Quick Query

Introduction

Overview

The "Quick Query" application is a comprehensive mobile platform designed to deliver detailed, real-time country data directly to users' fingertips. Utilizing a rich public API, the application provides information such as demographics, economic currencies, and geographical data in a structured and user-friendly manner. The app harnesses the power of modern Android development techniques to ensure a responsive, intuitive user experience.

Purpose

Quick Query is aimed at users ranging from students and researchers to curious individuals seeking to gain insights into different countries around the world. Whether it's for educational purposes, travel planning, or just general knowledge, the app provides valuable data conveniently on your mobile device. Key functionalities include:

- **Data Presentation**: Display up-to-date information about countries including capitals, population, area, and more.
- **Caching Mechanism**: Efficient data retrieval and storage using a local database and an in-memory cache to enhance offline capabilities.
- **User Interaction**: Interactive elements such as searchable lists and detailed entries for over 190 countries.
- **Custom Views**: Integration of web views and custom layouts to render data aesthetically.

The target audience includes:

- **Educational Institutions:** Facilitate learning by integrating geographical and cultural education in classrooms.
- **Travelers**: Plan trips with detailed information about destinations.
- **Data Enthusiasts**: Explore and compare data on various countries for personal projects or knowledge.

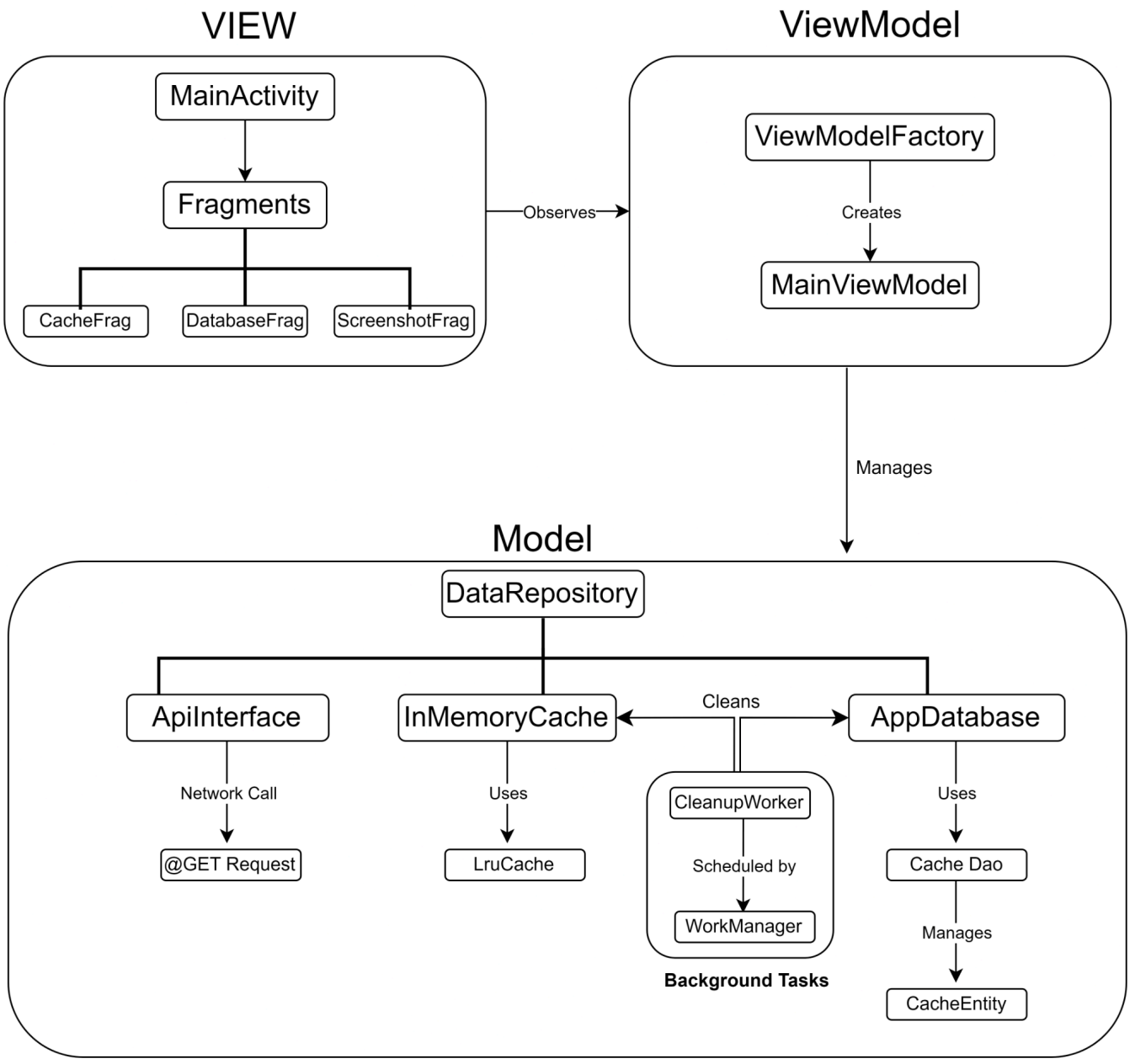
High-Level Architecture

Quick Query adopts the Model-View-ViewModel (MVVM) architecture pattern, which provides a robust and efficient way to structure Android applications. This pattern enhances the separation of concerns and improves the scalability and maintainability of the application. Here's how the application leverages MVVM:

- **Model:** This layer handles the business logic and data storage. It interacts with the database and network services to retrieve, process, and manage data. The model notifies the ViewModel of any data changes via LiveData or other observables.
- **View:** The View is responsible for rendering UI components and reacting to user inputs. It observes ViewModel to receive data updates and displays them accordingly. This layer includes activities, fragments, and custom views.
- **ViewModel:** It serves as the intermediary between the Model and the View. The ViewModel exposes streams of data relevant to the UI and handles user interactions by utilizing the Model. It manages the lifecycle-aware data flow to the View, ensuring that configuration changes such as rotations do not disrupt the user experience.

The MVVM pattern is particularly beneficial in this project for a few key reasons:

- **Decoupling of Components:** Views are isolated from the business logic, which means changes in ViewModel do not directly affect the View layer, making the codebase more robust and easier to manage.
- **Lifecycle Management:** ViewModel handles data-related operations based on the lifecycle of View components, reducing the risk of memory leaks and ensuring the UI state is preserved on configuration changes.
- **Improved Testability:** Each component can be tested independently, especially with the ViewModel handling most of the logic operations.



Flow Chart

Model

Components

- **AppDatabase** (AppDatabase.kt)
  Manages the application's SQLite database with the Room persistence library. It serves as an access point to the stored data by exposing various DAOs (Data Access Objects).

- **CacheDao** (CacheDao.kt)
  Interface providing access to the local cache stored in SQLite. It includes methods for retrieving, inserting, and deleting cached data.
- **DataRepository** (DataRepository.kt)
  Acts as an intermediary for network and database data transactions. It utilizes ApiInterface for network calls and CacheDao for local database interaction.

Functionality

- **Data Handling**
  The DataRepository fetches data through ApiInterface, which encapsulates all network operations with Retrofit. Depending on the operation, data is either fetched from the cache or database (using CacheDao) or directly from the network.
- **Synchronization**
  Regular syncs between the cached data and the server are handled by background workers (CleanupWorker), ensuring data consistency and freshness.

View

Components

- **MainActivity** (MainActivity.kt)
  Hosts the application's primary UI components. It initializes fragments and configures interactions between the ViewModel and user inputs.
- **Fragments** (CacheFragment.kt, DatabaseFragment.kt, ScreenshotFragment.kt)
  Manage specific user interfaces for cache viewing, database viewing, and displaying screenshots, respectively.

Functionality

- **User Interaction**
  Input is captured in MainActivity and relayed to the ViewModel.
- **UI Updates**
  Observes LiveData from the ViewModel to render items dynamically in response to data changes. Each fragment sets up its own RecyclerView and adapter to display data specific to its function.

ViewModel

Components

- **MainViewModel** (MainViewModel.kt)
  Connects UI components with the underlying data model. It handles UI logic, fetches data from the DataRepository, and prepares it for display.
- **ViewModelFactory** (ViewModelFactory.kt)
  Factory for creating ViewModel instances, ensuring they are equipped with necessary dependencies such as DataRepository.

Functionality

- **Data Management**
  MainViewModel requests data operations through DataRepository and handles LiveData to update the View based on data changes.
- **Lifecycle Handling**
  Ensures data is retained across configuration changes and manages data loading and error handling smoothly.

Integration and Interactions

- **Data Flow**
  Data flows from the DataRepository to MainViewModel and then to the UI (Activity/Fragment) via LiveData observables, ensuring that the UI reflects the most current data state.
- **Event Handling**
  User actions in the UI trigger events managed by the ViewModel, which uses the repository to fetch or update data, illustrating a clear separation of concerns.

Use of WorkManager

- **CleanupWorker** (CleanupWorker.kt)
  Scheduled via WorkManager to periodically clean up old cache entries from the database to prevent data bloating and maintain efficient performance.

Dependency Injection

- **Application Class Setup** (MyApp.kt)
  Initializes and provides dependencies like ApiInterface and AppDatabase universally throughout the app, facilitating easy access and management of singletons.

Libraries and External Tools

In the development of the Quick Query application, several external libraries and tools are utilized to enhance functionality, streamline the development process, and ensure robust architecture adherence. Below is a detailed explanation of each component integrated into the project's Gradle configuration:

Core Libraries

1. **AndroidX Libraries:**

- Core KTX: Kotlin extensions for the 'core' library, providing idiomatic Kotlin features for common tasks, simplifying code and reducing boilerplate.
- AppCompat: Ensures compatibility with various versions of Android, making it possible to use features on older API versions without manual backward-compatible checks.
- Activity KTX & ConstraintLayout: Facilitates modern, flexible, and performant layouts and interactions with components like activities.
- Material Components: Material Design components library that provides a wide range of ready to use material design UI components such as buttons, text fields, and much more that follow the material design guidelines.

- WebView: Provides a built-in browser to display web pages or to create complex user interfaces.
- WorkManager: Manages and schedules background tasks that need to run in a guaranteed manner, ensuring execution even if the app exits or the device restarts.

Network and Serialization

3. **Retrofit and OkHttp:**

- Retrofit: A type-safe HTTP client for Android and Java which makes it easier to consume RESTful web services.
- Gson Converter: A converter that uses Gson for serialization to and from JSON, facilitating the parsing and serialization of structured data like JSON directly into objects.
- OkHttp & Logging Interceptor: Provides an efficient HTTP client with capabilities to log network requests and responses which is crucial for debugging issues related to API interactions.

Asynchronous Programming

4. **Coroutines:**

- Kotlinx Coroutines Android: A rich library that supports coroutines, providing a way to perform asynchronous programming in a more linear, simplified manner, significantly improving the readability and maintainability of complex code.

Local Data Management

5. **Room Database:**

- Room Runtime & KTX: An abstraction layer on top of SQLite that allows for robust database access while harnessing the full power of SQLite.
- Room Compiler: Processes annotations from the Room library to generate code at compile time, reducing runtime overhead and potential for errors.

Miscellaneous

7. **Lottie:**

- Lottie: A library for Android, iOS, and React Native that parses Adobe After Effects animations exported as JSON and renders them natively on mobile.

Gradle Configuration Overview

This configuration ensures that the application is compiled against the latest SDK (SDK 34), targeting devices with a wide range of supported versions (minimum SDK 21), which covers nearly all active Android devices. The use of Java 1.8 compatibility mode allows the app to use Java 8 language features. Data Binding is enabled to decrease boilerplate and increase efficiency in UI data interactions.

Build and Deployment

Prerequisites

Before beginning the setup, ensure the following tools are installed and properly configured in your development environment:

- Android Studio: The official IDE for Android development, providing necessary tools for Android app development. Download and install from the Android Developer website.
- Java Development Kit (JDK): Ensure JDK 8 or newer is installed.

Clone the Project Repository

Using Git: 'git clone https://github.com/Krishna-Vamsi-G/Quick-Query.git'

1. This command clones the latest version of the Quick Query project into your local machine.

Using a ZIP File:

2. Alternatively, download the ZIP file of the project from the GitHub repository and extract it to your preferred location.

Project Setup in Android Studio

1. Open Android Studio and select Open an existing Android Studio project from the options.
2. Navigate to the directory where you have cloned or extracted the project.
3. Select the project root directory and click OK to open the project in Android Studio.

Resolve Dependencies

- Ensure all project dependencies are resolved. The Gradle sync should automatically handle dependency downloads and configurations.

Deployment Instructions

Prepare Device or Emulator

- Enable Developer options and USB debugging on the Android device to allow installations and debugging via USB. Click the Run icon or press Shift + F10 to compile, install, and run the application on the device.
- Use the AVD Manager in Android Studio to create or start an existing Android Virtual Device. Click the Run icon or press Shift + F10 to compile, install, and run the application on the emulator.

These detailed instructions provide a comprehensive guide to setting up, building, and deploying the Quick Query application using Android Studio.