Contents

What is the capstone project? 1
- Introduction . . . 1
- Elements . . . 2
  - Documentation . . . 2
    - High-level overview . . . 2
    - User stories . . . 2
    - Wireframe diagram . . . 2
    - Entity-relationship diagrams (ERDs) . . . 2
    - Data definition language (DDL) SQL . . . 3
    - Javadocs . . . 3
    - Service documentation . . . 4
    - Copyright & license information . . . 4
    - Build requirements & instructions . . . 4
    - User instructions . . . 5
    - State of completion . . . 5
    - State of testing . . . 5
    - Stretch goals . . . 5
  - Implementation . . . 6
    - Data persistence . . . 6
    - Multiple screens with intuitive navigation . . . 6
    - Data-driven, flexible display containers . . . 6
    - Integration with device or external services/data . . . 6
- Timetable . . . 7
  - Overview . . . 7
  - Weekly schedule . . . 7
- Grading . . . 10
  - Overview . . . 10
  - Rubrics & grading . . . 10
  - Extra credit . . . 10

What is the capstone project?

Introduction

In the Deep Dive Coding Java+Android Bootcamp, one of the 2 major projects you will plan and execute is the capstone project; the other is the personal Android project. These projects differ in 2 key aspects: First, each student is individually responsible for his or her personal Android project. but the capstone project is planned and executed by a team of students. Second, whereas the personal Android project is a standalone app development project, which must be usable even running on a single Android device, the capstone project is a *full-stack* project, including client and server components.

Elements

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

Documentation

Successful completion of the capstone 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 project description, including:

- application name;
- functional summary, with separate sub-sections for client-side and server-side ;
- 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 diagrams (ERDs)

These diagrams show the entities and relationships managed in the data models.

- There will always be one ERD for the server-side data model; if the client side has local persistent storage in a relational database, there must also be an ERD for the client side. In other words, each project's documentation must include 1–2 ERDs, depending on whether the client has local persistence of relational data.
- 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 create the tables, indices, and constraints for the data model implementation. As with the ERD, there will be 1–2 DDL documents, depending on whether the client persists relational data locally.

Javadocs

This is documentation embedded as comments in the Java source code, and the corresponding HTML (generated using the `javadoc` tool of the JDK). There must be 2 sets of generated HTML: 1 for the server side, and 1 for the client side.

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, regardless of the nesting level.

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.)

Service documentation

As a rule, the capstone project server-side implementation is expected to expose service methods as REST endpoints. These must be documented, with the following elements included for each endpoint:

- URL (including path placeholders, as needed)
- HTTP method (`GET`, `POST`, `PUT`, `PATCH`, `DELETE`, etc.)
- Authentication scheme (if any)
- Request specification:
    - Headers
    - Query-string parameters
    - Body (payload)
- Response specification
    - Headers
    - Body (payload)
    - Possible errors; for each, include:
        - Status code
        - Body

Copyright & license information

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

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

On the other hand, capstone teams will also be expected to make a deliberate decision on the licensing 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 **both** the server and client build environments, and building **both** the client and server applications must be documented in the final sprint. This documentation must address:

- satisfying all dependencies on 3$^{\text{rd}}$-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.

User instructions

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

- minimum client (e.g. Android) hardware and software requirements;
- peripheral hardware requirements and optional capabilities;
- operations (starting with launching the client) 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 (client hardware devices/emulators, client screen orientations, OS versions for client and server, 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 capstone 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

Every capstone project is expected to include relational data persistence on the server side (at least); it may also include relational data storage on the client side.

Server side persistence usually leverages the Apache Derby or Oracle MySQL relational database management system (RDBMS); in general persistence code in the server components will rely on JPA/Hibernate and Spring Data, rather than using JDBC directly.

Android persistence uses the SQLite RDBMS—though apps will typically use the Google Room ORM. (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. Teams will be expected to incorporate multiple screens, and to implement rational, intuitive navigation controls for moving between screens. (In general, navigation in Android clients will be expected to follow Material design guidelines.)

Data-driven, flexible display containers

Just as almost any UI can justifiably use multiple screens, almost any client application 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. Capstone client applications must have at least one such display, formatted as a scrolling list, grid, multi-page gallery, etc.

Integration with device or external services/data

The capstone application (considering both the client and server side together) is expected **not** to live in its own isolated world. For almost all such projects, there will be many opportunities to incorporate cloud-based services (including authentication servers), whether these are consumed by the client, server, or both. Additionally, some projects will need to preload a client or server database with data obtained (at system startup or as part of the deployment process) from an external source.

On the client side, there are also opportunities for integrating with device/platform services. In an Android client, for example, the app could use the device camera to capture images that are then uploaded to the server.

The capstone project 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 services, a specialized library that accesses specific services or devices, etc.

Timetable

Overview

Preliminary work on the capstone project begins with classroom discussions in weeks 1 & 2, then project proposal drafts written at the end of week 2 and refined in week 3, followed by voting and team formation at the end of week 3 or start of week 4. Formal work on the projects begins early in week 4 and is completed in week 12.

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. Under normal circumstances, grading is on a team basis: When grading the work for any milestone, the instructors will award the same number of points to all team members.

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. **Discussion**
    - Classroom discussions.
2. **Speculation**
    - Classroom & group discussions.
    - Write proposal first drafts.
3. **Articulation**

   - *Provide feedback on proposal drafts.*
   - Refine proposals (possibly changing topics), incorporating feedback and connecting proposals more closely to general project requirements.
   - *Filter proposals and sign-off on final proposals.*
   - Vote on short list of proposals (usually 8–10) for final selection.
   - *Assign members to teams, based on voting and other factors.*
4. **Team groundwork and initial UI design**
   - Create GitHub orgs and documentation repositories.
   - Write team ground rules.
   - Create first draft of wireframe diagrams.
   - *Provide feedback on wireframe drafts.*
   - Refine wireframes.
5. **Data modeling**
   - Create first draft of Entity-Relationships Diagram (ERD).
   - *Read ERD drafts and give feedback to students.*
   - Refine ERDs.
   - Complete CRUD operations inventory document.
   - *Provide feedback on CRUD operations inventory.*
   - Refine CRUD operations inventory.
6. **Data model implementation**
   - Create entity classes.
   - _Give feedback on entity classes to students.
   - Refine entity classes.
   - Create first draft of repository interfaces.
   - *Provide feedback on repository interfaces.*
   - Refine repository interfaces.
   - Submit milestone deliverable package to complete sprint 1.
   - *Give milestone 1 feedback to students.*
7. **Data model unit tests and services**
   - Create unit tests for CRUD operations; use unit tests to verify correctness and implement fixes.
   - Create services classes for implementation of authentication and business logic.
   - *Provide feedback on service classes.*
   - Refine service classes.
8. **Web service controllers**
   - Complete web services inventory document.
   - *Provide feedback on web services inventory.*
   - Refine web services inventory.
   - Create controller classes.
   - *Provide feedback on controller classes.*
   - Refine controllers.
9. **Testing server components**
   - Create tests for web services; use unit tests to verify correctness and implement fixes.

    - Submit milestone deliverable package to complete sprint 2.
    - *Give milestone 2 feedback to students.*
    - Create Android project & repository for client application.
10. **Client application core**
    - Create web service consumer interfaces.
    - *Provide feedback on web service consumer interfaces.*
    - Refine web service consumer interfaces.
    - Implement client-side persistent data model (if applicable).
    - 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.
11. **User interface**
    - Create UI controller classes and layouts implementing wireframes.
    - Create viewmodel classes.
    - Create adapter classes.
    - *Provide feedback on controllers, viewmodels, and adapters.*
    - Refine controllers, viewmodels, and adapters.
    - Create UI interaction tests; use tests to verify correctness and implement fixes.
12. **Final documentation**
    - Write build documentation.
    - Write user documentation.
    - Submit final deliverable package to complete sprint 3.
    - Present project to roundtable of program partners, potential employers, etc.
    - *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. (This is an important aspect of the capstone project team coordination: working together to plan and execute tasks.)

- 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 capstone project is worth 300 points in total. These are awarded upon grading of the individual milestones as follows:

- Milestone 1: 80 points
- Milestone 2: 110 points
- Milestone 3 (final): 110 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.

Under normal circumstances, capstone project work is graded on a team basis: every member of the team will receive the same amount of points in each milestone.

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.