**A-C.**

**SEE PROJECT FILES**

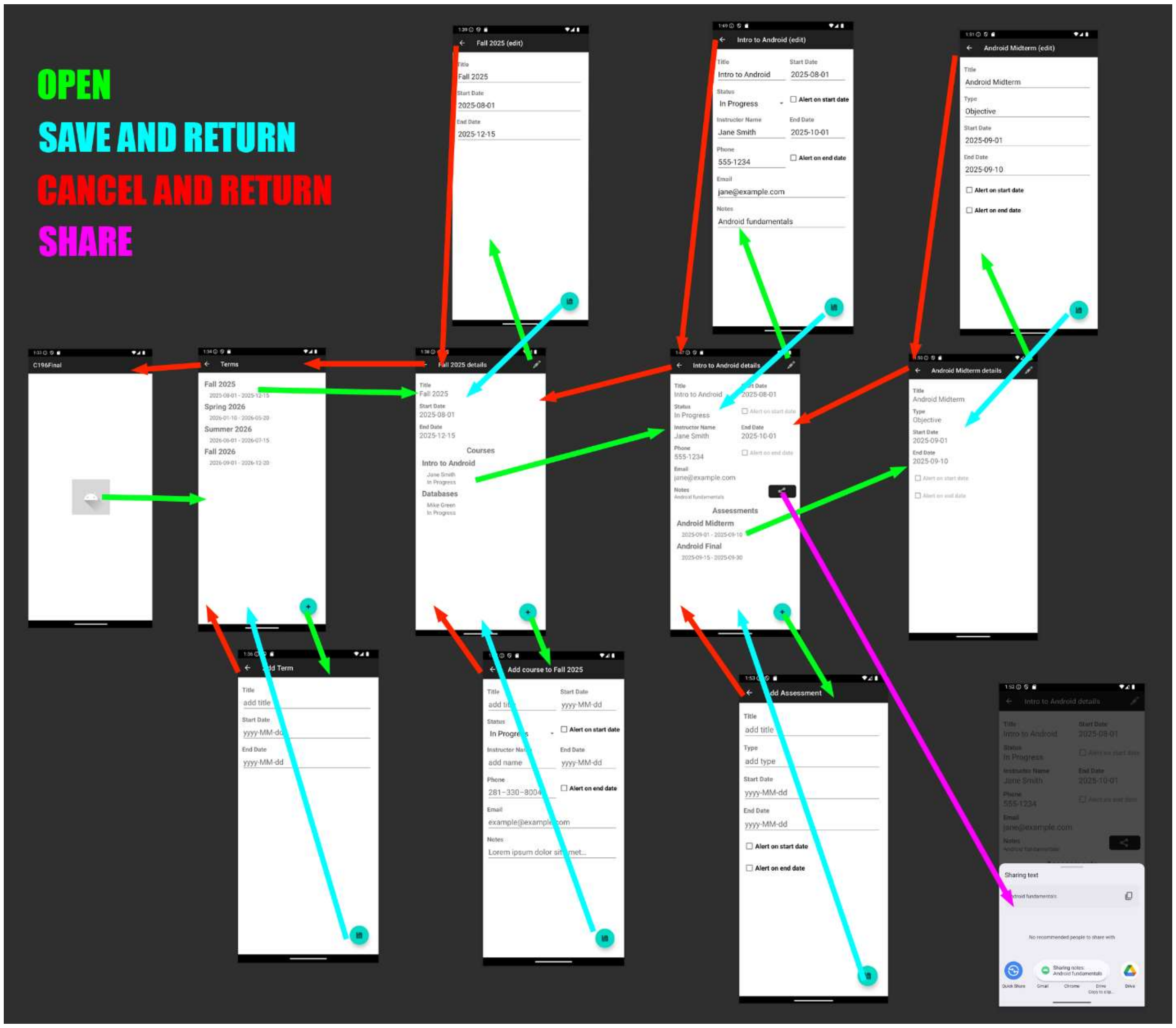
**Notes:**

ArrayList is used explicitly in CourseDetailActivity for storing assessments (ArrayList<Assessment>). It is unclear if a regular List counts as an ArrayList for grading purposes, hence its inclusion.

All screens work in portrait and landscape. I added layout-land XMLs where default portrait layouts caused visual issues; otherwise, the default portrait layout adapts correctly in landscape.

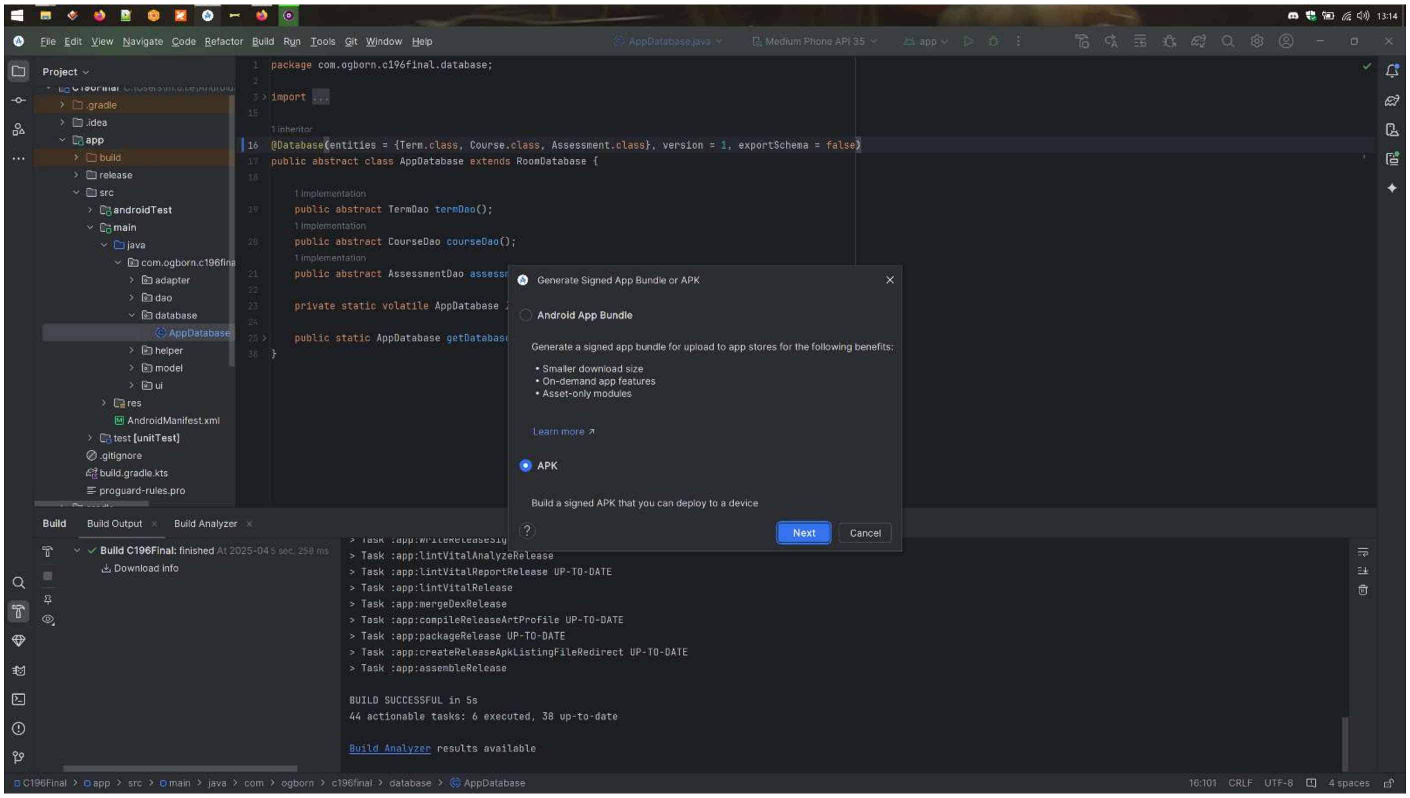
RecyclerViews provide vertical scrolling across the app. If this does not meet the "scroll vertically" requirement, please advise and I'll wrap content in a ScrollView.

**D. Create a storyboard to demonstrate application flow that includes each of the menus and screens from part B.**



**E. Provide screen shots of generating the signed APK to demonstrate that you have created a deployment package.**

If for some reason the screenshots are not sufficiently readable, **please find full-res screenshots in the /images/ directory in zip root.** The generated signed apk can be found at {root of zip}/app/release/app-release.apk. A copy will also be found in the zip root.



```
package com.ogborn.c196final.database;

import ...

1 inheritor
@Database(entities = {Term.class, Course.class, Assessment.class}, version = 1, exportSchema = false)
public abstract class AppDatabase extends RoomDatabase {

    1 implementation
    public abstract TermDao termDao();
    1 implementation
    public abstract CourseDao courseDao();
    1 implementation
    public abstract AssessmentDao assessm

    private static volatile AppDatabase 

    public static AppDatabase getDatabase
}
```

Generate Signed App Bundle or APK

( ) Android App Bundle

Generate a signed app bundle for upload to app stores for the following benefits:

- Smaller download size
- On-demand app features
- Asset-only modules

Learn more ↗

(●) APK

Build a signed APK that you can deploy to a device

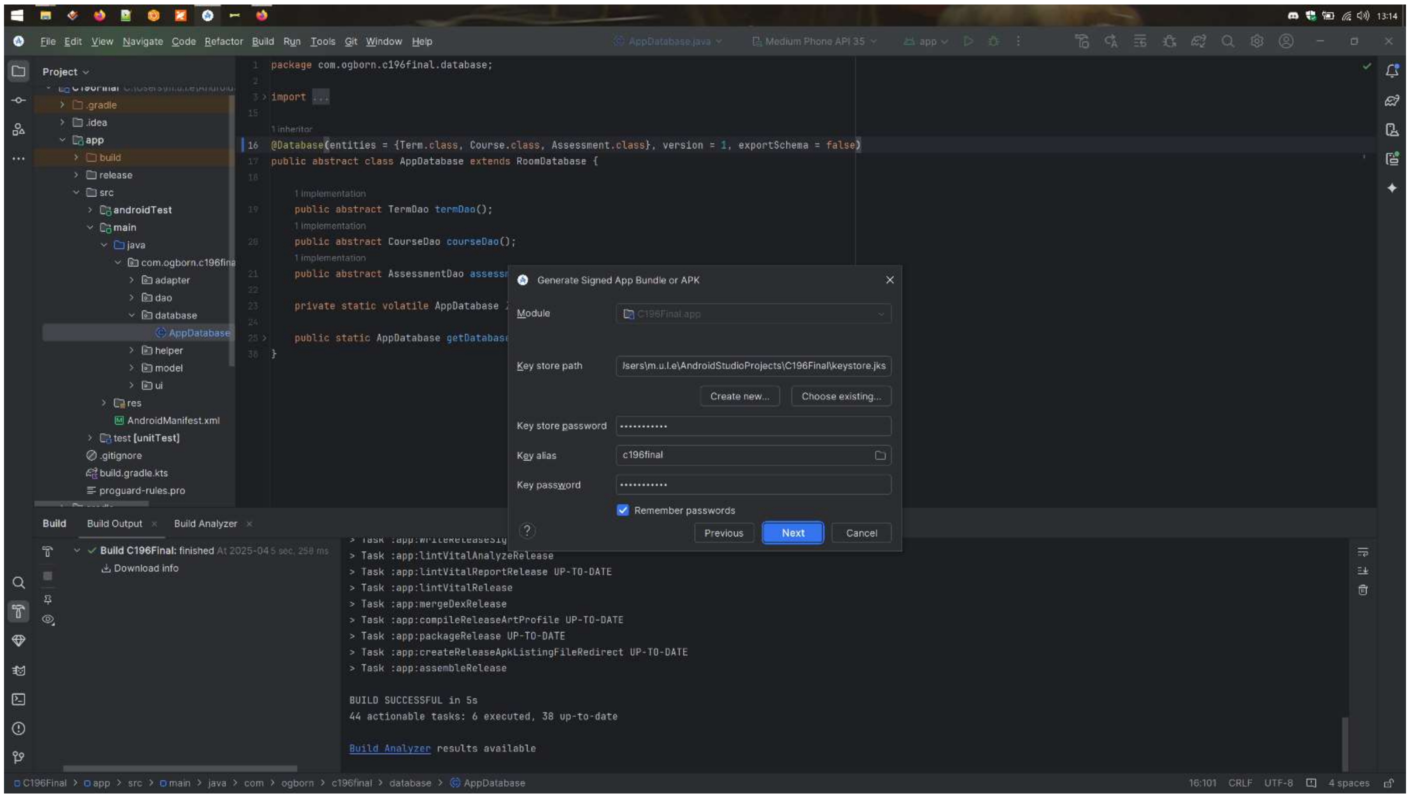
Build C196Final: finished At 2025-04 5 sec, 258 ms

```
> Task :app:writeReleaseSig
> Task :app:lintVitalAnalyzeRelease
> Task :app:lintVitalReportRelease UP-TO-DATE
> Task :app:lintVitalRelease
> Task :app:mergeDexRelease
> Task :app:compileReleaseArtProfile UP-TO-DATE
> Task :app:packageRelease UP-TO-DATE
> Task :app:createReleaseApkListingFileRedirect UP-TO-DATE
> Task :app:assembleRelease

BUILD SUCCESSFUL in 5s
44 actionable tasks: 6 executed, 38 up-to-date

Build Analyzer results available
```

File Edit View Navigate Code Refactor Build Run Tools Git Window Help
AppDatabase.java
Medium Phone API 35
Project
Next
Cancel
C196Final > app > src > main > java > com > ogborn > c196final > database > AppDatabase



Generate Signed App Bundle or APK

Module: C196Final.app

Key store path: /sers\m.u.l.e\AndroidStudioProjects\C196Final\keystore.jks

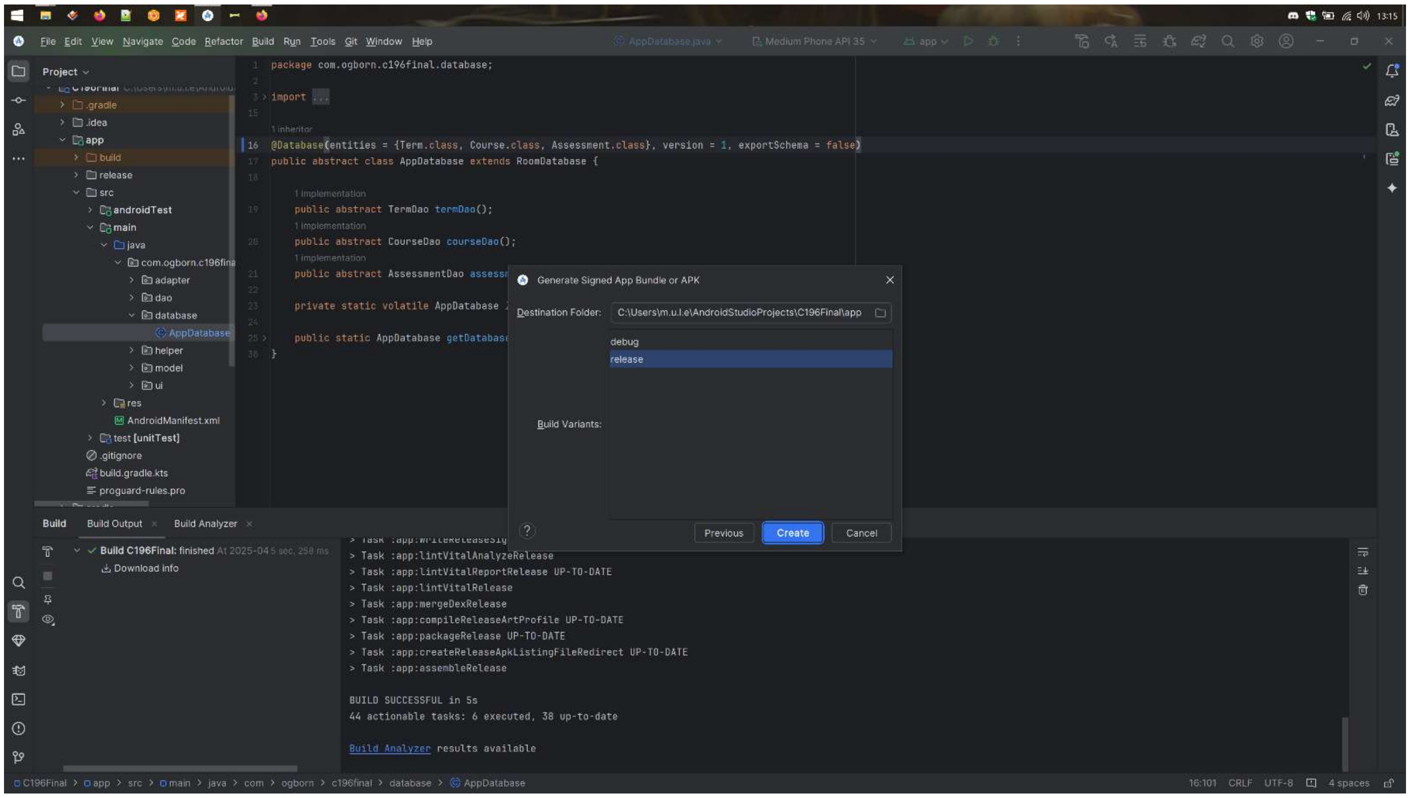
Key store password: •••••••••••

Key alias: c196final

Key password: •••••••••••

[✓] Remember passwords

Create new...
Choose existing...
Previous
Next
Cancel



```
package com.ogborn.c196final.database;

import ...

1 inheritor
@Database(entities = {Term.class, Course.class, Assessment.class}, version = 1, exportSchema = false)
public abstract class AppDatabase extends RoomDatabase {

    1 implementation
    public abstract TermDao termDao();
    1 implementation
    public abstract CourseDao courseDao();
    1 implementation
    public abstract AssessmentDao assessm

    private static volatile AppDatabase 

    public static AppDatabase getDatabase
}
```

Generate Signed App Bundle or APK

Destination Folder: C:\Users\m.u.l.e\AndroidStudioProjects\C196Final\app

Build Variants: debug, [✓ release]

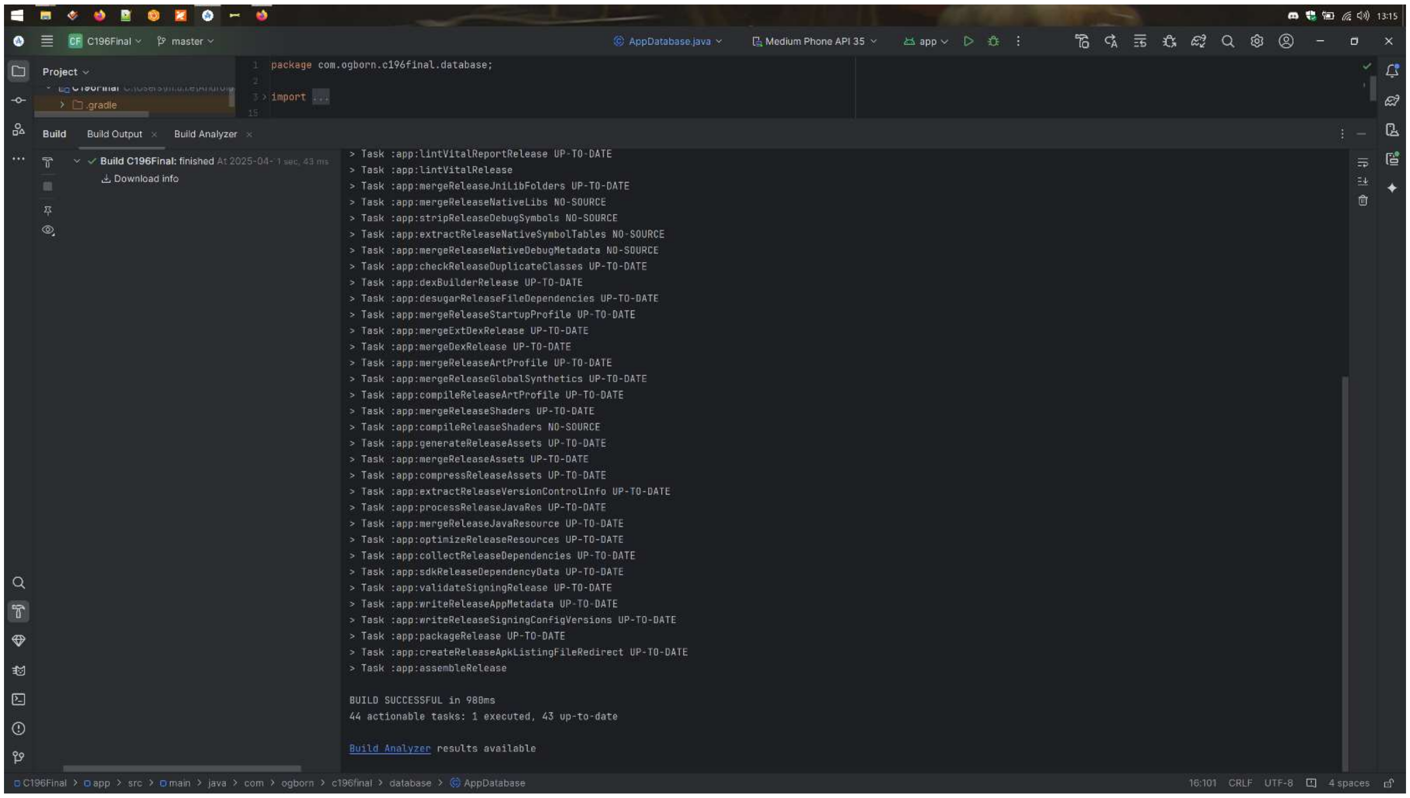
Previous | Create | Cancel

Build C196Final: finished At 2025-04 5 sec, 258 ms

```
> Task :app:lintVitalAnalyzeRelease
> Task :app:lintVitalReportRelease UP-TO-DATE
> Task :app:lintVitalRelease
> Task :app:mergeDexRelease
> Task :app:compileReleaseArtProfile UP-TO-DATE
> Task :app:packageRelease UP-TO-DATE
> Task :app:createReleaseApkListingFileRedirect UP-TO-DATE
> Task :app:assembleRelease

BUILD SUCCESSFUL in 5s
44 actionable tasks: 6 executed, 38 up-to-date

Build Analyzer results available
```

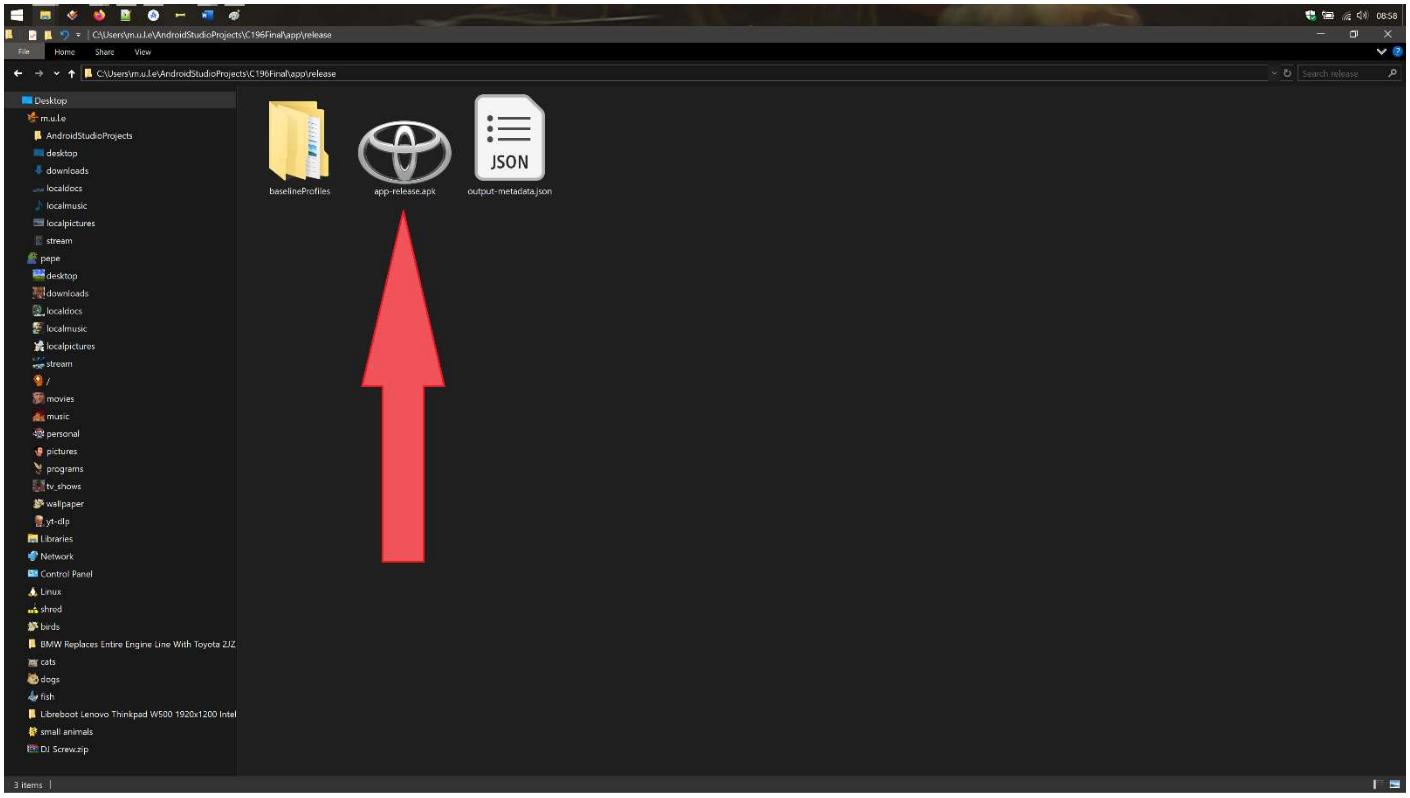


Build C196Final: finished At 2025-04- 1 sec, 43 ms

```
> Task :app:lintVitalReportRelease UP-TO-DATE
> Task :app:lintVitalRelease
> Task :app:mergeReleaseJniLibFolders UP-TO-DATE
> Task :app:mergeReleaseNativeLibs NO-SOURCE
> Task :app:stripReleaseDebugSymbols NO-SOURCE
> Task :app:extractReleaseNativeSymbolTables NO-SOURCE
> Task :app:mergeReleaseNativeDebugMetadata NO-SOURCE
> Task :app:checkReleaseDuplicateClasses UP-TO-DATE
> Task :app:dexBuilderRelease UP-TO-DATE
> Task :app:desugarReleaseFileDependencies UP-TO-DATE
> Task :app:mergeReleaseStartupProfile UP-TO-DATE
> Task :app:mergeExtDexRelease UP-TO-DATE
> Task :app:mergeDexRelease UP-TO-DATE
> Task :app:mergeReleaseArtProfile UP-TO-DATE
> Task :app:mergeReleaseGlobalSynthetics UP-TO-DATE
> Task :app:compileReleaseArtProfile UP-TO-DATE
> Task :app:mergeReleaseShaders UP-TO-DATE
> Task :app:compileReleaseShaders NO-SOURCE
> Task :app:generateReleaseAssets UP-TO-DATE
> Task :app:mergeReleaseAssets UP-TO-DATE
> Task :app:compressReleaseAssets UP-TO-DATE
> Task :app:extractReleaseVersionControlInfo UP-TO-DATE
> Task :app:processReleaseJavaRes UP-TO-DATE
> Task :app:mergeReleaseJavaResource UP-TO-DATE
> Task :app:optimizeReleaseResources UP-TO-DATE
> Task :app:collectReleaseDependencies UP-TO-DATE
> Task :app:sdkReleaseDependencyData UP-TO-DATE
> Task :app:validateSigningRelease UP-TO-DATE
> Task :app:writeReleaseAppMetadata UP-TO-DATE
> Task :app:writeReleaseSigningConfigVersions UP-TO-DATE
> Task :app:packageRelease UP-TO-DATE
> Task :app:createReleaseApkListingFileRedirect UP-TO-DATE
> Task :app:assembleRelease

BUILD SUCCESSFUL in 980ms
44 actionable tasks: 1 executed, 43 up-to-date

Build Analyzer results available
```


C:\Users\m.u.l.e\AndroidStudioProjects\C196Final\app\release
baselineProfiles
app-release.apk
output-metadata.json
3 items

**Note: Verify that all the required functions of your application are working by executing the apk file.**

--works, screenshots not required according to rubric

**F.**

**1. Explain how your application would be different if it were developed for a tablet rather than a phone, including a discussion of fragments and layouts.**

If my app were built specifically for tablets, I would restructure the layout to use multiple fragments on a single screen. Instead of launching a new activity for each navigation step, a tablet version would show the term list in a first-level sidebar on the left, the courses in a middle pane  or second-level left sidebar, and either course details or assessments on the right. This master-detail layout would allow the user to stay contextually aware without jumping between screens, significantly improving the experience on large displays.

Fragments would serve as independently updatable sections of the screen, each reacting to selections in the previous column. Layouts would be adapted to support both portrait and landscape modes using ConstraintLayouts and weight-based design. On phones, showing one screen at a time is appropriate due to limited space, but on tablets, laying out multiple “pages” side-by-side turns navigation into a fluid, document-like experience. This allows users to compare or interact with multiple pieces of data simultaneously—something not feasible on a smaller screen.

**2. Identify the minimum and target operating system your application was developed under and is compatible with.**

The app was developed with a minimum SDK version of API 26 (Android 8.0) and targets API 33 (Android 13) to maintain compatibility while benefiting from modern features and design practices.

**3. Describe (suggested length of 1–2 paragraphs) the challenges you faced during the development of the mobile application.**

**Gradle Dependency Issues:** One of the biggest challenges was caused by an automatic update to the Android Gradle Plugin early in development. I accepted the update prompt without fully understanding the impact. Afterward, the project no longer built correctly, and dependency compatibility issues surfaced that I couldn't explain. I tried downgrading and adjusting individual dependency versions, but error messages were vague and often led me in circles. Despite hours of manual trial-and-error with Gradle syncs, the project remained broken and unbuildable.

**Date Picker Behavior:** Another challenge was implementing clean, reliable date input using DatePickerDialog. Initially, the picker would reopen multiple times if tapped rapidly or if the activity recreated during interaction. Additionally, the EditText fields tied to the pickers were editable after the dialog closed, allowing users to enter invalid or misformatted dates. This undermined the goal of clean, consistent date input and caused parsing issues later in the app.

**4. Describe (suggested length of 1–2 paragraphs) how you overcame each challenge discussed in part F3.**

**Gradle Dependency Issues:** After failing to restore the project by downgrading the Gradle plugin or tweaking dependency versions, I completely uninstalled Android Studio. I cleared out all cache and configuration files to remove anything that might have been causing hidden issues. After reinstalling a fresh version, Rather than importing the entire project and risking the same problems, I built cleanly from scratch. I recreated the project structure manually and began copying my code back in one file at a time, starting with model classes and DAOs, then activities and layouts.  This process avoided lingering configuration issues and allowed me to verify each component as it was reintroduced. It was time-consuming but effective. While I still don't fully understand what broke during the original update, starting from zero proved to be the most stable path forward and gave me a clean foundation for final development and testing.

**Date Picker Behavior:** To fix the date picker issues, I made the input fields non-editable using setFocusable(false) and setClickable(true) to ensure the only way to enter a date was through the picker. I also implemented logic to prevent multiple instances of the DatePickerDialog from opening by disabling repeated triggers and isolating the picker launch to a single interaction. This created a more controlled and user-friendly experience, eliminated parsing errors from freeform input, and kept the UI consistent across devices.

**5. Discuss (suggested length of 1–2 paragraphs) what you would do differently if you did the project again.**

I would take care to put all user-facing strings into strings.xml upon first use of each string. I thought that development would be faster if I hardcoded them all, and as such would be able to remember what the content of the strings was. I eventually realized that Android Studio does put the content of the strings into the code, showing the values of the strings and not the string id.  Going back and fixing all the hardcoded strings was a nontrivial task and, while it was ultimately a necessary step, was a colossal waste of time.

**6. Describe how emulators are used and the pros and cons of using an emulator versus using a development device.**

**Emulators**: Emulators provide a versatile and efficient way to test applications across a range of Android versions, screen sizes, and hardware configurations. They are particularly effective for early-stage development, as they integrate seamlessly with tools like Android Studio and offer features like live debugging and layout inspection. Emulators are also cost-effective, eliminating the need for physical devices, and allow quick switching between configurations without additional hardware.

However, emulators have limitations. Their performance can lag, especially for resource-intensive apps, and they rely heavily on the host machine's processing power. They cannot accurately simulate real-world conditions, such as touch responsiveness, hardware-specific quirks, or battery performance. This makes it challenging to fully evaluate the user experience and identify edge cases that only occur on physical hardware.

**Development Devices:** Testing on physical devices offers a realistic view of app behavior under real-world conditions. Developers can evaluate touch interactions, responsiveness, and animations in ways that emulators cannot replicate. Physical devices also provide insights into hardware-specific behaviors like battery consumption, thermal performance, and unique quirks tied to certain manufacturers or models. This makes them invaluable for final testing before release.

The downside of using physical devices is the cost and complexity involved. Acquiring and maintaining a diverse range of devices to test compatibility across different screen sizes, hardware specs, and Android versions can be prohibitively expensive. Additionally, testing workflows with physical devices are often slower, as switching between configurations or devices requires manual setup.