Description

Intended User

Features

User Interface Mocks
- 1 - Main App View
- 2 - Main App View - Search
- 3 - Main App View - Filtering
- 4 - Details page
- 5 - Main app view
- 6 - Large screen display
- 13 - Widget

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: Basic navigation
- Task 3: Fetch and parse/store the data
- Task 4: Display the data
- Task 5: Search/filter/order
- Task 6: Details page
- Task 7: Save articles for offline use and display them
- Task 8: Add the widget
- Task 9: Start working on the UI
- Task 10: Add the free Ads
- Task 11: Add analytics
- Task 12: Final UI polishing, tests etc. + Prepare for submission

GitHub Username: jorjSB

Articles Registry

Description

Document, share, save and read offline articles as old as 1851. Powered by The New York Times.

Intended User

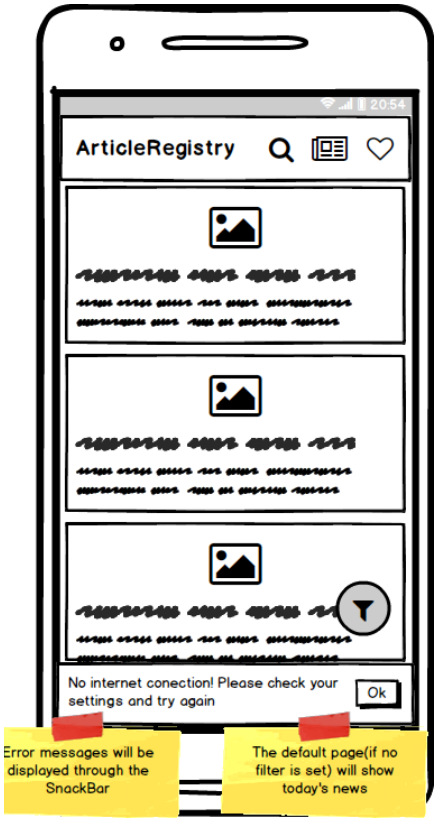
Any historian, student, teacher or just curious George can obtain news related to any subject as old as 1851 for academic purposes or just general knowledge, to satisfy one's curiosity or just a good conversation topic.

Features

- Accesses online, verified information;
- Saves data for offline use(option for each article);
- Offers a widget that displays the user's saved articles;
- Offers share capabilities;
- Supports tablet(large displays) with adapted UI;
- Supports RTL layout switching;
- Includes support for accessibility;
- Material design UI;
- Displays ads;
- Paged lists and LiveData backed by a database;
- Java language will be used for development
- App keeps all strings in a strings.xml file and enables RTL layout switching on all layouts
- App includes support for accessibility like content descriptions
- Will use (Retrofit's) Asynchronous requests for fetching data from the API.

User Interface Mocks

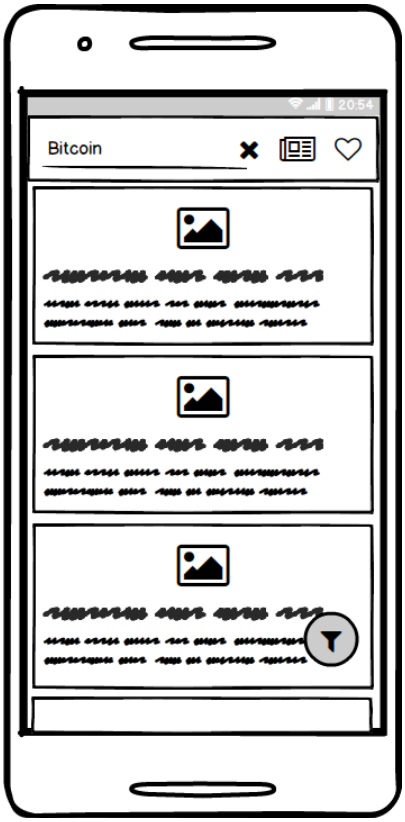
1 - Main App View



Main application view:

- Provides an AppBar with buttons for: “Searching Articles”, “Recent Articles” and “Saved articles”
- A list with the articles fetched from DB(saved) or cached in DB, fetched from the NY Api. Fetching and displaying articles will be paginated.
- A fab used for adding filters/sorting for a search(current).
- A SnackBar used to display informations to the user like Error messages or action feedback, like searched query;

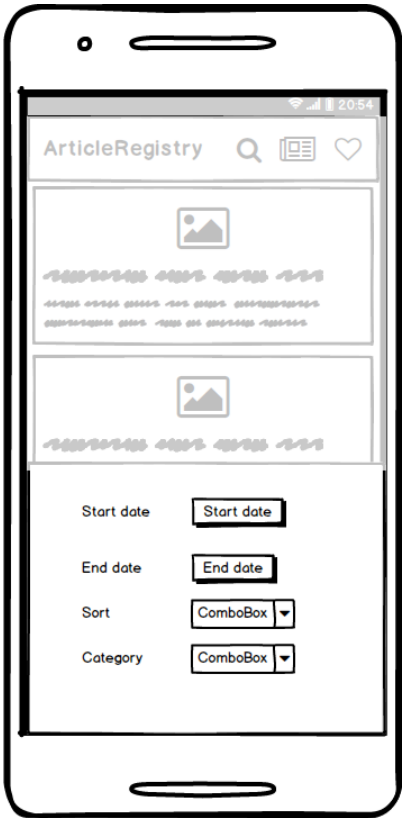
2 - Main App View - Search



Main application view - SEARCH:

- The user can click on the search button in the AppBar to open a SearchWidget. On submit a new “N.Y. API” search will be initialized and the results cached in the local DB, from which are displayed using LiveData and paging.

3 - Main App View - Filtering



Main application view - Filtering:

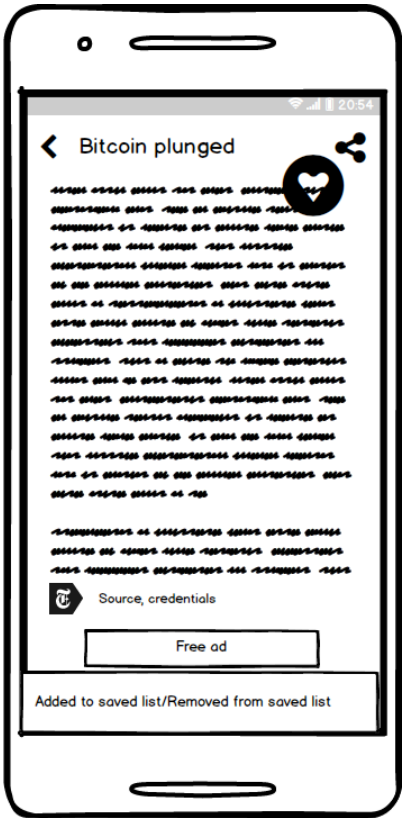
- When the fab is clicked, a popup with filtering and sorting options is displayed. After submitting, a new API search will be triggered, including the previous search term(if any) and the newly set filters.
- The results will replace the old results from the database(except the saved ones).

4 - Details page

Details page:

- When the user selects an article from the list, the article's details page will be shown, containing relevant data, an image, sharing button in the AppBar;
- Will also provide a fab for "saving to favourites/removing from favourites" - that action will flag the article in the DB so it won't get erased.
- Ads will be displayed at the bottom of the page;
- A SnackBar will give the user feedback regarding his actions;
- For the mobile version it will use the CollapsingToolbarLayout, showing the main image expanded, and will shrink/disappear on scroll;

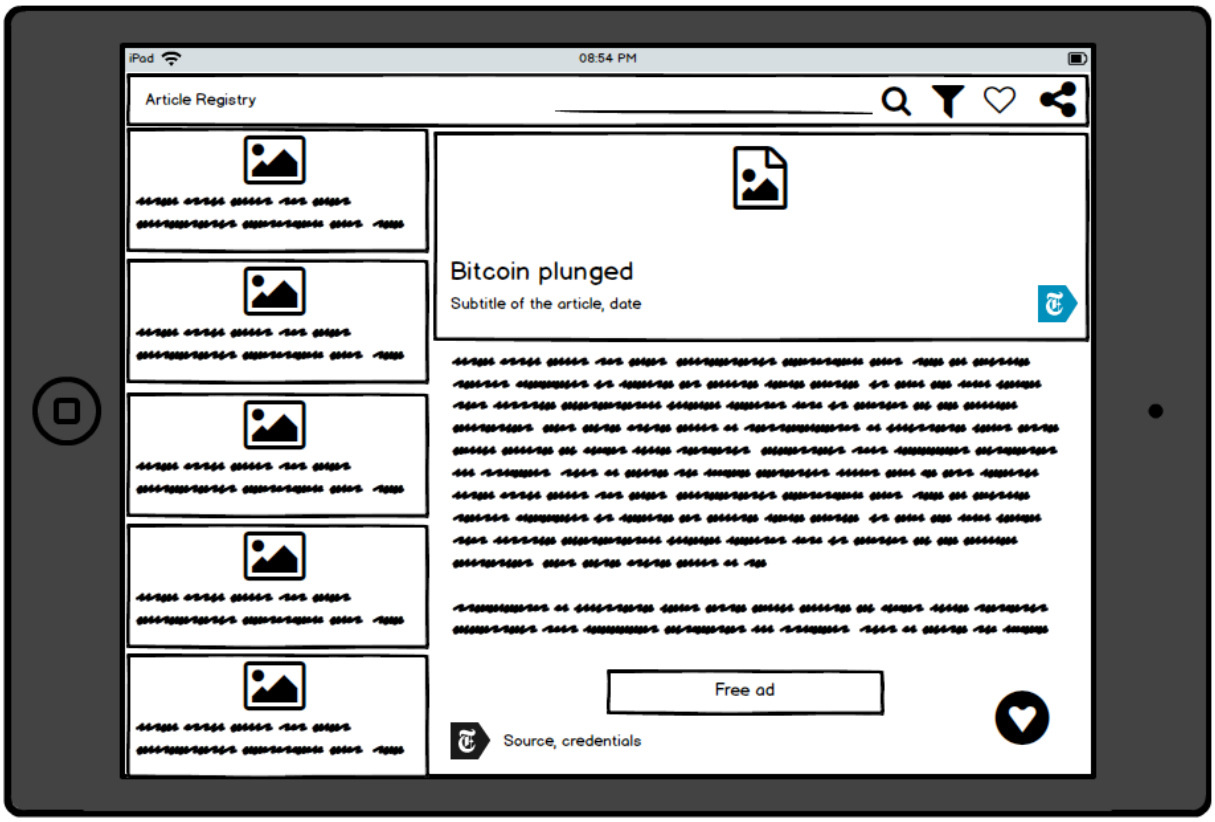
5 - Main app view



Details page:

- Whens scrolled down, the CollapsingToolbarLayout will hide the Image, leaving the entire display for the content of the article;

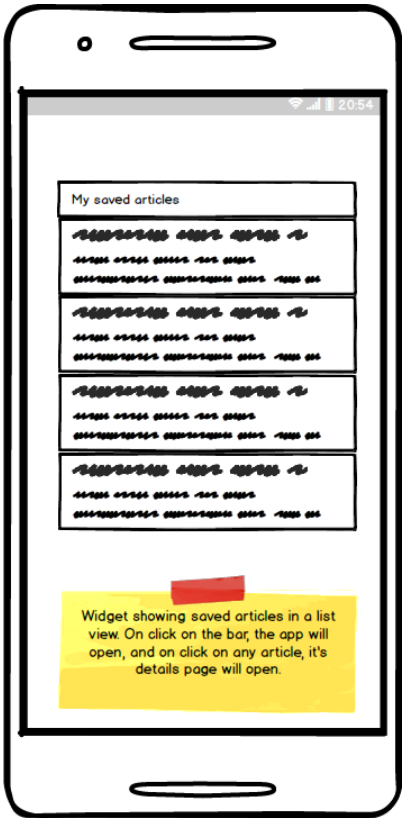
6 - Large screen display



Large screens (tablet):

- On large screen display, the master/detail UI design pattern applied. (*Balsamiq only offers iPad layout for large screens)
- The AppBar will be reconfigured to show the relevant actions for the navigation/list update
- The Fab will handle each displayed article add/remove from favourites action;

13 - Widget



Widget:

- Will display a list with user’s saved articles(if any);
- On click on the header the application will be launched;
- On click on individual list item, the corresponding article’s details page will be opened.

Key Considerations

How will your app handle data persistence?

Room will be used(along with its LiveData and ViewModel) for caching (storing / accessing / deleting) the articles fetched from the NYTimes API.
When retrieving the data from the database, Room's pagination(DataSource.Factory) will be used along with LiveData.
Whenever a new set of data(articles) is fetched, it will be parsed from Json to POJO's, and stored in the DB using DAO's, the entire DB will be erased except the articles that were flagged as "favourite" by the user.
The "favourite" articles, flagged accordingly can be accessed directly from the app or from the widget without internet connection;

Describe any edge or corner cases in the UX.

When opening the app(normal behaviour is to fetch the latest news), if no internet connection is available, a Snack will inform the user about that, and let him know that his saved articles are available even offline.

The navigation is pretty straight-forward: from the main page(list with articles), the user can navigate to the article's detail page or the filtering page, and return using back button.

This will not apply on the large screen layout: the back navigation will not be necessary as the "main list" and the other pages are always displayed or changed from the navbar or the articles list.

The user can navigate with the on-screen action buttons available and also using the device's back button for going back from a datepicker, filter screen, article details etc.

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

- **Retrofit** - transforms HTTP API into a Java interface. Needed to configure and access the API, turn results from JSON->GSON->POJO's;
    - implementation "com.squareup.retrofit2:retrofit:2.2.0"

    - implementation "com.squareup.retrofit2:converter-gson:2.2.0"
- **AndroidSupport/Design/RecyclerView** - support for older devices/Android versions; Design - Material components like FAB's erc. and provides RecyclerView.
    - com.android.support:appcompat-v7:27.1.1
    - com.android.support:support-v4:27.1.1
    - com.android.support:recyclerview-v7:27.1.1
    - com.android.support:design:27.1.1

- **LoggingInterceptor** - needed for logging the API calls/responses. Development without will be close to impossible.
    - com.squareup.okhttp3:logging-interceptor:3.9.1
- **DBDebugger** - can see the DB in browser, so I can analyse what and how is being stored into the DB, and that the DB actions are having the expected result.
    - com.amitshekhar.android:debug-db:1.0.4
- **Room** - over-database ORM layer. Offers a lot of uses like DAO's, listening to LiveData through a ViewModel, oaged lists with items stored etc.
    - android.arch.persistence.room:runtime:1.1.1
    - android.arch.persistence.room:compiler:1.1.1
- **Lifecycle** - offers architecture components for the MVVM architecture, LiveData, VM.
    - android.arch.lifecycle:extensions:1.1.1
    - android.arch.lifecycle:compiler:1.1.1
- **Paging** - loading paged(Live) data into RecyclerView
    - android.arch.paging:runtime:1.0.1
- **ConstrainLayout** - will use Constrain Layouts in the app's layouts
    - com.android.support.constraint:constraint-layout:1.1.2
- **Butterknife** - field and method binding for Android views, cleaner code;
    - com.jakewharton:butterknife:8.8.1
    - com.jakewharton:butterknife-compiler:8.8.1
- **Picasso** - image handling(fetching from url, placeholder, errors etc.)
    - com.squareup.picasso:picasso:2.71828
- **GoogleAds** - will display (test)ads in article's detail page using GoogleAds lib.
    - com.google.android.gms:play-services-ads:15.0.1
- **GoogleAnalytics** - will use GoogleAnalytics to gather data regarding the app usage;
    - com.google.android.gms:play-services-analytics:16.0.1
- **CardView** - will display articles preview in the RecyclerView through CardViews - looks better.
    - com.android.support:cardview-v7:27.1.1

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

1. I will implement **Google Analytics** so I can have a clear picture regarding the app usage, no. of users etc.
2. **Admob** - will display test ads. In the case of such an application, some revenue might be obtained, so Admob is a good starting place.

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

Create a new project in Android Studio and GitHub;

- Set the required initial permissions (like internet access);
- Add in Gradle the initial needed libraries I will use (Room, Architecture Components, OkHttp etc.);
- Add 2 activities and one fragment (the app will open a second activity on mobile, and just inflate another Fragment in order to obtain the Master/detail UI design pattern for large screens);

Task 2: Basic navigation

- Setup a RecyclerView, an adapter for it and link all the elements - setup the navigation;
- Add menu buttons, setup classes that display the Snack etc.
- Test and adjust the app navigation;

Task 3: Fetch and parse/store the data

I will start working on the main subject of the app: retrieving, storing and displaying articles;

- 1’st thing: Setup Room Database
- Start working on the data model for an articles(POJOs) and other objects needed to be parsed from the API;
- Use Retrofit/OkHttp library to configure a class that handles the network queries, fetches data using the API (http://developer.nytimes.com/) and deserialize it using GSON into the POJO’s; I will use just a hardcoded API url. for the beginning.
- Handle errors - will also be deserialized like the articles into POJO’s
- Store the fetched articles into Room DB;

Task 4: Display the data

After fetching and storing data, Il’l setup the MVMV architecture, setup an app Repository that serves as an interface for my ViewModels and object’s DAO’s - communication with the database.

- Use Room’s LiveData/PagedList/ViewModel/PagedListAdapter classes to listen/retrieve data from the DB and display it in a paged RecyclerView
- Handle the pagination (only 10 results are served one time, so pagination logic will have to implemented: while scrolling, before the end of the list is reached a new call will be made to fetch more data, the results will be stored into the DB and added in the RecyclerView through LiveData and VM’s observers);
- Test and handle errors/errors display, observe data etc.;
- Add and use Picasso to display images, thus the entire basic flow is built;

Task 5: Search/filter/order

I will need to add a basic search dialog in the header and another fragment for setting the filtering/ordering if the user desires so.

- Create a search Dialog and link it with the class that handles the network queries;
- In case of a new successful search, all the articles from the DB(except the ones saved by the user) will be deleted, and the new fetched ones will be stored/displayed;
- Create a new custom fragment or dialog that handles the filer/order. I will create a class that provides the filter values(categories that may interest the user) that will be displayed in a spinner, another spinner for the sorting by date and 2 additional fields for the start date and the end date. Those will use the android’s default Calendar View to set the dates. When the user submits the form, another call will be made with the selected values as described in the API’s documentation. The new URL will be created and handled like before, maintaining the previously searched query(if any).
- Test, handle all cases etc.

Task 6: Details page

Setup the fragment that handles the details page.

- Retrieve the article that's already stored in the DB on fragment start, using the article's ID passed from the list of articles displayed;
- Setup the FAB, other buttons, display of the data etc.

Task 7: Save articles for offline use and display them

I want to offer the users the option to save articles for offline use.

- Link the fab with a method that marks the article in the DB as "saved" or favourite;
- Handle the "unsave/ delete from DB case", triggered by the same button;
- Handle the data replacement in the DB (when a new set of data is fetched, it will replace the entire DB, except the "saved" articles);
- Handle the access of those (saved)articles. The saved articles list can be accessed by a button in the AppBar, on the Master Screen.
- Handle the display/messages of the saved articles list;

Task 8: Add the widget

As in that point the app is basically doing what we expect(a proof of concept), to complete it we can add the last main component: the widget;

- Add the widget and make the necessary adjustments;
- Display in the widget articles from the DB, flagged as "favourites";
- Setup a mechanism that triggers the widget update when one article's flag(favourite) changes;
- Check that everything works, is connected and no special cases need handling.

Task 9: Start working on the UI

- Start working on the AppBars, handle transitions, elevations, style the app, strings etc.;
- Add the NY Time's logo on the articles, as stated in the Terms and Conditions.
- Handle all orientation changes;
- Create and implement the fragment transition logic for the master/detail flow;

- Create other App Bar that will be used for large screens;
- Refine SnackBar messages/display for giving user feedback on his actions: searches, filters, save an article, unsave an article, share, no internet connection, errors etc.

Task 10: Add the free Ads

- Add the library if not yet added;
- Create/link the test account. project and what else is needed;
- Modify the layouts to display the ads in the detail page(s);

Task 11: Add analytics

- Add the library if not yet added;
- Create/link test account, project and what else is needed;
- Add the events wherever needed;
- Test that every event is sent in the console as I set it up to;

Task 12: Final UI polishing, tests etc. + Prepare for submission

- Manually test all the elements and functionality of the app and widget;
- Check the Large Screen display UI/UX elements;
- Check orientation changes etc.
- Accessibility UI enhancements;
- Configure `installRelease task;`
- Add a signing configuration, add keystore/password;
- Push to git.

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**"