---


Description

Intended User

Features

User Interface Mocks

- Screen 1
- Screen 2
- Screen 3
- Screen 4
- Screen 5
- Screen 6
- Screen 7

Key Considerations

- How will your app handle data persistence?
- Describe any edge or 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 or other external services.

Next Steps: Required Tasks

- Task 1: Project Setup
- Task 2: Get data from API
- Task 3: Implement UI for Each Activity and Fragment
- Task 4: Create ViewModels
- Task 5: Create final API endpoints and define methods
- Task 6: Create Repositories
- Task 7: Get reference of repository in ViewModel
- Task 8: Create database
- Task 9: Update repositories
- Task 10: Create Notification
- Task 11: Create background service
- Task 12: Implement testing

**GitHub Username**: https://github.com/PaloPodstreleny

# Artist

## Description

Artist is the right place for all art lovers and people who try to find inspiration in the works of well-known but also unknown authors. This app offers a large number of artwork that can be filtered based on a variety of criteria. In addition, it provides basic information about the authors and the periods in which they worked. If you are a lover of shows, you will certainly be pleased with the list of future shows you can visit.

## Intended User

Application is intended for regular people, people who seek inspiration and love art.

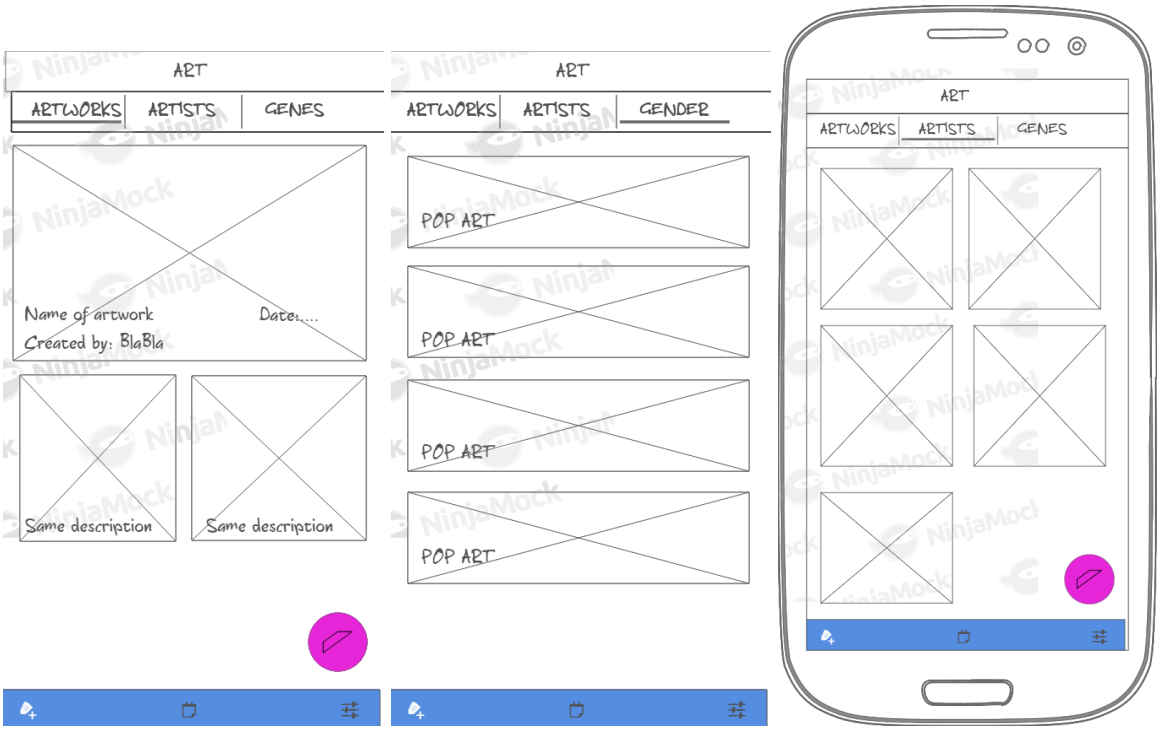
## Features

- App fetch information from https://www.artsy.net/ API
- App saves information for offline use
- Show art information to user (artists, artworks, shows)
- App shows notification to users who subscribe for daily inspirational art -> user can turn on/off this option in settings
- App uses Firebase Job Scheduler and intent service to re-fetch new artwork data once a day.

## User Interface Mocks

”Pop art”, ”Flatness”, ”Bright contrast” represent **Genes**

## Screen 1




Main page: This is the first screen of application. There are 5 important elements:

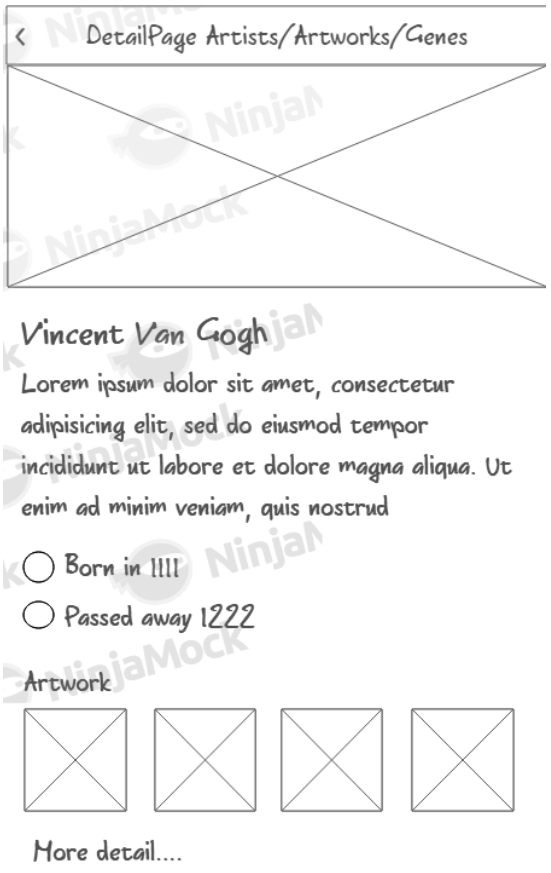
1) **AppBar**
2) **TabLayout**
3) **ViewPager / List of data**
4) **Fab**
5) **Bottom navigation**

1) AppBar contains title which describe in which section of bottom menu we are right now.
2) Art page contains 3 Tabs/Fragments (Artists, Artworks, Genes), we can switch between tabs by clicking on tab or swiping to left/right.
3) Each tab/fragment contains list of data. User can choose between:

- Artists -> list of artist (can be filtered by criteria (genes/release date)
- Artworks -> list of artworks (can be filtered by criteria (genes /release date))
- Genes-> list of art genes( can not be filtered by any criteria)
- After clicking on list item user will be moved to DetailPage

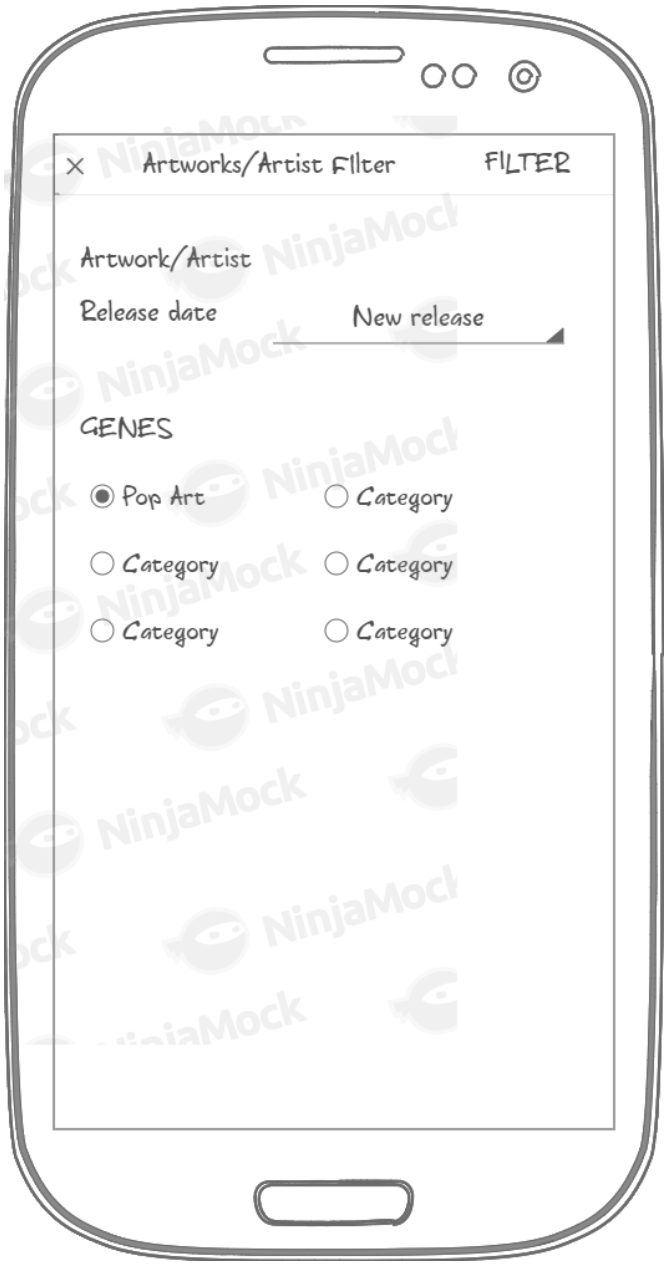
4) Fab button is used for filtering (Artist, Artworks) list of data. After clicking on Fab, user has to pick additional criteria for filtering data.
5) Bottom navigation contains 3 elements (Art, Shows, Settings).

## Screen 2



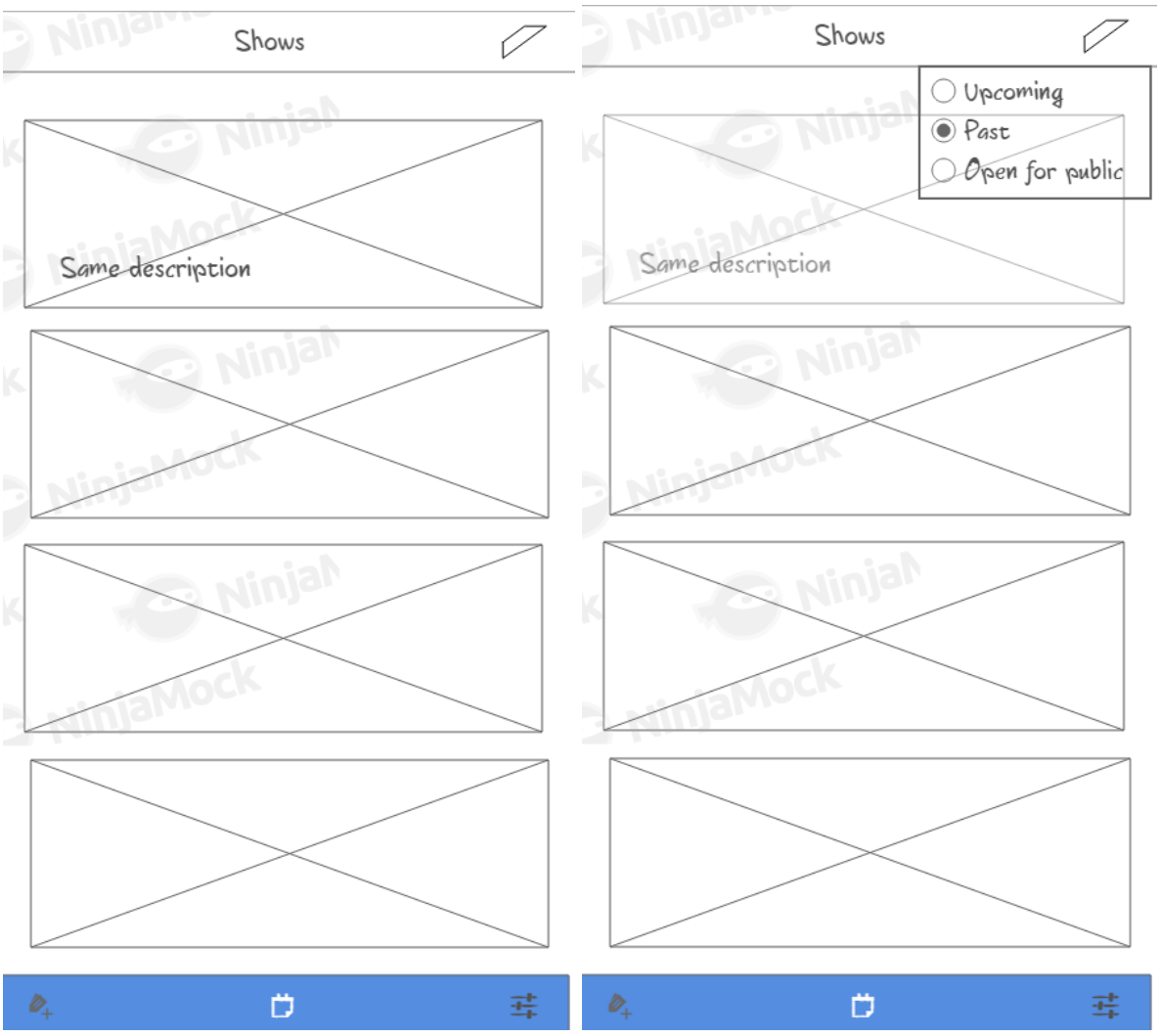

**Detail Page**: This screen appears after clicking on artists list item. Similar details screen will appear after clicking on artworks/genes but with different values. Artists details page contains information about artist. Fields like name, gender, biography, day of birth and death, artworks...are shown here.

## Screen 3




**Filter page**: This page appears after clicking on Fab which is located in artists/artworks section. User can filter list of artworks/artists according to genes or release date. After choosing right category, user can press filter button or cancel button which are in the app bar.

Screen 4




**Shows Page**: This section of the app is for displaying shows. User can filter shows according to time (upcoming, past shows...).

Screen 5

**Show Detail Page**: This page shows details of selected show. Fields like name, description, press description, ending date, status (running, finished), artworks are displayed here.

Screen 6

**Settings Page**: Settings page is the last section of the app.User will receive daily motivational art by default. If he doesn't like this option, he can simply turn off the notification here.

Screen 7

**Error page**: This is demonstration of error page use-case when there are no data to show. Retry button is for getting data from internet.

# Key Considerations

### How will your app handle data persistence?

The app will use Room persistence library (abstraction layer over SQLite). Everytime data will be fetched from API it will be saved to database which serves as the single source of truth. All parts of application are accessing database via repository.

### Describe any edge or corner cases in the UX.

If the user downloads/installs the app for the first time and he is not connected to the internet proper warning/error message with button has to be shown. In this application user will be able to fetch data by clicking on “retry button”. This button will fire network call to get data. Data will be saved to local database from where will be propagated to user page.

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

- Glide to handle the loading and caching of images
- Retrofit to handle network calls and connection to API
- Using Android architecture components:
    - Room - for easy local data persistence
    - LiveData -to ensure UI matches data in database, no more manual lifecycle handling and more.
    - ViewModel - to make abstraction layer which provide data for UI, resolve configuration change problems
- Support design library for using FABs and another great Views
- RecyclerView library to make list visualization fast and effective
- ButterKnife to simplify boilerplate code to getting Views
- Firebase to implement Analytics and performance monitoring
- FirebaseJobDispetcher for scheduling background jobs in the app
- Espresso for sofisticated UI testing
- Mockito for mock classes which will be used for tests

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

I will implement Firebase Analytics to collect and analyze information about users and Firebase performance monitoring to retrieve performance data from the app and review and analyze them in Firebase console.

# Next Steps: Required Tasks

The app architecture will be built with Android architecture components which helps to make robust, testable and easy to maintain application.

## Task 1: Project Setup

Before we start to code it's good to choose overall architecture and setup Gradle for future use. This app will use architecture which is defined in https://developer.android.com/jetpack/docs/guide. It's very similar to MVVM architecture.

First step will be to configure build.gradle files.

- provide repositories and libraries that will be used in the project
- define variables for libraries version (easy to change values of variables in the future)
- make sure project compiles after all changes we made

## Task 2: Get data from API

It's really great to know if we are able to get data from API we use before we start to make UI. Steps to check API calls:

- include INTERNET user-permission to manifest
- implement Retrofit Service (endpoint)
- add method which will return some data
- implement Retrofit provider
- check if we are getting data

## Task 3: Implement UI for Each Activity and Fragment

Implementation of the

**Main page:**

- Create Main page activity
- Create layout for Main page activity with bottom navigation
- Create fragments for Art/Shows/Settings
- Create layouts for Art/Shows/Settings
    - Art layout contains ViewPager for Artists/Artworks/Genes fragments

**Art fragment pages:**

- Create fragments for Artists/Artworks/Genes
- Create layouts for this fragments

**Detail pages:**

- Create detail activities for Artists/Artworks and Shows
- Create layouts for this activities

## Task 4: Create ViewModels

Next step is creation of ViewModel which provides data for UI.The app will contains

- ArtistViewModel
- ArtworkViewModel

- GenesViewModel
- ShowsViewModel

Next step is to observe data from ViewModel -> usage of LiveData data holder class. For now we can return fake data to UI.

## Task 5: Create final API endpoints and define methods

Specify final implementation of API calls:

- create own retrofit adapter which convert retrofit Call into LiveData
- update Retrofit builder, specify newly created adapter
- update endpoint methods to return LiveData<> objects

## Task 6: Create Repositories

Next step is creating repositories classes. Repository class will provide data from different sources. For now we will provide data just from network. Repository can make call to API and return data.

Repository should use Singleton pattern.

## Task 7: Get reference of repository in ViewModel

We can create reference to repository inside ViewModel and call repository method to retrieve data. We are able to show for the first time real data in the UI.

## Task 8: Create database

Create entities, daos and database class and specify relationships between tables.

## Task 9: Update repositories

It's time to update repository class. Change the logic of retrieving data:

- if data is not available in database -> retrieve data from network and save it to database. Then return saved data from database
- if data is not available in database and network problem -> retrieve error message
- if data is available in database -> return database data

The app should work and display proper data in this step.

## Task 10: Create Notification

Create notification which will show random artwork from database every day. If user's settings notification is off than does not show anything.

## Task 11: Create background service

Intent service will fetch newest artworks every day. JobDispatcher is used to schedule this job.

## Task 12: Implement testing

Provide basic UI  testing for activities/fragments.

### Task 13: Application signing

- create keystore and key
- create signing config in build.grade
- signing configuration of build type

---

**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**"