GitHub Username: sonyamoisset

N360

Description

N360 is a news app that allows users to access news from a large range of national and international news sources and websites everyday, share news with their friends easily, search by keywords and save their favorite articles to read it later on.

Intended User

This app is intended to anyone interested in the latest news, from students to professionals, and want to grasp different sources of news at one place.

Features

For the users:

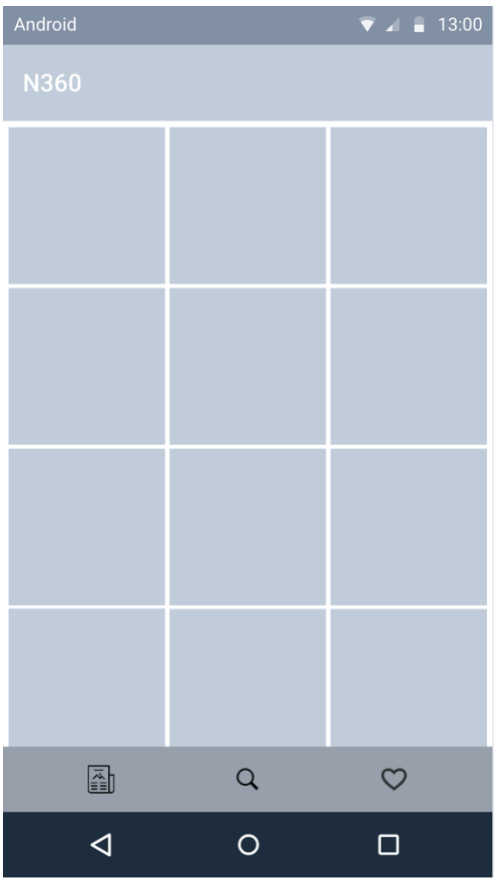
- Read news from national and international news sources/websites
- Search by keywords/sources
- Bookmark favorite articles
- Companion homescreen widget to list all favorites articles
- Free/paid versions of the app with interstitial ads.

For the developers:

- Analytics to check which activities/screens are the most popular

User Interface Mocks

Screen 1. Main screen with all media sources



The main page allows the user to choose among several news sources. When clicking on any news source, the user will be redirected to a detail activity of the chosen news source.
The app queries the newsapi.org api to fetch news stories.
A bottom navigation will be accessible on each screen (except for the webview), with a news button to access this screen, a search button to access the search screen (screen 5) and a favorite button to access the favorite screen (screen 6).

Screen 2. Detail Activity with one selected media source

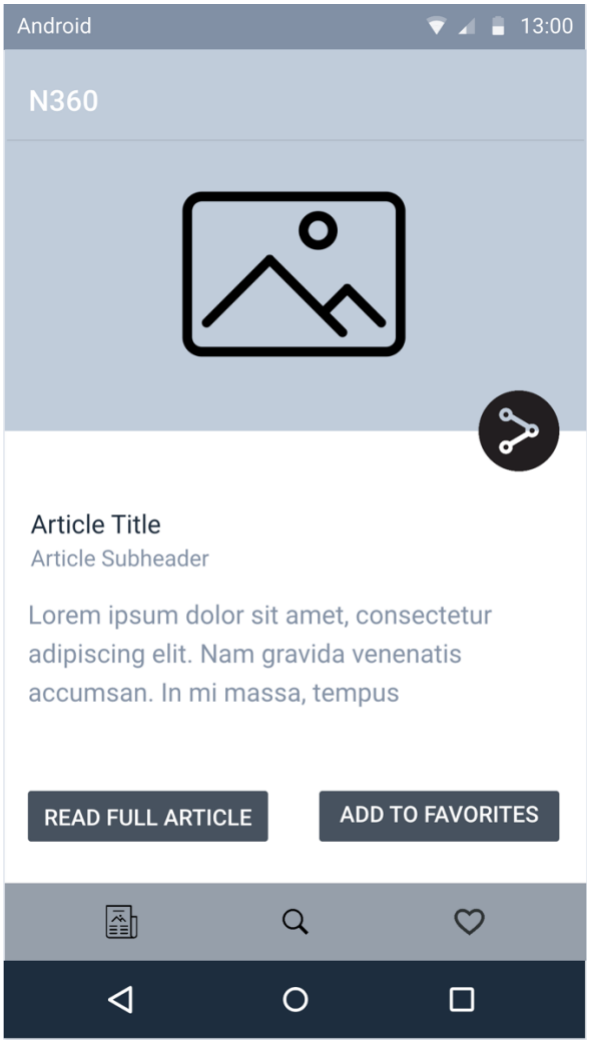
When the user has selected a media source, the next screen will show a detailed view of this selected media source with a description box at the top including the logo, the name of the source and a small description of the media.

Under this description box, the user will see a list of cards which includes each article from this selected media source. Each card includes a thumbnail, a title and a subtitle.

Each list item on this screen displays relevant text and information about the story

Stories shown on this screen update properly whenever new news data is fetched from the API.

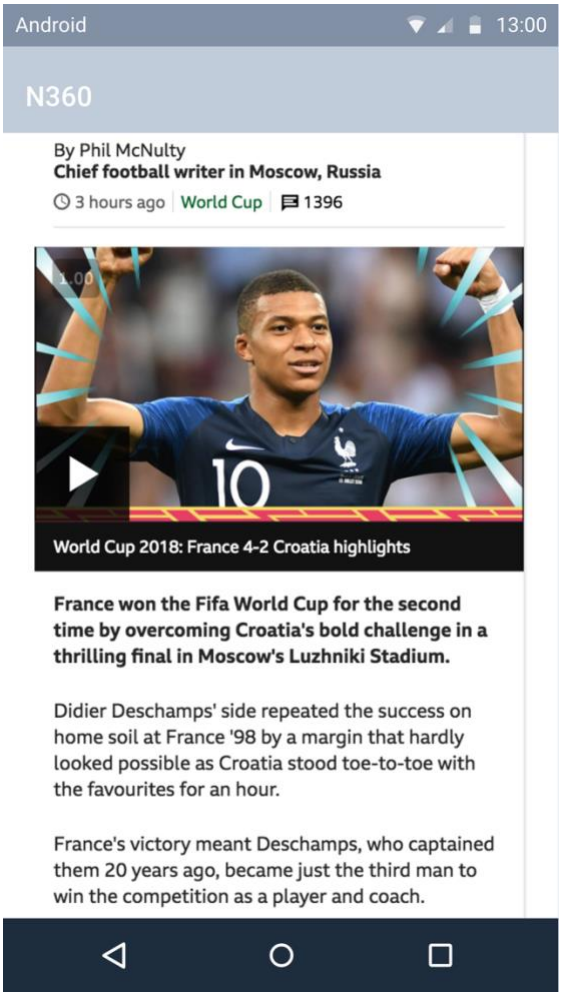
Screen 3. Detail Activity with one selected article



When the user has selected an article, the next screen will show a detailed view of this selected article with a picture and share button at the top. Under the picture, the user will see the title, subtitle and excerpt of the selected article.

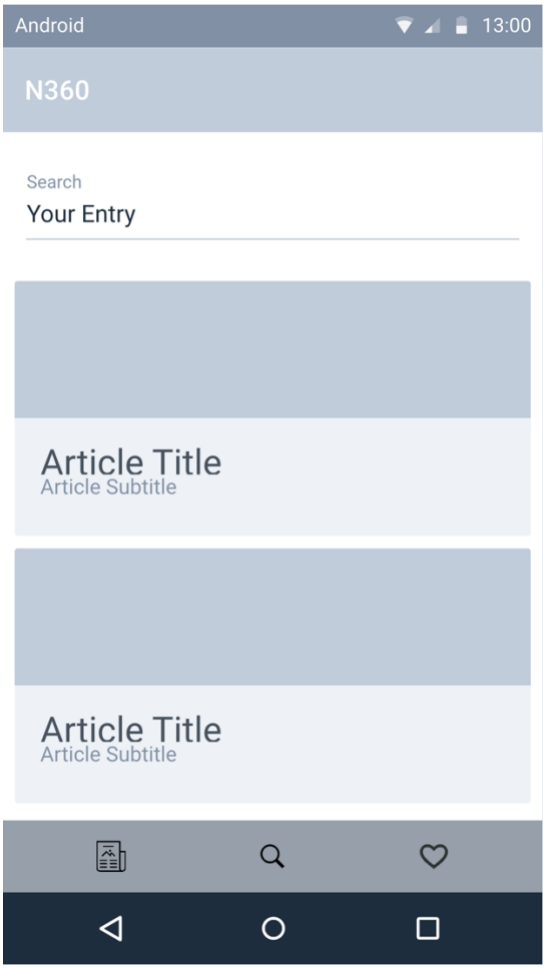
The user will also see two buttons, one to read the full article and one to add to favorites.

Screen 4. Detail Activity webview of the selected article



When the user has clicked on the button read full article, the next screen uses an intent to open in the user's browser a webview of the article.

Screen 5. Search Screen



When the user has clicked on the search button on the bottom navigation, the screen will show an input field at the top where the user can enter a search term/query.

This screen will first be empty before the user enters a query.

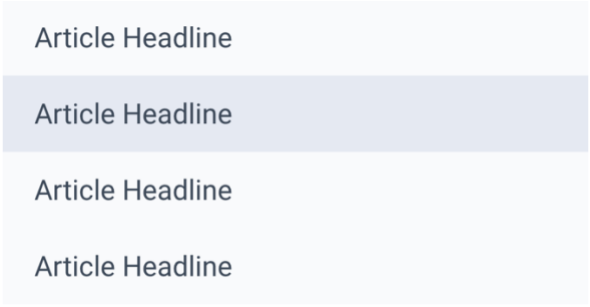
Once the search is performed, a list of articles/cards will appeared under the search box.

Screen 6. Favorite Screen

When the user has clicked on the favorite button on the bottom navigation, the screen will show a list of saved articles/cards.

If there is no article saved, the user will see a sentence letting them know they have to save articles to read them here.

Widget



When the user is on the homescreen, and creates a widget, he/she will be able to see a list of article headlines that has been saved.

When an article is clicked, it will open the news in a webview.

Key Considerations

How will your app handle data persistence?

Room will be used to handle data persistence. It will help create a cache of the app's data on the device that's running the app. This cache will serve as the app's single source of truth, allowing the users to view a consistent copy of key information within the app, regardless of whether users have an internet connection.

Describe any edge or corner cases in the UX.

The network state will be displayed.
In case there is no favorites marked by the user on the favorite screen, a message will be displayed.

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

- Android dependencies/UI dependencies
  - AppCompat (com.android.support:appcompat-v7:27.x.x)
  - Support (com.android.support:support-v4:27.x.x)
  - Constraint-layout (com.android.support.constraint:constraint-layout:1.x.x)
  - RecyclerView (com.android.support:recyclerview-v7:27.x.x)
  - CardView(com.android.support:cardview-v7:27.1.1)

- Retrofit: for the HTTP connection and JSON parsing. It handles the asynchronously on a separate thread.
- Picasso: for the images/thumbnails. It also allows caching data for a better user experience.

- Android Architecture Components (Android Jetpack) to implement a clean, maintainable and scalable code using the latest Android features.
    - Room
    - LiveData
    - ViewModel
    - Data Binding

- Dagger 2: for dependency injection

- Google Play Services to use AdMob and Analytics

Describe how you will implement Google Play Services or other external services.
Admob: the user will see an interstitial ad between screen 3 and screen 4 (the webview)
Analytics: to add screen tracking to see whenever the user opens or changes screens on the app.

Next Steps: Required Tasks

Task 1: Project Setup

The project will run on Android Studio 3.1.3 and the compiling will be done using Gradle 4.4 plug in.
The target SDK is 27 and the minimum SDK is 19.
The app will be written solely in the Java Programming Language and will use only stable release versions of all libraries, Gradle and Android Studio.

- Create a new project in Android Studio
- Add the Google Play Services dependency to the Gradle file, all necessary dependencies mentioned earlier, all Android Architecture Components dependencies and all Dagger dependencies
- Configure the manifest file to enable the necessary permissions for accessing internet and user's location.
- Create folders to start decoupling the code under different subfolders and keep a clean architecture. The following is to give a rough idea of the composition of what will be the app:
    - api (webservice)
        - ApiResponse.java
        - NewsWebService.java

- db (database/Room)
  - dao
    - NewsDao.java
  - entity
  - AppDatabase.java
- di (dependency injection/dagger boilerplate)
  - component
    - AppComponent.java
  - key
    - ViewModelKey.java
  - module
    - AppModule.java
    - BuildersModule.java
    - ViewModelModule.java
- network
  - LiveDataCallAdapter.java
  - LiveDataCallAdapterFactory.java
  - NetworkBoundResource.java
- repository
  - NewsRepository.java
- ui
  - common
    - BindingAdapters.java
    - ClickHandler.java
  - detail
    - adapter
    - fragment
    - module (Dagger)
  - main
    - NewsActivity.java
    - NewsActivityModule.java
    - NewsAdapter.java
    - NewsViewModel.java
  - widget
    - NewsPreferencesUtils.java
    - NewsRemoteViewsFactory.java
    - NewsRemoteViewsService.java
    - NewsWidget.java
- utils
  - IdlintResource.java
- viewmodel
  - NewsViewModelFactory.java
- vo (network)

        - Resource.java
        - Status.java
    - AppExecutors.java
    - NewsApp.java
- Note: this architecture design is just to give an idea on how the app will be decupled. There is room for changes and improvements that might be discovered during the development/coding phase. I have used this architecture for the Baking App with success, based on the Github Browser Sample with AAC (https://github.com/googlesamples/android-architecture-components/tree/master/GithubBrowserSample) and would like to apply it to this app as I think it is clean and scalable.

Task 2: Building the UI: Implement Each Activity and Fragment

- Build UI for the main screen
    - Grid view with RecyclerView
    - Bottom navigation with 3 buttons (news/search/favorites)
- Build UI for the detail view of a selected media
    - Reusable card component for the articles
- Build UI for the detail view of an article
    - Implement a share button
    - Implement a read full article button
    - Implement a add to favorite button
- Build UI for the search screen
    - Implement a search field input
- Build UI for the favorite screen
- Databinding will be used on the xml files

Task 3: Implement Data Fetching, Storage and Repository

- Get an API key on newsapi.org.
- Have a play with the different endpoints to see the JSON responses.
- The UI will be driven from a model, hence the implementation of entities, dao and one Room database.
- The database and the WebService will be singletons to only have one instance in the app (Dagger DI).
- Make sure the JSON response is parsed correctly, and that the relevant information is stored in the app.
- Implement the WebService with the HTTP GET requests.
- Implement RecyclerView to display the data into a list for the main screen and for the detailed views. (also as per following the folder architecture discussed in the first task)

- Implement a SyncAdapter or JobDispatcher to regularly pulls data from the newsapi WebService, and updates the data in its cache at regular intervals.
- Implement a NewsRepository module to handle data operations. The repository will know where to get the data from and what API calls to make when data is updated. It will connect the ViewModel to the WebService.
- Dependency Injection will allow classes to define their dependencies without constructing them. We will use Dagger 2 library for implementing DI in the app.
- Connect ViewModel to the repository.
- A rule of thumb would be that we should not store any app data in our app components and our app component should not depend on each other. A clear separations of concerns in the app should be followed, hence the use of AAC and the folder architecture described in task 1.

Task 4: Handle Error Cases

- Handle errors when there is no access to a network
    - The app will check whether the device is connected to the internet and responds appropriately. The results of the request will be validated to account for a bad server response or lack of server response.
- Handle errors when there is no data returned from the API
    - The app will make sure to validate all inputs from servers and users. If the data does not exist or is in the wrong format, the app logs it and does not crash.
- When there is no data to display, the app shows a default TextView that informs the user how to populate the list of favorites.

Task 5: Add Google Play Service 1. Admob

- Import the Google Mobile Ads SDK.
- Initialize MobileAds with AdMob App ID.
    - Sample AdMob app ID will be used in this case.
- Add a free flavor
    - Implement an ad format, an interstitial ad.
- Add a paid flavor
    - Remove the ad (and any dependencies) from the paid flavor.

Task 6: Add Google Play Service 2. Analytics

- Implement dependencies in build.gradle.
- Create global_tracker.xml and add tracking ID.
- Add screen tracking to any relevant screen on the app

  - Main screen with all different media sources to see which one are the most popular.
  - Detail screen of one article to check user's behavior (do they save the articles, do they click to read the full article, do they share the articles…)

Task 7: Implement the Widget

- Widget is declared in the manifest file.
- Add AppWidgetProviderInfo metadata.
- Create App Widget layout.
- Implement widget to display list of articles' headline on the homescreen, following folder architecture on task 1.

Task 8: Allowing for Localization and Making the App Accessible

- RTL layout switching is enabled for all layouts.
- Content descriptions provided for all icons and images
- All strings are in a strings.xml file.

Task 9: Polish the App Following Material Design Principles

- Specific theme will be used with primary and accent colors and will be defined in styles.xml. Color theme will not impact usability of the app.
- App will use Design Support library and its provided widget type (FloatingActionButton for the share button) and will properly specifies elevations.
- App will use fonts that are Android defaults.
- App will extends AppCompat.
- App will uses an app bar and associated toolbars.
- App will use standard and simple transitions between activities.
- Text sizes will be defined in sp.
- Lengths will be defined in dp.
- Padding and margins will be used appropriately, such that the views are not crammed up against each other.

General Tasks:

- The code will be made easily readable such that a fellow programmer can understand the purpose of the app.
- Clean code architecture will be used as much as possible following the guidance of the new Android Jetpack documentation.

- A constant naming convention will be used: all variables, methods, and resource IDs will be descriptively named such that another developer reading the code can easily understand their function.
- A constant formatting will be used: the code will be properly formatted (no unnecessary black lines, no unused variables or methods, no commented out code).
- A README file will be populated in the project to explain the features of the app, provide some screenshots, the libraries used and how to get a API key from the newsapi website.
- Note: if during the development, any frameworks, libraries, or SDKs have essential limitations while testing, I might have to evaluate other options if the first one didn't work.