Requirements

Fixed:

1. Max 2 page description of app
   1. Create mock screens for all user facing screens
2. Develop the app with functionality
   1. Min 2 distinct screens
   2. Use navigation component (move between screens)
   3. Use Intent to move outside of app
   4. Work with app lifecycle (e.g. rotate screen change)
   5. Use permissions
   6. Use local storage
   7. Create + use own ContentProvider
3. Use 3 of the following
   1. **Use Firebase (store and retrieve data)**
   2. Use Broadcast events
   3. Use Android Sharesheet
   4. Use your own Service
   5. **Use Notifications**
   6. **Adapt app for diff. screen sizes**
   7. Use touch gestures
4. App demo
   1. < 15 mins
   2. App intro, demo, implementation, reflective comments

**1. General Guidelines**

* **Code Structure:** Follow MVVM (Model-View-ViewModel) architecture to ensure a clean separation of concerns and easier management of UI components and data.
* **Naming Conventions:**
  + **Classes and Interfaces:** Use UpperCamelCase and should reflect their purpose clearly (e.g., **UserProfileViewModel**).
  + **Functions and Variables:** Use lowerCamelCase and be descriptive about the purpose (e.g., **loadUserProfile()**).
  + **Constants:** Use ALL\_CAPS with underscores (e.g., **MAX\_USER\_COUNT**).

**2. Code Formatting**

* **Indentation:** Use 4 spaces per indentation level.
* **Braces:** Use K&R style, where opening braces are on the same line as the statement and closing braces on their own line aligned with the statement.
* **Line Length:** Keep line length to a maximum of 100 characters.

**3. Documentation**

* **In-line Comments:** Use them sparingly to explain "why" something is done, not "what" is done.
* **Function/Method Comments:** Every function/method should have a comment explaining its purpose, parameters, and return value.
* **Class Comments:** Each class should have a top-level comment explaining its role within the app.

**4. Error Handling**

* **Consistent Strategy:** Use try/catch blocks judiciously and propagate errors that cannot be handled meaningfully.
* **Logging:** Use a logging framework (like Timber for Kotlin) for debugging purposes; ensure sensitive information is not logged.

**5. Version Control**

* **Commits:** Make small, atomic commits with clear, descriptive messages.
* **Branching:** Use feature branches for new features or bug fixes, merge back to the main branch using pull requests.

**6. Performance Optimization**

* **Avoid Overuse of Resources:** Be mindful of memory and battery use; avoid unnecessary background processing.
* **Efficient Data Handling:** Use efficient data structures and algorithms for processing data.

**7. Security Practices**

* **Data Storage:** Use encrypted storage (like EncryptedSharedPreferences) for sensitive data.
* **Network Security:** Ensure all network connections are secured using HTTPS.

**8. User Interface**

* **Consistency:** Follow Material Design guidelines consistently across all screens.
* **Responsiveness:** Design layouts to work across different device sizes using constraint layouts.