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**"

GitHub Username: cruddell

Museum of the Bible

Description

Explore the new Museum of the Bible. This app allows you to:
--Buy tickets: Use the app to buy tickets before you arrive and skip the line.
--Explore featured galleries: Get information on some of the featured exhibits at Museum of the Bible and listen to a guided audio tour about those exhibits

--Get directions to the Museum: We are located just two blocks from the National Mall in Washington D.C. Use the app to get here in no time!
Read the Bible: Never be without the Bible again. Get quick access to any verse in the Bible or get daily verses pushed directly to you.

Intended User

The intended user is someone planning a trip to Museum of the Bible

Features

List the main features of your app. For example:

- Buy Tickets (actual purchasing disabled for demo purposes)
- Featured exhibits
- Museum Location via Google Maps
- Facebook page for the Museum
- Digital Bible
- Push Notification Settings

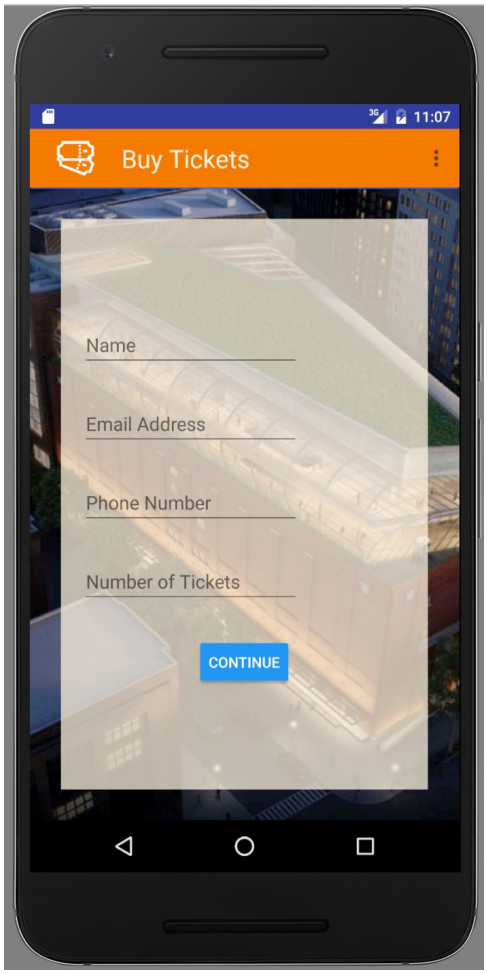
User Interface Mocks

These can be created by hand (take a photo of your drawings and insert them in this flow), or using a program like Photoshop or Balsamiq.

Screen 1

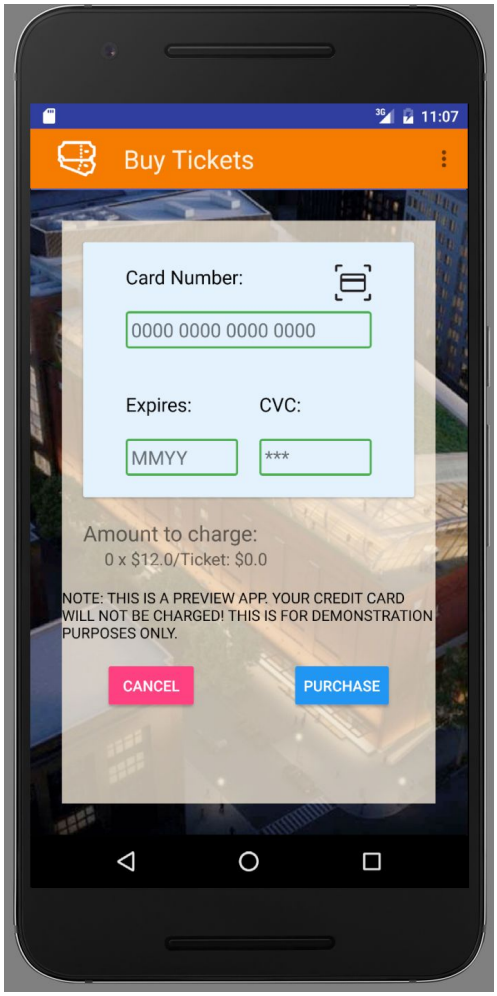
Home screen with buttons for each child screen

Screen 2



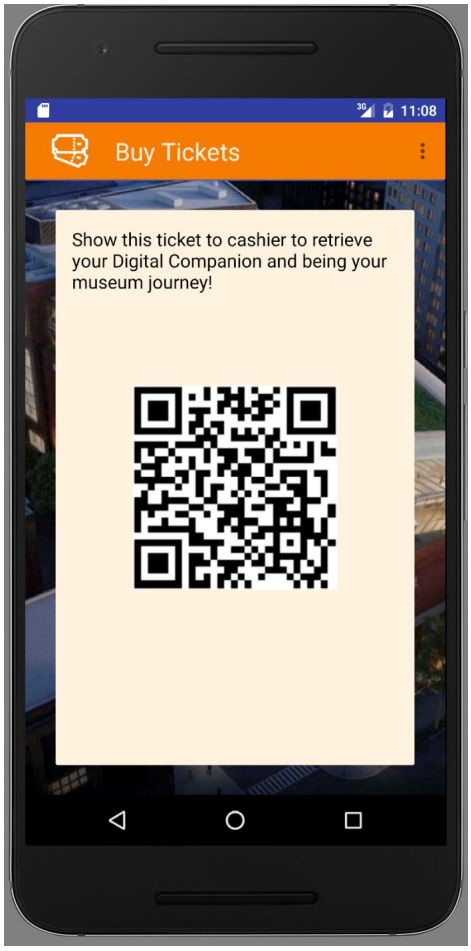
Ticketing Screen - first viewpager view; “continue” button advances the viewpager.

Screen 3



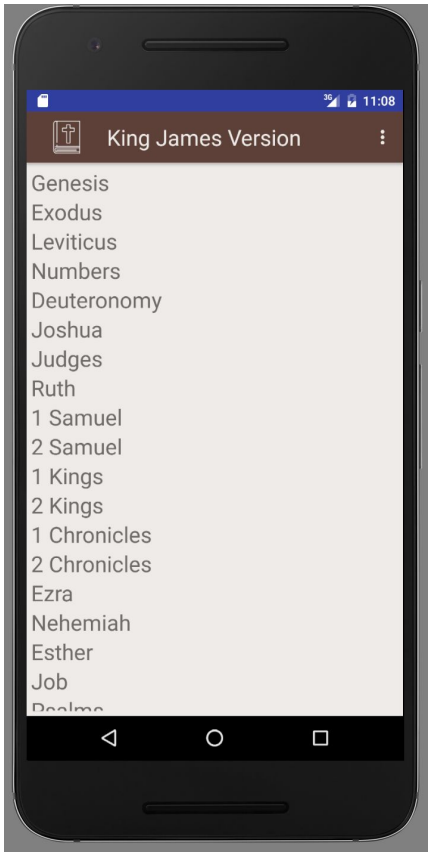
Ticketing Screen - second viewpager screen. Clicking “purchase” will advance the viewpager and show a QR Code image representing the ticket.

Screen 4



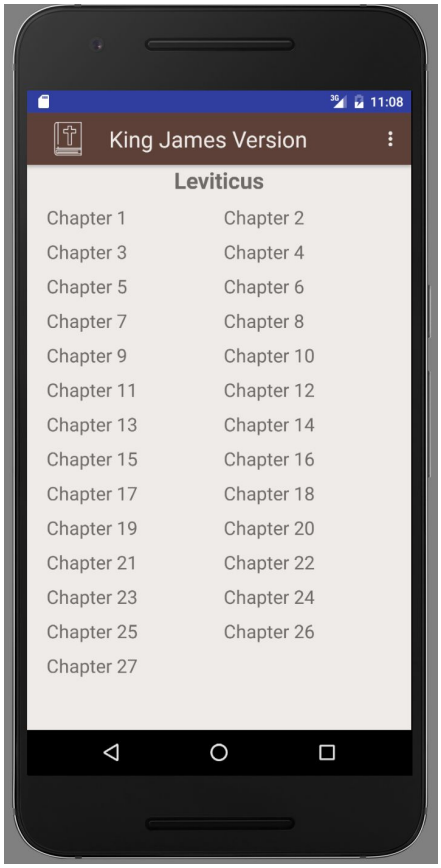
Ticketing Screen - third viewpager screen showing the QR code image on successful purchase.

Screen 5



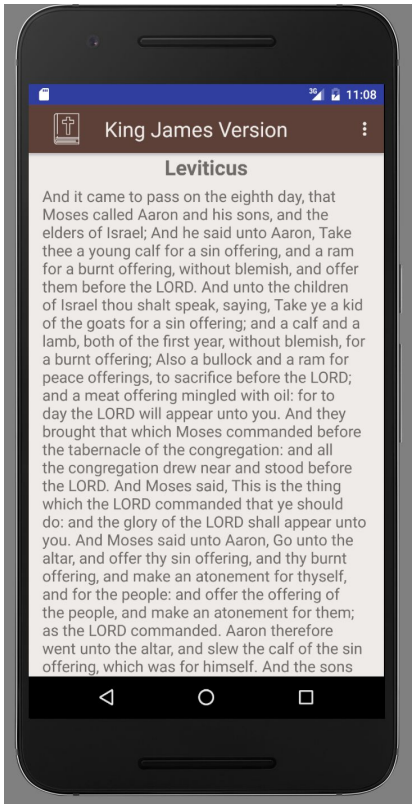
Bible Screen -- page 1: Shows the list of books in the Bible. Collapsable menu item shows different translations available.

Screen 6



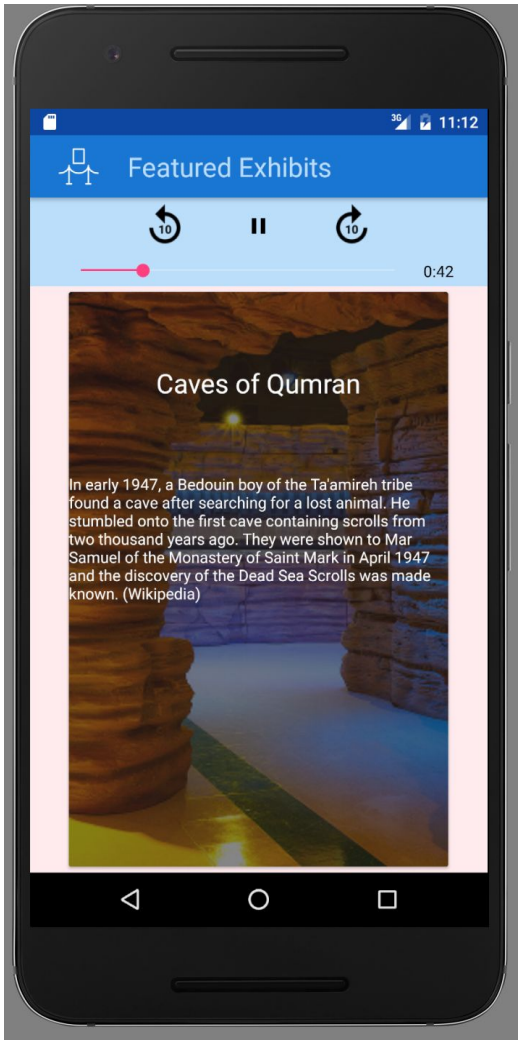
Bible Screen -- page 2: Shows the list of chapters for the selected book.

Screen 7



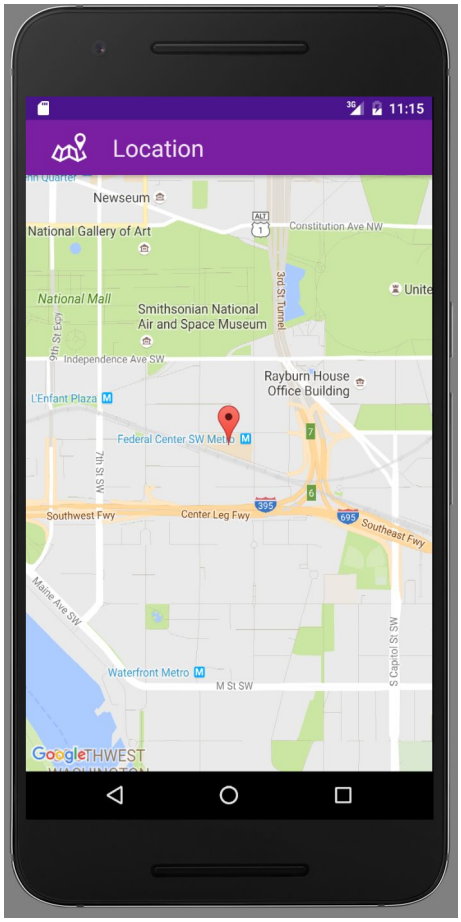
Bible Screen -- page 3: Shows text of the selected chapter.

Screen 8



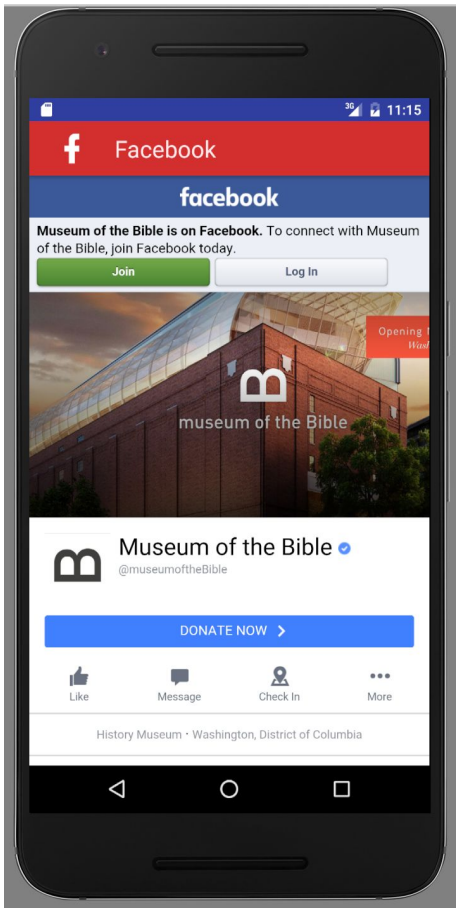
Featured Exhibits Screen - uses a ViewPager to show different exhibits downloaded from server API. Audio player begins playing first file automatically.

Screen 9



Map screen: shows location of museum on a Google Map.

Screen 10

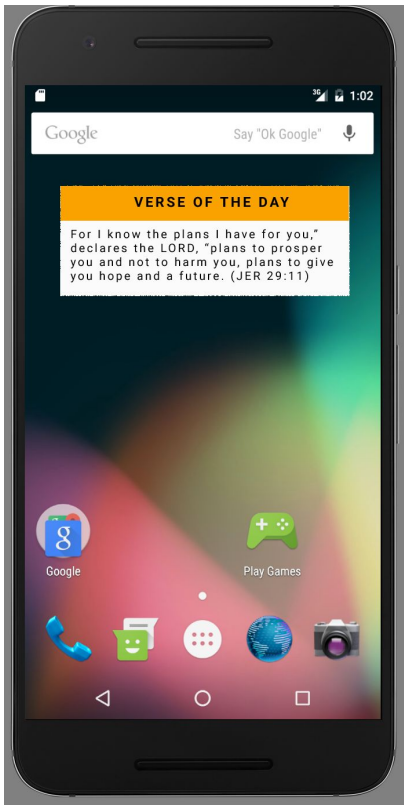


Facebook Activity -- displays webview with Facebook page (mobile version) loaded.

Screen 11

Settings screen -- allows user to unsubscribe from “verse of the day” notifications.

Screen 12



Widget -- shows “verse of the day” on the launcher screen

Add as many screens as you need to portray your app’s UI flow.

Key Considerations

How will your app handle data persistence?

The app will handle data persistence using a combination of sqlite databases and shared preferences. The sqlite databases will be accessed using a Content Provider design pattern and used for downloading featured exhibits and the digital Bible feature. The shared preferences will be used for saving simple values for later use, such as whether push notifications are turned on or off.

Describe any corner cases in the UX.

Normally users can explore verses from the Bible by navigating to the Bible screen, selecting a book, and then selecting a chapter. Pressing back will back up each step in the process until they return to the Home screen. However, if a Verse of the Day alert “Go to Bible” button is clicked it will bring them straight to the screen with all the verses displayed for the chapter containing that verse. Pressing back from this screen will return the user to the screen they were on when the alert came in.

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

Google Maps -- to show the Museum’s location on the map
Firebase Messaging -- to receive and handle “verse of the day” notifications
Card.IO -- to allow scanning of a credit card using the camera
Google App Engine -- to serve as a mock API server to handle credit card purchases for tickets (and to ensure no purchase is actually made for demo purposes)

Describe how you will implement Google Play Services.

Describe which Google Play Services you will use and how.
--Google Maps
--Firebase Messaging
--Google App Engine

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: Project Setup

- Open app with Android Studio
- Ensure JDK is set to 1.8
- Set configuration to “app”
- Click ‘run’ to compile and install on a device

Task 2: Implement UI for Each Activity and Fragment

List the subtasks. For example:

- Build UI for Home activity
- Build UI for Ticketing activity and fragments for each ViewPager page:
    - TicketingFragment
    - PaymentFragment
    - TicketFragment
- Build UI for Bible activity and each ViewPager page:
    - BibleBooksFragment
    - BibleChapterFragment
- Build UI for Featured Exhibits activity and Exhibit Detail Fragment
- Build UI for Map activity and Map fragment
- Build UI for Facebook activity
- Build UI for Settings activity

Task 3: Handle Ticketing activity

- Implement Card.io library into Ticketing Activity and related Purcahse Fragment
- Create Google App Engine api to handle payment requests and issue qr code image
- Implement networking code to connect to Google App Engine api
- Implement saved preferences to remember a ticket is available to display to ticketing agent at Museum.

Task 4: Handle Bible Activity

Describe the next task. List the subtasks. For example:

- Create Bible sqlite databases for at least two Bible translations that can be precompiled with project (assume Bible data will not change and thus does not need to be dynamic)
- Implement code to copy precompiled sqlite databases into database folder for app so it can be read and used.
- Implement Content Provider, Contract and Database helper classes for connecting to Bible databases
- Use Content Provider to perform queries from the Bible Activity and related fragments and display text as appropriate
- Assume only English is supported - if additional languages are supported in future, assume they will be treated as separate Bible translations and thus have their own Bible database
- Use a ViewPager to navigate between books of the Bible, chapters in each book, and verses for each chapter. Assume an entire chapter will be displayed at once.

Task 5: Handle Featured Exhibits Activity

Describe the next task. List the subtasks. For example:

- Implement a ViewPager to display each featured exhibit
- Implement an audio player using android.media.MediaPlayer
- Implement networking code to connect to Museum of the Bible development server to download featured exhibit data using a background thread

Task 6: Handle Facebook Activity

Describe the next task. List the subtasks. For example:

- Implement a WebView to connect to Museum's Facebook page and display as appropriate

Task 5: Handle Verse of the Day notifications

Describe the next task. List the subtasks. For example:

- Implement Google Cloud Messaging using new Firebase library
- Send Firebase token to development server
- Register BroadcastReceiver to handle notifications when in the foreground and Firebase notification receiver to handle notifications when in the background
- Add support to Bible Activity to display entire chapter containing the verse of the day when the AlertDialog button is clicked
- Implement Firebase server code to send notifications and setup CRON job to send one notification every day

Task 6: Handle Settings Activity

Describe the next task. List the subtasks. For example:

- Implement shared preferences to remember setting and display appropriate value on load
- Implement networking code to send value to development server when switch is clicked.
- Implement "test" button to validate setting has taken effect

Add as many tasks as you need to complete your app.

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**"