**Udacity Android Kotlin Developer Nanodegree**

**Capstone Project**

**PoetryApp Design Document**

**Overview:**

The PoetryApp is targeted for users that love to read poems. The app offers a collection of poems from the PoetryDatabase (<https://poetrydb.org/index.html>) and also has a Poem of the Day feature which is provided by Poems.One (<https://poems.one/api/poem/#poem-of-the-day>) The user can browse poems, browse favorites, read poems, mark as favorite and share a poem. The user is notified when the Poem of the Day changes.

**Credits:**

1. [@thundercomb](https://twitter.com/thundercomb) <https://poetrydb.org/index.html>) for providing a free API to access poems in their database.
2. Poems.One <https://poems.one/api/poem/#poem-of-the-day> for free access to the Poem of the Day.
3. Rashmi Dinesh for the Welcome Screen image - <https://docs.google.com/uc?id=1Om4xDZ6rLvLkV_i5oQkK3Un08wAhGysh>
4. Vecteezy for the Poem Vector used on Welcome Screen (as placeholder), Sign-in screen and as the Notification icon.

*<a href="https://www.vecteezy.com/free-vector/poem">Poem Vectors by Vecteezy</a>*

**Milestone Schedule**

*This schedule was created after Step 1(Ideation) was completed. I work during the week. So most of my project work is done mainly on weekends*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Milestone** | **Tasks** | **Target Date** | **Actual Date** |
| 1. | Ideation- Figure out the custom app to build | -Look for public API available to build app.  - Go over Project requirements and Rubric.  -Have a prototype of app features/offerings that will satisfy the rubric.  - | 4/1/2021 | 4/2/2021 |
| 2. | Data modeling | -Design the database  -Create data classes, domain objects, database and network objects and their mappings.  -Create a repository with functions for fetching the data from the database and for refreshing data from the network | 4/3/2021 | 4/3/2021 |
| 3. | Services API | -Create two network services using Retrofit, one for the PoetryDB endpoint and another for the Poem Of the Day endpoint.  -Parse the service results to load correctly | 4/3/2021 | 4/4/2021 |
| 4. | UI and ViewModel Design and Development (Core functionality) | -MainActivity that handles Authentication using Firebase  -PoetryActivity with Fragments that handle browsing poems and navigating to a poem  -Implement Recycler view for Browsing poems  -Use nav-graph for Fragment navigation  -Use the viewModel to fetch data from the repository  -get all of this working for the PoetryDB | 4/4/2021 | 4/5/2021 |
| 5. | UI Additional features | -Marking favorites  -Adding s Share feature  -Add overflow menu for filtering favorites and displaying Poem of the Day and LogOut | 4/10/2021 | 4/11/2021 |
| 6. | Notification | -Work on generating a notification when Poem of Day changes | 4/11/2021 | 4/11/2021 |
| 7. | Making it all pretty | -Welcome screen image  -Follow Material design guidelines  -Incorporate Motion Layout | 4/13/2021 | 4/13/2021 |
| 8. | Update Design Document | Update design document with Capstone Key Areas and Rubric Requirements | 4/14/2021 | 4/14/2021 |

|  |  |
| --- | --- |
| **Capstone Key Areas** | **Features** |
| * Application Architecture | The PoetryApp uses the the MVVM pattern and consists of the UI, PoetryViewModel and the PoetryRepository, that provide a clean separation of concerns. |
| * UI and Layout | The UI consists of two main activities – the MainActivity which handles the Authentication and the PoetryActivity that hosts all the fragments associated with the Poems.  The MainActivity uses a LinearLayout, while the fragments of the PoetryActivity use a ConstraintLayout. Navigation between the fragments of the PoetryActivity is driven by a navgraph.  An overflow menu is provided for filtering the poems by Favorites and for accessing the Poem of the Day and to Logout of the app.  The PoemFragment used the MotionLayout to provide a top to down flowing effect when displaying the poem. Material icons have been used in the PoemFragment and the material design guidelines at <https://material.io/design/color/the-color-system.html#color-usage-and-palettes> have been applied. |
| API Connectivity and Data Persistence | The app makes three different API calls:  1) PoetryDatabse <https://poetrydb.org/index.html> to load all poems. This is done in the background and all the data is inserted into a Room Database. Hence, only for the first time, the user waits a few seconds for the poems to load. Subsequent loads are from the database and instantaneous, i.e. poems are stored for offline use.  2) Poem Of the Day API - <https://poems.one/api/poem/#poem-of-the-day> 3) Loading of the Welcome Screen image https://docs.google.com/uc?id=1Om4xDZ6rLvLkV\_i5oQkK3Un08wAhGysh |
| Hardware Integration | The app uses the Notification hardware component to inform the user when a new poem of the day is available. This is managed by storing the value in shared preferences and checking for changes to the value. |
| * User-based Functionality | The following features are provided to the user:  1) Mark a poem as a favorite and browse favorite poems. The Poem of the day can also be marked as a favorite and is stored for future retrieval.  2) Share a poem |

**Rubric Requirements**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Rubric Requirement** | **Specification** | **App Feature** |
| 1. | Build a navigable interface consisting of multiple screens of functionality and data. | Application includes at least three screens with distinct features using either the Android Navigation Controller or Explicit Intents.  The Navigation Controller is used for Fragment-based navigation and intents are utilized for Activity-based navigation.  An application bundle is built to store data passed between Fragments and Activities. | MainActivity, PoetryListFragment and PoemFragment.  nav\_graph is used  bundle is used to pass the selectedPoem to the PoemFragment. |
| 2. | Construct interfaces that adhere to Android standards and display appropriately on screens of different size and resolution. | Application UI effectively utilizes ConstraintLayout to arrange UI elements effectively and efficiently across application features, avoiding nesting layouts and maintaining a flat UI structure where possible.  Data collections are displayed effectively, taking advantage of visual hierarchy and arrangement to display data in an easily consumable format.  Resources are stored appropriately using the internal res directory to store data in appropriate locations including string\* values, drawables, colors, dimensions, and more.  Every element within ConstraintLayout should include the id field and at least 1 vertical constraint.  Data collections should be loaded into the application using ViewHolder pattern and appropriate View, such as RecyclerView. | The PoetryListFragment and PoemFragment both use ConstraintLayout  and the PoetryListFragment uses a RecyclerView and the ViewHolder pattern.  An overflow menu has been provided to allow users to filter poems by favorites and to view the poem of the day and for logging out of the app.  All resources are stored in the appropriate resource files. |
| 3. | Animate UI components to better utilize screen real estate and create engaging content. | Application contains at least 1 feature utilizing *MotionLayout* to adapt UI elements to a given function. This could include animating control elements onto and off screen, displaying and hiding a form, or animation of complex UI transitions.  *MotionLayout* behaviors are defined in a *MotionScene* using one or more *Transition* nodes and *ConstraintSet* blocks.  *Constraints* are defined within the scenes and house all layout params for the animation. | PoemFragment uses MotionLayout for a flowing effect as the Poem is rendered. |
| 4. | Connect to and consume data from a remote data source such as a RESTful API. | The Application connects to at least 1 external data source using **Retrofit** or other appropriate library/component and retrieves data for use within the application.  Data retrieved from the remote source is held in local models with appropriate data types that are readily handled and manipulated within the application source. Helper libraries such as **Moshi** may be used to assist with this requirement.  The application performs work and handles network requests on the appropriate threads to avoid stalling the UI. | The App connects to 2 different datasources <https://poetrydb.org/index.html> and <https://poems.one/api/poem/#poem-of-the-day>  Retrofit and the ScalarConvertedFactory is used to fetch data and data is parsed in the NetworkUtils class.  All network requests are performed on the background thread using the viewModel scope |
| 5. | Load network resources, such as Bitmap Images, dynamically and on-demand. | The Application loads remote resources asynchronously using an appropriate library such as **Glide** or other library/component when needed.  Images display placeholder images while being loaded and handle failed network requests gracefully.  All requests are performed asynchronously and handled on the appropriate threads. | The Welcome screen image is stored in a google docs link and is retrieved using Glide. A placeholder is used. |
| 6. | Store data locally on the device for use between application sessions and/or offline use. | The application utilizes storage mechanisms that best fit the data stored to store data locally on the device. Example: SharedPreferences for user settings or an internal database for data persistence for application data. Libraries such as [**Room**](https://developer.android.com/topic/libraries/architecture/room) may be utilized to achieve this functionality.  Data stored is accessible across user sessions.  Data storage operations are performed on the appropriate threads as to not stall the UI thread.  Data is structured with appropriate data types and scope as required by application functionality. | All poems once loaded from the network are stored in the Room Database and hence available for offline use. This has been verified by setting device on Air-Plane mode.  All inserts and updates to the database are done using Kotlin co-routines.  Preferences have also been used in this app to store the title of the Poem of the Day, so that a Notification can be triggered for the user, when the poem changes. |
| 7. | Architect application functionality using MVVM. | Application separates responsibilities amongst classes and structures using the MVVM Pattern:   * *Fragments*/*Activities* control the *Views* * Models houses the data structures, * *ViewModel* controls business logic.   Application adheres to architecture best practices, such as the observer pattern, to prevent leaking components, such as Activity Contexts, and efficiently utilize system resources. | The MVVM pattern has been adhered to using the UI, PoetryViewModel and the PoetryRepository.  The Observer pattern has been followed in several places such as for displaying poems based upon the filter selected, for detecting a change to the Poem of the Day and for setting various navigation and feature(favorite) flags. |
| 8. | Implement logic to handle and respond to hardware and system events that impact the Android Lifecycle. | Beyond MVVM, the application handles system events, such as orientation changes, application switching, notifications, and similar events gracefully including, but not limited to:   * Storing and restoring state and information * Properly handling lifecycle events in regards to behavior and functionality   + Implement bundles to restore and save data * Handling interaction to and from the application via *Intents* * Handling Android Permissions | Orientation changes have been tested and handled.  Notification has been handled to navigate to the desired destination.  Intents have been used for activity navigation as well for the Share feature.  The app uses the Android INTERNET permission. |
| 9. | Utilize system hardware to provide the user with advanced functionality and features. | Application utilizes at least 1 hardware component to provide meaningful functionality to the application as a whole. Suggestion options include:   * Camera * Location * Accelerometer * Microphone * Gesture Capture * Notifications   Permissions to access hardware features are requested at the time of use for the feature.  Behaviors are accessed only after permissions are granted. | The app uses the Notifications component notifying the user when a new Poem of the Day is available. |