BulletPoints

GitHub Username: SBmore
Steven Blakemore
Udacity Capstone Stage 1

Product Proposal

March 18, 2016

Project Overview

Description

BulletPoints is an app that lets you keep up with the latest news and articles without having to trudge through the entire thing. BulletPoints presents you with a list of all the top stories that people are talking about and then summarises the content down to 5, easy to digest bullet points. If you feel like you never have time to read to read the full article; or you just want to get the gist, then this is the app for you.

Intended User

The intended user for BulletPoints is someone that typically scans through articles to try and get the gist but doesn't have the time, patience or interest to read the whole text.

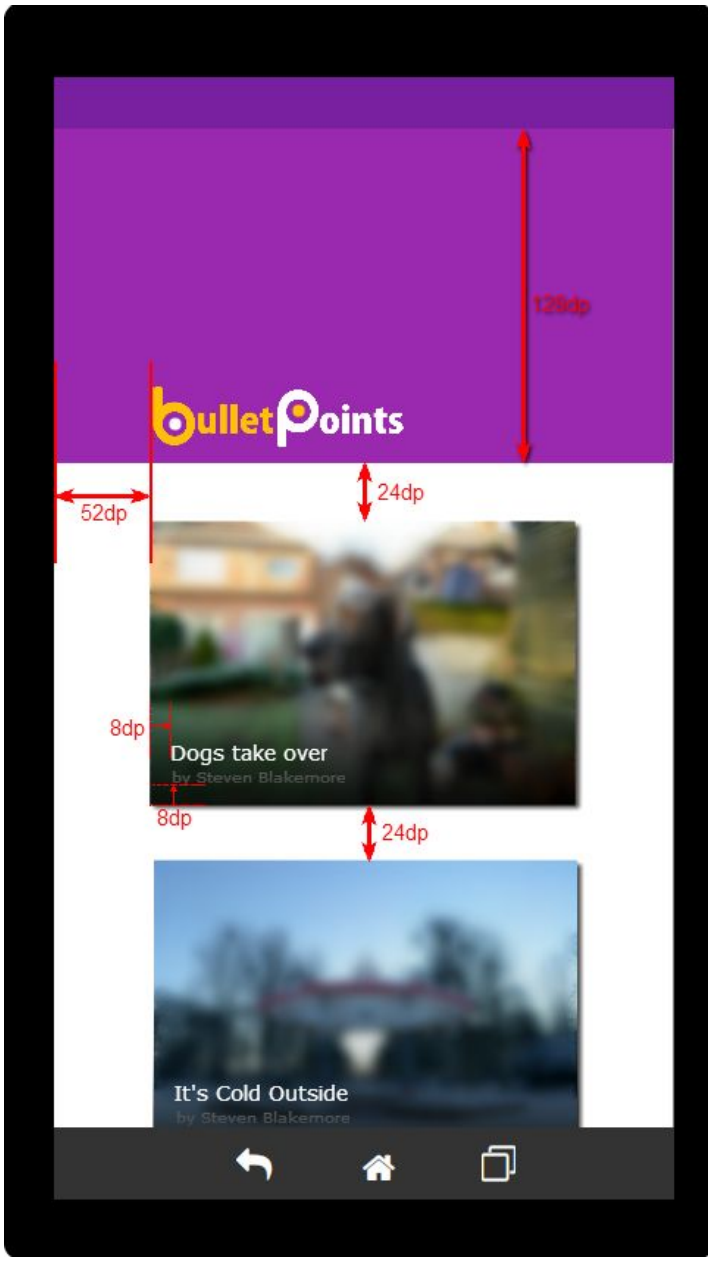
Features

- Pulls latest articles from news/RSS feeds and displays them in a list
- Summarises selected article into five bullet points
- Click on bullet point to get more information (full paragraph)
- Display whole article if interested enough to keep reading
- Articles can be shared to social media
- Latest article's bullet points are displayed in a widget

User Interface Mocks

These interface mockups were created using the Android Kitkat Template from Balsamiq and the Online Pixlr Editor. Images used are my own but have been blurred out to take attention away from the content so the focus can be on the layout and dimensions.

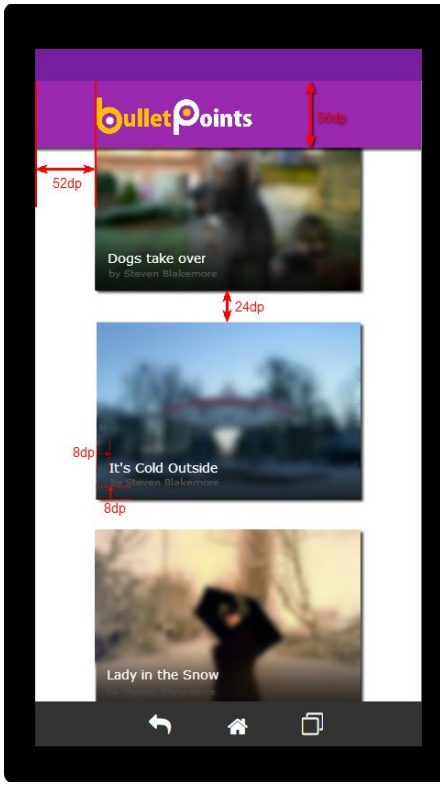
Screen 1 - Phone Portrait List



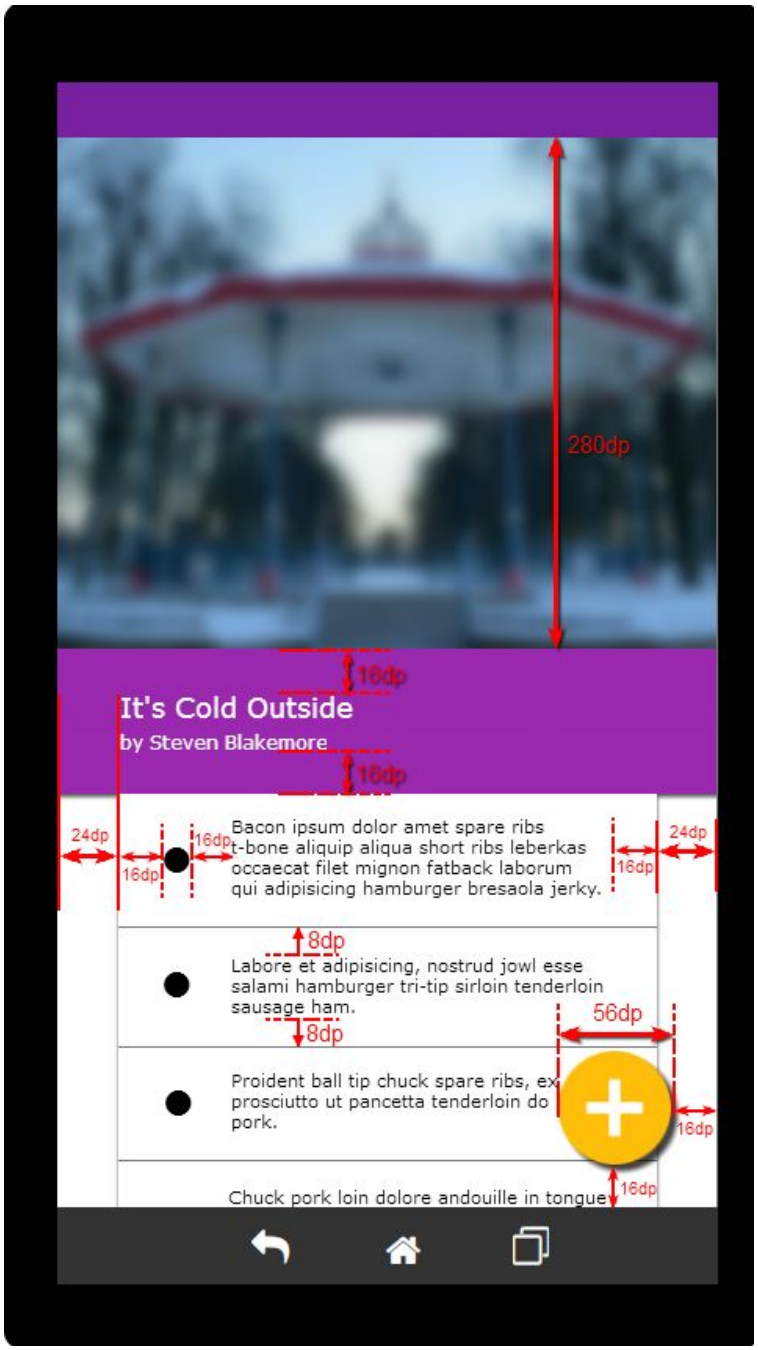
The app will load into this screen and it will show the different stories that can be selected.

The cards that can be clicked will be the article's image with a scrim to make the text visible, and the title text on top.

The toolbar will be extended and will pin and increase elevation when it has reached the top.



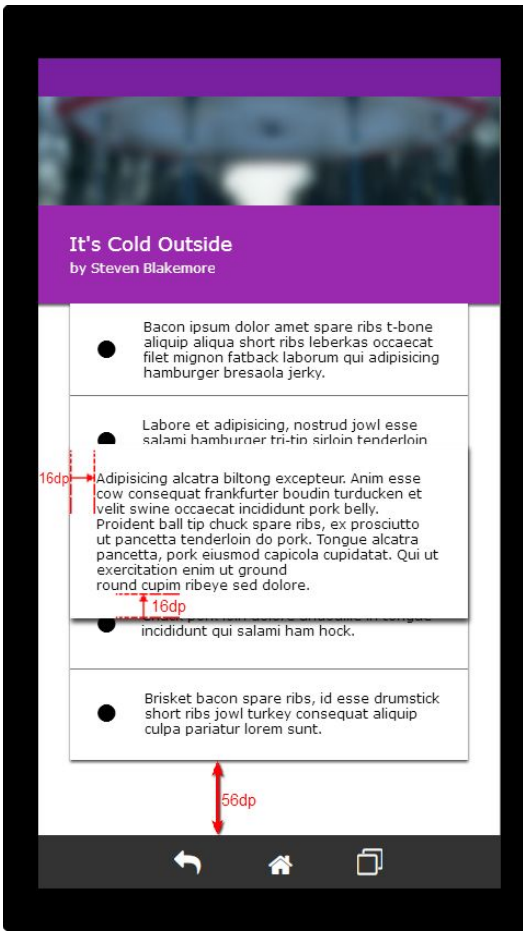
Screen 2 - Phone Portrait Detail



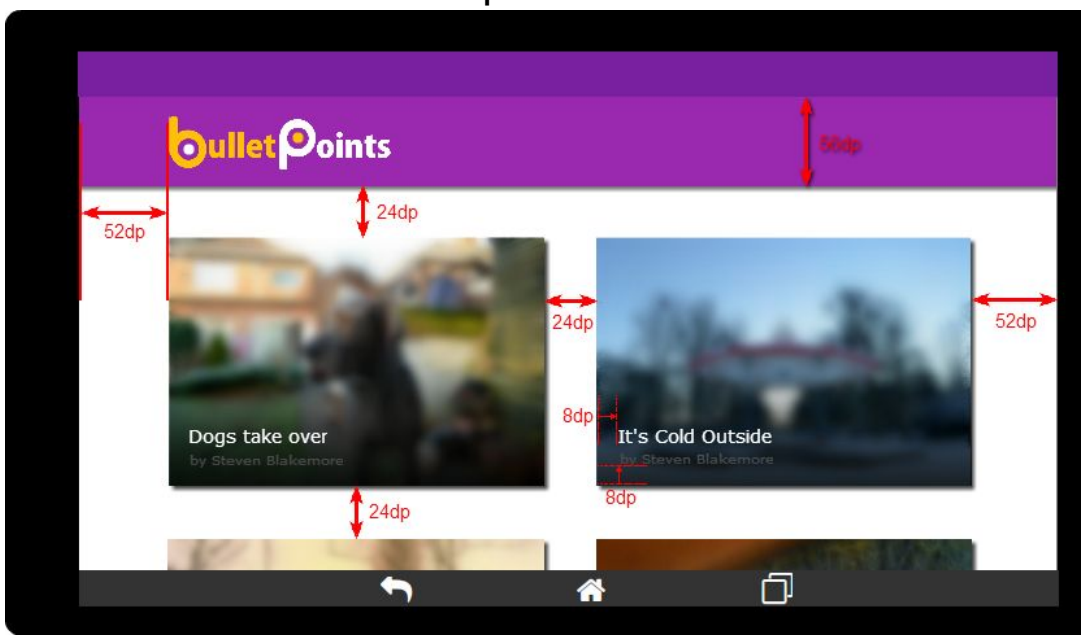
When a card is pressed on the list screen the detail page will show. This page will use the same article image as the card and will parallax scroll when the user scrolls the screen.

When the user presses a bullet point section a window will pop up with the whole paragraph.

Pressing the FAB will give the user the option of sharing content or expanding the article to view the whole text.

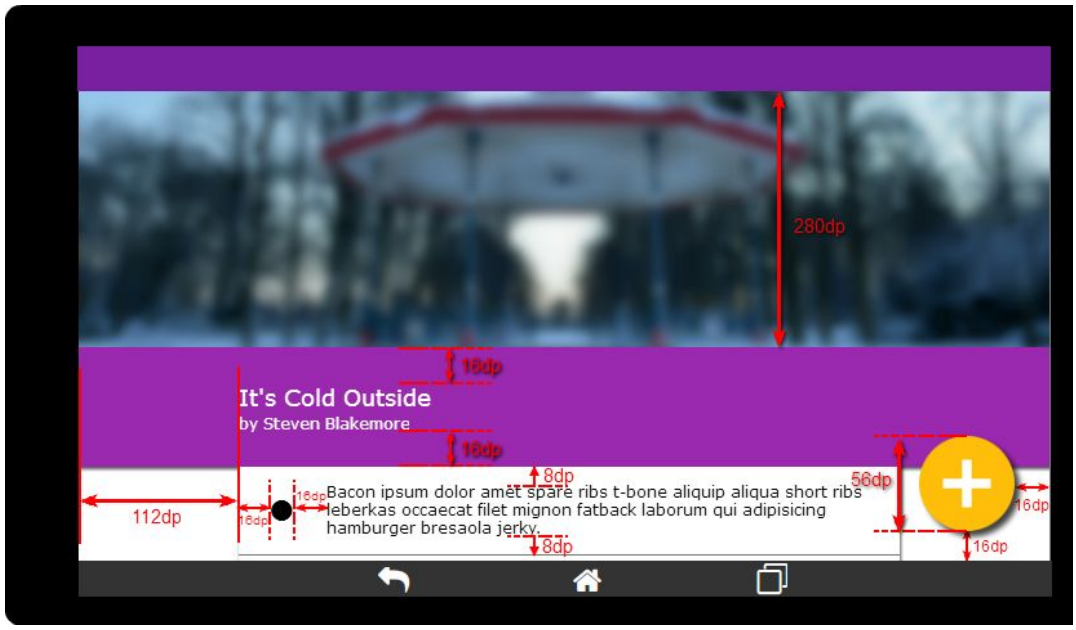


Screen 3 - Phone Landscape List



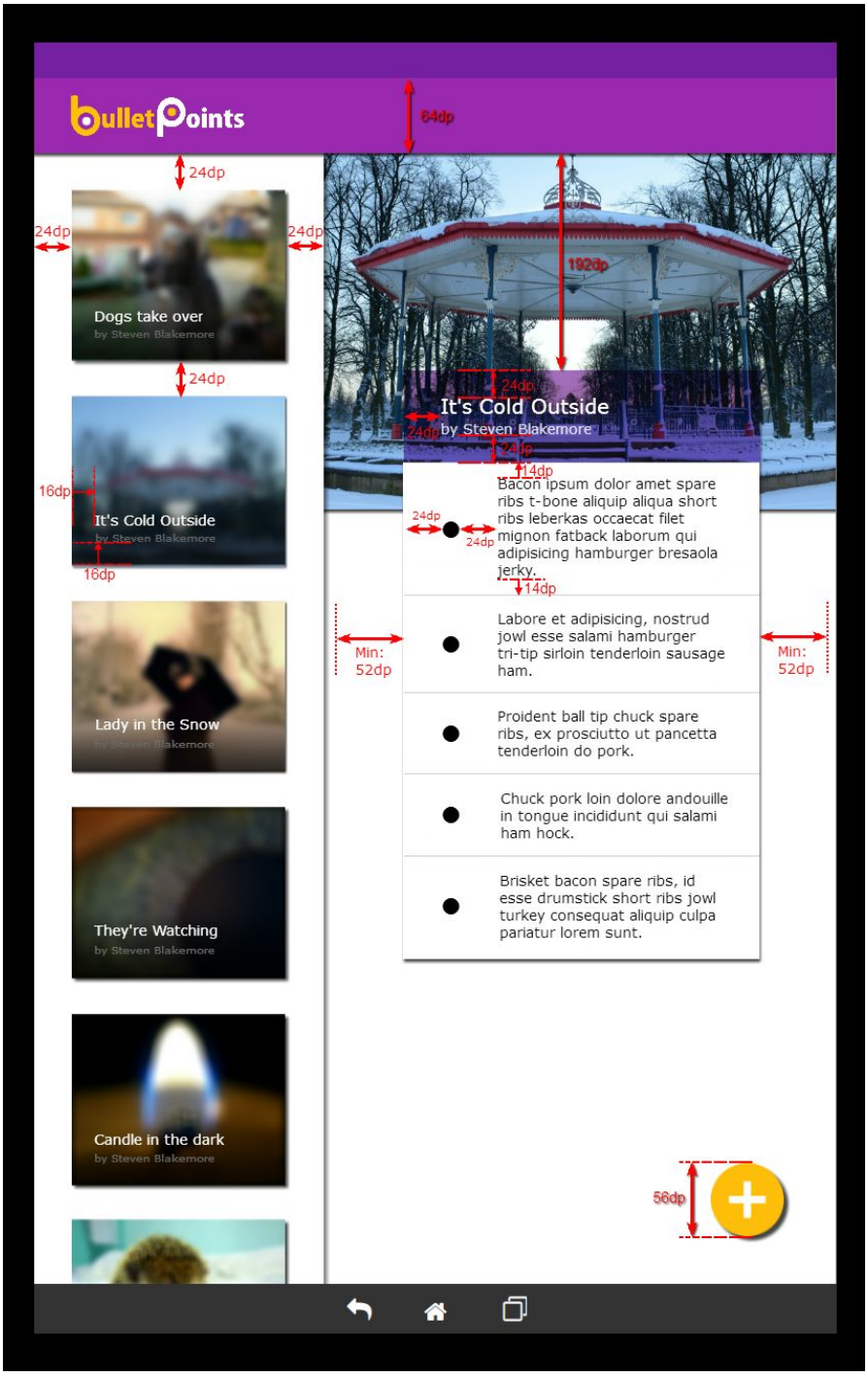
When a phone is rotated to landscape on the list page the number of columns will increase to two; to better use the screen space. This view will also not use the extended toolbar as it will use up too much of the screen in landscape mode.

Screen 4 - Phone Landscape Detail



The detail landscape should only affect dimensions and spacing.

Screen 5 - Tablet Portrait Master / Detail

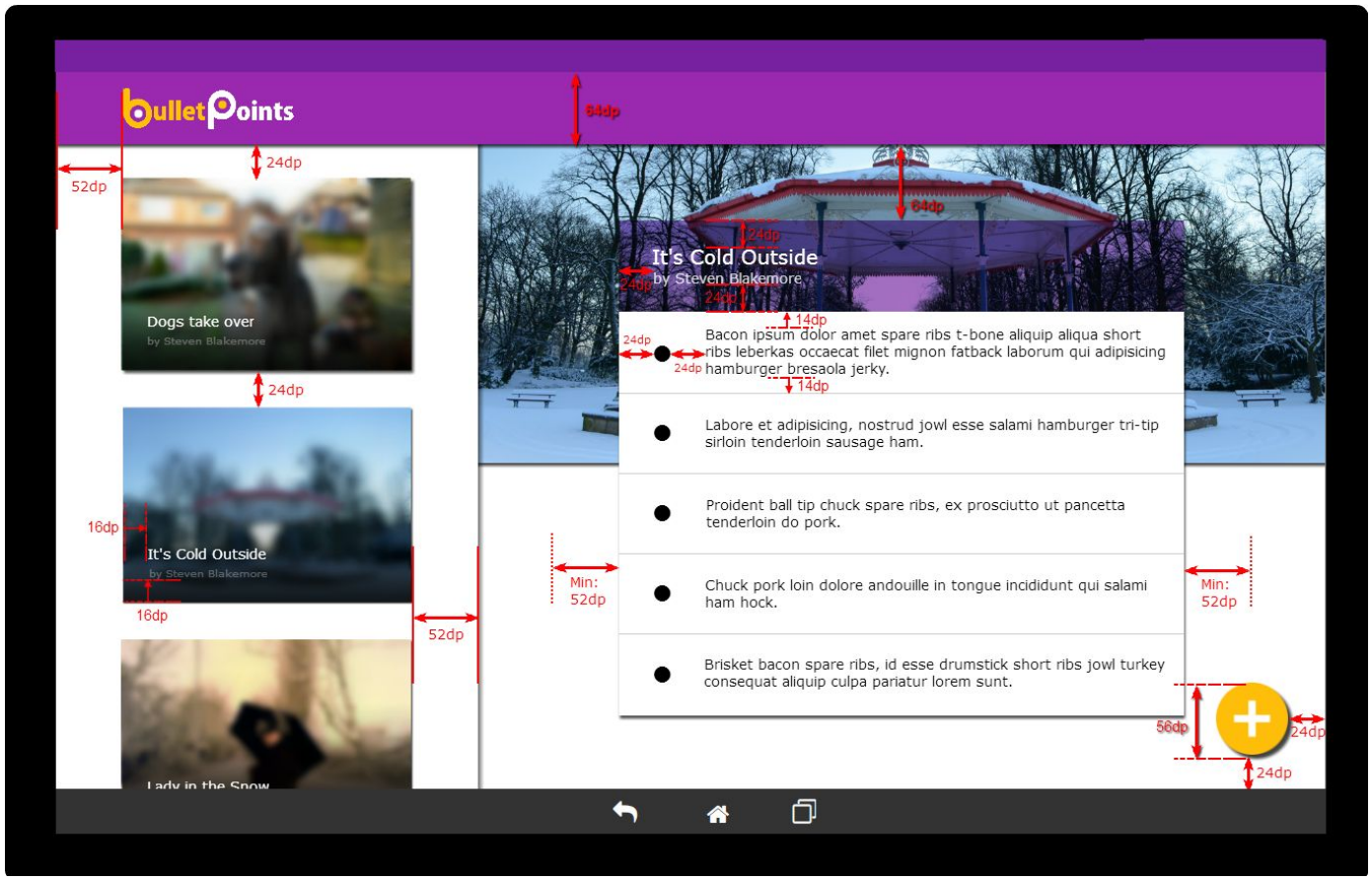


The tablet version of the app will utilise a Master/Detail layout, utilising the phone layout's list view on the left and a new detail layout on the right.

The article image will show as an elevated layer at the top of the detail page with the bullet points layer overlapping.

The FAB remains in the bottom right corner, and clicking on a bullet point will still open a floating layer with the full paragraph on it.

Screen 6 - Tablet Landscape Master / Detail



The landscape tablet view will follow the same layout as the portrait version but with altered sizes and margins to make it fit better.

Key Considerations

How will your app handle data persistence?

A Content Provider backed by an SQLite database, will be generated using the third party library Schematic and used to store the article data and the generated bulletpoints.

Describe any corner cases in the UX.

When the user clicks a bullet point and it opens a floating text view that shows the relevant paragraph, they can press anywhere on the screen to close it, or press the back button on the device.

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

android-rss

This library will be used to connect to the RSS feed and convert the returned data into a java object

okhttp

The okhttp library will be used to get the html source of each article that is returned from the RSS feed.

jsoup

Jsoup helps parse the HTML that is returned to extract the main body of the article so it can be parsed.

gson

Used to convert JSON strings, returned from the HTML, into a Java object

Schematic

Schematic will be used to automatically generate a Content Provider backed by an SQLite database.

Glide

Glide will be used to handle the loading and caching of images.

Next Steps: Required Tasks

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

Task 1: Project Setup

Android Studio:

- Update Android Studio to the latest version
- Create a new project called BulletPoints with a minimum SDK of API 16
- Create a new Android Library called TextBulletPointer with a minimum SDK of API 16

Libraries:

Import latest version of AppCompat and the support libraries

- add to the app-level build.gradle file in the dependencies block:
  - compile 'com.android.support:support-v4:19.1.0'

Import android-rss (https://github.com/ahorn/android-rss):

- add to the app-level build.gradle file in the dependencies block:
  - compile 'com.github.ahorn:android-rss:v1.0-rc1'

Import okhttp (https://github.com/square/okhttp)

- add to the app-level build.gradle file in the dependencies block:
  - compile 'com.squareup.okhttp3:okhttp:3.2.0'

Import jsoup (http://jsoup.org/)

- add to the app-level build.gradle file in the dependencies block:
  - compile 'org.jsoup:jsoup:1.8.3'

Import gson (https://github.com/google/gson)

- add to the app-level build.gradle file in the dependencies block:
  - compile 'com.google.code.gson:gson:2.6.2'

Import Schematic (https://github.com/SimonVT/schematic):

- Add the following to the app-level build.gradle file in the dependencies block:
  - apply plugin: 'com.android.application'
  - apply plugin: 'com.neenbedankt.android-apt'
- Add the following to the app-level build.gradle file in the buildscript's dependencies block:
  - classpath 'com.android.tools.build:gradle:{latest-version}'
  - classpath 'com.neenbedankt.gradle.plugins:android-apt:{latest-version}'
- Add the following to the app-level build.gradle file in the dependencies block:
  - apt 'net.simonvt.schematic:schematic-compiler:{latest-version}'
  - compile 'net.simonvt.schematic:schematic:{latest-version}'

Import Glide

- Add the following to the app-level build.gradle file in the dependencies block:
  - compile 'com.github.bumptech.glide:glide:3.7.0'

Google Services

Import Google Analytics (https://developers.google.com/analytics/devguides/collection/android/v4/):

- add to the project-level build.gradle file in the dependencies block:
  - classpath 'com.google.gms:google-services:2.0.0-alpha6'
- add to the app-level build.gradle file the plugin;
  - apply plugin: 'com.google.gms.google-services'
- add to the app-level build.gradle file in the dependencies block:
  - compile 'com.google.android.gms:play-services-analytics:8.4.0'
- make a Google Analytics account and register the app to get the configuration file
- copy the google-services.json file to the app/ directory of the BulletPoints project

Import Google AdMobs (https://developers.google.com/admob/android/start):

- add to the app-level build.gradle file in the dependencies block:
  - compile 'com.google.android.gms:play-services-ads:8.4.0'
- create an ad unit
  - sign in to AdMob account at https://apps.admob.com.
  - click the Monetise tab.

- click + Monetise new app.
- enter the app name and the platform, then click + Monetise.
- create an ad unit.

Create the Article Summarising library that I'll be writing and add the dependency

Task 2: Implement UI for Each Activity and Fragment

Creating the app will automatically create the activity and fragment for the list page.

Build the UI for the List page:

- Use a CoordinatorLayout with a CollapsingToolbarLayout to allow the App bar to get smaller when scrolled
- Use a RecyclerView to contain the list

Build the CardView that the RecyclerView will use to display the articles.

Build the landscape view of the list page, changing the column count to 2 and removing the collapsing toolbar.

Build the phone portrait details page:

- Use a CoordinatorLayout with a CollapsingToolbarLayout to allow the image to have parallax scrolling
- Use an ImageView for the article picture
- Use a ListView to display the bullet points
- Create a Floating Action Button in the bottom right

Create the view that will be used by the ListView on the details page.

Create the paragraph view that will appear when a bullet point is pressed:

- Use a TextView inside of a FrameLayout

Build the landscape view of the details page, adjusting margins and sizes as per the redlines.

Create the list (master) section of the tablet's portrait Master/Detail layout:

- Use the phone's list page as a base

Create the detail section of the tablet's portrait Master/Detail layout:

- Use an ImageView for the article image
- Use an overlapping frame layout that contains a ListView to display the bulletpoints
- Add a title bar above the ListView with reduced opacity for the effect seen in the mocks
- Create a Floating Action Button in the bottom right

Build the landscape view view of the tablet's Master/Detail page by adjusting the margins and sizes as per the redlines.

Task 3: Implement Google Ad Mobs

The Ad Unit that was created during setup will provide an Ad Unit ID which will be stored in strings.xml. To implement Ads:

- Create a class called App that extends Application and loads up an Ad instance when it is created
- Create a loadAd method that will load the Ad into the supplied view
- Place an Ad View in the layout files for the list page and the detail page fragments
- Use the App class to load the Ad in each of the fragments

Task 4: Implement the RSS feed

To connect to the RSS feeds the android-rss reader will be used. A class should be created that extends asyncTask that will fetch the RSS data from the feed's URL.

- An RSSReader object will load the feed's URL, which will return a RSSFeed object
- Pull all the RSS Items from the RSSFeed object
- Loop through each one and get the data to be stored in the database

Task 5: Implement the Content Provider

The content provider and SQLite database will be created using the Schematic library using the following steps:

- Create an interface to store the columns that will be held in the database table
- Create a class that will utilise the interface to create a database with those columns
- Create a class for the content provider that will connect to the database

Task 6: Build the TextBulletPointer library

The article's body will be passed into this library that will generate the bullet points that will be displayed when the user clicks on an article in a different thread.

- Create a method that takes the body of the email and splits it into a unique list of every word in the article
- Loop through each word and count how many times each word appears in the full article text. The count will become the "score" for that word
- Split the article into paragraphs and count how many times each of the words from the unique list appear in the paragraph
- Multiply the count by the word's "score" to give the paragraph a "score"
- Filter out any paragraphs with a score so low that it indicates the paragraph is too short to be useful (experiment with boundaries)
- Split the article into beginning, middle and end by taking the paragraph count and dividing it by 3
- Select the most useful paragraphs by selecting the lowest scoring paragraphs (as they will have the fewest throwaway words such as 'the', 'and' etc). The paragraphs picked will be:
    - The first paragraph (contains the introduction so pick regardless of score)
    - The lowest scoring paragraph from the beginning, excluding the first paragraph
    - The lowest scoring paragraph from the middle
    - The lowest scoring paragraph from the end, excluding the final paragraph
    - The final paragraph (contains the conclusion so pick regardless of score)
- Split each paragraph into sentences and then count how many times each of the words from the unique list appear in each sentence.
- Multiply the count by the word's "score" to give the sentence a "score"
- Return the following (this time picking the highest score as it indicates that the sentence contains the most content):
    - The first sentence from the first paragraph (introduction)
    - The highest scoring sentence from the second paragraph
    - The highest scoring sentence from the third paragraph
    - The highest scoring sentence from the fourth paragraph
    - The highest scoring sentence from the final paragraph

- Return these sentences, which will be the 5 bullet points, so they can be stored in the database and displayed when the user selects an article

Task 7: Implement a Cursor Loader

A loader will be used to asynchronously load data from the database. A new class called BulletLoader will be create that extends CursorLoader. The loadInBackground method will be overridden to return a Cursor containing rows of data from the database.

Loader callbacks will then be created to listen for events such as onCreateLoader, onLoadFinished and onLoaderReset and a custom adaptor will link the returned data to the user interface.

Task 8: Add the FAB functionality

Add a floating action button that expands to show two options when clicked:

1. Open the full article in the browser
2. Share the article link and some app specific information

When the user chooses to view the whole article an implicit intent will be created that will send the url as an extra to a browser application of the user's choosing.

When the user chooses to share an article a short message with the article name, link and app name will be sent as an extra via an intent with the ACTION_SEND action.

Task 9: Implement Google Analytics

Add the instance creation for the Google Analytics into the App class created earlier so that only one instance is created.

Create trackers that track:

- How many articles a user has clicked on
- How many times a user clicks a bullet point to get more information
- How many times a user clicks through to the main article
- How many times a user uses the share functionality

Task 10: Create a widget that shows the latest bullet points

- Create a Widget Provider that extends AppWidgetProvider
- Create an xml file for the widget provider and a layout xml file for the widget provider
- Create a layout file for the bullet point elements that will be displayed in the widget
- Register provider in the Manifest
- Create a data provider for the widget that implements RemoteViewsFactory
- Create a service for the widget that extends RemoteViewService
- Register the service in the Manifest
- Update the widget's provider to initialise the view
- Add click events to each bullet point in the widget's provider and data provider. These will open the app up to that article's detail page

Task 11: Develop Material Design Elements

- Add parallax scrolling to the article image on detail page
- Create standard and simple animations to ensure a smooth transition between the list page and the details page
- Ensure that the colour theme is defined in styles.xml.
- Work through mocks to ensure the look and feel of the app matches
- Create / Develop the extra layouts for different screen sizes and orientations

Submission Instructions

1. After you've completed all the sections, download this document as a PDF [File → Download as PDF]
2. Create a new GitHub repo for the capstone. Name it "**Capstone Project**"
3. Add this document to your repo. Make sure it's named "**Capstone_Stage1.pdf**"