CSC-306: Writing Mobile Apps

Guess The Song

ReadMe

Bahawal Waraich

880114

Content

1 API Classes

1.1 AddressApi.kt

This class is used to fetch the details of the initial address of the user, which has been set to Bay Campus. The address string is passed to the Geocode API, which returns details pertaining to the address string. By analyzing this response, the latitude and longitude value are returned to the calling activity.

1.2 GameDirectionApi.kt

This class is used to get direction from the current location to the destination location. The HTTP Request parameters are as follows:

I. Two locations, represented by their respective latitude and longitude values.
II. Google Maps API key.
III. The mode of the direction, which is equal to "walking".

Firstly, The AsyncTask is used to do this HTTP Request. So a new inner class is created, which extends AsyncTask to execute this request in the background thread. Then, an instance of the DirectionListener is used to return the result to the calling activity. In the case where onPostExecute() method is called, the response is examined to get all LatLng points of the Direction. Then a List of these points is passed to the listener method onFetchedLatLng(list) to be delivered back to its calling activity.

1.3 UserLocationAPI.kt

This class is used to listen to user location updates. There 4 main components:

I. **UserLocationListener instance**: Delivers the fetched location to the caller Activity by calling the onUpdateLocation(location), that will pass the last updated location.

II. **LocationCallback:** It's used in the location request method, which will receive all location updates.

III. **requestLocationUpdates():** Method called by the Activity from its onResumed() lifecycle method. A new LocationRequest is created with the required updates interval and priority. Then, this LocationRequest instance is used with LocationCallback to acquire the location updates from LocationServices.getFusedLocationProviderClient. To get a response from this location, the location permissions must be allowed and the Location Service of the device is enabled.

IV. **stopLocationUpdates()**: A method called by the Activity from its onStop() lifecycle method to stop the location request. This behaviour reduces battery consumption.

2 Database Classes

The SQLite is used to save all the collected lyrics and its data.

2.1 DbHelper.kt:

This class extends SQLiteOpenHelper to create the database.

I. **onCreate(SQLiteDatabase sqLiteDatabase):** To create a database if it has not is already been created, by passing SQL statements to the passed instance sqliteDatabase.execSQL(). Creates a database with a table for the lyrics. The SQL statement includes all the columns names and values of this table. The table includes columns for all the data lyrics plus the attempts state of this lyric.

II. **onUpgrade(SQLiteDatabase sqLiteDatabase, int i, int i1):** Method is invoked in the case when upgrading the database is required. This is done by dropping the saved table and creating it again from scratch.

2.2 LyricContract.kt

This class implements the BaseColumns interface. It contains all columns names, the table name, and the content authority. All these values are saved as constants to be used later easily to prevent any errors. The following are the columns names and their values:

I. **COLUMN_ID:** contains an auto-generated incrementally increasing integer value that is used as the primary key of the table. Unique value and used to identify each row in the table.

II. **COLUMN_TYPE :** contains not null String of the type of the lyric: current or classic.

III. **COLUMN_LYRIC:** contains not null String of the lyric text.

IV. **COLUMN_ARTIST:** contains not null String of the artist.

V. **COLUMN_ALBUM:** contains not null String of the album.

VI. **COLUMN_SONG:** contains not null String of the song title.

VII. **COLUMN_RELEASED:** contains not null String of the released date.

VIII. **COLUMN_GENRE:** contains not null String of the genre.

IX. **COLUMN_LENGTH:** contains not null String of the length.

X. **COLUMN_IS_SOLVED:** contains String of values "yes" or "no" which acts as a Boolean value to indicate if this lyric is solved or not. Its default value is "no".

XI. **COLUMN_IS_HALVED:** The same as the COLUMN_IS_SOLVED to indicate if the current attempt is halved or not.

XII. **COLUMN_IS_POINT_BOOSTED:** The same as the COLUMN_IS_SOLVED to indicate if the current attempt is point boosted or not.

XIII. **COLUMN_ATTEMPTS_LEFT:** contains an integer with a default value of 2, indicating the number of attempts left.

2.3 LyricProvider.kt

This class is declared in Manifest to receive database CRUD functions invoked by Content Resolver with the same Content Authority declared in the Manifest of the "lyrics" table. This class inherits from Content Provider and implements its abstract methods to handle CRUD functions related to lyrics table properly.

I. **query():** Querys the table to get a cursor of the data with selection and selection arguments values.
II. **Insert():** Inserts a new row to the table and returns its uri appended with the inserted row id. If the row is not inserted successfully, returns null. The values of the inserted row are included in the parameter of type ContentValue. The ContentValue object contains key-value pairs.
III. **Update():** Updates the rows in the table which meet the conditions specified by selection and selection arguments with the values saved on the ContentValue object. Returns the number of the updated rows.
IV. **Delete():** Deletes the rows in the table which meet conditions specified by selection and selection argument. Returns the number of the deleted rows.

3 Shared Preferences

3.1 GameSharedPreferences.kt

The Default Shared Preference is used to save the game state so that it can be be easily accessed by activities without using a background thread, using put and set proper methods. Saves values of the following:

I. Selected Game Mode.
II. Number of Collected Lyrics.
III. Number of Points.
IV. Number of Powers.

4 UI Classes

4.1 MainActivity.kt

initializeGameMode(): Sets the initial value to the shared preference of the game mode.

buildButtonsListeners(): sets onClickListeners for all the buttons in this Activity. "Play" button listener will start GameActivity. "Rules" buttons will start RulesActivity and "Switch" Checked Change Listener will save the selected Game mode to the Shared Preferences.

4.2 RulesActivity.kt:

initUI(): initialises an array of the layout resources ids to be used with ViewPager in order to display the fragments with the proper layout resource.

initActions(): To add a PageChangeListener to the ViewPager to get the selected page. If it's the last fragment, the Skip button will be invisible and the text of the next button will be set to "Let's Go".

addPagination(): Method is used to update the colour of the dots with the selected page position.

4.3 GameActivity.kt

This activity contains the Bottom Navigation Bar to display 3 different Fragments. The selected Fragment id is saved on the onSaveInstanceState() method to restore the same Fragment in case of configuration changes.

setUpBottomNavigationBar(): Sets setOnNavigationItemSelectedListener to the bottom navigation bar to return an object of the Fragment to be displayed depending on the selected menu icon id.

replaceFragment(fragment: Fragment): Replaces the proper Fragment by using Support Fragment Transaction replace method.

4.4 LyricsFragment.kt

This fragment displays all user collected lyrics of the selected game mode. Firstly it queries all the lyrics from the database with the proper selection and selection arguments and uses Content Resolver to get these lyrics. The main components of this class:

I. **getGameMode():** Gets the selected Game mode from Shared Preferences.

II. **buildToolbar():** Sets the title of the toolbar and OnClickListener on the navigation icon to finish the parent activity in case this icon is clicked.

III. **queeryUserCollectedLyrics():** Method is invoked from onViewCreated() and onResume() methods, to force the Loader Manager to reload the lyrics from the database. This is ensures that the lyrics Lists is always up-to-date.

IV. **updateAdapter(cursor: Cursor?):** In the case the Loader has finished loading lyrics from the database, and the Recycler View will display the loaded data and restore its state.

V. **buildRecyclerView() :** Builds the Recycler View and initialises LinearLayoutManager with horizontal orientation.

VI. **buildUserData():** Gets the values of collected lyrics, Points and Power from Shared Preferences then update their TextViews with their values.

VII. **LoaderManager.LoaderCallbacks:** The Fragment implements the interface and methods which are used to query data from Database using Cursor Loader.

4.5 MapFragment.kt

This Fragment has multiple functionalities, which are listed below:

I. Uses AddressAPI to get the latitude and longitude of the initial address.

II. Uses GameLocationAPI to get user location updates.

III. Uses GameDirectionAPI to draw the direction between two points on the map.

IV. Reads all the lyric from the JSON file and query all collected lyrics by the user. Then creates a list of lyrics which have not been collected by the user yet to be displayed as music markers on the map.

V. When the user taps on a music marker, the distance between the user and the marker is calculated. If this distance within 10 meters, the marker lyric will be collected, otherwise, direction route will be fetched and displayed.

VI. UserLocationListener and DirectionListener interfaces are used to get back the responses of the GameLocationAPI and GameDirectionAPI.

These are the main methods:

I. **initializeValidLyrics():** Gets all the Lyrics which are not collected.

II. **setupMap():** To set up Google Map and initialise it.

III. **getInitialLocation():** Gets initial LatLng from AddressAPI.

IV. **updateUserMarker():** Sets the new position to the user marker then update the lyrics locations depending on the user markers, just in case their locations has not been initialised.

V. **updateLyricsMarkers():** Clears all lyrics markers and then creates new markers with their new positions.

VI. **drawRoute(list: MutableList<LatLng>):** Draws the direction route between the user and the selected lyric using the PolylineOptions() and the list of the LatLng points.

VII. **collectLyricAtMarker(p0: Marker):** Saves the lyric data on the database as a collected lyric. Then increases the number of collected lyrics in SharedPrefrence by 1.

VIII. **requestLocationPermissions():** Requests LocationPermission on runtime for devices with Marshmallow version or higher.

4.6 SongsFragment.kt

It has the same functionalities as LyricFragment. Displays all user songs which have been solved correctly of the selected game mode. Firstly, it queries all the lyrics from the database with the proper selection and selection arguments then uses Content Resolver to get these lyrics.

If the user clicked on one song, The SongInfoActivity will be started passing the selected song on the bundle of the intent

4.7 SongInfoActivity.kt

It displays all the info about the selected song. Displays its title, artist, album, released date, genre and lyric.

4.8 GuessSongActivity.kt

The main functionality of this class is to build a spinner, providing some suggestion to guess the song. These suggestions include one correct answer. The other suggestions are selected randomly from the lyrics list while ensuring no repeated items.

buildSpinner(): Builds the spinner with generated suggestions songs, using "android.R.layout.simple_spinner_item" as a view to display the item.

halveSpinnerList(): If the user clicks on the Halve image, this method will be invoked. It halves the suggestion list using the Collections methods to handle the deleting of some items and ensuring not to delete the correct answer.

useBoostPointsPower(): It checks if the attempt is already boosted, if it has then it will display a toast to notify the user. Also checks if the power is 0, if so, then a Toast message will be displayed to notify the user that they have no Powers left. Otherwise, the Points are boosted and the attempt will be updated as boosted and the user power will be decreased by 1.

useHalvePower(): It checks if the attempt has already been halved or if the power is 0, then notify the user appropriately. Otherwise, the suggestions will be halved and the attempt will be updated as halved on the database along with the user power being decreased by 1.

checkUserAnswer(): This method is invoked if the user clicks on the checkmark image that is used to check their answer. If the selected item by the spinner is the correct answer, a dialogue will be displayed to notify the user that is the right answer and the points on Shared Preferences will be updated, and the lyric will be updated on the database as solved. The power value will also be increased by 1. If the answer is incorrect, a dialogue will be displayed to notify the user and the number of attempts will be decreased by 1. If the number of attempts left is 0, then the lyric will be removed from the database and the activity will be finished (i.e. the user would have to collect the lyric again), otherwise, the user will get another attempt to solve it.