GitHub Username: rclark1212
Doc Version: 1.0, 4/9/16

AppSyncr (backup name DeviceSyncr)

Description

This app allows users to synchronize and manage their installed apps across multiple android devices.

Generally owners of multiple devices want their devices to all have the same capabilities and apps installed on each device. This includes multiple sized tablets for different occasions (but you still want netflix, hulu, kindle installed on each device) or multiple Android TV devices deployed in the home. While it is possible to push one app to multiple devices via the web store, you cannot do this from the device google play store. And installed apps always seem to get out of sync per device.

It would be extremely convenient to have an app which, when installed on a device, would communicate with other devices owned by the user and provide the user with a single viewpoint of all their devices, the apps on each device along with whether their current device has all the apps which had been installed on other devices (and offer the user an easy way to install the missing apps).

Works with both phone, tablet and android tv.

Intended User

Users of multiple android devices. Generally power users.

Features

The main features of the app are:

- Collect installed app information on the device it is running on
- Pushing the installed app information to cloud storage
- Notifying/updating cloud storage when an app is installed or removed
- Getting a list of other devices (and apps on them) from cloud storage
- Notifies the user if an app has been installed or removed from another device
- Showing the user a detailed view of each device registered to the app (local and cloud)
- Showing the user a superset of all apps installed for that class device
- Allow the user to manage apps on devices
    - Locally will allow user to easily install or remove apps
        - Will have a superset row of apps to choose from
        - Will allow user to clone an app load from one device to another
    - Remotely will send install/remove notifications to the device which will be processed when app is run on that device

User Interface Mocks

Summary

UI mocks below (handdrawn). Showing the main view (item selector), the detail view container per platform and then the detail view. In the mocks below, there is some minimal text expanding upon the purpose and flow of each view.

I apologize in advance for my total lack of drawing skills.

Main Screen and Nav

The primary screen the user sees. For phone/tablet, contains a list of either devices or apps (depending on the selection of the nav drawer). For android TV, uses a browsefragment to show all list elements as a series of rows.

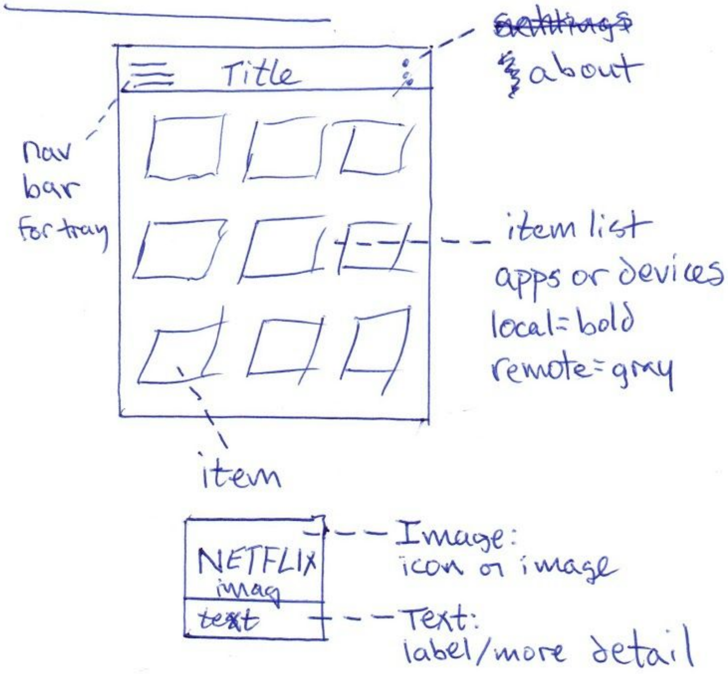
Each list element has graphics and text. The image section is either the app icon (apps) or an image of a tablet/android device (devices). The text is an identifying label for that app or device.

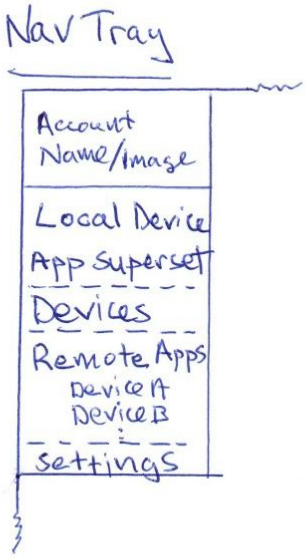
Nav drawer/browserow selections include:
Local Device - list elements will be the local apps installed on the device.
App Superset - list elements will be a superset of all devices that have this app installed (uses cloud data). Local list elements will be normal/bold, remote list elements will be slightly grayed (we will use this theme consistently).
Devices - list elements will be all the devices managed by this app (local bold, remote grayed)

When a list item is selected, you will be taken to the detail view.

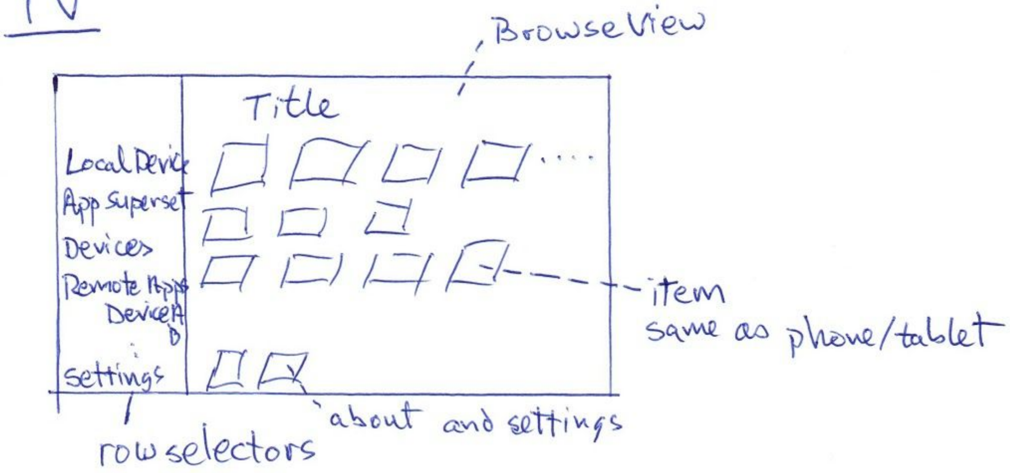
Main Screen + Nav

Phone/Tablet



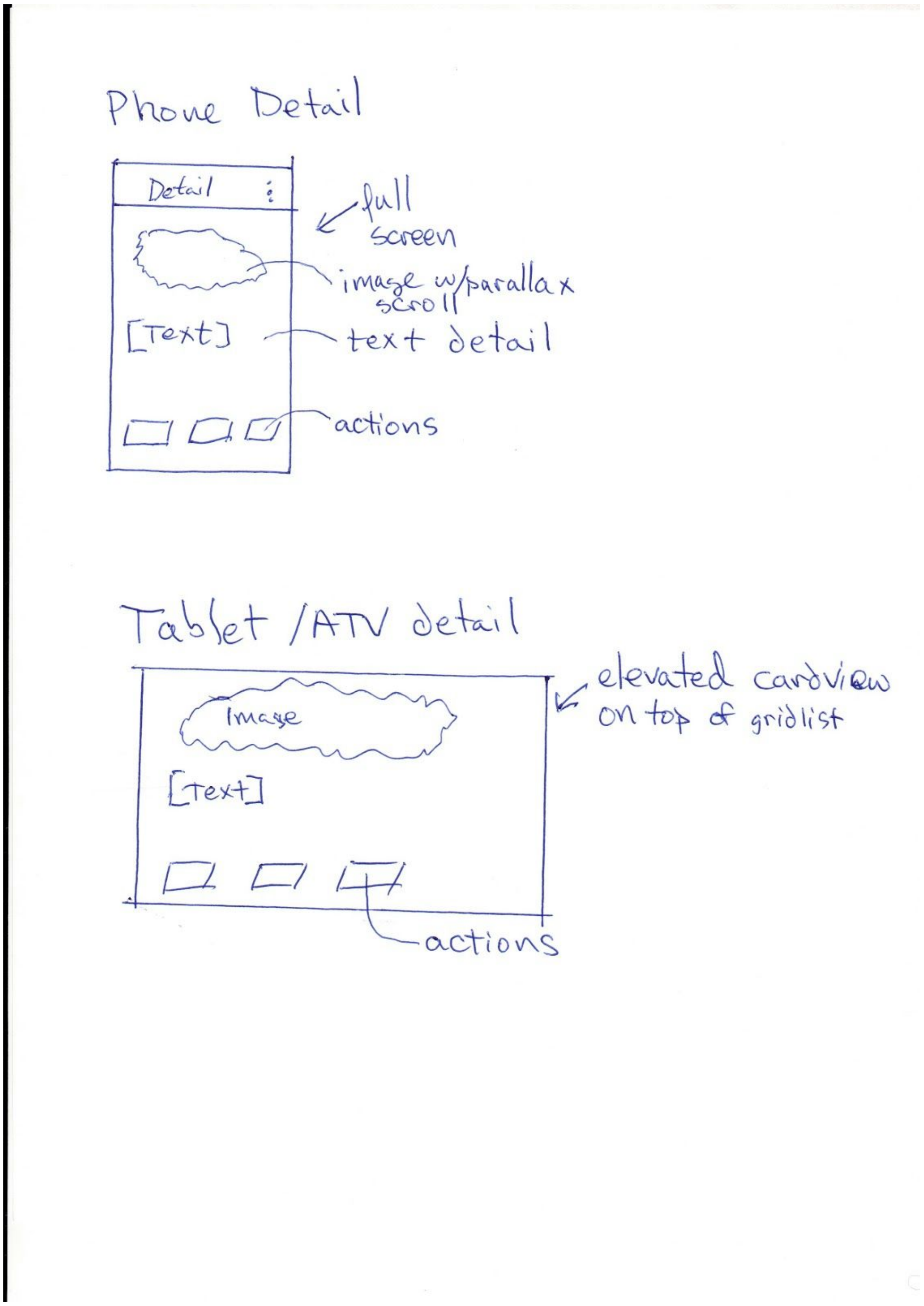


TV



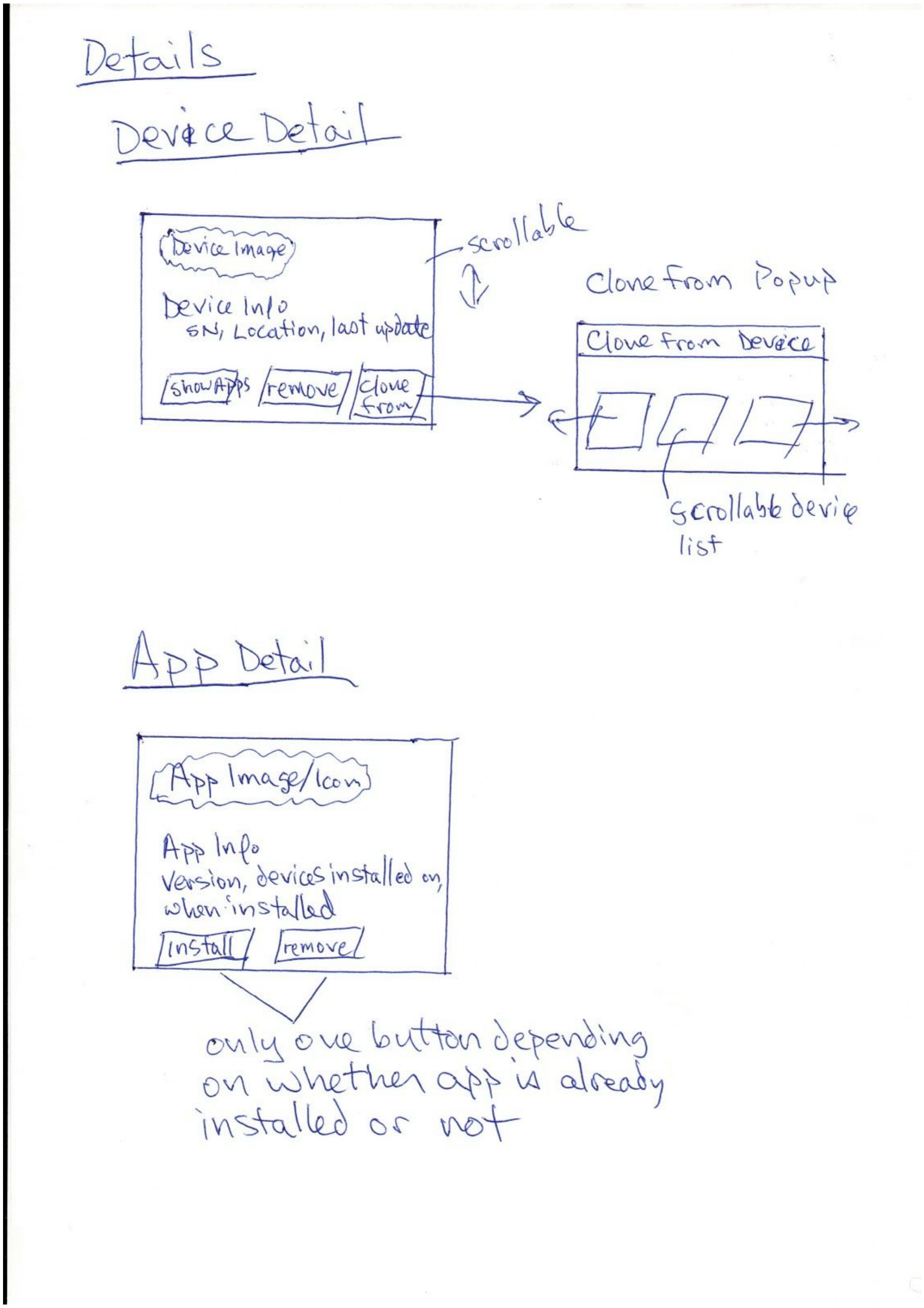
General Detail View per Platform

Invoked from selecting a list item from the main view. Phones (and small tablets) will use a full screen detail view. For large tablets and android tv, we will use a cardview that is elevated above the list selector. In general, there are 3 elements/areas on each detail view - image on top, text detail below, action buttons at bottom.



App and Device Detail View

If an app list element is selected, the app detail view will come up. If a device list element is selected, the device detail view will come up. See last section for differences between platforms on how the detail view is presented.



Device detail will offer “Show Apps” and “Clone From” if the local device is selected, “Show Apps”, “Remove” for a remote device.
Show Apps - will navigate the user to the device on the detail screen (will jump user back to the list view with the current device selected - so user will see that devices apps).
Remove (remote only) - will remove that device from both the local and cloud database (for use when you stop using a device and no longer want it tracked). Note, if the removed device runs this app again in the future, it will be re-added.
Clone Apps From (local only) - this will pull up another dialog popup on top. It will present a scrollable horizontal list of remote devices that you can select from. Once selected, app will fire off intents to package manager to make current device look like remote device.

App detail will offer only a single button. If the app exists upon your current local device, there will be an action button to remove it from the local device. If the app does not exist upon your current local device, there will be an action button to add it to the local device.

Key Considerations

How will your app handle installing/uninstalling apps?

Have to go through package manager so app install will not be automatic. User will need to confirm each install/uninstall. App will fire off intents to perform these install/uninstall operations.

How will your app handle data persistence?

Will build a content provider in the app to cache both local app data as well as cache cloud device data locally to the device. The GCE service will also maintain a database for all the devices in the group. Data included in content provider will be a device database as well as an app database which will include labels, package names and launch icons.

How will your app handle data synchronization?

Device->GCE: App will use a broadcast receiver to be notified of app changes on the device. Any app change (install, uninstall) will be immediately broadcast to the GCE. So the GCE will always have an up to date database of devices/apps. In the case of no connectivity, changes will be queued for later transmission.

GCE->Device: We will not pull data from GCE until the client app is opened? Or should we run a push service to notify that there has been an update and show a notification to the user? Should be a user configurable option.

Describe any corner cases in the UX.

When selecting showing apps from the device detail screen, it will jump the user to that device's nav bar entry to show the apps on the device.

Settings will be per client only (i.e. if I enable notifications on deviceA, that has no impact of notifications on deviceB).

Notifications of app install/remove will be filtered against the local app load of the device. If the app already exists on the device, notification of install will not show. Same logic applies for delete.

Deleting deviceB from deviceA even if deviceB is still actively being used. The delete device is really meant to delete old devices which are no longer used. In this case, delete will delete the data in the GCE database but if deviceB is turned back on, it will repopulate - warn user to uninstall app on deviceB for permanent deletion. May need to revisit.

It is still open on whether we want to allow a user to set up install/remove actions on remote devices. If we allow, these requests turn into pending requests at the remote when remote is activated. On the one hand, it provides a single management portal to manage devices. On the other hand, it could become quite complex for the user with little benefit. For the initial version 1.0, only allow management for the local device you are running on. Note that the remote devices will still get notifications of app installs on other products.

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

Will use Picasso to handle the loading and caching of images.
Will use google app engine endpoint libraries
Will use the gce library being built as part of this app
Will use standard android app libraries (appcompat, etc)

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

Will use GCE service for the cloud. And GCM for push notifications
Will use broadcast receiver for capturing install/uninstall events
Will use intents to kick off install/uninstall intents
Will use CP for local caching of data
Will use SyncAdapter to link CP to UX
Will use IntentService to run back end sync service to cloud along with SyncAdapter.
May use admob for interstitial ads
Will use location services to identify location of device
Need to figure out authentication service (Identity!, Account manager, Auth with endpoints, GoogleAccountCredential)
Will provide notification services

Will allow share services to send a list of apps on a device (gridview) to email
Custom view - is open...Once we start mocking up, can see if one is needed (maybe a grayed out cardview?). One possible custom view is a cardview with a horizontal scrolling list for selecting the device to clone from. See mock.

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 0: Research

Identify all tasks that have risk (services which I don't have experience with). Would include:

- Grabbing the app load on device
- Install intents and broadcast messages
- Location services for device
- Authentication for endpoint

Read up on using these services on the web to ensure that the plan on how to use them in this app is valid/reasonable. In some cases, do an implementation of a feature in code to understand how it works.

And, of course, write this POR/MRD document :)

Task 1: Project Setup

Create the initial empty project structure using android studio. Reasonable file/class names.

- Implement the directories for sync adapter, content provider.
- Ensure gradle, API level choices, etc are appropriate. Note, will require API 21 or higher (L).
- Set up gradle installRelease task (including signing config)

Task 2: Implement UI for Each Activity and Fragment

Creating a minimal UI implementing the gridview and menuing system for the app.

- Build UI for MainActivity/gridview/browseview
- Build UI placeholder for DetailView from gridview element
- Put in resource placeholders for strings/image data

- Populate with dummy/hack temporary data

Task 3: Implement local app data gathering routines

Replace the hack dummy data routines in last task with routines that get real data for the local device (note that endpoint tasks come later)

- Implement device data capture routines (name, serial number, etc)
- Implement app data capture routines (apps on device)

Task 4: Implement Local Content Provider

This will be the local cached data set used for both local data as well as remote data. Use with local data only for now.

- Create data layout
- Implement the contract, helper, provider functions
- Implement intent service call to update provider from local data
- Hook up CP to the UI. Move from array objects to SyncObjects on the gridview UI

Task 5: Hook up CP to the UI

Replace hack UI data population with a sync adapter pointing to the CP.

- Move from array objects to SyncObjects on the UI
- Create a hack dummy routine to populate CP with dummy cloud data for UI work

Task 6: Implement Detail View UI/functionality and Settings

Current DetailView is a placeholder. Implement UI per mocks and per needed functionality.

- Implement DeviceDetail (info, remove, applist, clone)
- Implement AppDetail (info, install, uninstall)
- Implement DeviceClone custom view (cardview with scrolling device list)
- Implement Share activities here as well (device app load share to another user)
- Implement Settings
    - Notification preferences
    - Publish to cloud
    - Doubtful we will have any account settings here as I plan to use google id for authentication through identity.

Task 7: Implement Endpoint Project w/Authentication

Build the endpoint for cloud support.

- Set up endpoint project (base on builditbigger and internets)
- Implement APIs/messaging between app and endpoint (includes syncadapter and GCM work)
- Implement identity/authentication for endpoint
- Set up endpoint content provider - risk here - assuming we can use CP in the endpoint and per internet this looks to be correct. Should be able to copy most of the code from local CP.
- Test locally with web
- Deploy to google

Task 8: Implement Sync Service to Endpoint

Basics of this work will have been done above as part of the testing. Complete it here.

- Hook up and verify GCM is working (local app getting notified on cloud change)
- Hook up and verify GCM above syncs/updates local CP
- Hook up and verify that local CP changes update cloud

Task 9: Implement multi-Platform UI

So far we have only been dealing with one platform (ATV - sidebar, ATV chosen as initial implementation for several personal reasons). At this point, implement the tablet/phone UI.

- Add tablet/phone app to project
- Implement gridview/detailview UIs
- Implement local app data gathering routines for tablet/phone
- Add resources placeholders for this platform
- Hook up to content provider
- Implement/verify material design transitions

Task 10: Implement Broadcast Receivers

Implement broadcast receivers for watching for app install/uninstall

- Implement service to watch for package manager messages
- Hook up broadcast receiver to CP/syncadapter to update cloud

Task 11: Implement Notification Service

Implement notification service for the platforms. Triggers on GCM cloud updates indicating addition/removal of apps from other devices.

- Implement basic notification service (with rules from settings)
- Consolidate multiple notifications into a single notification
- Implement rich notification (graphics) and launch point into appropriate UI location

Task 12: Implement Help/Review OOBE

Implement help. Review initial setup (OOBE) and iterate design with focus on minimal time/clicks required to get meaningful operation of the app for the user.

- Implement a first time use cling on the UI (both platforms). This will be the only help.
- Implement a very simple "about" screen describing functionality and how data is treated locally and in cloud (include data privacy terms here as well). This is a pretty specific utility app which addresses a pretty specific problem.
- Specifically should *not* have a help screen on use of the UX (UX is a failure if you have to have that)

Task 13: Cleanup!

App should be complete at this point modulo cleanup and additional testing.

- Replace resource placeholders - good images, good icons
- Ensure accessibility - verify all controls have descriptions, verify navigation and focus order
- RTL - verify all layouts/controls/formatting are RTL ready
- Comments - clean up comments, review FIXME, TODO
- Lint - run and fix lint
- Strings - verify all user facing strings are embedded in strings.xml (along with translation tags)
- review against rubric - verify all udacity rubric requirements met.

Task 14: eBeta!

Time to get others testing this app. Deploy app amongst friends/family/work. At minimum do 2 1 week beta cycles to flush out any usage bugs, unexpected behavior or unhandled use cases. Obviously evaluate feedback and fix.

Task 15: Release!

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**”