreset.UPCOMING_STREAMS),
    "Latest Music" to BrowseFilterState.create(ViewTypePreset.LATEST_STREAMS)
)

private fun saveBrowseFilters(filters: BrowseFilterState) =
    sharedPreferences.edit { putString(PREF_LAST_BROWSE_FILTERS, Gson().toJson(filters)) }

private fun loadLastSearchQuery(): String =
    sharedPreferences.getString(PREF_LAST_SEARCH_QUERY, "") ?: ""

private fun saveSearchQuery(query: String) =
    sharedPreferences.edit { putString(PREF_LAST_SEARCH_QUERY, query) }

private fun loadLastActiveListContextType(): MusicCategoryType = try {
    MusicCategoryType.valueOf(
```

```
            sharedPreferences.getString(
                PREF_LAST_CATEGORY_TYPE,
                MusicCategoryType.LATEST.name
            )!!
        )
    } catch (_: Exception) {
        MusicCategoryType.LATEST
    }

    private fun saveActiveListContextType(type: MusicCategoryType) =
        sharedPreferences.edit { putString(PREF_LAST_CATEGORY_TYPE, type.name) }

}

// File: java\com\example\holodex\viewmodel\autoplay\AutoplayItemProvider.kt
package com.example.holodex.viewmodel.autoplay

import com.example.holodex.data.repository.HolodexRepository
import com.example.holodex.playback.domain.model.PlaybackItem
import com.example.holodex.viewmodel.UnifiedDisplayItem
import com.example.holodex.data.model.HolodexSong
import com.example.holodex.data.model.HolodexVideoItem
import com.example.holodex.viewmodel.mappers.toPlaybackItem
import timber.log.Timber

internal sealed class AutoplaySearchResult {
    data class Found(val item: UnifiedDisplayItem) : AutoplaySearchResult()
    data class NotFound(val reason: String) : AutoplaySearchResult()
}

class AutoplayItemProvider(
    private val holodexRepository: HolodexRepository
) {
    companion object {
        private const val TAG = "AutoplayItemProvider"
    }

    suspend fun provideNextItemsForAutoplay(
        currentScreenItems: List<UnifiedDisplayItem>,
        lastPlayedItemIdInQueue: String?,
        unifiedDisplayItemToPlaybackItem: (UnifiedDisplayItem) -> PlaybackItem,
        holodexSongToPlaybackItem: (HolodexVideoItem, HolodexSong) -> PlaybackItem
    ): List<PlaybackItem>? {
        if (currentScreenItems.isEmpty()) {
            Timber.w("$TAG: Current screen list is empty.")
            return null
        }

        try {
            val nextCandidateResult = findNextCandidate(currentScreenItems, lastPlayedItemIdIn

            if (nextCandidateResult is AutoplaySearchResult.Found) {
                val candidateItem = nextCandidateResult.item
                Timber.i("$TAG: Found next autoplay candidate: '${candidateItem.title}'")
```

```
            if (shouldCheckForSegments(candidateItem)) {
                val videoWithSongsResult = holodexRepository.getVideoWithSongs(candidateIt
                if (videoWithSongsResult.isSuccess) {
                    val videoWithSongs = videoWithSongsResult.getOrThrow()
                    if (!videoWithSongs.songs.isNullOrEmpty()) {
                        Timber.i("$TAG: Video '${candidateItem.title}' has ${videoWithSong
                        return videoWithSongs.songs.map { song ->
                            holodexSongToPlaybackItem(videoWithSongs, song)
                        }
                    }
                }
            }

            // If not checking for segments or if it fails, play the single item
            val singlePlaybackItem = unifiedDisplayItemToPlaybackItem(candidateItem)
            if (validateAutoplayItem(singlePlaybackItem, lastPlayedItemIdInQueue)) {
                return listOf(singlePlaybackItem)
            }
            return null // Loop prevented
        } else {
            val reason = (nextCandidateResult as AutoplaySearchResult.NotFound).reason
            Timber.i("$TAG: No next autoplay candidate found. Reason: $reason.")
            return null
        }

    } catch (e: Exception) {
        Timber.e(e, "$TAG: Exception during autoplay provider execution.")
        return null
    }
}

private fun findNextCandidate(
    currentScreenItems: List<UnifiedDisplayItem>,
    lastPlayedItemIdInQueue: String?
): AutoplaySearchResult {
    if (lastPlayedItemIdInQueue == null) {
        return if (currentScreenItems.isNotEmpty()) {
            AutoplaySearchResult.Found(currentScreenItems.first())
        } else {
            AutoplaySearchResult.NotFound("screen_list_empty")
        }
    }

    val lastUnderscoreIndex = lastPlayedItemIdInQueue.lastIndexOf('_')

    // Check if there's an underscore and if the part after it is a number (the start time
    val lastPartIsNumeric = lastUnderscoreIndex != -1 &&
            lastPlayedItemIdInQueue.substring(lastUnderscoreIndex + 1).all { it.isDigit()

    val lastPlayedVideoId = if (lastPartIsNumeric) {
        // It's a song segment ID, so take the part before the last underscore.
        lastPlayedItemIdInQueue.substring(0, lastUnderscoreIndex)
    } else {
        // It's just a video ID, so use the whole string.
        lastPlayedItemIdInQueue
```

```
        }

        val indexOfLastPlayed = currentScreenItems.indexOfFirst { it.videoId == lastPlayedVide

        if (indexOfLastPlayed != -1 && indexOfLastPlayed + 1 < currentScreenItems.size) {
            return AutoplaySearchResult.Found(currentScreenItems[indexOfLastPlayed + 1])
        }

        return AutoplaySearchResult.NotFound("end_of_list_reached")
    }


    private fun shouldCheckForSegments(item: UnifiedDisplayItem): Boolean {
        return !item.isSegment && (item.songCount ?: 0) > 0
    }

    private fun validateAutoplayItem(
        itemToPlay: PlaybackItem?,
        lastPlayedItemIdInQueue: String?
    ): Boolean {
        if (itemToPlay == null) return false
        if (lastPlayedItemIdInQueue == null) return true
        val notLooping = itemToPlay.id != lastPlayedItemIdInQueue
        if (!notLooping) {
            Timber.w("$TAG: Loop prevention! Attempted to autoplay same item ID: ${itemToPlay.
        }
        return notLooping
    }
}

// File: java\com\example\holodex\viewmodel\autoplay\ContinuationManager.kt
// File: java/com/example/holodex/viewmodel/autoplay/ContinuationManager.kt
// (Create this new file)

package com.example.holodex.viewmodel.autoplay

import com.example.holodex.data.repository.HolodexRepository
import com.example.holodex.data.repository.UserPreferencesRepository
import com.example.holodex.playback.domain.model.PlaybackItem
import com.example.holodex.playback.player.PlaybackController
import com.example.holodex.viewmodel.UnifiedDisplayItem
import com.example.holodex.viewmodel.mappers.toPlaybackItem
import com.example.holodex.viewmodel.mappers.toVideoShell
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.Job
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.flow.collectLatest
import kotlinx.coroutines.flow.first
import kotlinx.coroutines.launch
import timber.log.Timber
import java.util.Collections
import javax.inject.Inject
```

```
import javax.inject.Singleton

@Singleton
class ContinuationManager @Inject constructor(
    private val holodexRepository: HolodexRepository,
    private val userPreferencesRepository: UserPreferencesRepository,
    private val autoplayItemProvider: AutoplayItemProvider
) {

    companion object {
        private const val TAG = "ContinuationManager"
        private const val RADIO_QUEUE_THRESHOLD = 5
    }

    private var autoplayContextItems: List<UnifiedDisplayItem> =
        Collections.synchronizedList(mutableListOf())

    private val _isRadioModeActive = MutableStateFlow(false)
    val isRadioModeActive: StateFlow<Boolean> = _isRadioModeActive.asStateFlow()

    private var currentRadioId: String? = null
    private var radioMonitorJob: Job? = null

    /**
     * Called by ViewModels to provide the current list of items on screen,
     * which will be used as the source for autoplay suggestions.
     */
    fun setAutoplayContext(items: List<UnifiedDisplayItem>) {
        // Only update context if the new list is not empty.
        // This prevents transient empty states from wiping a valid, existing context.
        if (items.isNotEmpty()) {
            Timber.d("$TAG: Setting autoplay context with ${items.size} items.")
            autoplayContextItems = Collections.synchronizedList(items.toMutableList())
        } else {
            Timber.d("$TAG: Ignoring empty context update. Keeping existing context with ${aut
        }
    }

    /**
     * Explicit method for intentionally clearing the autoplay context when a user
     * performs an action that should reset it, like a new search or pull-to-refresh.
     */
    fun clearAutoplayContext() {
        Timber.d("$TAG: Explicitly clearing autoplay context.")
        autoplayContextItems = Collections.synchronizedList(mutableListOf())
    }

    /**
     * Starts a new radio session. Fetches the initial batch of songs and begins monitoring th
     * @return The initial list of PlaybackItems to start the radio.
     */
    suspend fun startRadioSession(
        radioId: String,
        scope: CoroutineScope,
        controller: PlaybackController
```

```
): List<PlaybackItem>? {
    endCurrentSession()
    currentRadioId = radioId
    _isRadioModeActive.value = true
    Timber.d("$TAG: Starting radio session for ID: $radioId")

    val initialBatch = fetchRadioBatch(radioId)
    if (initialBatch.isNullOrEmpty()) {
        endCurrentSession()
        return null
    }

    radioMonitorJob = scope.launch(Dispatchers.IO) {
        // Observe Controller State directly
        controller.state.collectLatest { state ->
            handleQueueStateForRadio(state.activeQueue, state.currentIndex, controller)
        }
    }
    return initialBatch
}

/**
 * Ends the current radio session, stopping any background monitoring.
 */
fun endCurrentSession() {
    if (currentRadioId != null) {
        Timber.d("$TAG: Ending radio session for ID: $currentRadioId")
    }
    radioMonitorJob?.cancel()
    radioMonitorJob = null
    currentRadioId = null
    _isRadioModeActive.value = false
}

/**
 * Provides the next items to play when a finite queue ends, respecting the user's autopla
 * @return A list of new PlaybackItems to append, or null if autoplay is disabled or no it
 */
suspend fun provideAutoplayItems(currentQueue: List<PlaybackItem>): List<PlaybackItem>? {
    val isAutoplayEnabled = userPreferencesRepository.autoplayEnabled.first()
    if (!isAutoplayEnabled) {
        Timber.d("$TAG: Autoplay is disabled by user setting. Not providing items.")
        return null
    }

    Timber.d("$TAG: Autoplay is enabled. Attempting to provide next items.")


    return autoplayItemProvider.provideNextItemsForAutoplay(
        currentScreenItems = autoplayContextItems,
        lastPlayedItemIdInQueue = currentQueue.lastOrNull()?.id,
        { item -> item.toPlaybackItem() }, // Pass the mapper function
        { video, song -> song.toPlaybackItem(video) } // Pass the other mapper function
    )
}
```

```
    private suspend fun handleQueueStateForRadio(
        queue: List<PlaybackItem>,
        currentIndex: Int,
        controller: PlaybackController // <--- CHANGED
    ) {
        val radioId = currentRadioId ?: return

        val songsRemaining = queue.size - (currentIndex + 1)
        if (songsRemaining < RADIO_QUEUE_THRESHOLD) {
            val nextBatch = fetchRadioBatch(radioId)
            if (!nextBatch.isNullOrEmpty()) {
                // Use Controller to add items
                controller.addItemsToQueue(nextBatch)
            }
        }
    }

    private suspend fun fetchRadioBatch(radioId: String): List<PlaybackItem>? {
        val result = holodexRepository.getRadioContent(radioId)
        return result.getOrNull()?.content?.mapNotNull { song ->
            val videoShell = song.toVideoShell(result.getOrNull()?.title ?: "")
            song.toPlaybackItem(videoShell)
        }
    }
}

// File: java\com\example\holodex\viewmodel\mappers\UnifiedDisplayItemMapper.kt
package com.example.holodex.viewmodel.mappers

import com.example.holodex.data.db.LikedItemType
import com.example.holodex.data.db.PlaylistItemEntity
import com.example.holodex.data.db.UnifiedItemProjection
import com.example.holodex.data.model.HolodexChannelMin
import com.example.holodex.data.model.HolodexSong
import com.example.holodex.data.model.HolodexVideoItem
import com.example.holodex.data.model.discovery.MusicdexSong
import com.example.holodex.playback.domain.model.PlaybackItem
import com.example.holodex.playback.util.formatDurationSeconds
import com.example.holodex.util.ThumbnailQuality
import com.example.holodex.util.generateArtworkUrlList
import com.example.holodex.viewmodel.UnifiedDisplayItem
import timber.log.Timber
import kotlin.math.max

// ================================================================================================
// 1. DATABASE PROJECTION -> UI MODEL (The "Unified" Way)
// ================================================================================================

fun UnifiedItemProjection.toUnifiedDisplayItem(): UnifiedDisplayItem {
    val downloadInteraction = interactions.find { it.interactionType == "DOWNLOAD" }
    val likeInteraction = interactions.find { it.interactionType == "LIKE" }
    val isSegment = metadata.type == "SEGMENT"
    if (metadata.type == "CHANNEL") {
        Timber.d("MAPPER DEBUG: ID=${metadata.id}, Org=${metadata.org}, isExternal=${metadata.
```

```
    }
    return UnifiedDisplayItem(
        stableId = "${metadata.type.lowercase()}_${metadata.id}",
        playbackItemId = metadata.id,
        videoId = metadata.parentVideoId ?: metadata.id,
        channelId = metadata.channelId,
        title = metadata.title,
        artistText = metadata.artistName,
        artworkUrls = metadata.getComputedArtworkList(),
        durationText = formatDurationSeconds(metadata.duration),
        isSegment = isSegment,
        songCount = if (isSegment) null else metadata.songCount,
        isDownloaded = downloadInteraction?.downloadStatus == "COMPLETED",
        downloadStatus = downloadInteraction?.downloadStatus,
        localFilePath = downloadInteraction?.localFilePath, // Fix: Use the new field
        isLiked = likeInteraction != null,
        itemTypeForPlaylist = if (isSegment) LikedItemType.SONG_SEGMENT else LikedItemType.VID
        songStartSec = metadata.startSeconds?.toInt(),
        songEndSec = metadata.endSeconds?.toInt(),
        originalArtist = null,
        isExternal = metadata.org == "External"
    )
}

// ================================================================================================
// 2. UI MODEL -> PLAYER MODEL
// ================================================================================================

fun UnifiedDisplayItem.toPlaybackItem(): PlaybackItem {
    val finalStreamUri = if (this.isDownloaded && !this.localFilePath.isNullOrBlank()) {
        this.localFilePath
    } else {
        null
    }

    return PlaybackItem(
        id = this.playbackItemId,
        videoId = this.videoId,
        serverUuid = if (this.isSegment) this.playbackItemId else null,
        songId = if (this.isSegment) this.playbackItemId else null,
        title = this.title,
        artistText = this.artistText,
        albumText = if (!this.isSegment) this.title else null,
        artworkUri = this.artworkUrls.firstOrNull(),
        durationSec = this.songEndSec?.toLong()?.let { it - (this.songStartSec?.toLong() ?: 0)
        streamUri = finalStreamUri,
        clipStartSec = this.songStartSec?.toLong(),
        clipEndSec = this.songEndSec?.toLong(),
        description = null,
        channelId = this.channelId,
        originalArtist = this.originalArtist,
        isExternal = this.isExternal
    )
}
```

```
// ================================================================================================
// 3. API/LEGACY MODELS -> UI MODEL
// ================================================================================================

fun HolodexVideoItem.toUnifiedDisplayItem(
    isLiked: Boolean,
    downloadedSegmentIds: Set<String>
): UnifiedDisplayItem {
    val containsDownloadedSegments = this.songs?.any { song ->
        val segmentId = "${this.id}_${song.start}"
        downloadedSegmentIds.contains(segmentId)
    } == true

    return UnifiedDisplayItem(
        stableId = "video_${this.id}",
        playbackItemId = this.id,
        videoId = this.id,
        channelId = this.channel.id ?: this.id,
        title = this.title,
        artistText = this.channel.name,
        artworkUrls = generateArtworkUrlList(this.toPlaybackItem(), ThumbnailQuality.MEDIUM),
        durationText = formatDurationSeconds(this.duration),
        isSegment = false,
        songCount = this.songcount,
        isDownloaded = containsDownloadedSegments,
        downloadStatus = if (containsDownloadedSegments) "COMPLETED" else null,
        localFilePath = null,
        isLiked = isLiked,
        itemTypeForPlaylist = LikedItemType.VIDEO,
        songStartSec = null,
        songEndSec = null,
        originalArtist = null,
        isExternal = this.channel.org == "External"
    )
}

fun HolodexSong.toUnifiedDisplayItem(
    parentVideo: HolodexVideoItem,
    isLiked: Boolean,
    isDownloaded: Boolean
): UnifiedDisplayItem {
    val playbackItemId = "${parentVideo.id}_${this.start}"
    return UnifiedDisplayItem(
        stableId = "song_${playbackItemId}",
        playbackItemId = playbackItemId,
        videoId = parentVideo.id,
        channelId = parentVideo.channel.id ?: parentVideo.id,
        title = this.name,
        artistText = parentVideo.channel.name,
        artworkUrls = generateArtworkUrlList(this.toPlaybackItem(parentVideo), ThumbnailQualit
        durationText = formatDurationSeconds((this.end - this.start).toLong()),
        isSegment = true,
        songCount = null,
        isDownloaded = isDownloaded,
        downloadStatus = if (isDownloaded) "COMPLETED" else null,
```

```
        localFilePath = null,
        isLiked = isLiked,
        itemTypeForPlaylist = LikedItemType.SONG_SEGMENT,
        songStartSec = this.start,
        songEndSec = this.end,
        originalArtist = this.originalArtist,
        isExternal = parentVideo.channel.org == "External"
    )
}

fun PlaylistItemEntity.toUnifiedDisplayItem(
    isDownloaded: Boolean,
    isLiked: Boolean
): UnifiedDisplayItem {
    val isSegment = this.itemTypeInPlaylist == LikedItemType.SONG_SEGMENT
    val durationSec = if (isSegment && this.songStartSecondsPlaylist != null && this.songEndSe
        max(1, (this.songEndSecondsPlaylist - this.songStartSecondsPlaylist)).toLong()
    } else {
        0L
    }

    return UnifiedDisplayItem(
        stableId = "playlist_${this.playlistOwnerId}_${this.itemIdInPlaylist}",
        playbackItemId = this.itemIdInPlaylist,
        videoId = this.videoIdForItem,
        channelId = "",
        title = this.songNamePlaylist ?: "Unknown Title",
        artistText = this.songArtistTextPlaylist ?: "Unknown Artist",
        artworkUrls = listOfNotNull(this.songArtworkUrlPlaylist),
        durationText = formatDurationSeconds(durationSec),
        isSegment = isSegment,
        songCount = null,
        isDownloaded = isDownloaded,
        downloadStatus = if (isDownloaded) "COMPLETED" else null,
        localFilePath = null,
        isLiked = isLiked,
        itemTypeForPlaylist = this.itemTypeInPlaylist,
        songStartSec = this.songStartSecondsPlaylist,
        songEndSec = this.songEndSecondsPlaylist,
        originalArtist = this.songArtistTextPlaylist,
        isExternal = this.isLocalOnly
    )
}
fun MusicdexSong.toVideoShell(albumTitle: String = "Unknown Video"): HolodexVideoItem {
    return HolodexVideoItem(
        id = this.videoId,
        title = albumTitle,
        type = "stream",
        topicId = null,
        availableAt = "",
        publishedAt = null,
        duration = (this.end - this.start).toLong(),
        status = "past",
        channel = HolodexChannelMin(
            id = this.channel.id ?: this.channelId,
```

```
            name = this.channel.name,
            englishName = this.channel.englishName,
            org = null,
            type = "vtuber",
            photoUrl = this.channel.photoUrl
        ),
        songcount = 1,
        description = null,
        songs = null
    )
}
fun MusicdexSong.toUnifiedDisplayItem(
    parentVideo: HolodexVideoItem,
    isLiked: Boolean,
    isDownloaded: Boolean
): UnifiedDisplayItem {
    val playbackItemId = "${this.videoId}_${this.start}"
    return UnifiedDisplayItem(
        stableId = "song_${playbackItemId}",
        playbackItemId = playbackItemId,
        videoId = this.videoId,
        channelId = this.channel.id ?: "",
        title = this.name,
        artistText = this.channel.name,
        artworkUrls = generateArtworkUrlList(this.toPlaybackItem(parentVideo), ThumbnailQualit
        durationText = formatDurationSeconds((this.end - this.start).toLong()),
        isSegment = true,
        songCount = null,
        isDownloaded = isDownloaded,
        downloadStatus = if (isDownloaded) "COMPLETED" else null,
        localFilePath = null,
        isLiked = isLiked,
        itemTypeForPlaylist = LikedItemType.SONG_SEGMENT,
        songStartSec = this.start,
        songEndSec = this.end,
        originalArtist = this.originalArtist,
        isExternal = parentVideo.channel.org == "External"
    )
}

fun HolodexVideoItem.toVirtualSegmentUnifiedDisplayItem(
    isLiked: Boolean,
    isDownloaded: Boolean
): UnifiedDisplayItem {
    val playbackItemId = "${this.id}_0"
    return UnifiedDisplayItem(
        stableId = "video_as_segment_${this.id}",
        playbackItemId = playbackItemId,
        videoId = this.id,
        channelId = this.channel.id ?: "",
        title = this.title,
        artistText = this.channel.name,
        artworkUrls = generateArtworkUrlList(this.toPlaybackItem(), ThumbnailQuality.MEDIUM),
        durationText = formatDurationSeconds(this.duration),
        isSegment = true,
```

```
        songCount = 0,
        isDownloaded = isDownloaded,
        downloadStatus = if (isDownloaded) "COMPLETED" else null,
        localFilePath = null,
        isLiked = isLiked,
        itemTypeForPlaylist = LikedItemType.VIDEO,
        songStartSec = 0,
        songEndSec = this.duration.toInt(),
        originalArtist = null,
        isExternal = this.channel.org == "External"
    )
}

// ================================================================================================
// 4. API MODEL -> PLAYER MODEL
// ================================================================================================

fun HolodexVideoItem.toPlaybackItem(): PlaybackItem {
    return PlaybackItem(
        id = this.id,
        videoId = this.id,
        serverUuid = null,
        songId = null,
        title = this.title,
        artistText = this.channel.name,
        albumText = this.title,
        artworkUri = this.channel.photoUrl,
        durationSec = this.duration,
        streamUri = null,
        clipStartSec = null,
        clipEndSec = null,
        description = this.description,
        channelId = this.channel.id ?: "unknown_channel_${this.id}",
        originalArtist = null,
        isExternal = this.channel.org == "External"
    )
}

fun HolodexSong.toPlaybackItem(parentVideo: HolodexVideoItem): PlaybackItem {
    val playbackId = "${parentVideo.id}_${this.start}"
    return PlaybackItem(
        id = playbackId,
        videoId = parentVideo.id,
        serverUuid = playbackId, // Placeholder until server ID is known
        songId = playbackId,
        title = this.name,
        artistText = parentVideo.channel.name,
        albumText = parentVideo.title,
        artworkUri = this.artUrl,
        durationSec = (this.end - this.start).toLong(),
        streamUri = null,
        clipStartSec = this.start.toLong(),
        clipEndSec = this.end.toLong(),
        description = parentVideo.description,
        channelId = parentVideo.channel.id ?: "unknown_channel_${parentVideo.id}",
```

```
        originalArtist = this.originalArtist,
        isExternal = parentVideo.channel.org == "External"
    )
}

fun MusicdexSong.toPlaybackItem(parentVideo: HolodexVideoItem): PlaybackItem {
    val playbackId = "${this.videoId}_${this.start}"
    return PlaybackItem(
        id = playbackId,
        videoId = parentVideo.id,
        serverUuid = this.id,
        songId = playbackId,
        title = this.name,
        artistText = parentVideo.channel.name,
        albumText = parentVideo.title,
        artworkUri = this.artUrl,
        durationSec = (this.end - this.start).toLong(),
        streamUri = null,
        clipStartSec = this.start.toLong(),
        clipEndSec = this.end.toLong(),
        description = parentVideo.description,
        channelId = parentVideo.channel.id ?: "unknown",
        originalArtist = this.originalArtist,
        isExternal = parentVideo.channel.org == "External"
    )
}

// File: java\com\example\holodex\viewmodel\state\BrowseFilterState.kt
package com.example.holodex.viewmodel.state

enum class ViewTypePreset(
    val apiStatus: String?,
    val apiMaxUpcomingHours: Int?,
    val defaultSortField: VideoSortField,
    val defaultSortOrder: SortOrder,
    val defaultDisplayName: String
) {
    LATEST_STREAMS(
        apiStatus = "past",
        apiMaxUpcomingHours = null,
        defaultSortField = VideoSortField.AVAILABLE_AT,
        defaultSortOrder = SortOrder.DESC,
        defaultDisplayName = "Latest"
    ),
    UPCOMING_STREAMS(
        apiStatus = "upcoming",
        apiMaxUpcomingHours = 48,
        defaultSortField = VideoSortField.START_SCHEDULED,
        defaultSortOrder = SortOrder.ASC,
        defaultDisplayName = "Upcoming"
    );
}

// DELETED: enum class SongSegmentFilterMode
```

```
enum class VideoSortField(val apiValue: String, val displayName: String) {
    AVAILABLE_AT("available_at", "Date"),
    PUBLISHED_AT("published_at", "Published Date"),
    START_SCHEDULED("start_scheduled", "Scheduled Time"),
    START_ACTUAL("start_actual", "Actual Start"),
    DURATION("duration", "Duration"),
    LIVE_VIEWERS("live_viewers", "Live Viewers"),
    SONG_COUNT("songcount", "Song Count"),
    TITLE("title", "Title")
}

enum class SortOrder(val apiValue: String, val displayName: String) {
    ASC("asc", "Ascending"),
    DESC("desc", "Descending")
}

data class BrowseFilterState(
    val selectedOrganization: String? = null,
    val selectedPrimaryTopic: String? = null,
    val selectedViewPreset: ViewTypePreset,
    // DELETED: val songSegmentFilterMode: SongSegmentFilterMode,
    val sortField: VideoSortField,
    val sortOrder: SortOrder,
    val currentFilterDisplayName: String
) {
    val status: String? get() = selectedViewPreset.apiStatus
    val maxUpcomingHours: Int? get() = selectedViewPreset.apiMaxUpcomingHours

    companion object {
        fun create(
            preset: ViewTypePreset,
            organization: String? = null,
            primaryTopic: String? = null,
            sortFieldOverride: VideoSortField? = null,
            sortOrderOverride: SortOrder? = null
        ): BrowseFilterState {
            val effectiveSortField = sortFieldOverride ?: preset.defaultSortField
            val effectiveSortOrder = sortOrderOverride ?: preset.defaultSortOrder

            return BrowseFilterState(
                selectedOrganization = organization,
                selectedPrimaryTopic = primaryTopic,
                selectedViewPreset = preset,
                sortField = effectiveSortField,
                sortOrder = effectiveSortOrder,
                currentFilterDisplayName = preset.defaultDisplayName
            )
        }
    }

    @get:JvmName("getHasActiveFiltersProperty")
    val hasActiveFilters: Boolean
        get() {
            val default = create(this.selectedViewPreset, null, null)
            return this.selectedOrganization != default.selectedOrganization ||
```

```
                    this.selectedPrimaryTopic != default.selectedPrimaryTopic ||
                    this.sortField != default.sortField ||
                    this.sortOrder != default.sortOrder
        }

    @get:JvmName("getActiveFilterCountProperty")
    val activeFilterCount: Int
        get() {
            var count = 0
            val default = create(this.selectedViewPreset, null, null)
            if (this.selectedOrganization != default.selectedOrganization) count++
            if (this.selectedPrimaryTopic != default.selectedPrimaryTopic) count++
            if (this.sortField != default.sortField || this.sortOrder != default.sortOrder) co
            return count
        }
}

// File: java\com\example\holodex\viewmodel\state\UiState.kt
package com.example.holodex.viewmodel.state

/**
 * A generic class that represents a resource's state: Loading, Success, or Error.
 * This is used for individual shelves in the Discovery Hub and other async UI components.
 */
sealed class UiState<out T> {
    object Loading : UiState<Nothing>()
    data class Success<T>(val data: T) : UiState<T>()
    data class Error(val message: String) : UiState<Nothing>()
}

// File: res\drawable\ic_default_album_art_placeholder.xml
<vector xmlns:android="http://schemas.android.com/apk/res/android" android:height="24dp" andro

    <path android:fillColor="@android:color/white" android:pathData="M12,2C6.48,2 2,6.48 2,12s

</vector>


// File: res\drawable\ic_error_image.xml
<vector xmlns:android="http://schemas.android.com/apk/res/android" android:height="24dp" andro

    <path android:fillColor="@android:color/white" android:pathData="M11,15h2v2h-2zM11,7h2v6h-

</vector>


// File: res\drawable\ic_launcher_background.xml
<?xml version="1.0" encoding="utf-8"?>
<vector xmlns:android="http://schemas.android.com/apk/res/android"
    android:width="108dp"
    android:height="108dp"
    android:viewportWidth="108"
    android:viewportHeight="108">
    <path
        android:fillColor="#3DDC84"
```

```
    android:pathData="M0,0h108v108h-108z" />
<path
    android:fillColor="#00000000"
    android:pathData="M9,0L9,108"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M19,0L19,108"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M29,0L29,108"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M39,0L39,108"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M49,0L49,108"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M59,0L59,108"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M69,0L69,108"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M79,0L79,108"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M89,0L89,108"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M99,0L99,108"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M0,9L108,9"
    android:strokeWidth="0.8"
```

```
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M0,19L108,19"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M0,29L108,29"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M0,39L108,39"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M0,49L108,49"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M0,59L108,59"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M0,69L108,69"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M0,79L108,79"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M0,89L108,89"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M0,99L108,99"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M19,29L89,29"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M19,39L89,39"
    android:strokeWidth="0.8"
```

```
        android:strokeColor="#33FFFFFF" />
    <path
        android:fillColor="#00000000"
        android:pathData="M19,49L89,49"
        android:strokeWidth="0.8"
        android:strokeColor="#33FFFFFF" />
    <path
        android:fillColor="#00000000"
        android:pathData="M19,59L89,59"
        android:strokeWidth="0.8"
        android:strokeColor="#33FFFFFF" />
    <path
        android:fillColor="#00000000"
        android:pathData="M19,69L89,69"
        android:strokeWidth="0.8"
        android:strokeColor="#33FFFFFF" />
    <path
        android:fillColor="#00000000"
        android:pathData="M19,79L89,79"
        android:strokeWidth="0.8"
        android:strokeColor="#33FFFFFF" />
    <path
        android:fillColor="#00000000"
        android:pathData="M29,19L29,89"
        android:strokeWidth="0.8"
        android:strokeColor="#33FFFFFF" />
    <path
        android:fillColor="#00000000"
        android:pathData="M39,19L39,89"
        android:strokeWidth="0.8"
        android:strokeColor="#33FFFFFF" />
    <path
        android:fillColor="#00000000"
        android:pathData="M49,19L49,89"
        android:strokeWidth="0.8"
        android:strokeColor="#33FFFFFF" />
    <path
        android:fillColor="#00000000"
        android:pathData="M59,19L59,89"
        android:strokeWidth="0.8"
        android:strokeColor="#33FFFFFF" />
    <path
        android:fillColor="#00000000"
        android:pathData="M69,19L69,89"
        android:strokeWidth="0.8"
        android:strokeColor="#33FFFFFF" />
    <path
        android:fillColor="#00000000"
        android:pathData="M79,19L79,89"
        android:strokeWidth="0.8"
        android:strokeColor="#33FFFFFF" />
</vector>


// File: res\drawable\ic_launcher_foreground.xml
```

```
<vector xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:aapt="http://schemas.android.com/aapt"
    android:width="108dp"
    android:height="108dp"
    android:viewportWidth="108"
    android:viewportHeight="108">
    <path android:pathData="M31,63.928c0,0 6.4,-11 12.1,-13.1c7.2,-2.6 26,-1.4 26,-1.4l38.1,38
        <aapt:attr name="android:fillColor">
            <gradient
                android:endX="85.84757"
                android:endY="92.4963"
                android:startX="42.9492"
                android:startY="49.59793"
                android:type="linear">
                <item
                    android:color="#44000000"
                    android:offset="0.0" />
                <item
                    android:color="#00000000"
                    android:offset="1.0" />
            </gradient>
        </aapt:attr>
    </path>
    <path
        android:fillColor="#FFFFFF"
        android:fillType="nonZero"
        android:pathData="M65.3,45.828l3.8,-6.6c0.2,-0.4 0.1,-0.9 -0.3,-1.1c-0.4,-0.2 -0.9,-0.
        android:strokeWidth="1"
        android:strokeColor="#00000000" />
</vector>

// File: res\drawable\ic_like_empty.xml
<vector xmlns:android="http://schemas.android.com/apk/res/android" android:height="24dp" andro

    <path android:fillColor="@android:color/white" android:pathData="M480,480Q480,480 480,480Q

</vector>


// File: res\drawable\ic_notification_small.xml
<vector xmlns:android="http://schemas.android.com/apk/res/android" android:height="24dp" andro

    <path android:fillColor="@android:color/white" android:pathData="M12,3L4,9v12h16V9L12,3zM1

</vector>


// File: res\drawable\ic_pause.xml
<vector xmlns:android="http://schemas.android.com/apk/res/android" android:height="24dp" andro

    <path android:fillColor="@android:color/white" android:pathData="M12,2C6.48,2 2,6.48 2,12s

</vector>
```

```
// File: res\drawable\ic_placeholder_image.xml
<vector xmlns:android="http://schemas.android.com/apk/res/android" android:height="24dp" andro

    <path android:fillColor="@android:color/white" android:pathData="M20,2L8,2c-1.1,0 -2,0.9 -

</vector>


// File: res\drawable\ic_play_arrow.xml
<vector xmlns:android="http://schemas.android.com/apk/res/android" android:height="24dp" andro

    <path android:fillColor="@android:color/white" android:pathData="M8,5v14l11,-7z"/>

</vector>


// File: res\drawable\ic_repeat_off_24.xml
<vector xmlns:android="http://schemas.android.com/apk/res/android" android:height="24dp" andro

    <path android:fillColor="@android:color/white" android:pathData="M280,880L120,720L280,560L

</vector>


// File: res\drawable\ic_repeat_one_24.xml
<vector xmlns:android="http://schemas.android.com/apk/res/android" android:height="24dp" andro

    <path android:fillColor="@android:color/white" android:pathData="M7,7h10v3l4,-4 -4,-4v3L5,

</vector>


// File: res\drawable\ic_repeat_on_24.xml
<vector xmlns:android="http://schemas.android.com/apk/res/android" android:height="24dp" andro

    <path android:fillColor="@android:color/white" android:pathData="M120,920Q87,920 63.5,896.

</vector>


// File: res\drawable\ic_shuffle_off_24.xml
<vector xmlns:android="http://schemas.android.com/apk/res/android" android:height="24dp" andro

    <path android:fillColor="@android:color/white" android:pathData="M10.59,9.17L5.41,4 4,5.41

</vector>


// File: res\drawable\ic_shuffle_on_24.xml
<vector xmlns:android="http://schemas.android.com/apk/res/android" android:height="24dp" andro

    <path android:fillColor="@android:color/white" android:pathData="M120,920Q87,920 63.5,896.

</vector>
```

```
// File: res\drawable\ic_skip_next.xml
<vector xmlns:android="http://schemas.android.com/apk/res/android" android:height="24dp" andro

    <path android:fillColor="@android:color/white" android:pathData="M6,18l8.5,-6L6,6v12zM16,6

</vector>


// File: res\drawable\ic_skip_previous.xml
<vector xmlns:android="http://schemas.android.com/apk/res/android" android:height="24dp" andro

    <path android:fillColor="@android:color/white" android:pathData="M6,6h2v12L6,18zM9.5,12l8.

</vector>


// File: res\drawable\ic_stat_music_note.xml
<vector xmlns:android="http://schemas.android.com/apk/res/android" android:height="24dp" andro

    <path android:fillColor="@android:color/white" android:pathData="M12,3v10.55c-0.59,-0.34 -

</vector>


// File: res\drawable\ic_twitter.xml
<vector xmlns:android="http://schemas.android.com/apk/res/android"
    android:width="512dp"
    android:height="512dp"
    android:viewportWidth="512"
    android:viewportHeight="512">
    <path
        android:pathData="m0,0H512V512H0"
        android:fillColor="#fff"/>
    <path
        android:pathData="m458,140q-23,10 -45,12 25,-15 34,-43 -24,14 -50,19a79,79 0,0 0,-135
        android:fillColor="#1d9bf0"/>
</vector>

// File: res\drawable\ic_youtube.xml
<vector xmlns:android="http://schemas.android.com/apk/res/android"
    android:width="512dp"
    android:height="512dp"
    android:viewportWidth="512"
    android:viewportHeight="512">
    <path
        android:pathData="m0,0H512V512H0"
        android:fillColor="#fff"/>
    <path
        android:pathData="M313,256l-93,-53L220,309ZM427,169c9,37 9,138 0,174 -4,15 -17,27 -32,
        android:fillColor="#ed1d24"/>
</vector>

// File: res\drawable\twitter.xml
<vector xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:width="24dp"
    android:height="24dp"
    android:viewportWidth="24"
    android:viewportHeight="24">
  <path
      android:pathData="M23,3a10.9,10.9 0,0 1,-3.14 1.53,4.48 4.48,0 0,0 -7.86,3v1A10.66,10.66
      android:strokeLineJoin="round"
      android:strokeWidth="2"
      android:fillColor="#00000000"
      android:strokeColor="#44819F"
      android:strokeLineCap="round"/>
</vector>


// File: res\drawable\youtube.xml
<vector xmlns:android="http://schemas.android.com/apk/res/android"
    android:width="24dp"
    android:height="24dp"
    android:viewportWidth="24"
    android:viewportHeight="24">
  <path
      android:pathData="M22.54,6.42a2.78,2.78 0,0 0,-1.94 -2C18.88,4 12,4 12,4s-6.88,0 -8.6,0.
      android:strokeLineJoin="round"
      android:strokeWidth="2"
      android:fillColor="#00000000"
      android:strokeColor="#44819F"
      android:strokeLineCap="round"/>
  <path
      android:pathData="M9.75,15.02l5.75,-3.27l-5.75,-3.27l0,6.54z"
      android:strokeLineJoin="round"
      android:strokeWidth="2"
      android:fillColor="#00000000"
      android:strokeColor="#44819F"
      android:strokeLineCap="round"/>
</vector>


// File: res\mipmap-anydpi-v26\ic_launcher.xml
<?xml version="1.0" encoding="utf-8"?>
<adaptive-icon xmlns:android="http://schemas.android.com/apk/res/android">
  <background android:drawable="@mipmap/ic_launcher_background"/>
  <foreground android:drawable="@mipmap/ic_launcher_foreground"/>
  <monochrome android:drawable="@mipmap/ic_launcher_monochrome"/>
</adaptive-icon>

// File: res\values\attrs.xml
<!-- res/values/attrs.xml -->
<resources>
    <attr name="notificationSmallIcon" format="reference"/>
</resources>

// File: res\values\colors.xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- Basic semantic colors, often overridden by themes -->
```

```
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>

    <!-- Primary colors for XML themes (e.g., for Splash Screen or pre-Compose UI) -->
    <!-- These should ideally match or be inspired by your md_theme_light_primary -->
    <!-- from your Compose Color.kt for consistency if used by system UI elements -->
    <!-- before Compose takes over. -->
    <color name="seed">#6750A4</color> <!-- Your md_theme_light_primary -->

    <!-- Example: Colors matching your light theme (from your Color.kt) -->
    <!-- These can be used for things like windowSplashScreenBackground -->
    <color name="primary_light">#6750A4</color>
    <color name="on_primary_light">#FFFFFF</color>
    <color name="primary_container_light">#EADDFF</color>
    <color name="on_primary_container_light">#21005D</color>
    <color name="surface_light">#FFFBFE</color> <!-- Good for general backgrounds -->
    <color name="background_light">#FFFBFE</color>

    <!-- You might not need specific dark theme colors here if your -->
    <!-- DayNight theme handles it and Compose uses its own darkColorScheme. -->
    <!-- However, if you need to reference them from XML for some reason: -->
    <color name="primary_dark">#D0BCFF</color>
    <color name="on_primary_dark">#381E72</color>
    <color name="surface_dark">#1C1B1F</color> <!-- Good for general backgrounds in dark theme
    <color name="background_dark">#1C1B1F</color>

</resources>

// File: res\values\strings.xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- Existing strings -->
    <string name="app_name">Holodex Music</string>
    <string name="unknown_title">Unknown Title</string>
    <string name="unknown_artist">Unknown Artist</string>
    <string name="unknown_album">Unknown Album</string>
    <string name="loading">Loading?</string>

    <!-- API Key related -->
    <string name="hint_api_key">Enter Holodex API Key</string>
    <string name="button_save_key">Save Key</string>
    <string name="toast_api_key_saved">API Key saved!</string>
    <string name="toast_api_key_empty">API Key cannot be empty.</string>
    <string name="toast_api_key_required">Please save an API Key first.</string>
    <string name="status_api_key_required_fetch">Please enter and save your Holodex API Key to

    <!-- Fetching & Data States -->
    <string name="button_fetch_music">Fetch by Category</string>
    <string name="button_load_more">Load More</string>
    <string name="status_no_videos_found_filter">No videos found for this filter.</string>
    <string name="status_select_category_first">Select a category first</string>
    <string name="status_select_category_load_more">Fetch a category first to load more</strin
    <string name="toast_fetching_songs_for">Fetching songs for: %s</string>
    <string name="toast_playing_segment">Preparing to play segment: %s</string>
    <string name="toast_playing_selected_segment">Preparing selected: %s</string>
```

```
<string name="toast_no_segments_playing_full">No segments. Preparing full audio for: %s</s
<string name="toast_stream_url_ready">Audio stream URL obtained! Ready for playback.</stri

<!-- Organization Spinner Prompt (Optional) -->
<string name="spinner_prompt_organization">Select Organization</string>
<string name="unknown_channel">Unknown Channel</string>
<string name="notification_no_media_title">No media playing</string>
<!-- Song List Dialog -->
<string name="toast_playing_segment_from_stream">Playing stream segment: %s</string>
<string name="unknown_song_title">Unknown Song Title</string>
<string name="song_timestamp_format">%1$s - %2$s</string>
<string name="timestamp_not_available">N/A</string>
<string name="dialog_title_select_song">Select Song</string>
<string name="button_play_full_stream">Play Full Stream</string>
<string name="cancel">Cancel</string>
<string name="unknown_song_title_short">Untitled Song</string>
<string name="song_count_format" translatable="false">%d songs</string>
<string name="more_options">More options</string>
<!-- Search Feature -->
<string name="hint_search_videos">Search music videos (e.g., song title, artist)</string>
<string name="button_search">Search</string>
<string name="status_enter_search_term">Please enter a search term.</string>
<string name="content_desc_channel_thumbnail">Channel Thumbnail</string>
<!-- Playback Notification / Media Session related -->
<string name="playback_notification_channel_name">Music Playback</string>
<string name="playback_notification_channel_description">Controls for music currently play
<!-- Potential error messages or states -->
<string name="error_stream_resolution_failed">Could not load stream.</string>
<string name="error_playback_failed">Playback error occurred.</string>
<string name="error_player_service_not_ready">Player service not ready. Please try again.<
<string name="error_playback_command_failed">Playback command failed (code: %d).</string>
<string name="error_initiating_playback">Error initiating playback.</string>
<!-- For MediaItem default descriptions or titles if primary ones are missing -->
<string name="default_media_item_title">Untitled Track</string>
<string name="default_media_item_artist">Unknown Artist</string>
<string name="action_previous">Previous</string>
<string name="action_play">Play</string>
<string name="action_pause">Pause</string>
<string name="action_next">Next</string>
<string name="action_like">Like</string>

<string name="action_back">Back</string>
<string name="nothing_selected_to_play">Nothing selected to play.</string>
<string name="action_shuffle">Shuffle</string>
<string name="action_repeat">Repeat</string>
<string name="action_view_queue">View Queue</string>

<string name="settings_title">Settings</string>
<string name="settings_section_api_key">API Key</string>
<string name="settings_section_cache">Cache Management</string>
<string name="settings_button_clear_cache">Clear All Application Cache</string>
<string name="settings_desc_clear_cache">This will remove all downloaded media, cached API
<string name="settings_section_about">About</string>
<string name="settings_app_version">App Version: %s</string>
<string name="control_panel_search_videos">Search Videos</string>
```

```
<string name="control_panel_browse_category">Browse by Category</string>

<string name="status_api_key_required_main">API Key is required to use the app. Please set
<string name="button_go_to_settings">Go to Settings</string>
<string name="status_no_videos_for_category">No videos found for the selected category.</s
<string name="category_favorites">Favorites</string>
<string name="action_unlike">Unlike</string>

<string name="action_clear_search">Clear search text</string>
<string name="search_history_title">Recent Searches</string>
<string name="action_clear_history">Clear History</string>
<string name="content_desc_search_history_item">Search history item</string>

<string name="settings_label_version">Version</string>
<string name="settings_label_powered_by">Powered By / Acknowledgements</string>
<string name="settings_link_holodex">Holodex.net</string>
<string name="settings_link_newpipe">NewPipeExtractor</string>
<string name="settings_section_theme">Appearance</string>
<string name="settings_theme_light">Light</string>
<string name="settings_theme_dark">Dark</string>
<string name="settings_theme_system">Follow System</string>
<string name="content_desc_external_link">Open external link</string>

<string name="dialog_title_create_playlist">Create New Playlist</string>
<string name="hint_playlist_name">Playlist Name</string>
<string name="hint_playlist_description_optional">Description (Optional)</string>
<string name="create">Create</string>

<string name="dialog_title_add_to_playlist">Add to playlist</string>
<string name="action_create_new_playlist_dialog">Create new playlist?</string>
<string name="action_more_options">More options</string>
<string name="action_add_to_playlist_menu">Add to playlist</string>
<string name="apply_filters_button">Apply Filters</string>
<string name="screen_title_playlist_details">Playlist Details</string>
<string name="message_playlist_is_empty">This playlist is empty.</string>
<string name="action_play_item">Play item</string>
<string name="action_remove_from_playlist">Remove from playlist</string>
<plurals name="item_count">
    <item quantity="one">%d item</item>
    <item quantity="other">%d items</item>
</plurals>
<string name="action_play_all">Play All</string>
<string name="action_play_all_playlist">Play all items in playlist</string>
<string name="message_no_favorited_videos">You haven\'t favorited any videos yet.</string>
<string name="message_no_liked_segments">You haven\'t liked any song segments yet.</string
<string name="action_play_all_liked_segments">Play All Liked Segments</string>
<string name="screen_title_playlists">My Playlists</string>
<string name="action_create_playlist">Create Playlist</string>
<string name="message_no_playlists_yet">No playlists yet. Tap the \'+\' button to create o
<string name="action_delete_playlist">Delete playlist</string>
<string name="category_display_name_search_results">Search: %s</string>
<string name="search_prompt_start">Search for music!</string>
<string name="playlist_title_liked_segments">Liked Segments</string>
<string name="playlist_desc_liked_segments">All your liked song segments</string>
<string name="message_no_liked_segments_in_playlist_view">You haven\'t liked any song segm
```

```
<string name="search_your_music_hint">Search your music</string>
<string name="action_menu">Menu</string>
<string name="action_filter_sort">Filter or Sort</string>
<string name="bottom_nav_browse">Browse</string>
<string name="bottom_nav_favorites">Favorites</string>
<string name="bottom_nav_playlists">Playlists</string>
<string name="bottom_nav_settings">Settings</string>
<string name="screen_title_favorites">My Favorites</string>
<string name="category_liked_segments">Liked Segments</string>
<string name="bottom_nav_search">Search</string>
<string name="screen_title_search">Search Music</string>
<string name="status_no_videos_for_filter">No videos found for the current filters.</strin

<!-- For FullPlayerScreen -->
<string name="content_desc_volume">Volume control</string>
<string name="action_hide_lyrics">Hide lyrics</string>
<string name="action_show_lyrics">Show lyrics</string>
<string name="action_show_player">Show player</string>
<string name="action_audio_settings">Audio settings</string>
<string name="action_view_artist">View artist</string>
<string name="action_view_album">View album</string>

<!-- General Player Strings -->
<string name="content_desc_album_art">Album artwork</string>
<string name="loading_track">Loading track?</string>
<string name="now_playing">Now Playing</string>
<string name="up_next_queue_title">Up Next</string>
<string name="empty_queue">Queue is empty</string>
<string name="action_clear_queue">Clear Queue</string>
<string name="action_remove_from_queue">Remove from queue</string>
<string name="currently_playing_indicator">Currently Playing</string>
<string name="action_move_up">Move up</string>
<string name="action_move_down">Move down</string>
<string name="content_desc_navigate_back">Navigate back</string>
<string name="content_desc_like_button">Like button</string>
<string name="content_desc_unlike_button">Unlike button</string>
<string name="content_desc_view_queue">View queue</string>
<string name="content_desc_more_options">More options</string>
<string name="content_desc_lyrics_toggle">Toggle lyrics</string>
<string name="content_desc_shuffle_button">Shuffle button</string>
<string name="content_desc_skip_previous_button">Skip to previous</string>
<string name="content_desc_play_pause_button">Play or Pause</string>
<string name="content_desc_skip_next_button">Skip to next</string>
<string name="content_desc_repeat_button">Repeat mode</string>
<string name="lyrics_not_available">Lyrics not available for this track.</string> <!-- ADD
<!-- For MainBrowseScreen & SearchScreen -->
<string name="status_search_no_results">No results found for \'%s\'.</string>
<string name="action_refresh">Refresh</string>
<string name="action_retry">Retry</string>
<string name="unknown_error_occurred">An unknown error occurred</string>
<string name="message_youve_reached_the_end">You\'ve reached the end!</string>
<string name="action_search">Search</string>
<string name="scroll_to_top">Scroll to top</string>
<string name="loading_content_message">Loading content?</string>
<string name="status_no_videos_for_filter_try_refresh">No music found for these filters. T
```

```
<string name="error_loading_generic">Could not load content: %s</string>
<string name="error_search_failed">Search failed: %s</string>
<string name="status_no_results_try_refresh">No results for \'%s\'. Try refreshing.</strin
<string name="error_playlist_not_found">Playlist not found.</string>

<string name="settings_section_data_performance">Data and Performance</string>
<string name="error_artist_info_unavailable">Artist/Channel information not available.</st
<string name="original_artist_label">(Original Artist)</string>
<string name="channel_label">(Channel)</string>
<string name="settings_label_image_quality">Image Quality</string>
<string name="settings_desc_image_quality">Lower quality reduces data usage and may speed
<string name="settings_image_quality_auto">Auto (Recommended)</string>
<string name="settings_image_quality_medium">Medium (Faster loading)</string>
<string name="settings_image_quality_low">Low (Data saver)</string>

<string name="settings_label_audio_quality">Audio Quality</string>
<string name="settings_desc_audio_quality">Select preferred audio quality for streaming. L
<string name="settings_audio_quality_best">Best Available</string>
<string name="settings_audio_quality_standard">Standard (~128kbps)</string>
<string name="settings_audio_quality_saver">Data Saver (~64kbps)</string>
<string name="action_add_to_queue_short">Add to Queue</string>
<string name="error_no_equalizer_app_found">No equalizer app found.</string>
<string name="error_no_audio_session">Audio session not available.</string>
<!-- NEW: Strings for Autoplay feature -->
<string name="settings_section_playback">Playback</string>
<string name="settings_label_autoplay_next_video">Autoplay next video</string>
<string name="settings_desc_autoplay_next_video">Automatically play the next song in the q
<string name="settings_label_data_loading">List Loading Intensity</string>
<string name="settings_desc_data_loading">"Normal" preloads more items for smoother scroll
<string name="settings_data_loading_normal">Normal (Smooth scrolling)</string>
<string name="settings_data_loading_reduced">Reduced (Less data usage)</string>
<string name="settings_label_list_loading_config">List Loading</string>
<string name="settings_desc_list_loading_config">"Adjust how aggressively lists load data.
<string name="settings_label_shuffle_on_play">Shuffle on Play</string>
<string name="settings_desc_shuffle_on_play">Automatically shuffle any new album, playlist
<string name="login_with_discord">Login with Discord</string>
<string name="settings_section_account">Account</string>
<string name="action_login">Login with Discord</string>
<string name="action_logout">Logout</string>
<string name="settings_desc_login">Log in to synchronize your history, likes, and playlist
<string name="content_desc_sync_icon">Sync status icon</string>

<!-- Display names for ListLoadingConfigOptions will be in SettingsScreen.kt enum -->

<string name="settings_label_buffering_strategy">Playback Buffering</string>
<string name="settings_desc_buffering_strategy">"Controls how much audio is buffered befor

<plurals name="queue_items_count">
    <item quantity="one">%d song</item>
    <item quantity="other">%d songs</item>
</plurals>

<string name="empty_queue_description">Add songs from the browse or search screens to get
<string name="drag_to_reorder">Drag to reorder</string>
<string name="remove_from_queue">Remove from queue</string>
```

```
<!-- For VideoDetailsScreen -->
<string name="video_thumbnail_description">Video thumbnail</string>
<string name="no_song_segments_available">No song segments are available for this video.</

<!-- Using plurals for song count for better localization -->
<plurals name="song_count">
    <item quantity="one">%d song</item>
    <item quantity="other">%d songs</item>
</plurals>
<string name="playlist_title_downloads">Downloads</string>
<string name="playlist_desc_downloads">All your downloaded songs</string>
<!-- For Toast message -->
<string name="added_songs_to_queue">Added %d songs to queue</string>
<string name="bottom_nav_home">Home</string>
<string name="bottom_nav_library">Library</string>
<string name="bottom_nav_downloads">Downloads</string>
<string name="message_no_favorites_or_segments">Your favorites and liked segments will app
<string name="action_download_all">Download All Segments</string>
<string name="search_results_title">Results for: %s</string>
<string name="action_clear_all">Clear</string>
<string name="settings_label_storage_location">Storage Location</string>
<string name="settings_desc_location_default">App-private storage (Default)</string>
<string name="settings_desc_location_custom">Custom Location: %s</string>
<string name="settings_button_reset_to_default">Reset to Default</string>
<string name="settings_label_download_location">Download Location</string>
<string name="settings_desc_download_location">Choose a folder for downloaded audio. Defau
<string name="settings_download_location_default">Default (App-private storage)</string>
<string name="action_clear_location">Clear custom location</string>
<string name="action_download">Download</string>
<string name="toast_download_started">Download started!</string>
<string name="status_download_failed">Download failed</string>

<!-- Titles for different download states in the notification -->
<plurals name="download_notification_title_in_progress">
    <item quantity="one">Downloading %d file</item>
    <item quantity="other">Downloading %d files</item>
</plurals>
<plurals name="download_notification_title_failed">
    <item quantity="one">%d download failed</item>
    <item quantity="other">%d downloads failed</item>
</plurals>
<string name="download_notification_title_completed">All downloads complete</string>
<string name="download_notification_text_failed">Could not download content. Tap to see de


<!-- Download Notification Channel details -->
<string name="download_notification_channel_name">Holodex Downloads</string>
<string name="download_notification_channel_description">Notifications for Holodex song do

<!-- Download Notification messages -->
<string name="download_notification_no_active_downloads">No active downloads</string>
<string name="download_notification_paused_reason">Downloads paused: %1$s</string>
<string name="download_notification_downloading_items_progress">Downloading %1$d items (%2
<string name="download_notification_processing_downloads">Processing downloads</string>
```

```
<!-- Download Requirement messages -->
<string name="download_requirement_network_connection">Network connection</string>
<string name="download_requirement_wifi_connection">Wi-Fi connection</string>
<string name="download_requirement_device_charging">Device charging</string>
<string name="download_requirement_battery_not_low">Battery not low</string>
<!-- Download Screen Strings -->
<string name="action_view_as_list">View as List</string>
<string name="action_view_as_grid">View as Grid</string>
<string name="search_your_downloads_hint">Search your downloads?</string>
<string name="action_cancel_download">Cancel download</string>
<string name="action_resume_download">Resume download</string>
<string name="action_retry_download">Retry download</string>
<string name="action_delete">Delete</string>
<string name="message_no_search_results_downloads">No downloads match your search</string>
<string name="message_no_downloads">No downloads yet\n\nStart downloading songs to see the

<!-- Download Status Strings -->
<string name="status_queued">Queued</string>
<string name="status_downloading">Downloading</string>
<string name="status_paused">Paused</string>
<string name="status_completed">Completed</string>
<string name="status_failed">Failed</string>
<string name="status_deleting">Deleting?</string>



<!-- Additional Action Strings -->
<string name="action_clear">Clear</string>
<string name="action_resume">Resume</string>
<string name="action_cancel">Cancel</string>

<!-- Progress and Status Messages -->
<string name="download_progress_percent">%1$d%%</string>
<string name="download_speed_format">%1$s/s</string>
<string name="download_eta_format">%1$s remaining</string>



<!-- Error Messages -->
<string name="error_download_failed">Download failed</string>
<string name="error_download_cancelled">Download cancelled</string>
<string name="error_network_unavailable">Network unavailable</string>
<string name="error_storage_full">Storage full</string>

<!-- Accessibility Strings -->
<string name="content_description_download_artwork">Artwork for %1$s</string>
<string name="content_description_download_progress">Download progress: %1$d percent</stri
<string name="content_description_download_status">Download status: %1$s</string>

<string name="screen_title_history">History</string>
<string name="message_no_history">Your listening history is empty.\n\nPlayed songs will ap
<string name="recently_played_songs">Recently Played Songs</string>
<string name="action_playAll">Play All</string>
<string name="action_add_to_queue">Add to Queue</string>
<plurals name="song_count_label">
```

```
        <item quantity="one">%d song</item>
        <item quantity="other">%d songs</item>
    </plurals>
    <string name="category_favorite_channels">Favorite Channels</string>

    <string name="bottom_nav_discover">Discover</string>
    <string name="action_show_more">Show More</string>
    <string name="shelf_title_for_you">For You</string>
    <string name="shelf_title_trending_songs">Trending Now</string>
    <string name="shelf_title_recent_streams">Recent Live Performances</string>
    <string name="shelf_title_fan_playlists">Community Mixes</string>
    <string name="shelf_title_artist_radios">Artist Radios</string>
    <string name="shelf_title_daily_mixes">Daily Mixes</string>
    <string name="screen_title_more_results">More Results</string>
    <string name="shelf_title_recent_streams_favorites">Recent streams from your favorites</st
    <string name="content_desc_album_art_for">Album artwork for %1$s</string>

    <string name="song_is_already_correctly_downloaded">\'%1$s\' is already correctly download
    <string name="metadata_incorrect_reprocessing">Metadata incorrect, re-processing \'%1$s\'?
    <string name="already_in_download_queue">\'%1$s\' is already in the download queue.</strin
    <string name="retrying_export_for">Retrying export for \'%1$s\'?</string>
    <string name="starting_download_for">Starting download for \'%1$s\'?</string>
    <string name="verifying_songs_for_download">Verifying %1$d songs for download/repair?</str


    <string name="sgp_artist_radio_title">%1$s Radio</string>
    <string name="sgp_artist_radio_desc">Radio featuring %1$s</string>
    <string name="sgp_daily_mix_title">Daily Mix: %1$s</string>
    <string name="sgp_daily_mix_desc">Your daily dose of %1$s</string>
    <string name="sgp_mv_random_title">Best of %1$s</string>
    <string name="sgp_mv_latest_title">Recent %1$s Covers &amp;amp; Originals</string>
    <string name="sgp_mv_random_desc">Relive the top hits from %1$s</string>
    <string name="sgp_mv_latest_desc">Latest released covers &amp;amp; originals from %1$s</string
    <string name="sgp_latest_title">Catch up on %1$s</string>
    <string name="sgp_latest_desc">Latest tagged songs in %1$s</string>
    <string name="sgp_weekly_mix_title">%1$s Weekly Mix</string>
    <string name="sgp_weekly_mix_desc">Explore this week in %1$s</string>
    <string name="sgp_my_weekly_mix_title">My Weekly Mix</string>
    <string name="sgp_my_weekly_mix_desc">Crafted for you based on your listening habits</stri
    <string name="sgp_history_title">Recently Played</string>
    <string name="sgp_history_desc">Your recently played songs</string>
    <string name="sgp_hot_title">Trending Songs</string>
    <string name="sgp_hot_desc">Trending songs radio</string>
    <string name="sgp_video_desc">Sang by %1$s</string>
    <string name="sgp_daily_mix_type">Daily Mix</string>
    <string name="sgp_weekly_mix_type">Weekly Mix</string>
    <string name="sgp_radio_type">Radio</string>
    <string name="action_share">Share</string>
    <string name="action_view_video">Go to Video</string>
</resources>

// File: res\values\Theme.xml
<resources xmlns:tools="http://schemas.android.com/tools">
    <!-- Try this parent name -->
    <style name="Base.Theme.Holodex" parent="Theme.Material3.DayNight.NoActionBar">
```

```
        <!-- OR, if the above doesn't work, try the direct Material Components one (less likel
        <!-- <style name="Base.Theme.Holodex" parent="Theme.MaterialComponents.DayNight.NoActi
        <!-- Note: Theme.Material3.DayNight.NoActionBar IS the correct one for M3 -->

        <!-- Your items -->
        <item name="android:windowFullscreen">true</item>
        <item name="android:windowContentOverlay">@null</item>
        <item name="android:windowTranslucentStatus">false</item>
        <item name="android:windowTranslucentNavigation">false</item>
        <item name="android:fitsSystemWindows">false</item>
        <item name="android:windowLayoutInDisplayCutoutMode" tools:targetApi="p">shortEdges</i
    </style>

    <style name="Theme.Holodex" parent="Base.Theme.Holodex" />

    <!-- SplashScreen theme -->
    <style name="Theme.App.Starting" parent="Theme.SplashScreen">
        <item name="windowSplashScreenBackground">@color/primary_container_light</item>
        <item name="windowSplashScreenAnimatedIcon">@drawable/ic_launcher_foreground</item>
        <item name="postSplashScreenTheme">@style/Theme.Holodex</item>
    </style>
</resources>

// File: res\values-night\themes.xml
<resources xmlns:tools="http://schemas.android.com/tools">
    <style name="Base.Theme.Holodex" parent="Theme.Material3.DayNight.NoActionBar">
        <!-- Parent should be the same, DayNight handles the switch -->
        <!-- ... your dark theme specific attributes if any, usually handled by Compose ... --
    </style>
    <!-- No need to redefine Theme.Holodex here if it just inherits Base.Theme.Holodex -->
    <!-- If you had a Theme.App.Starting for night, it would go here too -->
</resources>

// File: res\xml\backup_rules.xml
<?xml version="1.0" encoding="utf-8"?><!--
   Sample backup rules file; uncomment and customize as necessary.
   See https://developer.android.com/guide/topics/data/autobackup
   for details.
   Note: This file is ignored for devices older than API 31
   See https://developer.android.com/about/versions/12/backup-restore
-->
<full-backup-content>
    <!--
   <include domain="sharedpref" path="."/>
   <exclude domain="sharedpref" path="device.xml"/>
-->
</full-backup-content>

// File: res\xml\data_extraction_rules.xml
<?xml version="1.0" encoding="utf-8"?><!--
   Sample data extraction rules file; uncomment and customize as necessary.
   See https://developer.android.com/about/versions/12/backup-restore#xml-changes
   for details.
-->
<data-extraction-rules>
```

```
    <cloud-backup>
        <!-- TODO: Use <include> and <exclude> to control what is backed up.
        <include .../>
        <exclude .../>
        -->
    </cloud-backup>
    <!--
    <device-transfer>
        <include .../>
        <exclude .../>
    </device-transfer>
    -->
</data-extraction-rules>
```