How to Use this Template

1. Make a copy [File → Make a copy...]
2. Rename this file: “**Capstone_Stage1**”
3. Replace the text in green

Submission Instructions

1. After you’ve completed all the sections, download this document as a PDF [File → Download as PDF]
2. Create a new GitHub repo for the capstone. Name it “**Capstone Project**”
3. Add this document to your repo. Make sure it’s named “**Capstone_Stage1.pdf**”

Description
Intended User
Features
User Interface Mocks
Screen 1
Screen 2
Key Considerations
How will your app handle data persistence?
Describe any corner cases in the UX.
Describe any libraries you’ll be using and share your reasoning for including them.
Describe how you will implement Google Play Services.
Next Steps: Required Tasks
Task 1: Project Setup
Task 2: Implement UI for Each Activity and Fragment
Task 3: Your Next Task
Task 4: Your Next Task
Task 5: Your Next Task

GitHub Username: robert.lagrasse

Metronome

Description

In quantum gravity, the **problem of time** is a conceptual conflict between general relativity and quantum mechanics. Roughly speaking, the problem of time is that there *is none* in general relativity. This is because in general relativity, the Hamiltonian is an energy constraint that must vanish to allow for general covariance. However, in theories of quantum mechanics, the Hamiltonian generates the time evolution of quantum states. Therefore, we arrive at the conclusion that "nothing moves" ("there is no time") in general relativity. Since "there is no time", the usual interpretation of quantum mechanics measurements at given moments of time breaks down. This problem of time is the broad banner for all interpretational problems of the formalism.

Fortunately, we don't need to solve time problems quite this complex in order to keep a beat.

Metronome is intended to solve the most important problem in the world - my problem. Namely, I'm a musician, and all of the metronome apps on the play store stink. Ugly, clunky UI's, inconsistent performance, limited options beyond click click click. What I envision is something that sits between a metronome and a full on drum machine. I want to enable my users to set the tempo, pick kits built from multiple high-quality sound samples, select patterns for those kits to play, and save jams comprised of tempo, kit, and pattern - all from a single screen UI so simple and intuitive a child can operate it.

Intended User

This app is intended for any musician, at any age.

Features

List the main features of your app. For example:

- Saves information regarding kits, patterns, jams in a local database
- Leverages multiple recyclerviews to build UI, all feeding from local DB. LoaderManager.LoaderCallbacks will be implemented to feed all of these RecyclerViews.
- Leverages a graphing library to present the user with a visual representation of the pattern being played.
- Allows users to share jams.
- Allows users to create custom kits, patterns, and jams.

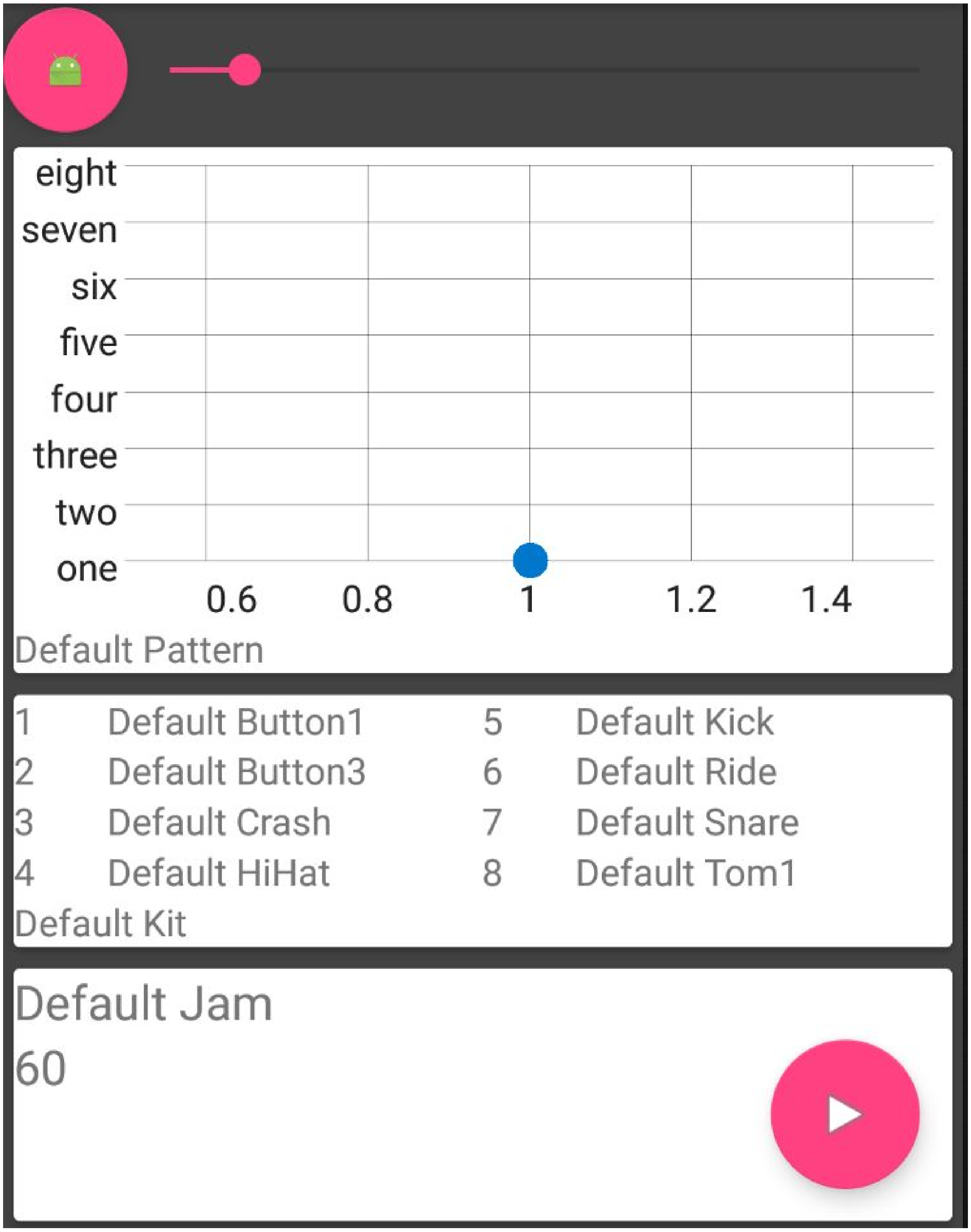
User Interface Mocks

This is the concept that most closely inspires my design.

I want the interface to be this simple and approachable. A slider along the top to select tempo, and three stacked recycler views to select the pattern, kit, and jam. The only button on the screen will be the FAB to start/stop the timer.

Swiping left or right on any of the recycler views will result in an immediate change. No buttons to push, no confirmations. You swipe, the recyclerview snaps into position, and on the very next beat, you're hearing the new combination. It's as easy as changing your clown-pants.

Screen 1



1	Default Button1	5	Default Kick
2	Default Button3	6	Default Ride
3	Default Crash	7	Default Snare
4	Default HiHat	8	Default Tom1

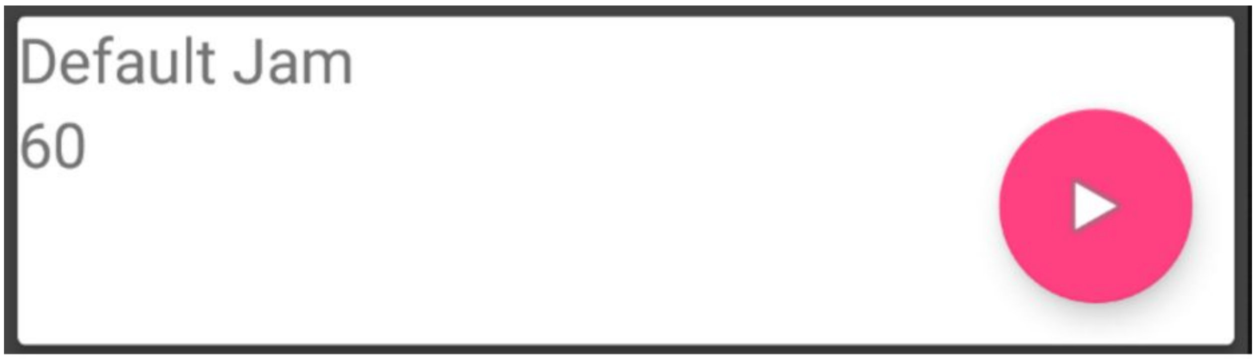
Default Kit

Default Jam

60

Widget -

The intent is to drive control and screen time to the main screen, so the widget has been limited to displaying the current/last jam, and a start/stop button



I'll implement a custom toolbar along the top, where I can implement less often used features like sharing. Sharing will happen on demand, and will leverage an Intent Service. I'll also leave some banner space on the bottom as I intend for this to be an ad delivery platform

Key Considerations

How will your app handle data persistence?

This is actually where things get kinda cool. I'm designing classes to do the following:

Component - A component is just a sound. It's got a name, a corresponding resource, and a 2-digit hex ID. The app will come bundled with pre-recorded components.

Kit - A Kit is a collection of 8 different components. Users pick the kit they want to use with the recyclerview as above.

Beat - A beat represents what happens when the timer goes off. A beat represents the state of 8 different components.

Pattern - A Pattern is a collection of beats. The pattern can be of any length greater than 0. Users pick the pattern they want to play using the recyclerview as above. The pattern is visually represented in the recyclerview using a graphing library.

You'll notice that a beat simply represents the state of the components, and that there are 8 components. I can treat these states like bits, and represent every possible beat as a number between 0 and 255, or more precisely, a two digit hex value between 00 and FF.

Since a Pattern is nothing more than a series of beats, I can store a pattern in a database as a series of two digit hex values, and reconstruct the pattern and individual beats quickly and easily.

The same logic applies to Kits. Kits are comprised of 8 components, and each of those components has a static hex ID. Therefore, any kit can be described as a string of 8 hex ID's.

This allows jams of infinite complexity to be stored and shared using an incredibly small amount of space. Anything with a pattern length of less than 100 beats (and that would be pretty ridiculous) could be shared using less data than a single tweet.

All of this information will sit on an SQLite database with tables for components, kits, patterns, and jams. The SQLite database will be front-ended by a content provider.

Describe any corner cases in the UX.

The UI is designed to keep everything in one place, so there really aren't any corner cases.

Describe any libraries you'll be using and share your reasoning for including them.

I'll be using the support library for recyclerview and cardview, material design library for FAB, and graphview to build a visual representation of the patterns.

Describe how you will implement Google Play Services.

I intend to use adMob to squeeze every dime I can out of the masses of musicians who will love this application more than their own pets.

Next Steps: Required Tasks

This is the section where you can take the main features of your app (declared above) and decompose them into tangible technical tasks that you can complete incrementally until you have a finished app.

Task 1: Core Elements

Build Classes for
Beat
Component
Jam
Kit
Pattern

Task 2: Data Storage

Define tables in database contract
Build a DatabaseManager
Build a Content Provider
Push some basic defaults into the database

Task 3: Layouts

Activity_main will contain:
Seek_bar
FAB
Three Frame Layouts to house RecyclerViews for kit, pattern, and jam.
Final frame layout sits on the bottom - adMob space, and possible branding.

CardView layouts for Pattern Chooser, Kit Chooser, and Jam Chooser

Task 4: Adapters

Build CursorAdapters to feed the recyclerviews from the database

Task 5: Make the FAB do something.

The Fab will either start or stop the timer, based on its current state.
The timer, on each tick, will move to the next position in the pattern, and interpret that beat. The sounds that correspond with the components marked true in the beat will be played. When the end of the pattern is reached, it goes back to the start.

Task 6: Sharing

The app will package up user created jams and allow them to be shared. As mentioned earlier, the classes have been structured to facilitate easy sharing. The jam will be packaged up into a JSON string, and sent either to a server, or via messaging platform.

Task 7: Profit ;)

Google AdMob will be integrated

Task 8: Wearable integration

Watch app will provide start/stop for metronome.

Submission Instructions

1. After you've completed all the sections, download this document as a PDF [File → Download as PDF]
2. Create a new GitHub repo for the capstone. Name it "**Capstone Project**"
3. Add this document to your repo. Make sure it's named "**Capstone_Stage1.pdf**"