Contents

What is the personal Android project?

Introduction

In the Deep Dive Coding Java+Android Bootcamp, one of the 2 major projects you will plan and execute is the personal Android project; the other is the capstone project. These projects differ in 2 key aspects: First, each capstone project is planned and executed by a team of students, while each student is individually responsible for his or her personal Android project. Second, whereas the capstone project is a *full-stack* project, including client and server components, the personal Android project is a standalone app development project, which cannot include or depend on custom development on any other platform.

Elements

Note: In the summaries that follow, sentences like "The Android app must implement functionality *X*" should be interpreted as "For full credit, the Android app must implement functionality *X*."

Documentation

Successful completion of the Android project requires a wide range of design, technical, and user documentation. In general, the following are required for full credit. Each is introduced in a relevant milestone, and is expected to be updated as necessary in subsequent milestones.

Many of the elements listed here will be expected to follow particular naming conventions or style guides. This level of detail is not included in this document, but will be specified in the milestone rubrics.

High-level overview

This is the general app description, including:

- app name;
- functional summary;
- intended users;
- any other background or supporting information not addressed by the more specific items that follow.

User stories

These are simple statements of key functionality required by the intended users, and the benefit that they will obtain from that functionality.

Wireframe diagram

This is a high-level diagram of UI layout and flow; it does not require detail in fonts, color, icon images, etc.

Entity-relationship diagram (ERD)

This diagram shows the entities and relationships managed in the data model.

- Each entity must include a name and a list of attributes.
- Each attribute must include a name, type, any index & key indicators, default value (if any), and nullability indicator (if relevant).
- Relationships must be written using *crow's foot notation* to indicate cardinality. In most cases, relationships should also include text annotations, summarizing the conceptual relationships.

Data definition language (DDL) SQL

These are the SQL statements that can be used to create the application database.

Javadocs

This is documentation embedded as comments in the Java source code, and the corresponding HTML (generated using the `javadoc` tool of the JDK).

Java code elements requiring documentation are:

- top-level `public` classes (including `enum` classes) & interfaces;
- `public` and `protected` members (including nested classes and interfaces) of the above;
- `public` and `protected` members (including nested classes and interfaces) of `public` or `protected` nested classes and interfaces.

The only permitted exceptions to the above are:

- methods annotated with `@Override`, as long as the overriding methods do not significantly alter the functionality of the method as described in the inherited definitions;
- accessors and mutators (aka getters and setters), in that the `@return` tags need not be included in the Javadoc comments of the former, and the `@param` tags need not be included in the Javadoc comments of the latter. (The Javadoc summary comments for these methods should include sufficient information for `@return` and `@param` to be unnecessary.)

Copyright & license information

In executing their projects, students are expected to demonstrate respect for their own and others' intellectual property.

All of the libraries used in class-wide coding activities are open-sourced; however, all have some license notice requirements. Students will learn how to research these requirements, and will be expected to fulfill them in their Android projects.

On the other hand, students will also be expected to make a deliberate decision on the licensing (or not) of their own intellectual property, as embodied in the code and other artifacts of their Android projects, and include copyright and license notices accordingly.

Build requirements & instructions

The process of preparing the build environment and building the app must be documented in the final sprint. This documentation must address:

- satisfying all dependencies on 3rd-party libraries and other software assets;
- cloning the repository;
- incorporating into the project any sensitive data not included in the repository (including instructions for obtaining any necessary keys);
- steps to execute the build and deploy the app using IntelliJ IDEA or Android Studio.

User instructions

The final sprint includes this documentation, which is expected to include:

- minimum Android hardware and software requirements;
- peripheral hardware requirements and optional capabilities;
- operations (starting with launching the app) required to perform the actions for at least 2 user stories.

State of completion

We recognize that it may be the case that not all planned elements of the app will be completed during the project. As a rule, this will not be penalized, as long as key technical requirements are met, and as long as the state of completion is accurately described on final delivery. This must include not only unimplemented functionality, but known deficiencies (especially conditions under which the app is known to crash).

State of testing

This element is related to the state of completion, but is focused on formal and informal test conditions and outcomes in the implementation (not at all on unimplemented functionality), including:

- tested configurations (hardware devices, emulators, Android versions, screen orientations, etc.);
- code or UI elements exercised via unit tests and other code-based tests, along with the test results;
- user-level tests (operations performed) and results.

Stretch goals

Virtually every non-trivial system has some possibility for expanding in an organic fashion: new functionality that adds to whole in a manner coherent with the primary aims, that doesn't significantly change its fundamental nature. The final deliverable should include a short description of one or more functional enhancements of this type.

Implementation

We expect that students will propose and execute Android projects in a wide variety of application domains. In signing off on on proposals at the start, and in grading project work at the milestone, we have no interest in prioritizing productivity apps over games, utilities over interesting wastes of time, "serious" apps over eye candy. However, because we are committed to helping students build skills that will position them well for employment as developers, we do require that all the personal Android app include a number of key implementation elements.

Data persistence with SQLite

Virtually every Android device incorporates the SQLite relational database management system (RDBMS). All personal Android apps are expected to utilize SQLite for master lists, transactional data, key event logs, etc. This may be done using the native SQLite libraries available on the Android platform, or with the Room Object-Relational Mapping (ORM) library; the latter is recommended. (JDBC is not available on the Android platform.)

Multiple screens with intuitive navigation

Even for apps with relatively simple UIs, there are opportunities to incorporate multiple screens—e.g. settings/preferences screens, high score displays, even license information screens. Students will be expected to incorporate multiple screens, and to implement rational, intuitive navigation controls for moving between screens. (In general, navigation will be expected to follow Material design guidelines.)

Data-driven, flexible display containers

Just as almost any app can justifiably use multiple screens, almost any app can effectively incorporate a data-driven display of a non-hard-coded size—e.g. a list of notification events, a list of images or documents, a game board of a user-configurable size, a list of high scores. Students apps must have at least one such display, formatted as a scrolling list, grid, multi-page gallery, etc.

Integration with device or external services/data

By default, an Android app runs entirely in its own *sandbox*, unable to see or modify the behavior or data of any other app on the device, unable to leverage the shared hardware/software services of the device (e.g. camera, contacts, messaging, sensors), unable even to access the Internet. However, an app that can escape this sandbox—using the supported facilities for doing so—becomes (at least potentially) much more powerful and interesting.

A student's Android app is expected to include some functionality that integrates with one of these device or external services or data sources. This may involve the use of a general-purpose client library for accessing web services, or a specialized library that accesses some very specific web service; it may involve reading or writing files in the device's *external storage* (a portion of the device's non-volatile storage which can be accessed by multiple apps); it may involve invoking the registered camera app on the device to capture an image, or even display the live camera image in the app's own screens and capture images or video directly into the app's memory or file storage; it may involve reading or updating the user's list of contacts; the list of possibilities is practically endless.

Storage and use of preferences/settings

Many elements—directly visible in the UI or not—of an Android app can be implemented in such a way that their future operation can be configured by current user actions. Students' Android apps must include the persistence and use of one or more such configuration settings. This may be implemented using the Android Preferences framework (the recommended approach), or in a SQLite database.

Timetable

Overview

Preliminary work on this project begins with classroom discussion in week 1, followed by project proposal drafts written and refined from the end of week 1 to the end of week 2. Formal work on the projects begins with instructor sign-off on the proposals at the end of week 2, and completes in week 11.

The project development calendar is divided into 3 *sprints*, of approximately 3 weeks each. Each sprint ends with a milestone deliverable, which is graded; the sum of the 3 milestone grades is the total grade for the project. (In addition, some tasks in the project—particularly the early design tasks—will be assigned as homework; completion of those individual tasks will be reflected in the homework & in-class activities grade, which is separate from the Android project grade.)

Weekly schedule

The key activities of the project follow (roughly) the schedule below, where the numbers denote the numbered weeks of the bootcamp. Student activities are shown in normal type, and *instructor activities appear in italics.*

1. **Speculation**
   - Classroom discussions.
   - Write proposal first drafts.
2. **Articulation**
   - *Provide feedback on proposal drafts.*
   - Research web and device services.
   - Refine proposals (possibly changing topics), incorporating feedback and research.
   - *Sign-off on final proposals.*
3. **Initial UI design and GitHub Pages setup**
   - Create Android projects in IntelliJ.
   - Transfer relevant content from proposal repositories to project repositories.
   - Enable GitHub Pages in project repository and customize (if desired) theme.
   - Create first draft of wireframe diagrams.
   - *Provide feedback on wireframe drafts.*
   - Refine wireframes.
4. **Data modeling and basic navigation**
   - Create first draft of Entity-Relationships Diagram (ERD).
   - *Provide feedback on ERD drafts.*
   - Implement basic navigation, based on wireframes.
   - Refine ERDs.
   - Complete CRUD operations inventory document.
   - Create first draft of entity classes.
5. **Entity class implementation**
   - _Give feedback on entity classes to students.
   - Refine entity classes.
   - *Provide feedback on CRUD operations inventory.*
   - Refine CRUD operations inventory.
   - Submit milestone deliverable package to complete sprint 1.

- *Give milestone 1 feedback to students.*

6. **Data model implementation**
   - Create first draft of Data Access Object (DAO) interfaces.
   - *Provide feedback on DAO interfaces.*
   - Refine DOA interfaces.
   - Create database class.
   - Incorporate Stetho library for external inspection in Chromium-based browsers.
7. **Data model unit tests and repositories**
   - Create unit tests for CRUD operations; use unit tests to verify correctness and implement fixes.
   - Create Retrofit service interfaces for consuming external services (if relevant).
   - Create repository classes to broker requests to internal and external data.
   - *Provide feedback on data- and service-access stack.*
   - Refine data- and service-access stack.
8. **Authentication and viewmodels**
   - Implement authentication (if applicable).
   - Submit milestone deliverable package to complete sprint 2.
   - *Give milestone 2 feedback to students.*
   - Create viewmodel classes.
   - Create adapter classes.
   - Implement layouts from up-to-date wireframes.
9. **User interface**
   - *Provide feedback on controllers, viewmodels, and adapters.*
   - Refine controllers, viewmodels, and adapters.
   - Create UI interaction tests; use tests to verify correctness and implement fixes.
10. **Fixing and polishing**
    - Continue refining application, incorporating test results.
    - Run code analysis.
    - Perform simple code reformatting, reorganization, and refactoring, based on code analysis.
    - Assemble technical documentation.
11. **Final documentation**
    - Write build documentation.
    - Write user documentation.
    - Submit final deliverable package to complete sprint 3.
    - *Grade & provide feedback on final deliverable.*

Schedule notes

- Students are expected to complete the required work with a combination of in-class and out-of-class time; there will **not** be sufficient time to complete all of the Android work during class hours. Some tasks—early in the bootcamp, in particular—will be assigned as homework, but students are expected to work on some without explicit assignment.

- While the tasks are listed sequentially, many tasks overlap in actual execution, and/or are revisited multiple times.
- The calendar of in-class and homework tasks may be adjusted, based on class progress, unforeseen events and conflicts, etc.

Grading

Overview

The personal Android project is worth 200 points in total. These are awarded upon grading of the individual milestones as follows:

- Milestone 1: 50 points
- Milestone 2: 75 points
- Milestone 3 (final): 75 points

These subtotals—and the overall total—may be adjusted as necessary, based on unforeseen events, resulting in changes to the course calendar, instruction mode, and available hours per week of instruction & homework.

Rubrics & grading

The rubric for a given milestone will be published early in the sprint that ends with the milestone. The elements included in that rubric will generally be those listed before the milestone in question, and after the preceding milestone (if any), in the weekly schedule; however, we reserve the right to adjust this, as circumstances dictate.

Similarly, we reserve the right to set the points available for each graded element on a cohort-by-cohort basis. In general, however, 40–45% of the points available in each milestone are based on documentation elements, while the remaining points are based on code and app functionality.

Extra credit

Up to 10% extra credit may be earned in each milestone for exemplary execution of one or more elements. In the final milestone, additional points may also be available for early submission. All such extra credit opportunities will be declared in the rubrics.