Description

Intended User

Features

User Interface Mocks

- News List Screen
- News Details Screen
- Fixtures List Screen
- Results List Screen
- Log Table Screen
- App Widget UI

Key Considerations

- Tools Requirements
- 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.

Required Tasks

- Task 1: Project Setup
- Task 2: Implement UI for Each Activity and Fragment
- Task 3: Firebase Setup
- Task 3: Implement the Business Logic
- Task 4: Implement the App Widget Service
- Task 5: Test the App

GitHub Username: NalediMadlopha

Diski

Description

Diski is a mobile app that provides users with real time updates about anything related to soccer in South Africa. It mainly focuses on the Premier Soccer League (PSL), providing real time news feeds, live score updates, fixtures and results as well as the latest league table standings.

We live in an information age, where people have access to almost any kind of information that is of interest to them. However most of the information is offered with a certain level of delay, e.g. Back in the days people who would have missed a soccer match would have to wait till Monday morning to read about all the weekend soccer action in a newspaper. Now are days, people have televisions (TV), radios and the Internet to keep them posted about the current happenings of the matches taking place. However, they would have to wait for a specific time for a TV or radio programme to find out what transpired in a specific match of their interest. Some may even go online to read about it which is arguably the best approach to find the latest news of one's interest. The Diski app provides real time information so the user does not have to wait for a specific time to get information or even go online to search for soccer related information. The app provides the user with information to the user as it happens.

Intended User

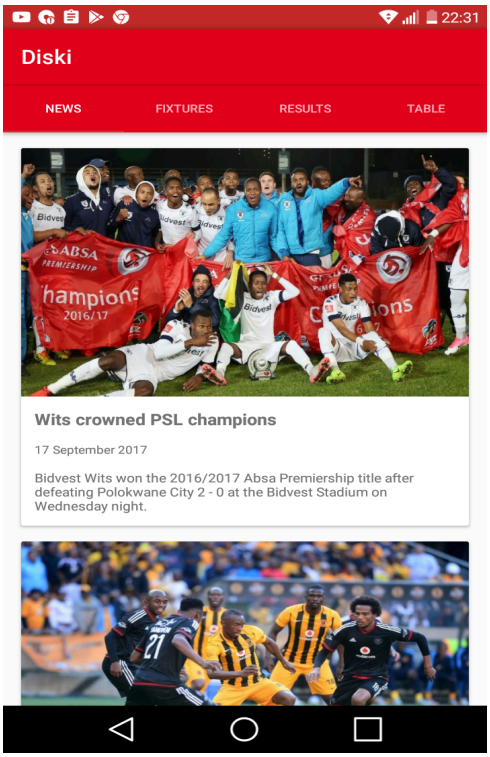
This app is intended for anyone who has interest in South African football and wants to keep themselves up to date with what's happening in the world of football in this country.

Features

- **News Feeds –** the news feeds feature provides soccer news feeds to the user. The user will be able to select an article from a list of articles to display its full content allowing the user to read the story.
- **Match Fixtures –** this feature provides the user will a list of fixtures for different competitions.
- **Match Results –** the match results feature provides the user with a list of match results, display the scores between two teams that we playing against each other.
- **League Table Standing –** displays the log table detailing the games played, wins, loses, draws as well as the total points each team in the league has.

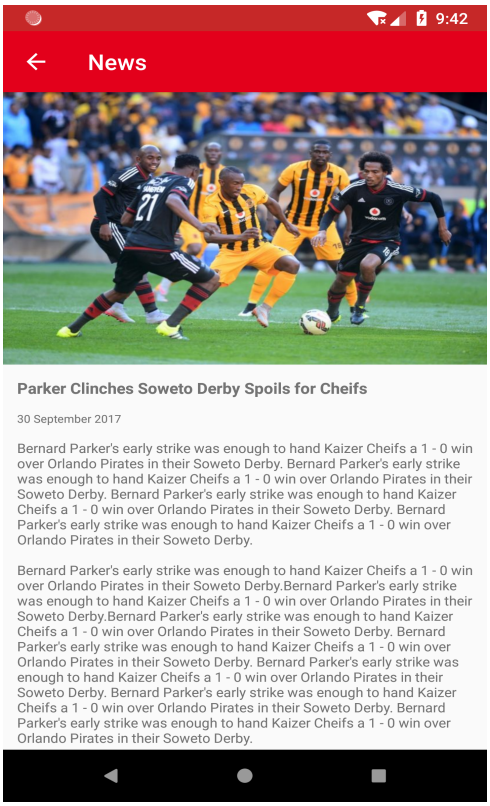
User Interface Mocks

News List Screen



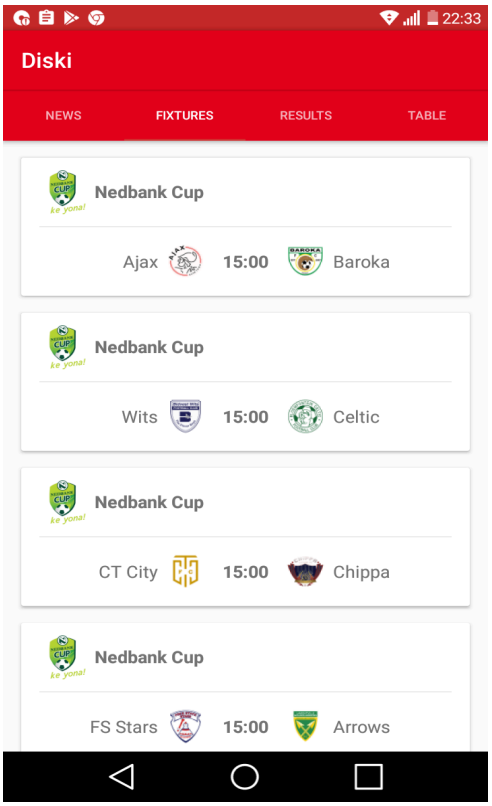
The news list screen is the land screen. When the user launches the app, they will be presented with the screen. The user can then click on the news cards to read the entire story.

News Details Screen



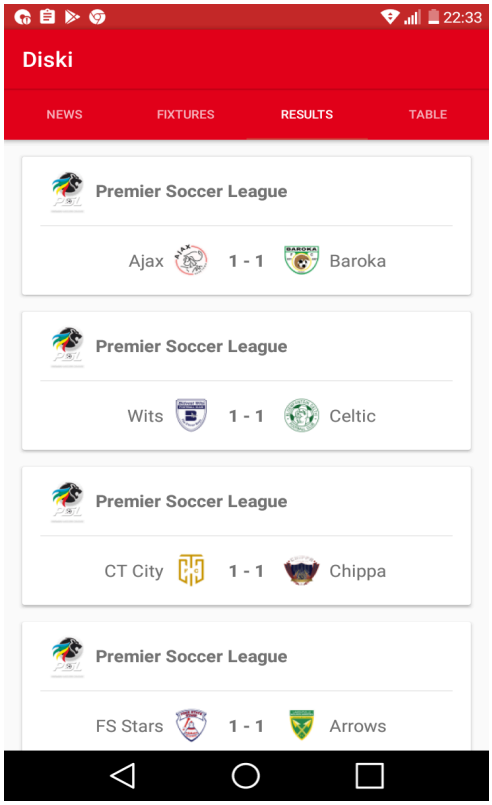
The news details screen displays the entire story of the article that was selected from the news list screen.

Fixtures List Screen



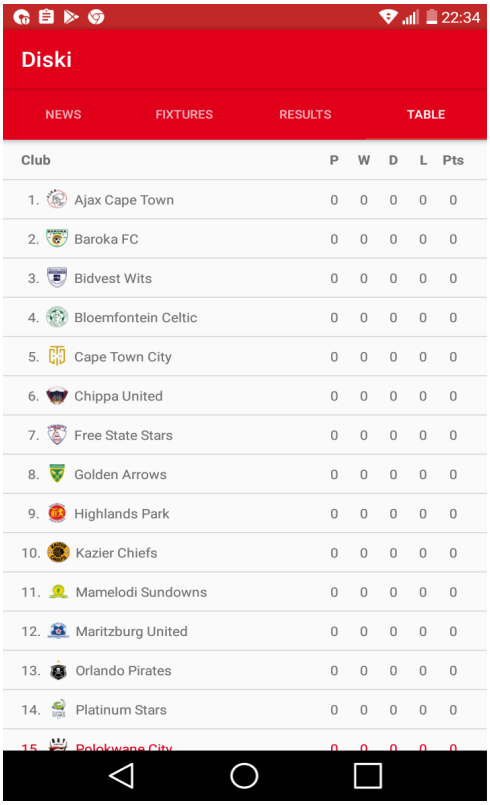
The fixture screen displays a list of upcoming fixtures for different tournament. This screen provides the user with the name of the teams playing, the time as well as the competition of fixture.

Results List Screen



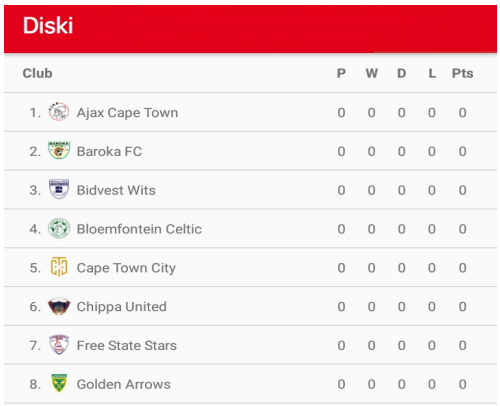
The results list screen displays a result list of the most recently played matches. It displays the result score, competition of the match as well as the teams that were playing against each other.

Log Table Screen



The log table screen displays the log standings of the league, detailing the matches played, wins, loses, draws as well as points for each team.

App Widget UI



This is an 8 x 8 app widget which will display the log table standings.

Key Considerations

Tools Requirements

- Android Studio 3.1.3
- Java 8 Programming Language
- Gradle version 3.1.3 or higher
- Compile version: 27
- Target SDK version: 27 (Oreo)
- Minimum SDK version: 15 (Ice Cream Sandwich)

How will your app handle data persistence?

The app will be using Firebase Realtime Database to store data remotely. This will allow the app to sync data while online and persist it to disk, making it available offline, even when the user or operating system restarts the app. The app will work as it would online by using the local data stored in the cache.

Describe any edge or corner cases in the UX.

The app will land the user on the news list screen when it is first launched. The user can will be able to click on certain tab then the app will slide to the screen linked to that tab. E.g. If the user launches the app for the first time then click on the fixtures tab, the app will slide to the fixtures fragment display a list of upcoming fixtures. While on the fixtures screen if the user clicks on the fixtures tab, the app will not change the screen.

The user will also be able to slide left and right to switch between the tabs. If there news screen is displayed, the user will not be able to slide to the left though because this is the first screen of the view pager. Similarly the user will not be able to slide to the right when the table log screen is displayed since this is the last screen of the view pager.

The user will be able to select a news article to read by clicking on a desired news article card on the news list screen. A screen displaying the entire article will be presented to the user the user. If the user hits the back button from this screen they will be taken back to the news list screen.

If the user hits back while any other screen (news list, fixtures list, results & table log) is displayed, the app will be exited.

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

Mockito (version 2.18.3) – to mock out dependencies when unit testing

Butterknife (version 8.8.1) – to annotate fields and cast corresponding view in the layouts

Glide (version 4.6.1) – to handle the loading and caching of images

Dagger (version 2.16) – for dependency injection

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

Firebase Database - for real time data storage
`com.google.firebase:firebase-database:16.0.1`

Firebase Storage – to store the images
`com.google.firebase:firebase-storage:16.0.1`

Firebase Analytics – for app usage reporting
`com.google.firebase:firebase-core:16.0.1`

Firebase Crash – to report on crashes
`com.google.firebase:firebase-crash:16.0.1`

Firebase Cloud Messaging – for push notifications
`com.google.firebase:firebase-messaging:17.1.0`

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 project
- Configure libraries
- Implement the package structure using the MVVM architecture design pattern:
    - *Service*
    - *Repository*
    - *Dependency*
    - *Firebase*
    - *Utils*
    - *Model*
    - *View*
    - *ViewModel*
- Configure product flavors
    - Release
    - Development

Task 2: Implement UI for Each Activity and Fragment

- Implement News Feed Feature
    - Build UI for MainActivity
    - Build UI for NewsListFragment
    - Build UI for NewsDetailsActivity
    - Build UI for NewsDetailsActivityFragment
- Implement Fixture List Feature
    - Build UI for FixtureListFragment
- Implement Result List Feature
    - Build UI for ResultListFragment
- Implement Log Table Feature
    - Build UI for LogTableFragment
- Build UI for ErrorScreenActivity
- Build UI for AppWidget

Task 3: Firebase Setup

- Create Firebase project on the Google developer console
- Add Firebase to the app
- Store club logo images and article images in Firebase storage
- Create entities in Firebase
- Create POJO in Android Studio to match the Firebase entities
- Configure Firebase cloud messaging
- Setup Firebase dependency injection module

Task 3: Implement the Business Logic

- Implement News Feed Feature Business Logic
    - Implement AsyncTask to execute a task to fetch remote data and synchronize that data with the local data
    - Implement a ViewModel to handle the data transfer logic
    - Implement a ViewModelFactory to inject into the UI components to provide the viewmodel
- Implement Fixture list Feature Business Logic
    - Implement AsyncTask to execute a task to fetch remote data and synchronize that data with the local data
    - Implement a ViewModel to handle the data transfer logic
    - Implement a ViewModelFactory to inject into the UI components to provide the viewmodel
- Implement Results list Feature Business Logic
    - Implement AsyncTask to execute a task to fetch remote data and synchronize that data with the local data
    - Implement a ViewModel to handle the data transfer logic
    - Implement a ViewModelFactory to inject into the UI components to provide the viewmodel
- Implement Log table Feature Business Logic
    - Implement AsyncTask to execute a task to fetch remote data and synchronize that data with the local data
    - Implement a ViewModel to handle the data transfer logic
    - Implement a ViewModelFactory to inject into the UI components to provide the viewmodel

Task 4: Implement the App Widget Service

- Implement a Remote View Service for the app widget

Task 5: Test the App