**Project Specification**

**Political Preparedness/Custom Application**

Android UI/UX

| **Criteria** | **Meets Specifications** |
| --- | --- |
| 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. |
| 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. |
| 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. |

Local and Network data

| **Criteria** | **Meets Specifications** |
| --- | --- |
| 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. |
| 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. |
| 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. |

Android system and hardware integration

| **Criteria** | **Meets Specifications** |
| --- | --- |
| 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. |
| 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 |
| 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. |