Riyaz-ali [github.com/riyaz-ali]

HN Browser

Description

Lightweight, ad-free and open source Hacker News client that is optimized for speed and network usage. Designed for simple browsing in mind, this app doesn't intend to be the all-in-one Hacker News client, and is specifically designed (and optimized) for *browsing* the posts on Hacker News

Intended User

Users who just want have a simple and lightweight experience while browsing through their daily feeds from Hacker News. This app is not intended for more *hardcore users*, and currently (in the first version) doesn't provide features such as voting and commenting.

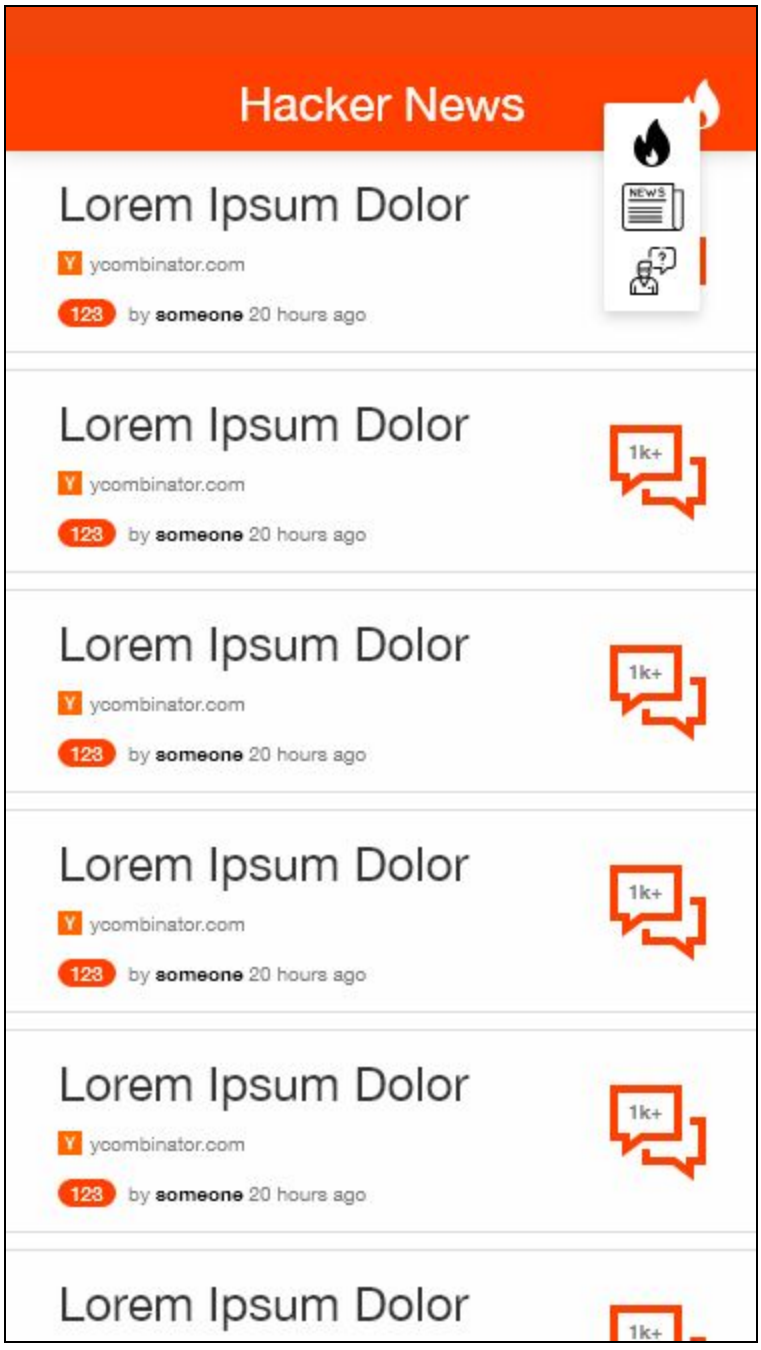
Features

- Conforming to Udacity guidelines, the app will solely be written in Java.

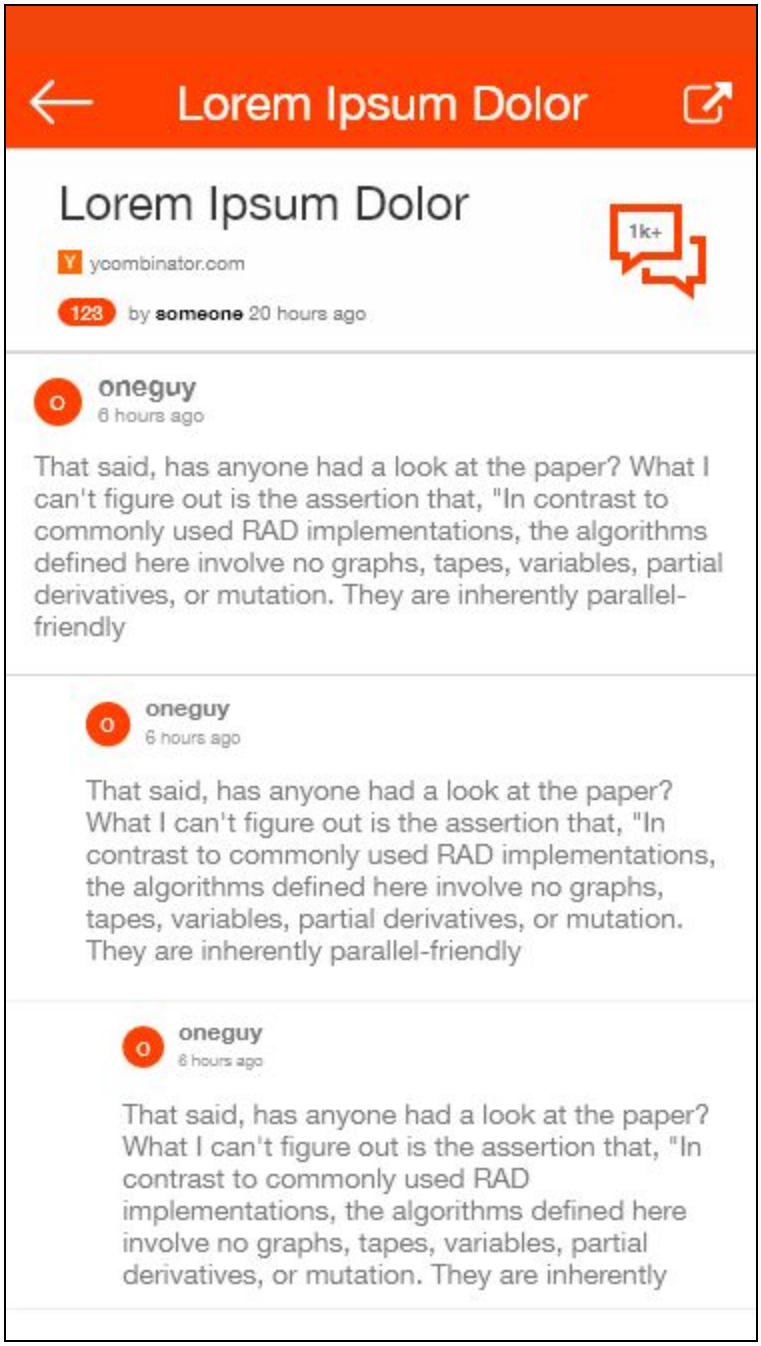
- Simple and modern looking UI providing a lightweight, clutter-free browsing experience
- Support for night mode (dark theme) for easier reading.
- Seamless (and opt in) integration for opening Hacker News posts from anywhere.
- Seamless transitioning to the main Hacker News site to support advance use cases such voting and commenting.

User Interface Mocks

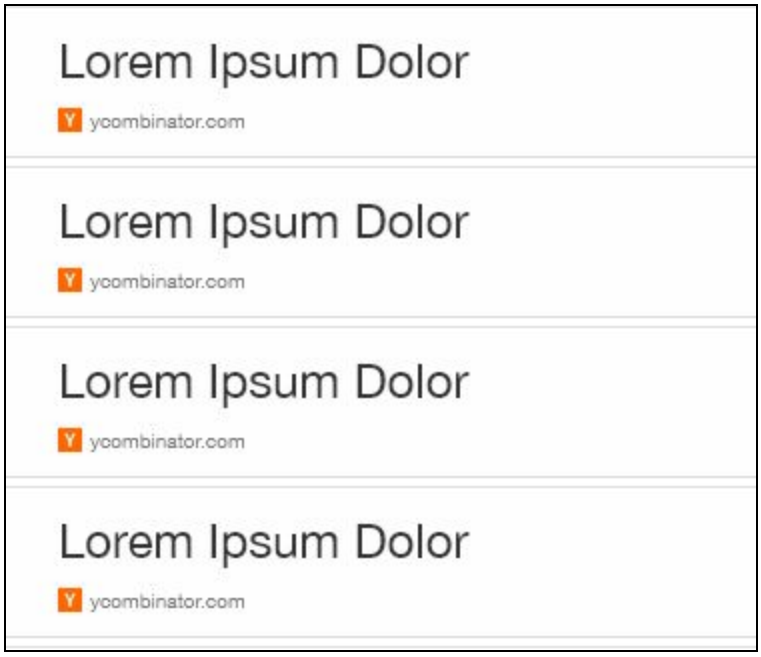
Home Screen



Details Screen



Widget UI



Key Considerations

How will your app handle data persistence?

Initially, the data will queried from Hacker News using the HNPWA API. The application will use an implementation of AsyncTask to load data on demand instead of periodically polling the API. Once the data is loaded, it will be cached locally in an SQLite DataBase using Room persistence library and further requests will return both cached and new data from API.

Describe any edge or corner cases in the UX.

As mentioned earlier, the intended use case for the app is only to browse posts on Hacker News and does not allow for voting and commenting directly. Instead, if the user clicks on a comment or vote count, the app opens a link to the post in the default browser on the device.

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

I intend to use several libraries, some of the key libraries are:

- Retrofit - For handling network and API communications
- Groupie - For displaying nested comments in a RecyclerView
- Android Architecture components - For using ViewModel and Lifecycle components, along with Room for handling persistence
- Android Design Support - For material design elements
- Android Support libraries - For compatibility and other components (like RecyclerView, CardView, etc.)

Library	**Version**
Retrofit	2.4.0
Gson	2.8.5
Groupie	2.1.0
Architecture Components - Lifecycle	`2.0.0`
Support Library - CardView	`28.0.0`
Support Library - RecyclerView	`28.0.0`
Support Library - AppCompat	`28.0.0`
Architecture Components - Room	`2.1.0`

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

I plan to add an optional flavor with Google Analytics library for gathering analytical data and Crashlytics for gathering application crash data. This flavor is intended to publish to the PlayStore.

Next Steps: Required Tasks

Task 1: Project Setup

The project setup task includes setting up an Android Studio project and a corresponding Github repository to host the project online.

1. Create a new Android Studio project using the 'New Project' wizard in the Android Studio.
2. Create new Github repository and check in the project there.
3. Also set up a README.md file, describing the project, it's LICENSE and the steps required to build it from source.

Task 2: Implement UI for Each Activity and Fragment

Next task would be to create a UI based on the above design mocks.

1. Create a layout for main (home) activity. This layout includes a RecyclerView the adapter for which is to be designed in the next step.
2. Create layout for the adapter that lists the posts.
3. Create a details activity. This activity shows the post's details alongwith the comments retrieved from the API. Comments are displayed using a RecyclerView which is managed using Groupie the layout for which is designed in the next step.
4. Create adapter layout for the Comments.
5. Move all hardcoded strings from UI to string.xml and also add support for RTL layouts.

Task 3: Integrate HNPWA API

In this step, we need to create integrations for the API. As mentioned earlier, we will be going to use the HNPWA API. Since the number of results can be very large, we will also use the Paging Support Library from the Architecture Components to add paging feature. The API will be created using Retrofit.

1. Create POJO models to store de-serialized values received from the API
2. Create API interfaces for Retrofit.
3. Implement caching layer with Room.
4. Add Paging Support Library.
5. Integrate with the UI (directly)

Task 4: Implement ViewModel

In this task, we will move the networking code out from the Activity (UI) and move it into the ViewModel class. The ViewModel also makes use of LiveData class to query cached data from Room.

1. Create ViewModel for main activity and refactor main Activity.
2. Create ViewModel for details activity and refactor details Activity.

Task 5: Add new Build Variant

In this task, we add a new build variant which enables Google Analytics.

1. Create a new variant and add an Application class that registers Analytics.