MarvelNow

- Description
- Intended User
- Features
- User Interface Mocks
- Onboarding Screen
- Authentication Screen
- Man Activity Screen (is searchFragment)
- Main Activity Navigation Drawer
- ItemDescriptionFragment, CharacterFragment
- CharacterDetailActivity (Contains pager with CharacterFragment , ComicsFragment, SeriesFragment, StoriesFragment)
- ItemDetailActivity (Contains pager with ItemDescriptionFragment, CharactersFragment)
- SettingsActivity
- Widget
- Key Considerations
- Next Steps: Required Tasks
- Task 1: Project Setup
- Task 2: Data Model
- Task 3: Implement UI for Each Activity and Fragment
- Task 4: Make API Calls and Wire to UI
- Task 5: Firebase Real-time Database
- Task 6: Crashlytics
- Task 7: UI Optimisation - Transitions and Animations
- Task 8: Final Steps

GitHub Username: cdemetriou

MarvelNow

Description

The app will be a gateway to enjoy extensive content for Marvels superheroes. Every action will be based around your favourite heroes. Content provided will include a browsable directory of characters, comics, series and storylines and their respective detailed information, keeping track of new material for each of these categories. A user will be able to build a team of favourite heroes making the app more personalised. Content provided in the app will be fetched from Marvel Comics API.

Intended User

The intended user is any fan of Marvel content, that may be the comics, series or even just a specific character. Marvel fans will be able to keep track of their favourite characters and content relevant to them.

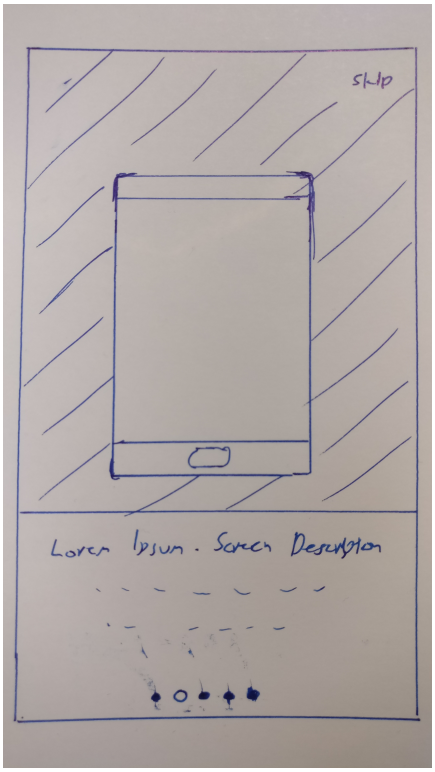
Features

Use Firebase Authorisation for user log in which will provide the app with a user to store persona info under.
Use Firebase Real-Time Database to store user data both online and offline
Use Firebase Crashlytics to track crashes
Build a team of superheroes (favourites)
Personalise app with most favourite character as user avatar
Search for a character
Search for a series
Search for a comic
Search for a storyline
Widget with your team
Share your team or a specific content such as a series with a friend

User Interface Mocks

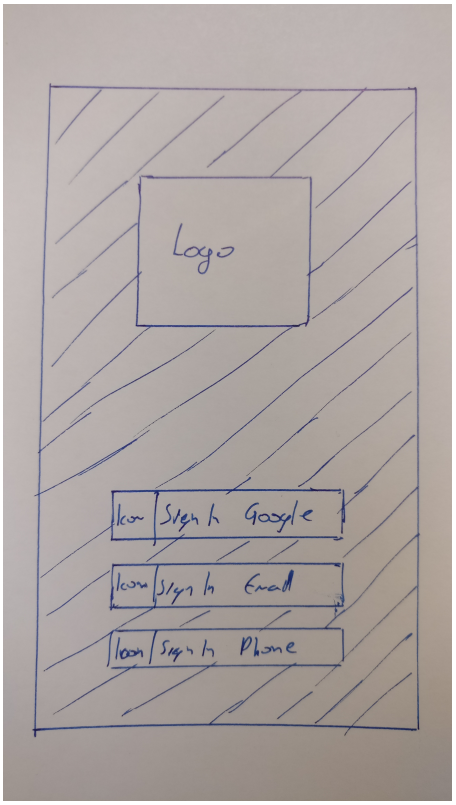
These can be created by hand (take a photo of your drawings and insert them in this flow), or using a program like Google Drawings, www.ninjamock.com, Paper by 53, Photoshop or Balsamiq.

Onboarding Screen



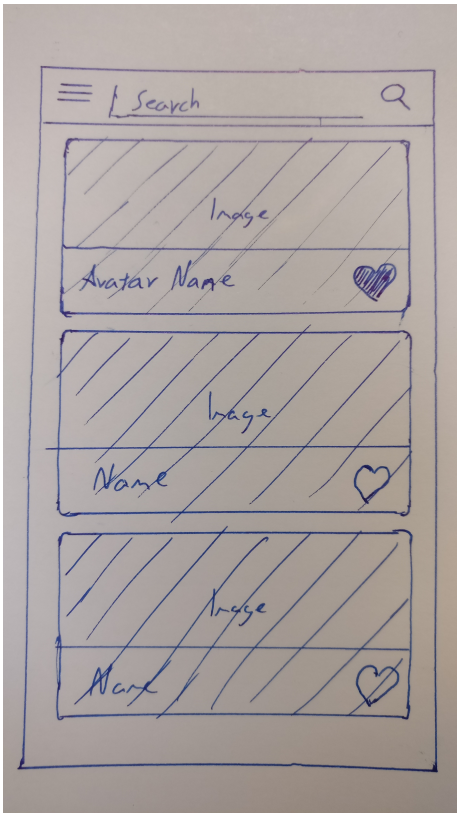
Pager with a number of screen shots of the apps main functionality with description underneath. Skip button to the top right.

Authentication Screen



Firebase authentication for register and log in.

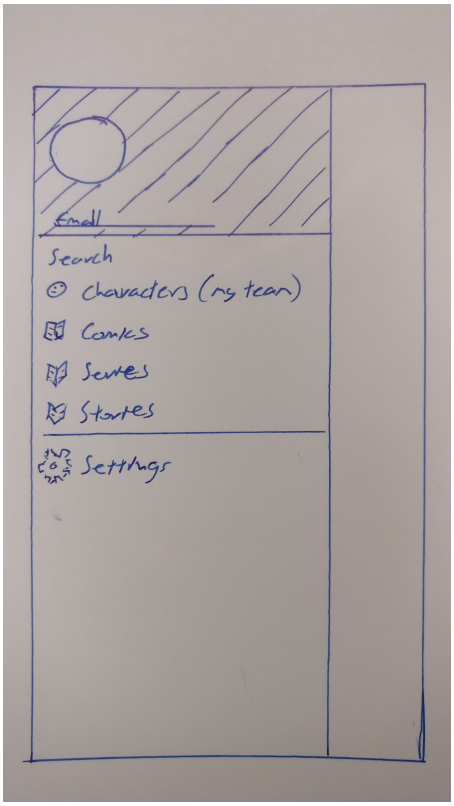
Man Activity Screen (is searchFragment)



SearchFragment, contains the search bar and navigation drawer and another fragment inside. In the main activity case its the CharactersFragment showing a list of characters initially and when a team is selected, shows the team characters.

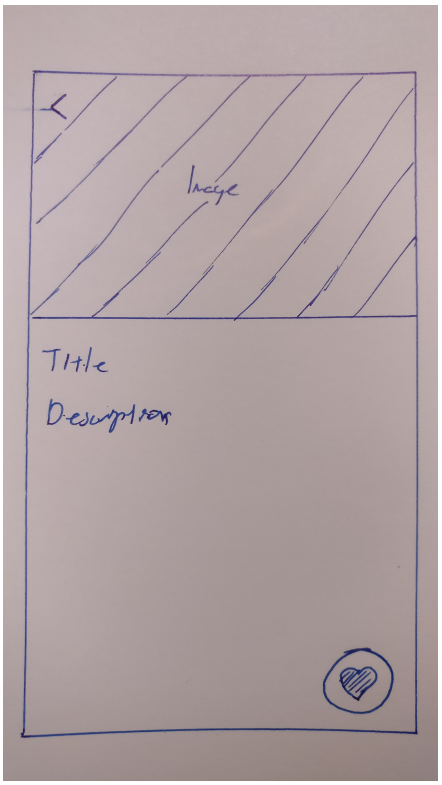
The CharactersFragment will be the same one used for ComicsFragment, SeriesFragment, StoriesFragment with different viewHolders for the recycler view.

Main Activity Navigation Drawer



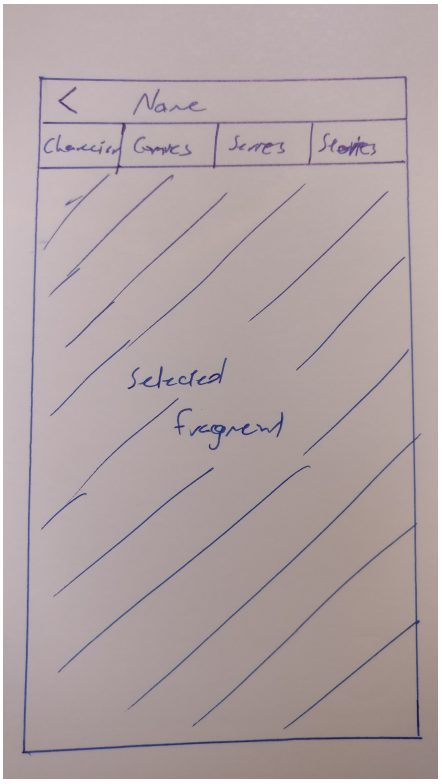
Navigation drawer showing the available actions.

ItemDescriptionFragment, CharacterFragment



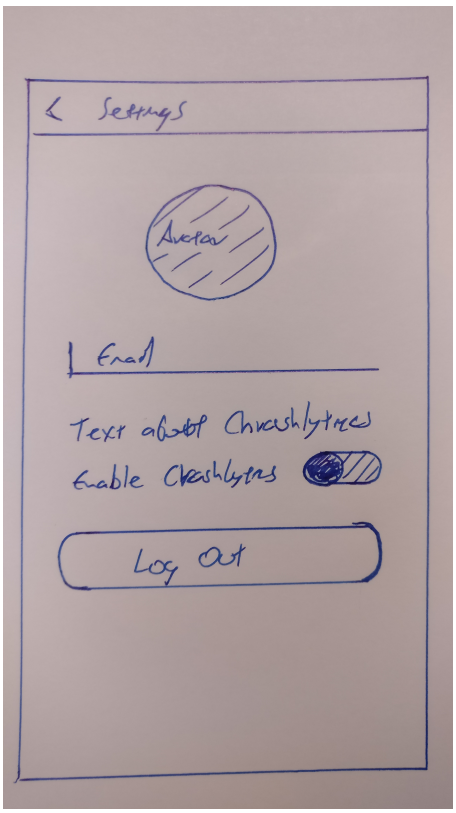
Fragment showing details for the item selected. The character one will have a floating action button for adding the character to the team.

CharacterDetailActivity (Contains pager with CharacterFragment , ComicsFragment, SeriesFragment, StoriesFragment)

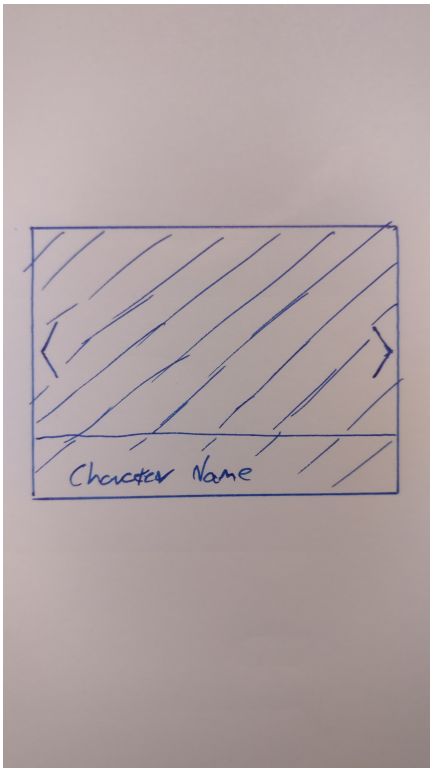


ItemDetailActivity (Contains pager with ItemDescriptionFragment, CharactersFragment)

SettingsActivity



Widget



The widget will show the current characters in the users team.

Key Considerations

How will your app handle data persistence?

Data will be stored in Firebase Realtime Database with local persistence enable for offline availability.

Describe any edge or corner cases in the UX.

- Keep instance state to correctly handle orientation changes.
- Use asynchronous operations for fetching and storing data in order to keep the UI responsive, and provide indications of background operations underway.
- Track network connection availability to prevent data changes from queueing up, as firebase realtime database transactions are not guaranteed to be present after an app restart which might introduce data corruption.

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

- Firebase - For database persistence, user authentication and analytics
- Retrofit2 - for asynchronous API calls.
- Retrofit2 Gson Converter - For converting data received to Json
- Logging Interceptor - To monitor API requests
- Dagger2 - For dependancy injection of modules such as the Firebase database and the retrofit instances
- Glide - For efficient loading and possible processing on images
- Data binding - For simplified communication between activities and xml
- Parceler - Annotation based library for simplified conversion of POJO class to Parcelable
- IcePick - For instance state restoration to handle orientation changes
- Timber - For cleaner debugging code
- MaterialDrawer - For extensive customisability of navigation drawer
- MaterialSearchView - Simplified search view with optional voice recognition

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

- Firebase authentication will be used to register and sign in users.
  - User credentials will be used as the identifier for the database user info.
- Data will be stored in Firebase Realtime Database with local persistence enabled for offline availability.
- Firebase analytics will be used to track crashes.

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 new Project
  - Configure Gradle for data banding and add dependancies for the libraries discussed
  - Create BaseActivity with IcePick - For instance state restoration
  - Initialise Dagger
    - Add Dagger dependancies for Retrofit with Interceptor and Gson converter and Firebase realtime database

Task 2: Data Model

- Build data model
  - Build data model for characters, comics, series, stories.
  - Use Parceler library.

Task 3: Implement UI for Each Activity and Fragment

- Build UI for SplashActivity showing a logo for the app
- Build UI for OnboardingActivity introducing main app functionality in 3 to 5 steps
- Build UI for AuthenticationActivity which will use Firebase Authentication
- Build UI for MainActivity containing a navigation drawer.
  - MainActivity will contain your team, initially contain a list of characters and when not empty, your team of characters.
  - Navigation drawer will contain options for searching characters, comics, series, storylines
- Build UI for SearchFragment.
  - Default state has a number of random items of the specific type (eg. Comics).
  - Search state shows search results in CharactersFragment.
- Build UI for CharacterDetailActivity for when you click on a character.
  - Contains pager with CharacterFragment, ComicsFragment, SeriesFragment, StoriesFragment.
- Build UI for ItemDetailActivity for when you click on a item (eg. Comics).
  - Contains pager with ItemDescriptionFragment, CharactersFragment.
- Build UI for SettingsActivity
  - Option to change avatar
  - Option to enable/disable analytics
  - Option to log out
- Build UI for AboutActivity giving credit to the API provider
- Build UI for Widget.

Task 4: Make API Calls and Wire to UI

- Add API endpoints to be used in the retrofit interface.
- Add async calls for the appropriate endpoints based on UI interaction.
- Present data in recycler views using a custom adapter with a custom view.

Task 5: Firebase Real-time Database

- Make the calls for storing and retrieving user favourite characters.
- Handle connection availability and prevent updates when offline or store transactions to keep them in order when app restarts.

Task 6: Crashlytics

- Add Firebase Crashlytics to the app

Task 7: UI Optimisation - Transitions and Animations

- Add transitions and animations where appropriate
- Add RTL support
- Add all strings int strings.xml
- Make any other potential UI optimisation

Task 8: Final Steps

Create release key
Add signing configuration inside Gradle
Refer to keystone by relative path

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**"