How to Use this Template

1. Create a new document, and copy and paste the text from this template into your new document [Select All → Copy → Paste into new document]
2. Name your document file: "**Capstone_Stage1**"
3. Replace the text in green

GitHub Username: fireflyfif

Art Place

Description

Art Place is an app for art lovers, artists or people who just enjoy seeing a piece of artwork.
Browse through famous artworks and see their location and other details.
Find out who are their artists and see more details about them too.

Intended User

This app is perfect for Artsy people who love art and want to find their calm place of art everyday.

Features

Main features of the app:

- Displays data from the **Artsy API**
- Displays endless scrollview with artworks
- Saves favorite items into a local database
- Shows details for each artwork
- Shows details for each artists

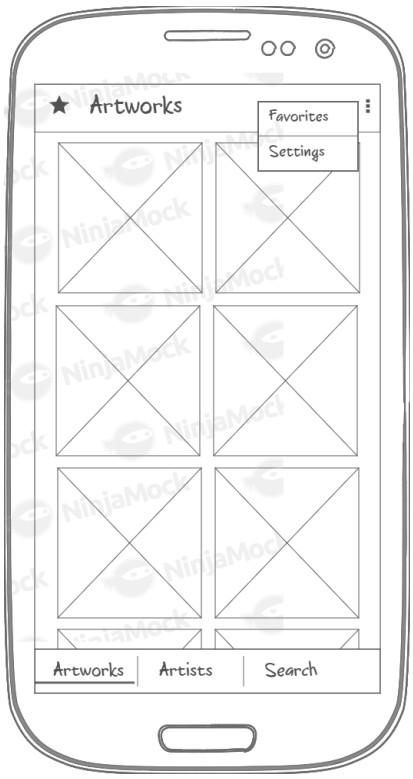
Additional notes:

- App will be written solely in the Java Programming Language.
- App will utilize stable release versions of all libraries, Gradle, and Android Studio.
- App will keep all strings, colors, dimens, etc in the equivalent resource folders.
- App will include support for accessibility. That includes content descriptions, navigation using a D-pad.
- All app dependencies will be managed by Gradle.
- Room will be is used with LiveData and ViewModel, and no unnecessary calls to the database will be made.
- App integrates a third-party library.
- Each service that will be imported in the build.gradle will be used in the app.

User Interface Mocks

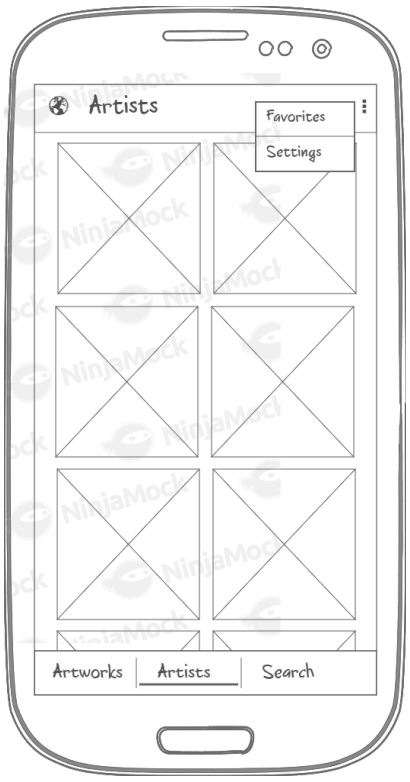
All UI Mocks are created using the NinjaMock program.

Screen 1



The Main Screen will display three Fragments with a Bottom Navigation. The first Fragment will have a Gridview that will display Artworks. The Artworks are fetched from Artsy's API. Each item will contain an image of the corresponding artwork and its title. There will be a menu that will have ***Favorites*** where the user can find all of the their saved items in the local database.

Screen 2



The second Fragment will have again a Gridview that will display Artists fetched from Artsy API. Each item will contain an image that is associated with the artist's profile and name.

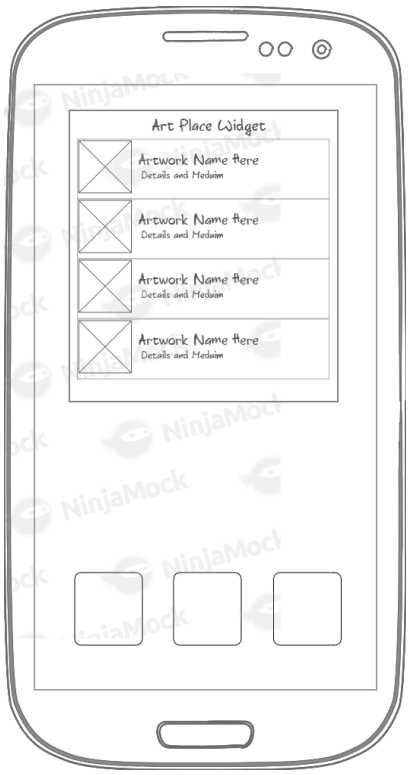
Screen 3

The detail screen will display additional information on the selected artwork or artist from the Gridview. There will be an image at the top, also all included details of the selected item such as title, category, medium, size, year of creation, location, etc. The screen will also have a Floating Action Button that will allow the user to save the item into a local database. There will be a button with the name of the current artist.

Screen 4

This screen will display the artist that was clicked from Screen 3. It will show an image that is associated with the artist, the name of the artist, year of birth and death, nationality, etc.

Screen 5



This is the App Widget that Art Place will offer to the users. It will display a list of the saved favorite items and a quick access to them.

Key Considerations

How will your app handle data persistence?

The app will save data locally using the Room persistence library. Only the selected items by the user will be saved and the app will offer offline access to them. The user will have the ability to delete items one by one or all at once.

Describe any edge or corner cases in the UX.

The user will navigate between the screens by either clicking on an item in a RecyclerView or on a Button. Each detail screen will have an Up Button that will navigate the user to the previous screen. Back buttons will be handled in a similar way.

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

- **Android Support Library**: Handle all of the UI elements that are supported by the Android Library, such as RecyclerView, CardView, ConstraintLayout, CoordinatorLayout, etc.
  The following stable versions will be used:
  ```
  implementation 'com.android.support:appcompat-v7:27.1.1'
  implementation 'com.android.support:recyclerview-v7:27.1.1'
  implementation 'com.android.support:design:27.1.1'
  implementation 'com.android.support.constraint:constraint-layout:1.1.2'
  implementation 'com.android.support:cardview-v7:27.1.1'
  ```

- **Android Architecture Component**: Lifecycle, including LiveData and ViewModel. These libraries will be used so that the app will implement the Android Architecture Components such as LiveData and ViewModel, Repository, etc.
  The following stable versions will be used:
  ```
  implementation "android.arch.lifecycle:extensions:1.1.1"
  implementation "android.arch.lifecycle:viewmodel:1.1.1"
  ```

- **Retrofit2** and **Gson**: Handle easily the network calls on a background thread.
  The following stable versions will be used:
  ```
  implementation 'com.squareup.retrofit2:retrofit:2.4.0'
  implementation 'com.google.code.gson:gson:2.8.4'
  implementation 'com.squareup.retrofit2:converter-gson:2.4.0'
  ```

- **ButterKnife**: Field and method binding for Android views which uses annotation processing to generate boilerplate code.
  The following stable versions will be used:
  ```
  implementation 'com.jakewharton:butterknife:8.8.1'
  annotationProcessor 'com.jakewharton:butterknife-compiler:8.8.1'
  ```

- **Picasso** - Handle the loading and caching of images
  The following stable version will be used:
  ```
  implementation 'com.squareup.picasso:picasso:2.71828'
  ```

- **Room** - Persistence library that provides an abstraction layer over SQLite to allow for more robust database access while harnessing the full power of SQLite.
  The following stable versions will be used:
  ```
  implementation 'android.arch.persistence.room:runtime:1.1.1'
  annotationProcessor 'android.arch.persistence.room:compiler:1.1.1'
  ```

- **Paging Library** - Consume data from a data source that contains a large number of items, but only display a small portion at a time. Help the app to observe and display a reasonable subset of the data.
  The following stable version will be used:

```
implementation 'android.arch.paging:runtime:1.0.1'
```

Describe how you will implement Google Play Services or other external services.

The app will be using **Google Ads** and **Google Analytics.** Ads will be visible on the bottom edge of almost every screen. Analytics will be used for detecting the most used features.

Next Steps: Required Tasks

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

Task 1: Project Setup

List the subtasks such as:

- Find an API that will offer good endpoints for art displaying app
- Configure libraries that will be used in the app in the build.gradle App module
- Add Internet permission to the Manifest
- Create a class that will extend from Application and then connect it with the `android:name` attribute in the Manifest

Task 2: Implement UI for Each Activity and Fragment

- Build UI for MainActivity that will display a gridview with images for each item.
- App theme will extend AppCompat.
- App will be using an app bar and associated toolbars.
- Build UI for the second Detail Activity that will display detail information for the clicked item.
- Build UI for the third Activity that will hold information for the clicked Artist.
- Build UI for the Favorites Activity that will hold all saved items from the user.
- Build UI for the Detail screen that will show the saved information from the Favorites.
- App will be using standard and simple transitions between activities.

Task 3: Get familiar with the chosen API

- Make a developer's account and retrieve an authorisation Token that will be used to access the API content
- Make a few example calls to the API that will be used in the app
- Make POJO classes from the returned JSON response

Task 4: Make the network call using Retrofit2

- Create the remote service interface that will hold the methods to different endpoints
- Create the API Manager class that will configure the Retrofit calls
- Create the classes that will hold the Retrofit calls, in my case: a DataSource will be needed for storing the fetched data in a PagedList using the Paging library

Task 5: Implement the Android Architecture Components

- Add a ViewModel that will store and manage UI-related data in a lifecycle conscious way
- Use LiveData for observing mutable data throughout the app
- Use the Repository pattern

Task 6: Implement Room persistence library for storing data locally

- Create an Entity model class that will represent a table within the database
- Create a database class by extending RoomDatabase that will contain the database holder and serves as the main access point for the underlying connection to the app's persisted, relational data
- Create the DAO interface that will contain the methods used for accessing the database
- Make calls to the database using an AsyncTask

Task 7: Implement Google Play Services

- Google Admob will be used, the app will display test ads banners on the bottom of some screens.
- Implement Google Analytics for logging the most used features. The app will create only one analytics instance.

Task 8: Handle Error Cases

- Display Empty View for no Internet connection cases, provide adequate messages for the user
- Test the app for errors
- Provide placeholders for no image cases
- Check if there is no internet connectivity with a separate asynchronous method (using an AsyncTask)
- App validates all input from servers and users. If data does not exist or is in the wrong format, the app logs this fact and does not crash.

Task 9: Implement Widget

- Create layout for the widget that will display a list of favorite items
- Create a Remote Adapter
- Create a WidgetProvider

Task 10: Provide Support for Accessibility

- Provide content descriptions
- Provide support for navigation using a D-pad
- Enable RTL layout switching on all layouts

Task 11: Prepare a Clean Build for the Reviewer

- App will build from a clean repository checkout with no additional configuration.
- App will build and deploy using the installRelease Gradle task.
- App will be equipped with a signing configuration, and the keystore and passwords are included in the repository. Keystore is referred to by a relative path.

Submission Instructions

- After you've completed all the sections, download this document as a PDF [File → Download as PDF]
    - Make sure the PDF is named "**Capstone_Stage1.pdf**"
- Submit the PDF as a zip or in a GitHub project repo using the project submission portal

If using GitHub:

- Create a new GitHub repo for the capstone. Name it "**Capstone Project**"
- Add this document to your repo. Make sure it's named "**Capstone_Stage1.pdf**"