

Preliminary Design Review (PDR)

Cause And Effect



By:

Keith Adler

Matthew Brannick

Tomin Kozhimala

William Vennes

Table of Contents

Vision Statement.....	4
Executive Summary.....	5
User Needs.....	6
High Priority User Stories.....	6
Medium Priority User Stories.....	7
Low Priority User Stories.....	8
Phone/Physical Rules.....	8
Account Sync Rules.....	8
Engineering Specifications.....	9
Acceptance Test Plan.....	11
Initial Design Ideas.....	18
Rule View Activity.....	18
Drag and Drop View.....	19
Rule Creation Activity.....	20
My Rules Activity (Version 2).....	21
Account Sync Activity.....	22
My Rules Activity.....	23
Help Activity.....	24
Android Screen Flow Diagram.....	25
Rule Handling Workflow (Backend).....	26
Current Design.....	27
Broadcast Receiver.....	27
Rules Engine.....	28
Overview.....	28
Structural View.....	29
Thread View.....	30
Expression Tree.....	31
Individual Cause Evaluation.....	32
Database.....	33
Action Executer.....	36
Thread Diagram.....	37

Risk Management Plan	38
Agile Development Schedule	40
Budget.....	43
Source Code Repository.....	45
References	46
Appendix	52
Phone/Physical Rule User Stories	52
Account Sync Rule User Stories	55

Vision Statement

Cause and Effect: For people wanting to make their smart devices truly intelligent. Today, smart phones are overly complex and jammed pack with functionality. Despite having the power to do so many spectacular things, smart devices lack the ability for users to take control and put the phone to work for them. Cause and Effect is a software solution that let users easily program their smart devices to respond to a set of rules with a chosen effect. With Cause and Effect, smart devices will become more intelligent and make user's lives easier by intuitively automating tasks, gathering information, displaying information, etc. Unlike other rule based software, Cause and Effect will provide more functionality and a better user interface to allow any type of user to take control of their phone.

Executive Summary

Today, smart phones are a part of our everyday lives. Users of smart phones are able to do a variety of functions, whether it is talking to friends on Facebook, listening to music, or sharing and watching videos on YouTube. However, smart phones currently do not have the ability to automatically perform user tasks based on changes that occur in the phone's environment.

Users often pay large amounts of money for their smart phones but never utilize their phone's potential. The typical smart phone user uses only functions such as the phone, the text messaging, email, and some fun games. There are so many features and APIs available for smart phone development but rarely are these integrated to enrich the life of the user by using the entire phone's brilliance.

The solution to this is Cause and Effect for Android. Cause and Effect allows users to take more control over their phones. Users of Cause and Effect will be able to set "rules" for their phone, which enables the phone to automatically respond to changes in the environment, such as a change in Wi-Fi connection or current phone location via GPS. Users will also be able to share their phone rules with other users of Cause and Effect, whether it is via email, text, or NFC between other Android devices. Cause and Effect will enable users to automate desired functionality and will increase the usability of their smart phone.

Partnered with our client, Samsung Mobile, we have been given the tools and direction to develop Cause and Effect for Android. We plan on revolutionizing the way how users see their phones and the smart phone development industry. Through much planning and hard work, we have a working prototype here today.

The application is simple and intuitive. It is so easy that your grandma could use it. The user can create rules that can be toggled on and off. These rules consist of Causes and Effects, which is where the name comes from. The Causes are actions from the phone's operating system or from an external source such as Facebook, Gmail, or Twitter. The operating system broadcasts can come from simple calls or messages, or many of the phones sensors and services. Users can choose causes with their own parameters to trigger effects. These effects are actions that the phone can execute. These effect actions allow users to really gain a lot of benefit from this application. With various types of causes and effects, users can create complex rule structures to fit their needs. Further variety can be attained using the logical AND and OR structure integrated into the causes chosen by the user. These causes are arranged in a tree structure so that users can apply user friendly Boolean logic to their own rules. Since you only have to setup a rule once, this functionality allows users to really take advantage of their phone from then on. Users can also share their rules and lists of rules with other Cause and Effect users so a community could form around rule creation and rule types that solve everyday problems.

User Needs

We met with Samsung and outlined our requirements for the project. These requirements are organized into user stories, so that we can develop a project in a structured manner that abides by our agile development methodology. Some user stories can be considered "epic." Epic user stories are high level requirements and are intended to be broken into sub user stories and tasks when developed. For organizational purposes, the stories are organized into three categories. High Importance user stories are user stories that the client specifically asked for, are epic, or are "need to haves." Medium important stories are the "nice to haves" and complete "if extra time is available." Low importance user stories consist of our project backlog and end product ideas.

High Priority User Stories

- As a client, I want a rules engine, so that users can automate phone actions.
- As a client, I want the rule creation process to be as easy as possible.
- As a client, I want a fluid user interface, so that all users can use our application with ease.
- Extensible to Third Parties
- As a client, I want rules to work with Boolean algebra, so that users can create complex rules to tailor to their needs.
- As a client, I want rules to be sharable, so that users can easily distribute their rule frameworks to others
- As a client, I want the application to require a minimum of Android version 4.0 (Ice Cream Sandwich), so that we can utilize the next generation of Android devices.
- As a client, I want the application to run on all Android v4.0+ devices, so that we can have as much user compatibility as possible
- As a client, I want to include NFC based rules and sharing, so that the app can make use of Samsung's latest hardware sharing technology.
- As a client, I want rules that react to the user's location, so that users can create intricate rules using any given location.
- As a client, I want to include Wi-Fi based rules and sharing, so that the app can make use of local network capabilities.
- As a client, I want to be able to put many rules in (stress test) and have it work as quickly as with very few rules.

- As a developer, I want an easily accessible database (such as SQLite).
- As a developer, I want to be able to decipher and encode rules with Boolean algebra.

Medium Priority User Stories

- As a client, I want the application to be optimized for battery efficient, so that the application will not drain the phone's battery life.
- As a client, I want an additional drag and drop rule creation interface, so that the user can choose to have a fluid look and feel while creating rules.
- As a client, I want the application to run quickly and not to slow down my phone's other processes.
- As a client, I want the application to not take up much space on my phone.
- As a developer, I want to be able to also share using email, MMS, Facebook, and Dropbox, so that users can have options on the sharing medium.
- As a developer, I want to create user notifications, so that users are aware of the application's activity and progress.
- As a developer, I want to include a user preferences activity, so that users can tweak and enable settings for the application.
- As a developer, I want to have text message based rules, so that I can debug rules using the phone's mobile network.
- As a developer, I want to have time-based rules, so that I can debug the application easily.
- As a developer, I want to have vibrate-based rules, so that I can debug the application easily.
- As a developer, I want to have "Toast" based rules, so that I can debug the application easily.
- As a developer, I want to have Facebook-based rules, so that I can test the account sync rules.
- As a developer, I want to have Google-based rules, so that I can test the account sync rules.
- As a developer, I want to have Dropbox-based rules, so that I can test the account sync rules.

Low Priority User Stories

- As a developer, I want to have screen animations, so that the application is fluid and aesthetically pleasing.
- As a developer, I want a cool logo, so that our application is memorable and aesthetically pleasing.
- As a developer, I want a cool application icon, so that our application is easy to recognize on the Android home screen.
- As a client, I want to know the application is working correctly when closed.

Phone/Physical Rules

There are many phone features that will be included in the final product. Some of the rule types are considered more important, and there are listed above and are not included in this list. For the sake of space and organization, the list of phone-based rules is listed as an appendix at the end of this document.

Account Sync Rules

There are also numerous account types that will be integrated in the final product. The plethora of accounts and rule types even prompted the creation of a "Category View" that will be the default view for browsing rule options. For the sake of space and organization, the list of account types is listed as an appendix at the end of this document.

Engineering Specifications

- A. The UI is simple and effective.
 - a. The user interface must be simple enough for all of the users to be able to use it or no one will use it. It must also be effective at doing what it does and not just be a bunch of pretty buttons.
- B. Implement certain rules that were given to us by Samsung.
 - a. Wifi and Location must end up in the final implementation or as soon as possible since Samsung wanted to see these especially.
- C. Check that the rules engine is working properly.
 - a. The rules engine must be working properly or the application will crash and there will be no processes running and you just have an app that is taking up space until the user removes it.
- D. Evaluate battery life while the app is running.
 - a. The battery life in phones is of a major concern to most users as no one remembers to fully charge it daily and they definitely do not want to have to plug it in every few hours.
- E. How often the app checks to evaluate rules.
 - a. This extends back to battery life; you must balance between battery life and how often you need to check the rules.
- F. Test the sharing functions.
 - a. People want to be able to share their rules with other people who have phones without manually typing it into another phone.
- G. Check extensibility to 3rd party programs.
 - a. The application must work with other programs such as Facebook and Twitter for a lot of users to consider it a useful application of this type.
- H. Check extensibility to tablets.
 - a. The application must be usable also by tablets so that Samsung can put it on all of their smart devices.
- I. Set up Bitbucket.
 - a. We must have somewhere to store all of the code that we are working on while keeping it agile. It is also important to make this code private so that no competing companies can get their hands on it.
- J. Set up Mercurial.

- a. Mercurial works with Bitbucket to allow us to update more easily with a better user interface.
- K. Make sure the Boolean algebra works correctly.
 - a. We need to make sure that the Boolean algebra is encoded correctly and parsed correctly afterwards. This must also be true for much larger rule sets.
- L. Check the drag and drop UI.
 - a. We must check that this user interface in particular is both user friendly and works correctly.
- M. Check that the accounts connect correctly.
 - a. When we connect the accounts to the phone, we must make sure that these are functioning accounts and not just a set of data taking up space. It also affects our rules as some of them need this info to work.
- N. Make it so that a single receiver receives all of the rules instead of multiple receivers.
 - a. There is no need to waste space and processing power to have multiple receivers. Afterwards, it is also important to check that the intent is then handled correctly.
- O. Check that the event handler correctly receives callbacks from the underlying system.
 - a. We must make sure that the event handler is getting all of the data correctly so that we can use all of the necessary information for the application.
- P. Understand what an action is in respect to the rules database.
 - a. What does the database do with an action that is waiting?

Acceptance Test Plan

Test A: The UI is simple and effective.

Test by asking users if they have any problems using the application.

Process

Give the application or paper prototype to a random person (prefer non-engineer) to see if they have any problems using it.

Results

Able to make rule (yes/no): _____

Comments:

[Passed/Failed]

Test B: Implement certain rules that were given to us by Samsung.

Test by making sure that the rules work in our application.

Process

Run the application and make rules that use those functionalities, make them activate.

Results

Wifi (yes/no): _____

Location (yes/no): _____

[Passed/Failed]

Test C: Check that the rules engine is working properly.

Test by making any rule become active.

Process

Set up one of the rules to easily activate, then activate and make sure nothing crashes.

Results

Able to add rule (yes/no): _____

Rule is activated when appropriate (yes/no): _____

[Passed/Failed]

Test D: Evaluate battery life while the app is running.

Test by leaving a phone out that has the application running on it.

Process

Set up one of the phones with the application on it and compare battery life to when the phone does not have the application on it.

Results

Battery life without app: _____

Battery life with app: _____

[Passed/Failed]

Test E: How often the app checks to evaluate rules.

Test by waiting for any rule become active.

Process

Set up one of the rules to easily activate and then wait for activation.

Results

How long it takes after proper time before it activates: _____

[Passed/Failed]

Test F: Test the sharing functions.

Test by trying to share a rule.

Process

Set up one of the rules on a phone and use some service to try to share that rule.

Results

Able to share rule via email (yes/no): _____

Able to share rule via NFC (yes/no): _____

[Passed/Failed]

Test G: Check extensibility to 3rd party programs.

Test by making a rule that uses a 3rd party program.

Process

Set up one of the rules with a 3rd party program that you can easily activate, then activate it.

Results

Able to add rule (yes/no): _____

Rule is activated when appropriate (yes/no): _____

Rule functions correctly (yes/no): _____

[Passed/Failed]

Test H: Check extensibility to tablets.

Test by trying to put on tablet.

Process

Install the app on a tablet then try to run the app.

Results

Able to install app (yes/no): _____

Able to run app (yes/no): _____

All functionality still present (yes/no): _____

[Passed/Failed]

Test I: Set up Bitbucket.

Test by making the Bitbucket account and creating the project.

Process

Set up the project on Bitbucket and make sure it updates correctly.

Results

Able to make project (yes/no): _____

Updates (yes/no): _____

[Passed/Failed]

Test J: Set up Mercurial.

Test by making Mercurial update the Bitbucket project.

Process

Install Mercurial and try to pull all of the information and then update the Bitbucket through it.

Results

Able to pull information (yes/no): _____

Able to update Bitbucket (yes/no): _____

[Passed/Failed]

Test K: Make sure the Boolean algebra works correctly.

Test by making a rule with Boolean algebra present and trying to activate.

Process

Set up one of the rules to easily activate that has multiple causes, then activate and make sure nothing crashes.

Results

Able to add rule (yes/no): _____

Rule is activated when appropriate following Boolean algebra (yes/no): _____

[Passed/Failed]

Test L: Check the drag and drop UI.

Test by asking users if they have any problems using the drag and drop UI.

Process

Give the application to a random person (prefer non-engineer) to see if they have any problems using it.

Results

Able to make rule using drag and drop (yes/no): _____

Functionality of rule still present (yes/no): _____

Comments:

[Passed/Failed]

Test M: Check that the accounts connect correctly.

Test by making any rule that uses accounts.

Process

Set up one of the rules to easily activate that uses accounts, then activate.

Results

Able to add account (yes/no): _____

Account is able to connect to service (yes/no): _____

[Passed/Failed]

Test N: Make it so that a single receiver receives all of the rules instead of multiple receivers.

Test by checking output of the receiver to what it should be getting.

Process

Have the receiver print out all of the stuff it receives and compare to what it should be getting.

Results

Receiver receives all information (yes/no): _____

[Passed/Failed]

Test O: Check that the event handler correctly receives callbacks from the underlying system.

Test by having the event handler print out what it receives.

Process

Set up the application and having the event handler print out all the callbacks it receives.

Results

Event handler prints out desired callbacks (yes/no): _____

[Passed/Failed]

Test P: Understand what an action is in respect to the rules database.

Test by queuing up multiple intents.

Process

Set up multiple rules that you can then quickly queue up to see if they are being stored in database correctly.

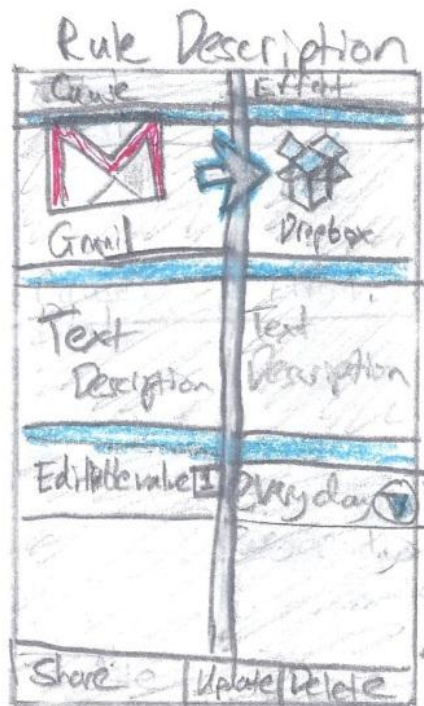
Results

Database is storing intent queue correctly (yes/no): _____

[Passed/Failed]

Initial Design Ideas

Rule View Activity

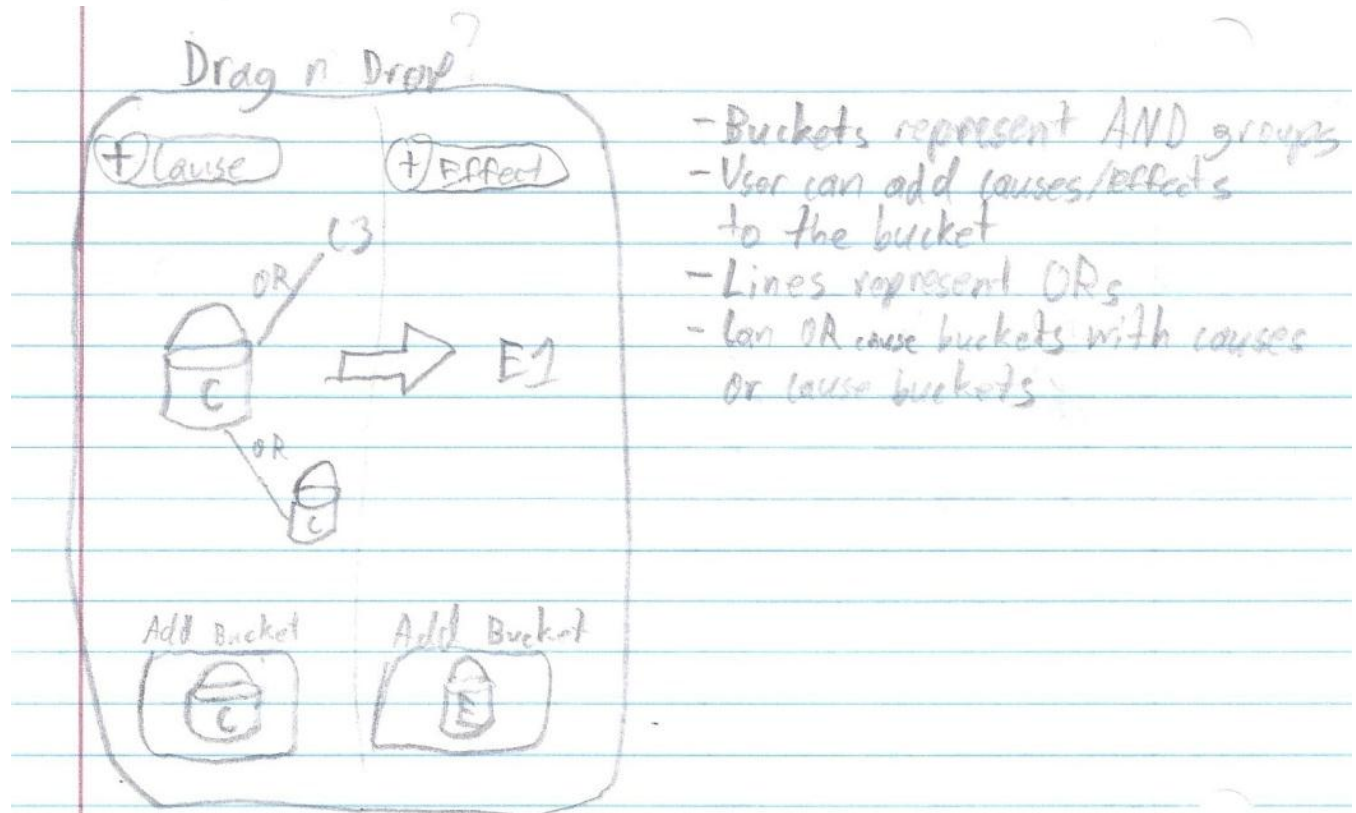


Rule View

- views one rule
- easy to read view
- share/update/delete the rule
- lower section for editable fields

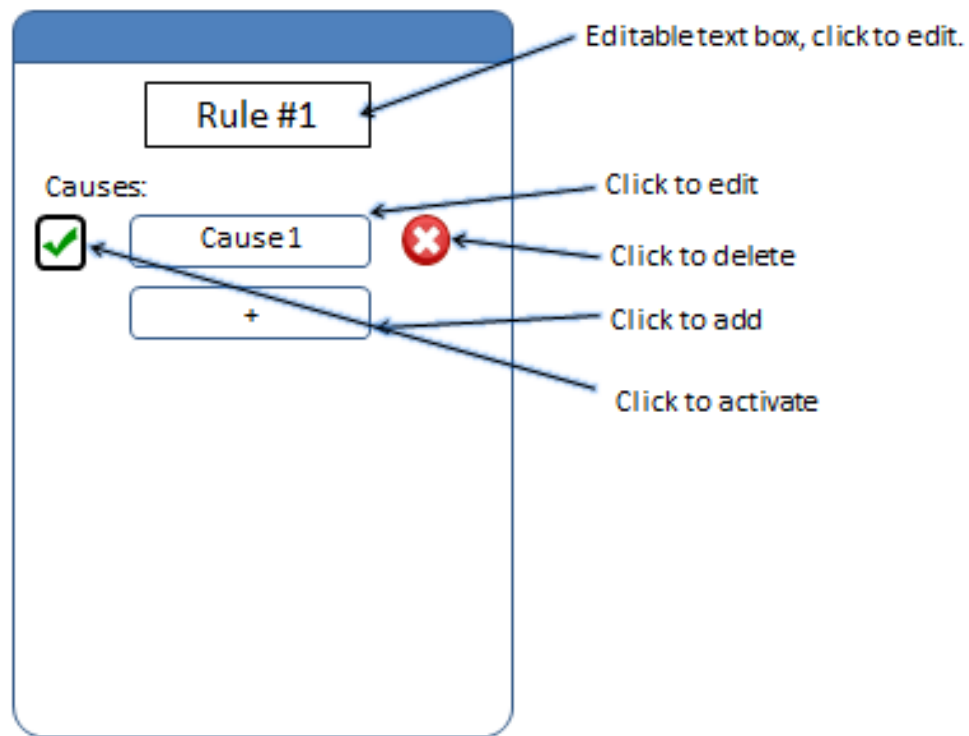
- Here, the user will follow the rule description page as they create the rule. They click on the cause size to choose a cause, and the effect page to choose an effect. As the user adds causes and effects, they will return to this page to see how their rule has progressed as they create it.

Drag and Drop View



- Here is an idea for a drag and drop implementation of the rule creation page, in which users place selections in “buckets” to represent groups of causes or effects, and link them to other buckets or individual items by an OR line. This would create a more abstract view for rule creation and could benefit visual-based users.

Rule Creation Activity



- Here is a rule creation page in which causes and effects are separated by different pages and takes a more list view approach.

My Rules Activity (Version 2)

Wi-Fi On

My Rules

Preferences

Help

Wi-Fi On

On

Edit

Delete

Cause:

At Home

At Work

Effect:

Wi-Fi On

New

Global On

- Another conceptual rule view where causes and effects are listed sequentially in a list format.

21 | Page

Account Sync Activity

The wireframe shows a mobile app interface for 'My Accounts'. At the top is a blue header bar. Below it, the title 'My Accounts' is centered. Under the title is a row of icons: Gmail, Google, a robot icon, and Twitter. To the right of these icons are three black dots, representing a scrollable list of accounts. Below this row is a form with two input fields. The first field is labeled 'Account Name:' and the second is labeled 'Password:'. Below the password field is a large empty rectangular area. Three blue arrows point from text annotations to the interface: one points to the title 'My Accounts', another points to the row of icons, and a third points to the large empty area below the password field.

My Accounts

Account Name:

Password:

Make buttons (like tabs) that will display information for below.

Replace with more icons that they are using.

Additional asked info can go here

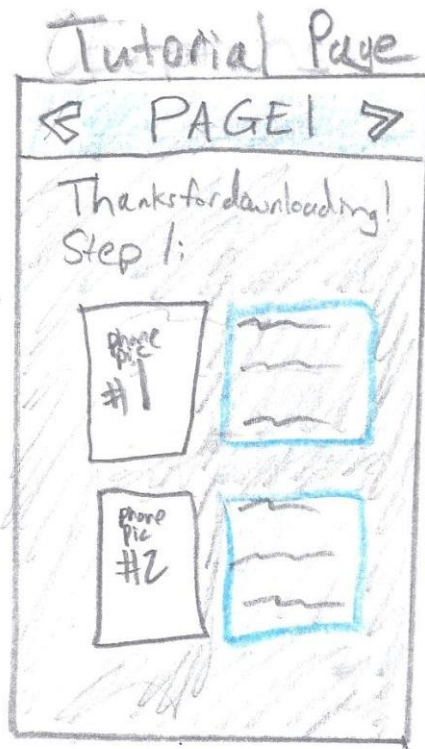
-
- This is an accounts page in which the user syncs their Google, Facebook, etc. accounts for use with rule creation. Another idea was moving the list of accounts to a separate page accessed from preferences, rather than a slide at the top.

My Rules Activity



- This was a concept screen for the viewing of existing rules. From this screen, users can turn a rule on or off, go to the edit menu, or delete the rule.

Help Activity

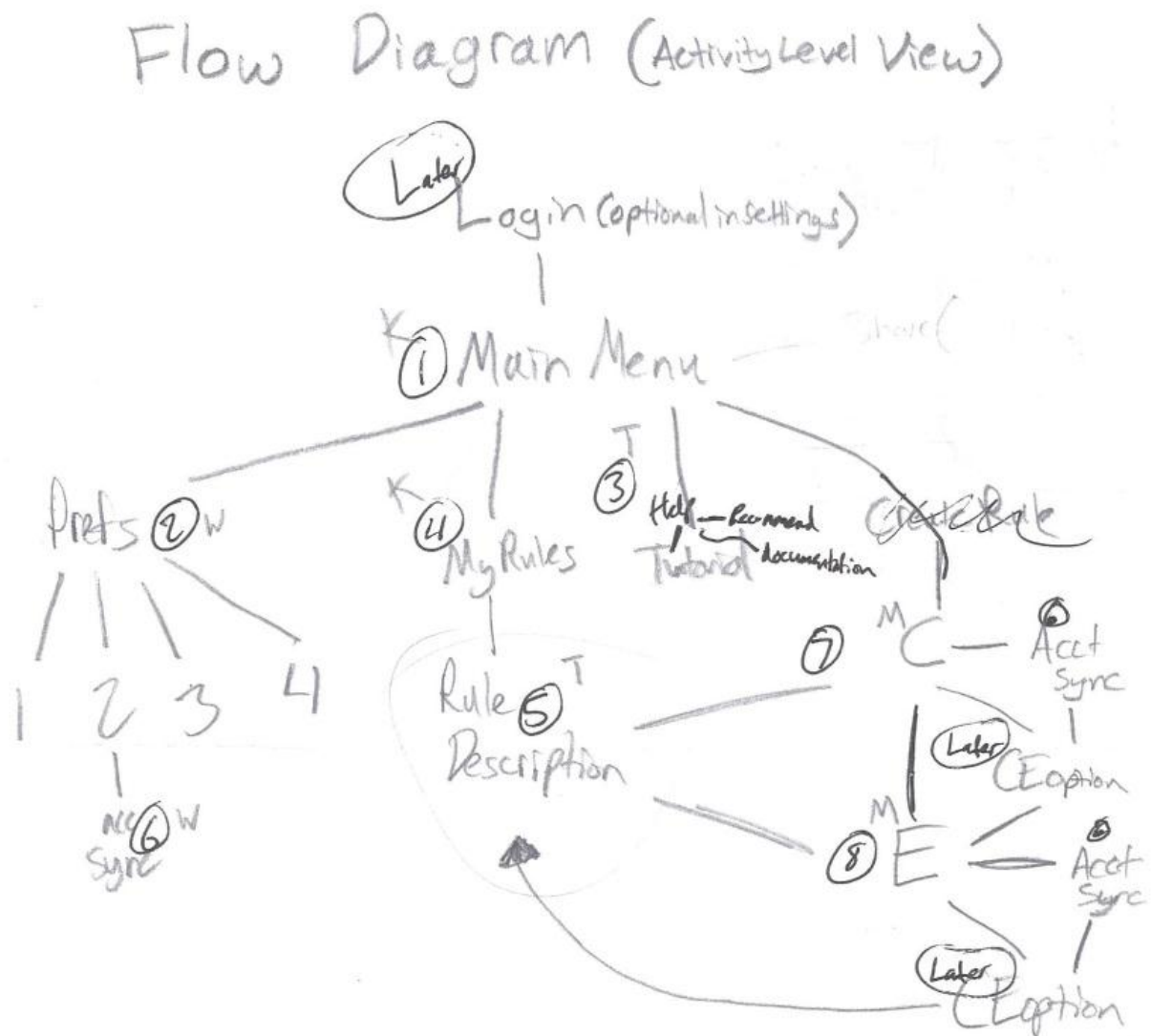


Tutorial

- A "So easy your grandma could use it" guide to the app.
- Page turning per step/direction set
- ★ editable page selection? I.E: Page 1/6
- has picture guide
- ★ video guide? link? Embed? Youtube integration?

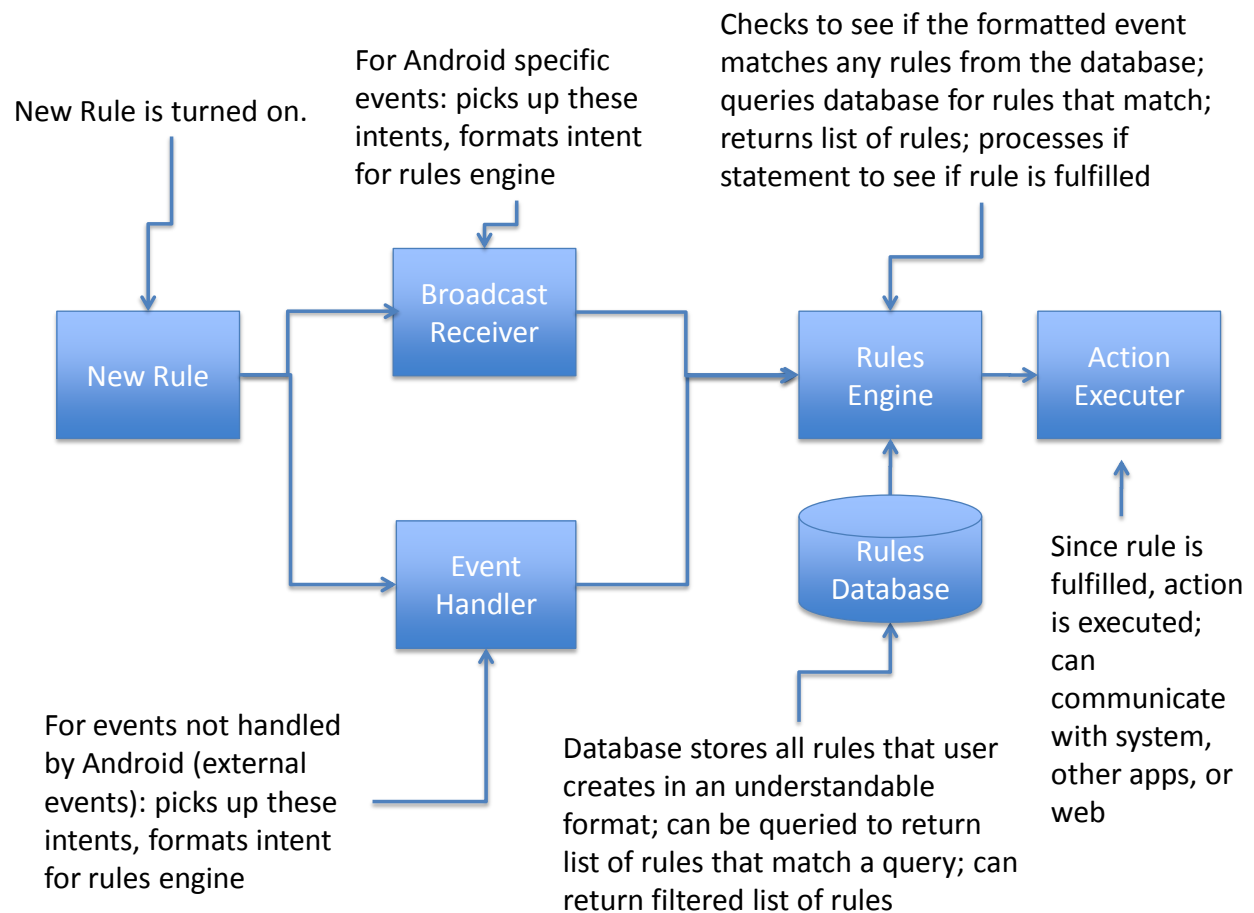
-
- The tutorial page is an idea we had that would help first time users understand how to operate the app. There was also an idea for a tutorial overlay for the app that would direct users towards options and functions for app operation.

Android Screen Flow Diagram



- This flow diagram represents the expected usage and screen flow for users operating the app.

Rule Handling Workflow (Backend)



-
- This flow diagram represents the initial backend design for the app.

Current Design

Broadcast Receiver

The first part of the rules engine is the broadcast receiver. The broadcaster receiver is a class built into the Android API that either sends or receives messages to/from the Android operation system. In our case, the broadcast receiver receives the different cause types and sends the data to the Rules Engine to process the causes if they exist in the database. The reason why the broadcast receiver is so light is because the broadcast receiver is only given 10 seconds to operate after it receives a broadcast. If the broadcaster receiver is still active after 10 seconds, the Android operating system will kill the process and the intent is lost. Therefore, we kept the broadcast receiver light and send off the commands with the minimal amount of processing. The broadcast receiver is also designed to work in the background so that it will be active even when the application is no longer focused.

Rules Engine

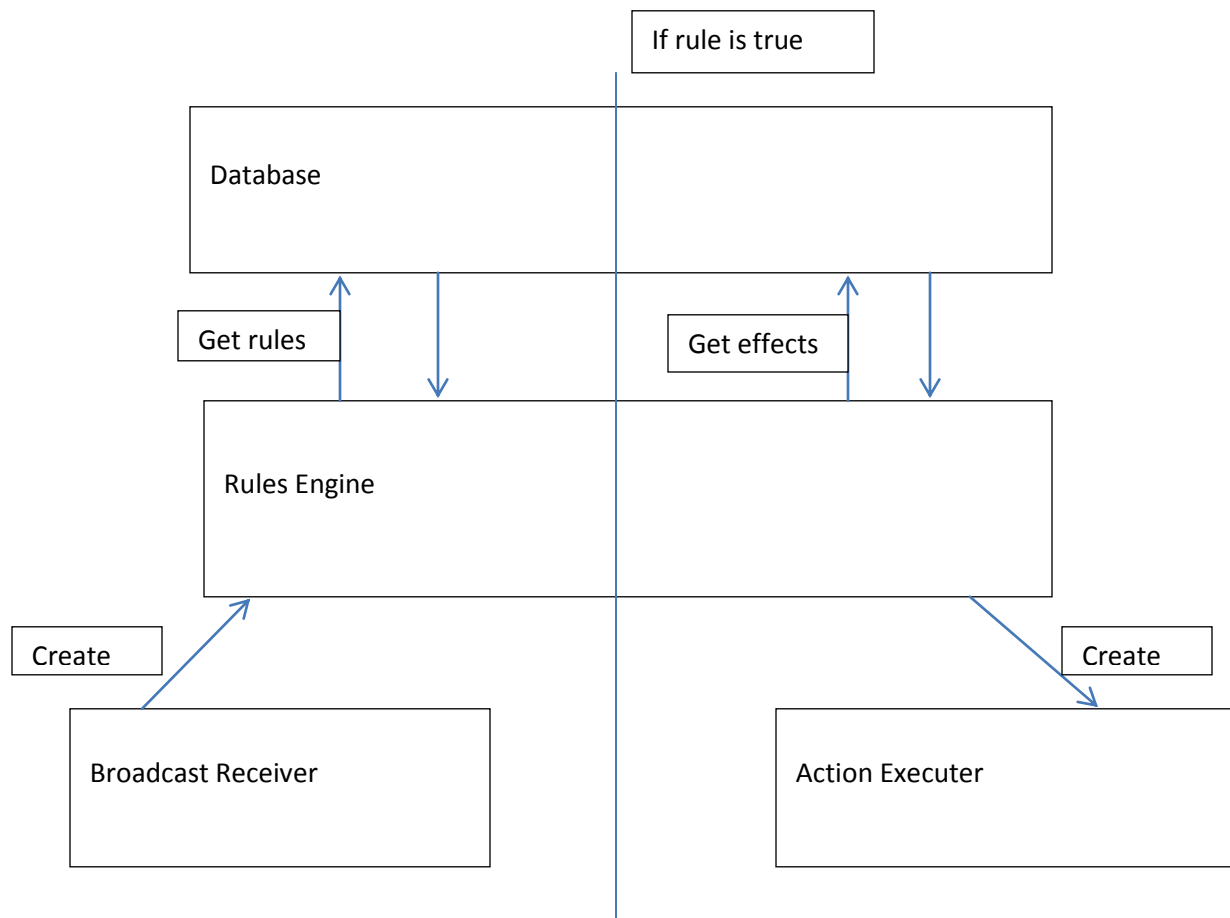
The main function of the rules engine is the evaluation of rules. The rules engine object is what determines whether a rule is true or false.

Overview

A rules engine object is created by the broadcast receiver once an event has been triggered. The broadcast receiver passes in the cause type, parameter and the context to the rules engine. The rules engine uses the type to send a request to the database for all rules of that type. Once the list of rules has been received, the rules engine evaluates each rule in separate threads using Android's AsyncTask.

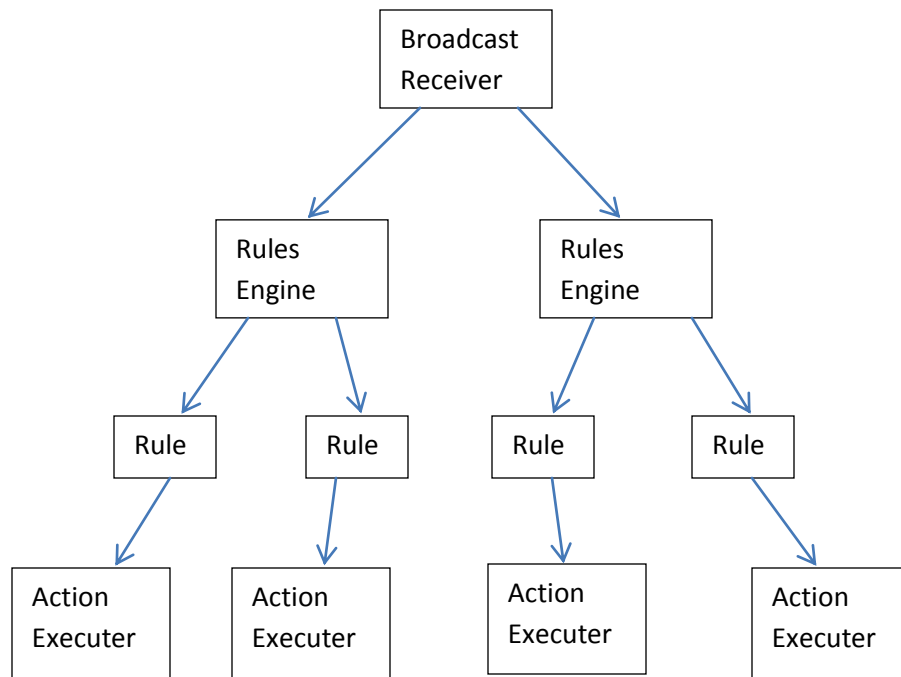
In each thread, an expression tree of rule causes is built using a stored character string in each rule. This expression tree is then evaluated recursively in a depth-first fashion. Each individual cause object within the tree has its own `isTrue` method that returns either true or false depending on what value is evaluated for the cause. Using these returned values, the expression tree will return a final value, true or false, to the AsyncTask. If the tree of the rule returns true, then the task will request the effect list for the rule from the database. This effect list is then sent to an `ActionExecuter` object for execution, along with the context initially passed from the broadcast receiver.

Structural View



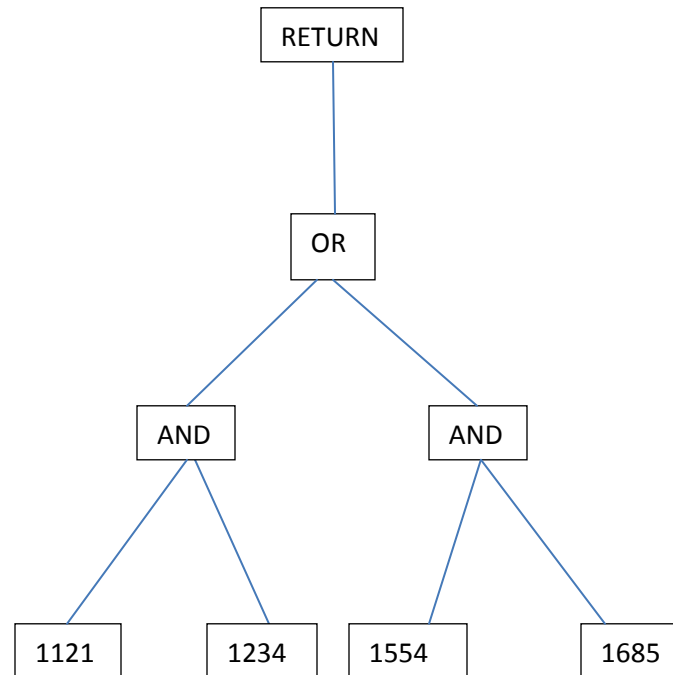
The rules engine interacts with the broadcast receiver, the database, and the action executor depending on if the rule cause tree returns true or not. If the rule evaluates to be false, then the right half of the view does not occur.

Thread View



Each rule stemming from a rules engine object is evaluated in its own thread. Should the rule return true, then an effect list is fetched from the database and an action executer object is created, which takes in the effect list and executes each effect.

Expression Tree



This is a view of the expression tree used to evaluate rule causes. This particular tree is the tree for a Boolean algebra equivalent of $(1121 \& 1234) + (1554 \& 1685)$. The character string stored in the rule for this tree would be “+:&:1121@:1234@@:&:1554@:1685\$”. ‘+’ indicates an OR node. ‘&’ indicates an AND node. ‘:’ indicates moving down a level while ‘@’ indicates moving up one level. The numbers are the cause ids that will be used to receive actual cause objects from the database for evaluation. The ‘\$’ character indicates the end of the string.

Individual Cause Evaluation

Causes are evaluated using a function containing a switch statement that chooses evaluation methods based on the rule type.

Pseudocode for cause evaluation:

```
Bool isTrue(eventType, param)
{
    // event type is given to use by the broadcast receiver
    Switch(eventType)
    {
        // each case returns true or false depending on specific
        //conditions of each case
        Case wifi: // do something
            Break;
        Case phoneCall: // do something
            Break;
        Case time: // do something
            Break;
        Default: return false;
    }
}
```


Database

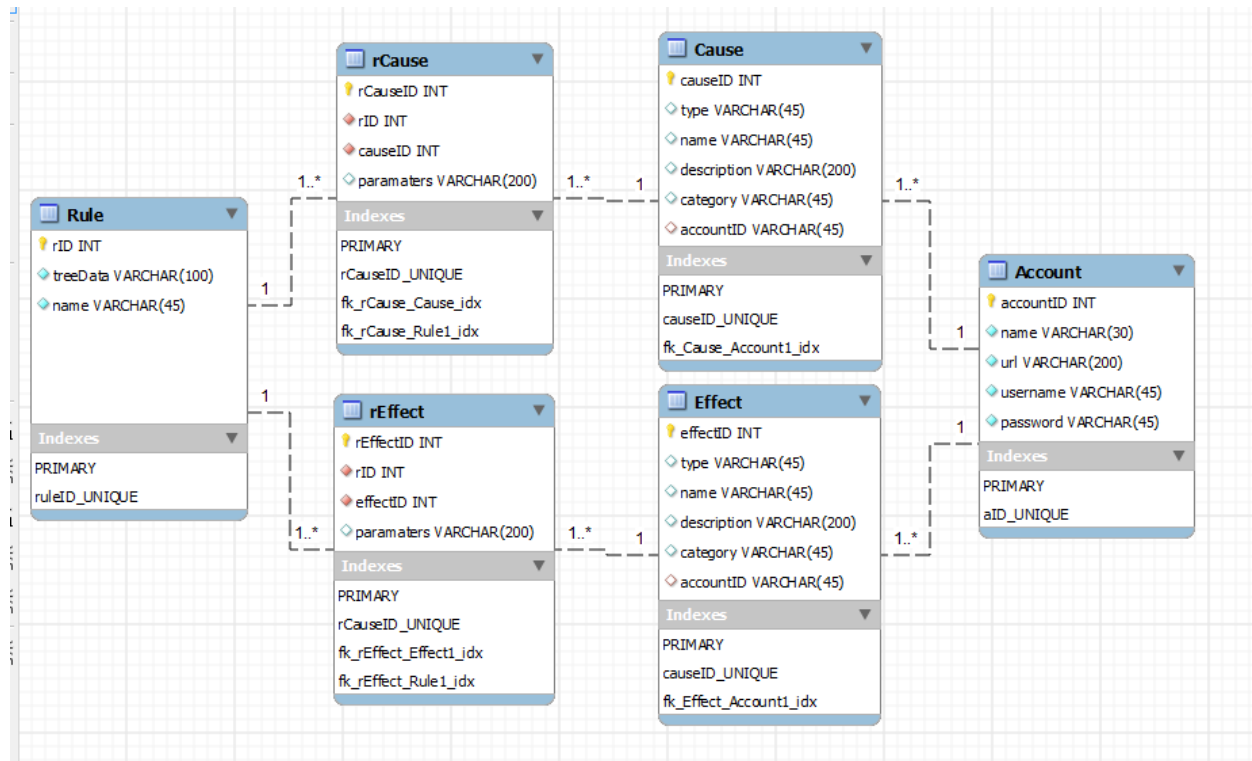
While developing the structure of the android application we decided that we needed a full database to hold all of the relevant information that we would need. This database would keep track of all causes, effects, rules, and account information so we needed it not only to be large but secure as well. We inquired with Samsung to see if they had any preferences on what we should use to accomplish this task and were presented with TouchDB (<https://github.com/couchbaselabs/TouchDB-Android>). We started playing around with it to try to get used to it but found the documentation for it to be subpar. After this setback we decided to go with SQLite since we needed to keep the database lightweight to keep our app from taking up too much memory on the phone as we remember it has to be constantly running in the background.

So getting started with this, we had to set up a structure for how we wanted the databases to look. This included how we would want our pulls to appear from the database. So we started with creating the objects that we would be using in our program to decide how many objects we would need to keep it lightweight while balancing that with how many variables were in each object. We also had to keep track of links between the objects to finally pull a full rule out of the database if we were to need that much information. I, personally, got to learn a lot about databases since I had never even been introduced to any sort of database language before this. There was a bit of a steep learning curve between not only learning SQL, but then to learn the limitations that were present with using SQLite as well as making the database so large with a number of tables and foreign keys.

Getting a little more specific into how the database was made, we have to first start by looking at how the objects were made that would basically be held by the tables. The cause object is one of the base objects so it needed to hold a lot of information including: cause id, type, name, description, category, and account id (which we will be implementing later). The cause id would of course be the primary key to keep track of which cause you would be talking about in the rCause table. It was at this time the auto-increment functionality became our friend as we discovered that it was already implemented in the tables and assigning a primary key actually renames the column to whatever you called it (led to many headaches and head scratching). The cause table would be kept separate from the rCause table since it needs to be an overarching list while the specifics that would needed to be implemented in each rule would be handled by the rCause table. The effect tables on the other side of the chart were basically mirror images of the cause tables.

When looking at the rules objects you realize that you need the string to keep track of how the rules interact (and/or) as well as a name to display and of course a primary key. A rule id is kept

as well in the rCause/rEffect tables. Finally, the account table is not fully implemented yet in our application, but soon will be. The breakdown of the account (as we know right now) is that it must contain a (service) name, URL, username, and password. All of these are essential in logging into certain other services such as Facebook, Twitter, etc. Below, you will see a display of how the tables talk to each other:



After that whole database is set up, you then need to be able to do multiple functions to accommodate for the database handler which is what the program needed to be able to get their information. Some of the most obvious ones were to insert causes and effects into the table. Whenever you start up the application for the first time you will be inserting all of the causes and effects from either a text file (to possibly be implemented later) or just being read in hard coded (current implementation). Now for inserting the rule you also need to be inserting the specific rCause and rEffect information since these are important parts of the rule. The application will be set to remember all of the rules from previous uses, because who really wants to have to input the rules every time they start up their phone.

Once you are done inserting all of the information for the application, there were the actual parts of the database that the other parts of the program needed to be able to access. The first ones I was told to implement were to return all cause, effect, and rule objects (separately). This was pretty easy as you just use an SQL statement to pull all of the information out of a certain

table and put the information into an ArrayList. After those methods got done, we started to get a little more complicated in how we wanted to call from the database. The first method was to return all cause objects where the category of that cause is equal to a given category. So we passed a string in, used it in a SQL query to create a result table that you would pull from to get the wanted information. So of course the same thing also needed to be done for effects and rules. The rules one was actually more complicated since you had to go into the rCause and rEffect tables with external columns from other tables using joins (since there is no category column in the rules table or the rCause/rEffect).

So taking a break from the more complicated calls I had to return a full cause object given a cause id. After taking a break, it was back to complicated calls with receiving all rule objects where the type of that rule is equal to the given type, making sure that the effect list and cause tree are null, but the cause tree string is still there. This was for in the early parts of the application when you are pulling up some information quickly to display to the user while it doesn't take up much processing power. And finally the last method I've had to make was to return an effect list given the rule id, which just made sure that we had all the method calls for the other parts of the program.

Action Executer

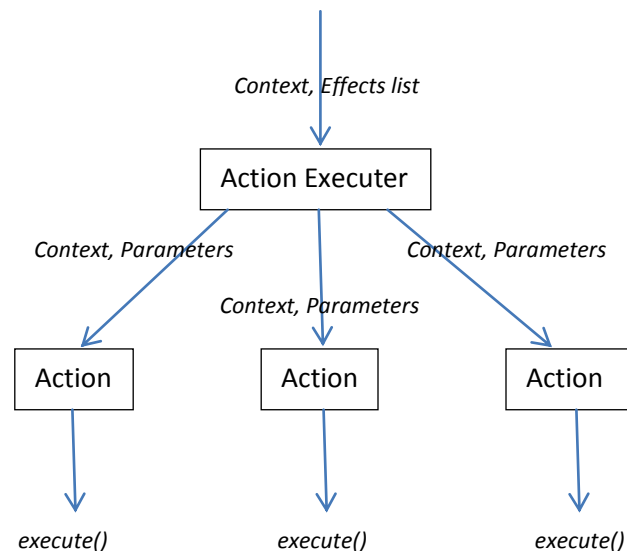
The main function of the action executer is the carrying out of effects of a positively evaluated rule. The action executer is what carries out the actual actions that are associated with a rule.

The action executer object is created by the rules engine each time a rule has been evaluated at true. Many action executers can exist at one time if multiple rules have been triggered. The rules engine passes the context of the current state of the application as well as an array list of the rEffect objects from the rule. Once the object has been constructed, the action executer will loop through all the effects in the array list. In each loop, the “type” of the effect is enumerated and evaluated as the argument in a switch statement. This is more efficient than using many if statements. The individual case that is evaluated would create a new action object and execute the action.

Each rule type has a unique action class that inherits from AsyncTask. AsyncTasks are Android’s native method of threading and allows for the parallelization of actions being executed. Each thread will be made of an action that can be executed. For instance, a separate thread is created to execute a “toast” (Android’s most simple notification to the user). The individual action classes are passed both the context of the action and the parameters needed to execute that action in the form of a string. The constructor of the action will parse the parameters to make sense of one long string before executing the action. Each action type has a unique method for parsing the parameters because different rules require different parameters (some rules do not have any parameters). The “execute” function of the action class contains source code that will carry out the individual task, using both the application context and the parsed parameters.

The specifications of the action executer require it to be very lightweight and adaptable. The possibility of several action executers existing in memory and executing effects at one time is very probable. The action executer must also be able to execute any type of action. It must be universal enough to create an action object of any type needed and pass the necessary application context and parameters.

Thread Diagram



Each action object stemming from the action executor is executed on its own thread.

Risk Assessment and Management Plan

Risk	Type	Probability	Consequence	Strategy
The application does not adapt to all targeted Android devices.	Technology	High probability	Tolerable	We will program our application keeping in mind that our application must be dynamic and usable across a variety of devices, regardless of hardware, API level, and screen resolution.
The application leaks personal information to hackers.	Technology	Moderate probability	Serious	We will be using the highest available levels of security to encrypt user data and account information.
Team members are frequently sick or unavailable to work on the project or meet.	People	High probability	Tolerable	We will communicate frequently about our schoolwork and upcoming assignments when assigning project tasks. We will set times and importance for our meetings, so that all team members will be available and accountable for each other.
Changes to requirements that require major design rework are proposed.	Requirements	Low probability	Serious	We will frequently be communicating with our client to be sure they are satisfied with our progress and results. Any changes to requirements will be up for discussion and source code will be distributed and dynamic enough to adapt.

Risk	Type	Probability	Consequence	Strategy
The time required to develop is underestimated.	Estimation	Moderate probability	Serious	Team members will regularly meet to discuss current status and estimate deadlines. As different tasks and components are completed, team members will take on new tasks to complete.
The source code is inefficient and the application is slow when executed.	Technology	Moderate probability	Insignificant	Our team will program efficiently and constantly review each other's source code to look for areas of improvement.
The test devices are not sufficient for testing.	Tools	Low probability	Insignificant	Members of the team have personal devices to test on. New devices may be purchased for the purpose of testing.

Agile Development Schedule

Agile Development has been quite an interesting trip so far. Agile development is essentially the waterfall cycle in short sprints called “iterations.” The entire requirements, design, implementation, testing/verification, and release process is included in a single sprint of agile development. The requirements are broken into development needs called “User Stories.” User stories are one of the most unique parts about agile development. When a new iteration is started, user stories are chosen and ranked in a game called “Playing Poker” in the field. This allows developers to come to coordinate which tasks will be achieved during any given iteration by group decision and analysis of user story difficulty. The group ideally takes on an appropriate amount of user stories and begins the iteration with these user stories in mind. Each user story is very similar in concept to the change request forms idea from other development cycles. Each user story is its own change to the last release that has been decided upon, and ranked by the group. Typically (and ironically), there is little or no documentation for agile development projects.

One feature of agile development is the SCRUM architecture. The typical agile development team meets every day in meetings called “Standup Meetings.” The daily standup meeting is critical to the success of the team. Since agile is such a fast moving development structure, these meetings are used to both check on the progress of development and discuss development issues, clarify user stories, clarify client’s needs, coordinate the group’s efforts, and unite the team members.

The SCRUM master (our team leader, Keith), is in charge of running the scrum meetings for our team. Normally, the SCRUM master is not a developer. However, all 4 members of our team are developers and are responsible for code. This has led to an interesting group dynamic. To spread out duties from the typical agile development team, the acceptance of user stories is handled by Matthew and William. The note taking for all of our meetings internally and with the client are handled by Tomin. Our team meets 4 times a week, at 12:30PM on Tuesday and Thursday, and at 3:00PM Monday and Friday.

Our iterations are 2 weeks in length. Iterations in agile development vary from 2-4 weeks but each team decides on a length and sticks to it. Our iterations start on Saturday and go through the Friday that is 13 days later. The finished code with all the iteration's user stories is due by the Wednesday on day 12 of the iteration. Between day 12 and day 13, all acceptance testing is handled. If a user story is not accepted, the story is either patch fixed (if possible) or pushed into the next iteration's user stories as a high priority user story. Between day 13 and day 14, the team members submit documentation on each user story they completed and on day 14, we plan the next iteration's user stories and rank them using Playing Poker. The next iteration then starts immediately on the following Saturday.

Our iteration plan is to have 5 iterations in fall and 6 in spring. This gives us a total of 11 iterations, so 22 weeks of formal development. During our first iteration, we defined most of our user stories and developed a mock GUI so we could begin testing functionality. During our second iteration, we were advised by our client to spend additional time to design the rules engine structure. Our release for iteration 2 was just documentation and a few cleaned up features on the mock GUI. This structure is the foundation of the functionality of our application and is complex to parallelize and integrate. To account for this, iterations three and four were designed to be the first and second halves of the rules engine. With school ramping up, iteration 5 was designed to merge, clean, test, and optimize the application before the PDR. Since we are doing agile development, this presentation will also include a demo of our iteration 5 prototype release. It just so happens (thanks to signing up early) that iteration 5 ends on the same day as the PDR.

We planned for a lighter iteration during Thanksgiving (to prepare for the PDR), and made the iteration over spring break a 3 week iteration to account for members out of town. The great thing about agile development is that all of the user stories that have not been assigned to an iteration live in the backlog. The backlog currently consists of a few functional user stories and the entire list of appendices at the end of this document. The benefit to this is that winter break doesn't have a formal structure encapsulating it (to respect the holidays and difficulty of meeting for SCRUM standups), but instead the backlog will be updated with priorities so team members can work on user stories during the break and clear up that list. Our intention is to set a minimum amount of stories for each person to complete over break and let everyone develop when they have time. So yes, we are have a plan and are working over winter break.

After winter break, our plan can change depending on the iteration. Our plan is to spend the sixth iteration to integrate the account tables that already exist in our database and NFC if these have not been already completed over winter break. Iteration six or seven could contain the drag and drop interface specified by our client. Iteration seven to iteration 10 will consist of lots of user testing, usability features, and tons of new rule types. Iteration 11 will be planned to work on the FDR demo and documentation. Agile development is dynamic so with client feedback, testing feedback, and the clearing of our backlog, we will have plenty to do.

Budget

Proposal Budget

Project Name: Cause & Effect

Period #1: 9/22/2012 - 4/20/2013

Description	Rate	Hours	Subtotal	Total
Software Engineer	\$25.00			
Keith Adler		550	\$13,750.00	\$13,750.00
Matthew Brannick		440	\$11,000.00	\$11,000.00
Tomin Kozhimala		440	\$11,000.00	\$11,000.00
William Vennes		440	\$11,000.00	\$11,000.00
Employee Salaries Subtotal		1,870	\$46,750.00	\$46,750.00
Emp Ben, FT Exempt Staff (42%)			\$19,635.00	\$19,635.00
Emp Ben, Non-Exempt Staff (42%)			\$0.00	\$0.00
Employee Benefits Subtotal			\$19,635.00	\$19,635.00
Employee Salaries and Benefits Subtotal		1,870		\$66,385.00
Equipment	Cost	Quantity	Sub-Total	Total
Samsung Nexus S (on Loan)	\$0.00	2	\$0.00	\$0.00
Samsung Epic 4G (on Loan)	\$0.00	1	\$0.00	\$0.00
Samsung Galaxy Tab 10.1 (on Loan)	\$0.00	1	\$0.00	\$0.00

Equipment Costs Subtotal		4	\$0.00	\$0.00
Software	Cost	Quantity	Sub-Total	Total
Eclipse IDE	\$0.00	4	\$0.00	\$0.00
Android SDK	\$0.00	4	\$0.00	\$0.00
Bitbucket Repository	\$0.00	4	\$0.00	\$0.00
Mercurial Source Control	\$0.00	4	\$0.00	\$0.00
Android Open Source Project Code	\$0.00	4	\$0.00	\$0.00
Software Costs Subtotal			\$0.00	\$0.00
Total Equipment and Software Costs				\$0.00
Contingency Costs (20%)				\$0.00
Project Costs Total				\$66,385.00

Our current proposed budget is \$66,385. The budget above represents our total projected budget for the duration of the Cause & Effect Project. These figures for employee salary and benefits were estimated using an hourly wage of \$25.00 per hour. This is the approximate starting salary for a Software Engineer at Samsung Mobile (Glassdoor). The group leader is estimated to work 25 hours per week, while the remaining group members are estimated to work 20 hours per week. Employee benefits were estimated to be close to 42% of the rate of wages and salaries (Bureau of Labor Statistics).

Our equipment consists of four devices given on loan from Samsung Mobile to assist us in testing our application. These include two Samsung Nexus S developer phones, a Samsung Epic 4G, and a Samsung Galaxy Tab 10.1.

In order to construct a software environment that was both robust and economical, our team took advantage of many free and open source software programs. Android Developers provide the necessary Android Software Development Kit (SDK). We are currently using Eclipse as our Integrated Development Environment (IDE) with the Android Development Tools (ADT) plugin, which provides us with a powerful integrated environment to develop out application, as well as build an app UI, debug, and export app packages (APK) to distribute and test. Our environment also includes Bitbucket hosting service and our team was able to take advantage of using our .edu email accounts to register with Bitbucket and create a free private repository to host our source code. We chose to use the free Mercurial distributed source control to manage our software code at our client's recommendation and our own interest. Finally, the Android Open Source Project provides us with a free developer's website with many resources for reference about the operating system (developer.android.com).

Source Code Repository

Our Repository can be found at:

<https://Talador12@bitbucket.org/Talador12/ceandroid>

Feel free to request read privileges and/or follow our project.

References

[Glassdoor]

http://www.glassdoor.com/Salary/Samsung-Group-Software-Engineer-Salaries-E3363_D_KO14,31.htm

The screenshot displays the Glassdoor website interface for Samsung Group Software Engineer Salaries. The page includes a navigation bar with links for Home, Jobs, Companies & Reviews, Salaries, and Interviews. A search bar is present with filters for Salaries, Job Title or Company, and Location. The main content area is titled "Samsung Group Software Engineer Salary" and includes a "Post a Salary" button. Below the title, there is a section for "Overview" with links to Salaries, Reviews, Interviews, Photos, Jobs, and Connections. The "Salaries" section shows "14 Salaries" and a "Back to all Samsung Group Salaries" link. A "Hide Filters" button is also visible. The "Find by Years Of Experience" and "Find by Location" sections allow users to filter results. The "Salaries in USD" table shows the average salary of \$85,393 and a range from \$52k to \$105k. The table also lists bonuses and other pay components. A key at the bottom explains the symbols used in the table. A sidebar on the right provides a "Samsung Group Overview" with details about the company, including its website, HQ, industry, size, and competitors. It also features a "Featured Jobs" section with links to various job openings. At the bottom, there is a "Samsung Group Connections" section with "20 Inside Connections" and a "Jobs by Title" section listing various engineering roles.

Samsung Group Software Engineer Salary

Web www.samsung.com | HQ Seoul, South Korea [Post a Salary](#)

[Overview](#) | [Salaries](#) | [Reviews](#) | [Interviews](#) | [Photos](#) | [Jobs](#) | [Connections](#)

Updated Nov 19, 2012 – Salaries posted anonymously by employees and employers.

14 Salaries | [Back to all Samsung Group Salaries](#)

▼ Hide Filters

Find by Years Of Experience Find by Location

- Any Experience - United States – All Cities [Apply Filters](#) [Clear](#)

Salaries in USD	Average	\$10k	\$50k	\$90k
Total Pay — Salary / Bonus / Other (14)	\$85,393		\$52k	\$105k
Salary (14)	\$80,643		\$52k	\$105k
▼ Bonuses (2)	\$9,500	\$9,000	\$10k	
Cash Bonus (2)	\$9,500	\$9,000	\$10k	
Stock Bonus (0)	n/a		No Reports	
Profit Sharing (0)	n/a		No Reports	
▶ Other Pay (0)	n/a		No Reports	

Salaries in USD [USD](#)

Key: = Range = Anonymous Range [?] = Average

Note: Pay may vary based on location, years of experience, and/or other factors.

See salary by office:

- [Samsung Group Software Engineer Salary in San Jose](#)
- [Samsung Group Software Engineer Salary in Bangalore](#)
- [Samsung Group Software Engineer Salary in Delhi](#)
- [Samsung Group Software Engineer Salary in Suwon](#)

Or looking beyond Samsung Group?

Check out the latest [Software Engineer Jobs](#) or see [Software Engineer Salaries](#) at other companies.

You might also be interested in: [Salaries](#), [Software Engineer Salaries](#), [Samsung Group Salaries](#)

Samsung Group Overview [Edit](#)

Web www.samsung.com
HQ Seoul, South Korea
Industry Semiconductor Manufacturing
Size 300+ Employees, \$119B+ Revenue
Competitors [LG Group](#), [SK Group](#), [Hyundai Corporation](#)

Work here? Learn about our [Enhanced Employer Profile](#)

Featured Jobs [Personalize](#)

Search for jobs by title or keyword

[Computer Hardware / Software Sales](#)
GHA Technologies – Dallas, TX
From GHA Technologies

[Administrative Assistant](#)
Autodesk – Plano, TX
From Autodesk – 10 days ago

[Staffing Specialist](#)
Texas Instruments – Dallas, TX
From Texas Instruments – 19 days ago

[Sr Support Engineer](#)
TIBCO Software – Arlington, TX
From TIBCO Software

Hiring? [Post a Job](#)

Samsung Group Connections

20 Inside Connections
Find more connections by adding friends.

Jobs by Title

[Software Engineer Jobs](#)
[Staff Engineer Jobs](#)
[Senior Software Engineer Jobs](#)
[Senior Engineer Jobs](#)
[Engineer Jobs](#)
[Design Engineer Jobs](#)
[Senior Design Engineer Jobs](#)

[Bureau of Labor Statistics]

<http://www.bls.gov/news.release/ecec.nr0.htm>

The screenshot shows a web browser window with the address bar displaying www.bls.gov/news.release/ecec.nr0.htm. The page header includes the United States Department of Labor and Bureau of Labor Statistics logos, along with navigation links like "A to Z Index", "FAQs", "About BLS", "Contact Us", "Subscribe to E-mail Updates", "Follow Us", "What's New", "Release Calendar", and "Site Map". A search bar is also present. The main content area is titled "Economic News Release" and "Employer Costs for Employee Compensation news release text". It includes technical information such as the release date (September 11, 2012), contact details for NCSinfo@bls.gov and PressOffice@bls.gov, and the release number USDL-12-1830. The body of the release discusses the average cost per hour worked for employee compensation in June 2012, broken down by industry and benefit type.

Employer Costs for E x

www.bls.gov/news.release/ecec.nr0.htm

UNITED STATES DEPARTMENT OF LABOR

BUREAU OF LABOR STATISTICS

Home Subject Areas Databases & Tools Publications Economic Releases Beta

Economic News Release

Employer Costs for Employee Compensation news release text

FOR RELEASE 10:00 A.M. (EST) TUESDAY, SEPTEMBER 11, 2012 USDL-12-1830

Technical information:
(202) 691-6199 NCSinfo@bls.gov <http://www.bls.gov/ect>
Media contact:
(202) 691-5902 PressOffice@bls.gov

EMPLOYER COSTS FOR EMPLOYEE COMPENSATION - JUNE 2012

Private industry employers spent an average of \$28.80 per hour worked for employee compensation in June 2012, the U.S. Bureau of Labor Statistics reported today. Wages and salaries averaged \$20.27 per hour worked and accounted for 70.4 percent of these costs, while benefits averaged \$8.52 and accounted for the remaining 29.6 percent. Total compensation costs for state and local government workers averaged \$41.10 per hour worked in June 2012. Total compensation costs for civilian workers, which include private industry and state and local government workers, averaged \$30.61 per hour worked in June 2012.

Employer Costs for Employee Compensation (ECEC), a product of the National Compensation Survey, measures employer costs for wages, salaries, and employee benefits for nonfarm private and state and local government workers.

Retirement and savings benefit costs in private industry

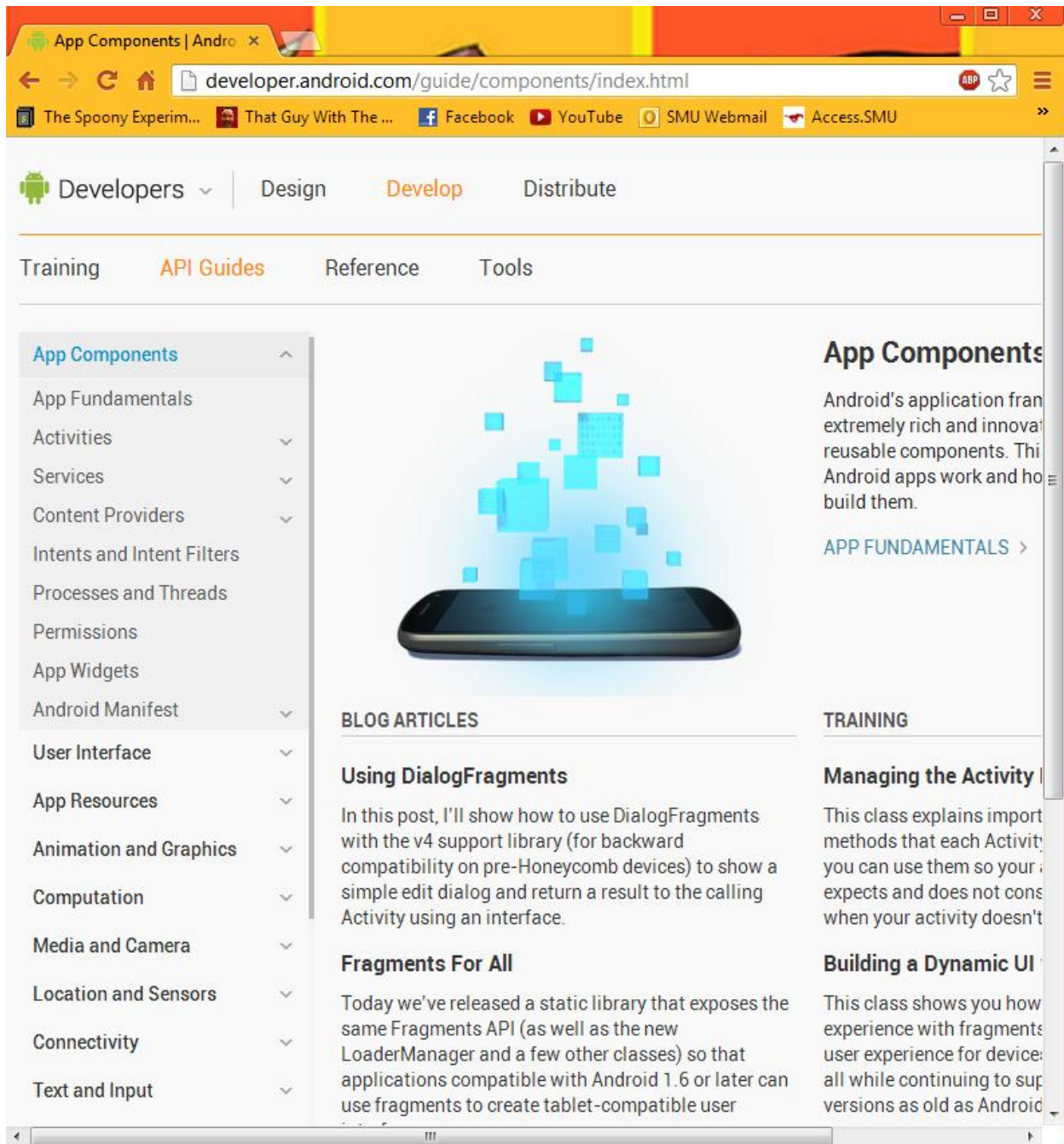
In June 2012, average costs in private industry for retirement and savings benefits were \$1.02 per hour worked, or 3.5 percent of total compensation. Private industry retirement and savings benefit costs for management, professional, and related occupations were \$2.07 per hour, or 4.0 percent of total compensation in June 2012. Costs were lowest among service occupations, 22 cents or 1.6 percent of total compensation. (See table 5.) Included in retirement and savings benefit costs were employer costs for defined benefit and defined contribution plans. Employer costs for retirement and savings plans are affected by several factors, including the percentage of employees that participate in the plans offered by their employer. (The National Compensation Survey produces comprehensive data on the percentage of workers with access to and that participate in retirement plans. Data for March 2012 are available at <http://www.bls.gov/news.release/pdf/ebs2.pdf>).

In June 2012, the average cost per hour worked for defined benefit plans—retirement plans that typically specify a benefit based on age, years of service, and earnings—was 43 cents (1.5 percent of total compensation). The average cost for defined contribution plans—retirement plans usually based on employer contributions to individual employee accounts—was 59 cents (2.1 percent of total compensation). (See table 5.)

Retirement and savings benefit costs were higher, both in amount and as a proportion of total compensation, for union workers (\$2.89 and 7.4 percent of total compensation) than for nonunion workers (83 cents and 3.0 percent of total compensation). Defined benefit plan costs were significantly higher for union workers (\$2.11 and 5.4 percent of total compensation) than for nonunion workers (26 cents and 0.9 percent of total compensation). Defined contribution costs for union workers were higher

[Android Developers]

<http://developer.android.com/guide/components/index.html>



The screenshot shows a web browser window displaying the Android Developers website. The address bar shows the URL <http://developer.android.com/guide/components/index.html>. The page features a navigation bar with tabs for Design, Develop, and Distribute. Below this, there are links for Training, API Guides, Reference, and Tools. The main content area is titled "App Components" and includes a large illustration of a smartphone with blue cubes floating above it. To the left of the main content is a sidebar menu listing various topics under "App Components", including App Fundamentals, Activities, Services, Content Providers, Intents and Intent Filters, Processes and Threads, Permissions, App Widgets, Android Manifest, User Interface, App Resources, Animation and Graphics, Computation, Media and Camera, Location and Sensors, Connectivity, and Text and Input. The main content area also includes sections for "BLOG ARTICLES" (with links to "Using DialogFragments" and "Fragments For All") and "TRAINING" (with links to "Managing the Activity" and "Building a Dynamic UI").

App Components | Android

developer.android.com/guide/components/index.html

The Spoony Experim... That Guy With The ... Facebook YouTube SMU Webmail Access.SMU

Developers Design Develop Distribute

Training API Guides Reference Tools

App Components

Android's application framework provides an extremely rich and innovative set of reusable components. This guide explains how Android apps work and how to build them.

[APP FUNDAMENTALS >](#)

BLOG ARTICLES

Using DialogFragments

In this post, I'll show how to use DialogFragments with the v4 support library (for backward compatibility on pre-Honeycomb devices) to show a simple edit dialog and return a result to the calling Activity using an interface.

Fragments For All

Today we've released a static library that exposes the same Fragments API (as well as the new LoaderManager and a few other classes) so that applications compatible with Android 1.6 or later can use fragments to create tablet-compatible user

TRAINING

Managing the Activity

This class explains important methods that each Activity you can use them so your app expects and does not crash when your activity doesn't

Building a Dynamic UI

This class shows you how to experience with fragments to user experience for device all while continuing to support versions as old as Android

[TouchDB-Android]

<https://github.com/couchbaselabs/TouchDB-Android>

The screenshot shows the GitHub repository page for **couchbaselabs / TouchDB-Android**. The page is viewed in a web browser with the address bar showing <https://github.com/couchbaselabs/TouchDB-Android>. The repository is public and has 1 pull request and 34 issues. The description states: "CouchDB-compatible mobile database; Android version — [Read more](#)".

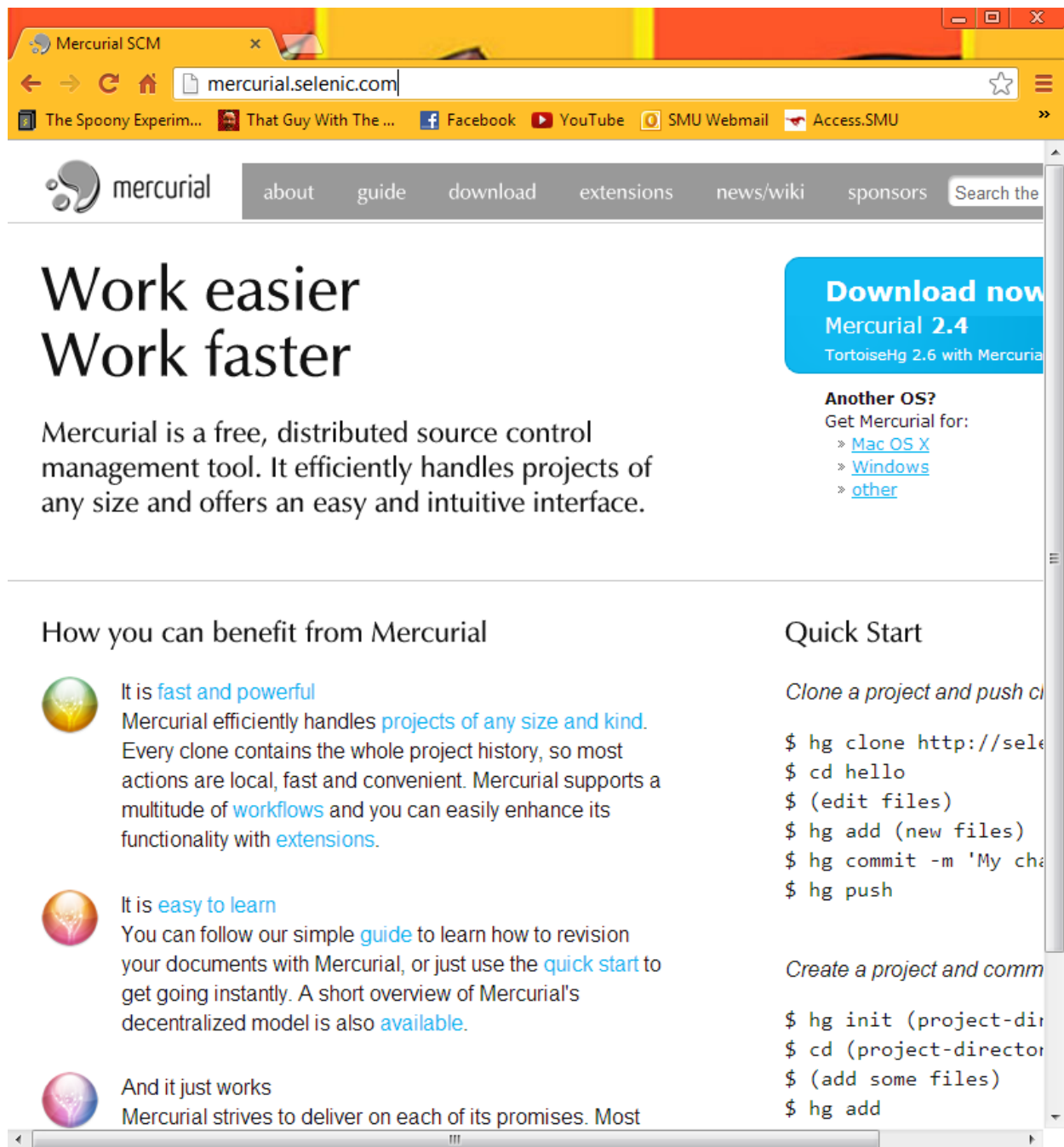
Clone options are available: Clone in Windows, ZIP, HTTP, SSH, and Git Read-Only. The current branch is **master**. The file list shows:

File	Time	Description
TouchDB-Android-Ekorp	4 months ago	added eclipse project specific settings to force 1.6 compiler [Marty]
TouchDB-Android-JavaScript	3 months ago	Fix handing over json to rhino by serializing it from the native java...
TouchDB-Android-Listener	4 months ago	added eclipse project specific settings to force 1.6 compiler [Marty]
TouchDB-Android-TestApp	3 months ago	Fix handing over json to rhino by serializing it from the native java...
TouchDB-Android	4 months ago	added eclipse project specific settings to force 1.6 compiler [Marty]
README.md	7 months ago	updated readme to point ot wiki [mschoch]

The README.md file is partially visible at the bottom of the screenshot, showing the title **TouchDB-Android**.

[Mercurial Source Control]

<http://mercurial.selenic.com/>



The screenshot shows the Mercurial website homepage in a web browser. The browser's address bar displays 'mercurial.selenic.com'. The website features a navigation bar with links for 'about', 'guide', 'download', 'extensions', 'news/wiki', and 'sponsors'. The main heading reads 'Work easier Work faster'. Below this, a paragraph describes Mercurial as a free, distributed source control management tool. To the right, a blue box promotes 'Download now Mercurial 2.4' and 'TortoiseHg 2.6 with Mercurial'. Further down, a section titled 'How you can benefit from Mercurial' lists three points: 'It is fast and powerful', 'It is easy to learn', and 'And it just works'. To the right of this section is a 'Quick Start' section with terminal commands for cloning a project and creating a new project.

Mercurial SCM

mercurial.selenic.com

The Spoony Experim... That Guy With The ... Facebook YouTube SMU Webmail Access.SMU

mercurial

about guide download extensions news/wiki sponsors Search the




Work easier Work faster

Mercurial is a free, distributed source control management tool. It efficiently handles projects of any size and offers an easy and intuitive interface.

Download now
Mercurial 2.4
TortoiseHg 2.6 with Mercurial

Another OS?
Get Mercurial for:
» [Mac OS X](#)
» [Windows](#)
» [other](#)

How you can benefit from Mercurial

-  It is **fast and powerful**
Mercurial efficiently handles **projects of any size and kind**. Every clone contains the whole project history, so most actions are **local**, fast and convenient. Mercurial supports a multitude of **workflows** and you can easily enhance its functionality with **extensions**.
-  It is **easy to learn**
You can follow our simple **guide** to learn how to revision your documents with Mercurial, or just use the **quick start** to get going instantly. A short overview of Mercurial's decentralized model is also **available**.
-  And it just works
Mercurial strives to deliver on each of its promises. Most

Quick Start

Clone a project and push changes

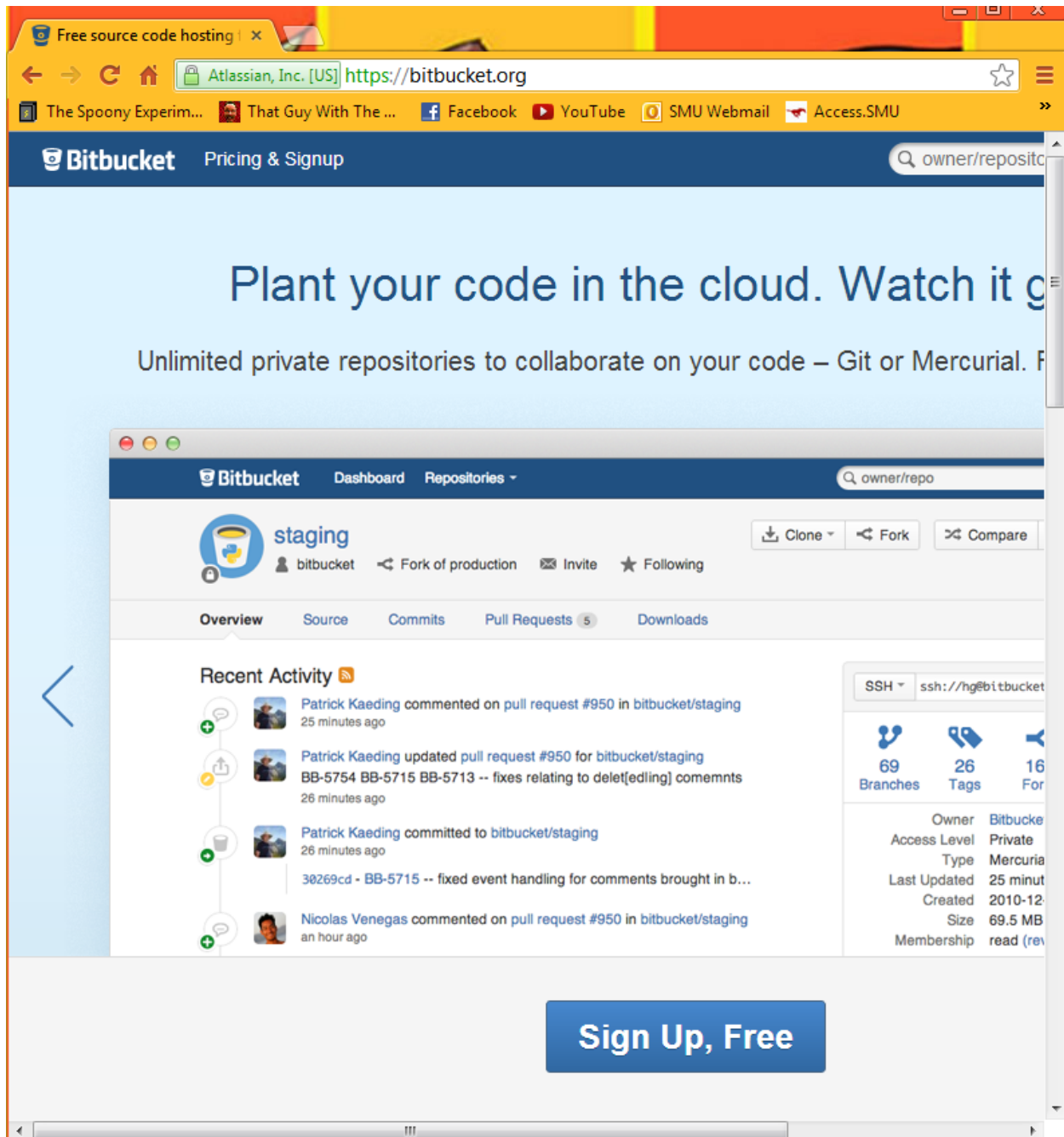
```
$ hg clone http://selenic.org/mercurial
$ cd hello
$ (edit files)
$ hg add (new files)
$ hg commit -m 'My changes'
$ hg push
```

Create a project and commit

```
$ hg init (project-directory)
$ cd (project-directory)
$ (add some files)
$ hg add
```

[Bitbucket]

<https://bitbucket.org/>



Here is the list of all of the user stories that are included in the low priority list. This list is enormous but it includes all of the rule types implemented by our competitors and much more.

Phone/Physical Rule User Stories

- As a developer, I want to be able to access location using the mobile service, mobile data, Wi-Fi, and GPS, so that I can regulate a relationship between battery use and accuracy.
- As a developer, I want to be able to open a given location in a maps application, so that the user may view location data on their phone.
- As a developer, I want to be able to open a given location in a navigation application, so that the user may access navigation features on their phone.
- As a developer, I want to be able to integrate Bluetooth functionality, so that users can use rules and sharing that utilizes the phone's Bluetooth capabilities.
- As a developer, I want phone call based rules, so that the user can take communicate effectively using our application.
- As a developer, I want sound based rules, so that the user can get clear notification of events, even when not looking at the device.
- As a developer, I want the app to feature media message based rules, so that the user can handle complex message types.
- As a developer, I want the application to access the display features, so that the display can be turned on or off as desired by the user.
- As a developer, I want the application to recognize when it is docked, so that the user can create rules based on docking status.
- As a developer, I want the application to be able to send fax messages, so that our users can use this functionality.
- As a developer, I want the application to recognize when headphones are plugged in, so that the user can create rules based on headphones.
- As a developer, I want the application to recognize when an external display is plugged in, so that the user can create rules that interact with the external display.
- As a developer, I want to include rules associated with the calendar, so that users can interact with their schedules using our application.

- As a developer, I want to take advantage of Android (and especially Android 4.1)'s voice commands, so that the user can interact with rules using their voice.
- As a developer, I want the application to be able to open or terminate other applications, so that the user can optimize application flow on their phone.
- As a developer, I want the application to be able to access the phone's memory and storage, so that the user can interact with the phone's data hardware.
- As a developer, I want the application to interact with the audio player, so that the user can create rules based on audio.
- As a developer, I want the user to be able to change phone preferences with our application, so that the operating system is editable by rules.
- As a developer, I want rules that affect the battery usage, so that the user can optimize battery life using our application.
- As a developer, I want to access the LEDs, because that would be a really cool feature that all users could enjoy.
- As a developer, I want to access the phone's alarm system, so that the user can edit their alarm settings using rules.
- As a developer, I want to be able to toggle airplane mode, so that the user can access this feature using rules.
- As a developer, I want to be able to change the wallpaper using our app, so that users can have dynamically changing or updating wallpapers.
- As a developer, I want to be able to manage files on the device use our app, so that users can manage their file structure using rules.
- As a developer, I want to access the camera, so that users can create rules that can utilize the camera's features (new photo, flash, etc).
- As a developer, I want rules that can access the current time zone, so that the user can utilize different time zones as they change.
- As a developer, I want rules that access email, so that the user can manage their email using our rules engine.
- As a developer, I want rules that can access the web, so that the user can access web content using our rule system.
- As a developer, I want the access the state of the USB tethering, so that the user can build rules from this.
- As a developer, I want to be able to build rules for a wireless hotspot, so that the user can utilize their over expensive data plan or rooted phone.
- As a developer, I want to be able to access the phone's Radio, so that there can be rules that use the radio.

- As a developer, I want to be able to access different profiles, so that the user can enable or disable sets of rules at any time (even with rules).
- As a developer, I want to integrate rule types for the lock screen, so that the user can create rules using the lock functionality.
- As a developer, I want rules that access the motion sensors/accelerometers, so that the user can create rules based on the motion of the phone.
- As a developer, I want rules that access the gyroscope, so that the user can create rules based on the device's orientation.
- As a developer, I want to access when the phone is shutdown or restarted, so that the user can create rules based on these actions/events.
- As a developer, I want to know when the first boots up, so the user can respond using a set of rules.
- As a developer, I want to access the phone's current state, so that the user can use that state in rules.
- As a developer, I want language based rules, so that the application can be used internationally and for translation.
- As a developer, I want debug based rules, so that the user can create sets of rules for debugging their applications (including this one).
- As a developer, I want to be able to interact with a printer, so that the user can create rules that interact with the printer.
- As a developer, I want to allow the user to directly input JavaScript, so that a technical user can use the rules engine to its fullest.
- As a developer, I want to access the weather status, so that the user can use rules based on this.
- As a developer, I want to access emergency messages, so that the user is quickly informed of hazards.
- As a developer, I want to access the news, so that the user can create rules based on this.
- As a developer, I want to create rules based on traffic, so that the user can create rules for this.
- As a developer, I want to access Stocks, so that the user can integrate with any stock service.
- As a developer, I want to access Banking, so that the user can manage their banking using rules.
- As a developer, I want to store VPN settings, so that the user can manage VPN connections.
- As a developer, I want to react to Data roaming, so the user can include this state in rules.

- As a developer, I want to access Voicemail, so that the user can manage voicemail using rules.
 - As a developer, I want to know when something is downloaded and uploaded, so the user can create rules for this.
 - As a developer, I want to be able to access movie times, so that the user can create rules for this.
-

Account Sync Rule User Stories

- As a developer, I want to access Dropbox, so that the user can manage their files stored on the cloud.
- As a developer, I want to access Evernote, so that the user can manage their notes using this rules engine.
- As a developer, I want to access Flickr, so that the user can manage photos using this application.
- As a developer, I want to access Craigslist, so that the user can interact with this service using our application.
- As a developer, I want to access Blogger, so that the user can interact with blogs using rules.
- As a developer, I want to access RSS feeds, so that users can integrate their rules with the feeds.
- As a developer, I want to access Gmail, so that the user can use Google email as an email source for rules.
- As a developer, I want to access eBay, so that the user can interact with this service using our rules.
- As a developer, I want to access Instagram, so that the user can manage their sepia stained photos.
- As a developer, I want to access Twitter, so that the user can tweet using our rule system.
- As a developer, I want to access Steam, so that users can interact with their gaming communities using this application.
- As a developer, I want to access Foursquare, so that users can interact with this service and sync the user's location using this service.
- As a developer, I want to access Google+, so that users can interact with social media.
- As a developer, I want to access Pinterest, so users can manage their boards using our app.
- As a developer, I want to access Path, so users can stay connected with family & close friends.
- As a developer, I want to access last.fm, so users can manage music using our application.

- As a developer, I want to access LinkedIn, so users can manage their professional identities.
- As a developer, I want to access Google and Yahoo Finance, so users can manage their financial interests using our app.
- As a developer, I want to access YouTube, so they can manage their video accounts.
- As a developer, I want to access Word Press, so they can manage their blogs.
- As a developer, I want to access Tumblr, so that users can manage their Tumblr content.
- As a developer, I want to access StumbleUpon, so that users can stumble using our application.
- As a developer, I want to access Yahoo, so that users can manage their Yahoo accounts.
- As a developer, I want to access Windows live, so that users can integrate with their Window's live features.
- As a developer, I want to access ESPN, so that users can manage their sports.
- As a developer, I want to access GroupMe, so that users can manage group text messaging using our app.
- As a developer, I want to access Samsung Kies, so that users can update their hardware using this application.
- As a developer, I want to access Amazon, so that users access this service.
- As a developer, I want to access Newegg, so that users can interact with this online store.
- As a developer, I want to access Google play store, so that the user can create rules involving the Android marketplace.
- As a developer, I want to access Google voice, so that users can integrate with voice accounts.
- As a developer, I want to access AIM, so that users can interact with this messaging service.
- As a developer, I want to access Google Talk, so that users can interact with this service.
- As a developer, I want to access Skype, so that users can text/audio/video with other users.
- As a developer, I want to access Netflix, so that users can build rules for this type.
- As a developer, I want to access Hulu Plus, so that users can build rules for this type.
- As a developer, I want to access Kindles, so that users can interact with these accounts and devices.
- As a developer, I want to access Wikipedia, so that users can be well informed.
- As a developer, I want to access App.net, so that other developers can create rules based on this wonderful service.
- As a developer, I want to access Bit.ly, so that users can save, share, & discover links.
- As a developer, I want to access Buffer, so that users can have a smart way to share on social media.
- As a developer, I want to access Delicious.com, so that users can access cooking and recipe features.
- As a developer, I want to access Diigo, so that users can collect and organize using our app.

- As a developer, I want users to be able to ordering food (Example: Pizza Hut), so that users can automate this process.
- As a developer, I want to access Readability, so users can read comfortably if they prefer to.
- As a developer, I want to access Storify, so that users can create custom scenarios using social media.
- As a developer, I want to access Spotify, so that users can manage their media libraries.
- As a developer, I want to access Rdio, so that users can listen to music whenever they want.
- As a developer, I want to access Svvply, so that users can shop a wide variety of products.
- As a developer, I want to access Vimeo, so that users can have multiple options for viewing media.
- As a developer, I want to access ZooTool, so that users can bookmark and share content with ease.
- As a developer, I want to access Myspace, so that users can access music and video based social media.
- As a developer, I want to access Friendster, so that users can access this social media source.
- As a developer, I want to access 4chan, so that our application can address a wide user base.
- As a developer, I want to access Classmates.com, so that users can communicate with old classmates using our app.
- As a developer, I want to access Deviant art, so that users can manage their art portfolios using our application.
- As a developer, I want to access CrunchyRoll, so that users can be notified of new release dates.
- As a developer, I want to access Blogster, so that users can manage their blogs.
- As a developer, I want to access IGN, so that users can use this game database functionality.
- As a developer, I want to access GameFly, so that users can order games using our application.
- As a developer, I want to access IMDB, so that users can use this movie database functionality.
- As a developer, I want to access Grindr, so that our application is usable for many audiences.
- As a developer, I want to access Live journal, so that users can manage their personal publishing.
- As a developer, I want to access Xanga, so that users can manage their own blogs.
- As a developer, I want to access Yelp, so that users can access local reviews and information.
- As a developer, I want to access Bebo, so that users can use this social media outlet.

- As a developer, I want to access Badoo, so that users can communicate with local people.
- As a developer, I want to access Stackoverflow, so that users can communicate on these developer forums.
- As a developer, I want to access XDA developers, so that developers can communicate on this medium.
- As a developer, I want to access the MSDN, so that developers can use this means of communication.
- As a developer, I want to access Digg, so that users can stay up to date on current events.
- As a developer, I want to access Orkut, so that users can social network and discuss using Google's engine.
- As a developer, I want to access PlayStation Network, so that we can accompany PlayStation users.
- As a developer, I want to access MSN, so that users can use access this functionality.
- As a developer, I want to access Bing, so that users can choose which search engine they wish to use.
- As a developer, I want to access AOL, so that we can address all user types.
- As a developer, I want to access Ask.com, so that users can have their questions answered.
- As a developer, I want to access Google's basic search, so that users can use this search engine.
- As a developer, I want to access CNN, so that users can receive updates about news.
- As a developer, I want to access FOX, so that users can receive updates about entertainment.
- As a developer, I want to access ABC, so that users can receive updates about news.
- As a developer, I want to access BBC, so that users can integrate with an international news source.
- As a developer, I want to access Wolfram Alpha, so that users can do complex math using our application.
- As a developer, I want to access Whitepages, so that users can quickly search this database.
- As a developer, I want to access Yellowpages, so that users can quickly search this database.
- As a developer, I want to access RememberTheMilk.com, so that users can update and receive reminder notifications.
- As a developer, I want to access iTunes, so that users can integrate their phone with this music application.
- As a developer, I want to access Apple accounts, so that users can integrate their Android device with Apple hardware.
- As a developer, I want to access PayPal, so that users can manage purchases using this application.

- As a developer, I want to access GoDaddy, so that users can manage website domains using our application.
- As a developer, I want to access 4shared, so that users can manage files on this service.
- As a developer, I want to access The Weather Channel, so that users can have more accurate weather on demand.
- As a developer, I want to access SourceForge, so that developers can manage their projects from their phone.
- As a developer, I want to access GitHub, so that developers can manage their code repositories.
- As a developer, I want to access Bitbucket, so that developers can manage their code repositories.
- As a developer, I want to access Walmart, so that users can be aware of updates and make purchases.
- As a developer, I want to access Target, so that users can utilize this department store's API.
- As a developer, I want to access Fedex, so that users can monitor shipping status.
- As a developer, I want to access UPS, so that users can monitor shipping status.
- As a developer, I want to access Dictionary.com, so that users can define words.
- As a developer, I want to access Urban dictionary, so that users can understand local colloquialisms.
- As a developer, I want to access Thesaurus.com, so that users can use this service.
- As a developer, I want to access Google Documents, so that users can manage their documents from our app.
- As a developer, I want to access Google Drive, so that users can manage their files on this service.
- As a developer, I want to access GroupOn, so that users can get deals from local businesses.
- As a developer, I want to access Living Social, so that users can receive deals on local services.
- As a developer, I want to access Urban Spoon, so that users can have information on cooking at hand.
- As a developer, I want to access Rhapsody, so that users can use this music player.
- As a developer, I want to access Imugr, so that users can manage their image files.
- As a developer, I want to access Heroku, so that users can manage their projects using this service.
- As a developer, I want to access Speedtest.net, so that users can check their current connection status with detail.
- As a developer, I want to access Photobucket, so that users can manage their image and video files.
- As a developer, I want to access Pandora, so that users can use this music service.
- As a developer, I want to access CNet, so that users can view technical reviews on products.

- As a developer, I want to access Battle.net, so that these users can manage their accounts.
- As a developer, I want to access Grooveshark, so that users can use this music service.
- As a developer, I want to access Monster, so that users can find the job that's right for them.
- As a developer, I want to access the NFL, so that users can receive updates from this feed.
- As a developer, I want to access Expedia, so that users can manage travel details.
- As a developer, I want to access Hotels.com, so that users can manage travel details.
- As a developer, I want to access Kayak, so that users can manage travel details.
- As a developer, I want to access Travepedia, so that users can manage travel details.
- As a developer, I want to access Engadget, so that users can be up to date on technology news.
- As a developer, I want to access Slashdot, so that users can be up to date on technology news.
- As a developer, I want to access Fandango, so that users can manage movie tickets.
- As a developer, I want to access Ticketmaster, so that users can book and manage tickets.
- As a developer, I want to access StubHub, so that users can book and manage tickets.
- As a developer, I want to access Match.com, so that users can communicate with local individuals.
- As a developer, I want to access Rotten Tomatoes, so that users can use this movie review service.
- As a developer, I want to access Twilio, so that users can manage VoIP, Voice, and Text applications over the web.
- As a developer, I want to access Metro Lyrics, so that users can search for song lyrics.
- As a developer, I want to access Shazam, so that users can recognize songs using this service.
- As a developer, I want to access the NBA, so that users can receive updates from this service.
- As a developer, I want to access Dailymotion, so that users can upload, share, and embed their own videos.
- As a developer, I want to access Chacha, so that they can have their questions answered.
- As a developer, I want to access Best Buy, so that users can access this API service.
- As a developer, I want to access Quora, so that users can connect and share content using this service.
- As a developer, I want to access Voxeo, so that users can manage their cloud computing profiles.
- As a developer, I want to access "Let me Google that for you," so that users can share knowledge with peers.
- As a developer, I want to access About.com, so that users can solve their needs.
- As a developer, I want to access eHow, so they can use this service for information.

- As a developer, I want to access Mustang TRAK, so that users can manage their SMU job profiles.
- As a developer, I want to access to Access.smu.edu, so that users can manage their student accounts using our application.
- As a developer, I want to access Blackboard, so that users can manage their classes using this rules engine.