

Overview

Android Performance Evaluator is a service that allows developers how efficiently their application is running on the android OS.

This product is not complete. All of the wording will need to be revised several times before it is finalized. The graphics and layout of the screens is shown here are not the final product. The actual look and feel will be developed over time with different input from product managers and programmers.

Installation Instructions

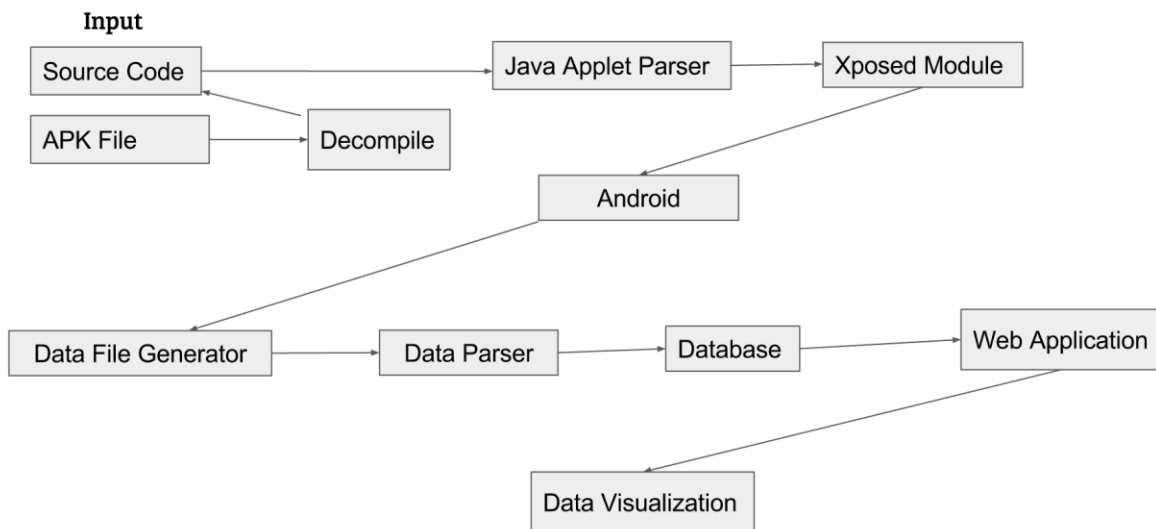
A brief high level “how-to-use” guide for the current build.

Last revised 3/30/2016

1. Start the Java app. It will open the file explorer. Select the source code you wish to parse or the apk you wish to analyze. Select the entire package if passing in source code - DO NOT navigate to the /src folder or any other subdirectory. The parser needs to the entire project structure to generate the parsed output.
2. After selecting the code/apk to analyze, select Parse. Parser will parse through your uploaded code (and decompile it if necessary). It will then produce an output file that ModuleBuilder will analyze in order to generate your Xposed module.
3. When the status on the file explorer states that the module has been built, close the file explorer. The module file (moduleFile.java) will be ready for installation.
4. Install moduleFile.java on your Android phone or emulator.
 - a. Currently, Android Studio installs the module for us. Eventually, the installation will be automated.
5. Once the module is installed, restart your phone or emulator to complete the installation. You are now ready to analyze the app.
6. Open a terminal and navigate to where your adb resides (likely android-sdks/platform-tools).
7. Run “\$./adb logcat -s Xposed”. You should see the module hook onto the package you are analyzing and produce the method start and end times.
8. Play with the app to produce more output for the logcat. It will record the method times related to the buttons you select.
9. Copy the output to data.txt and upload to dataBaseListener. dataBaseListener will parse this output and upload it to the database.
10. Log onto <http://localhost/softDev/index.php>. Username is username, password is password.
11. Select Projects and then the app you wish to analyze. Charts will appear with your uploaded data. Select Make CSV or Make PDF to get the stats as raw data.

Data Flow

Below is the data flow of the program and should give a general sense of how to use the Android Performance Evaluator. Details of each component will be described in detail later.



Technologies Used

Below are the technologies currently used in Android Performance Evaluator.



Parser Components

Before anything else, once launching the Java Applet Parser, the user is prompted to log in as shown below in Figure 0. The user will log in and in order to authenticate the user, it will call the membership database class in order to verify. The passwords are hashed with a custom MD5 function and stored in the database, so the Java Applet Parser will also hash the password with the custom MD5 function.

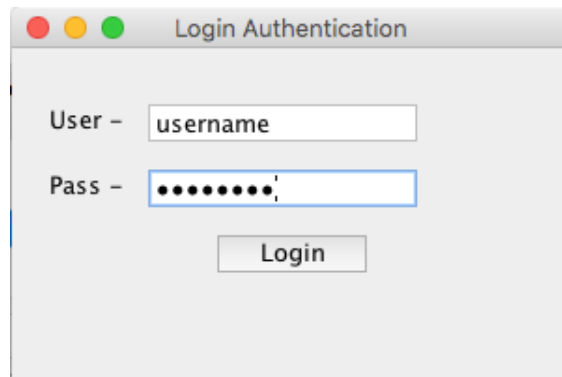


Fig 0) File Explorer in choosing the apk or source code

Once a user has been authenticated, the Java Applet Parser takes in either source code or an apk as input. If the input is an apk, it will decompile said apk and parse its source code. An apk or source code directory can be selected through the file explorer after hitting the “select file” button as shown in Figure 1.

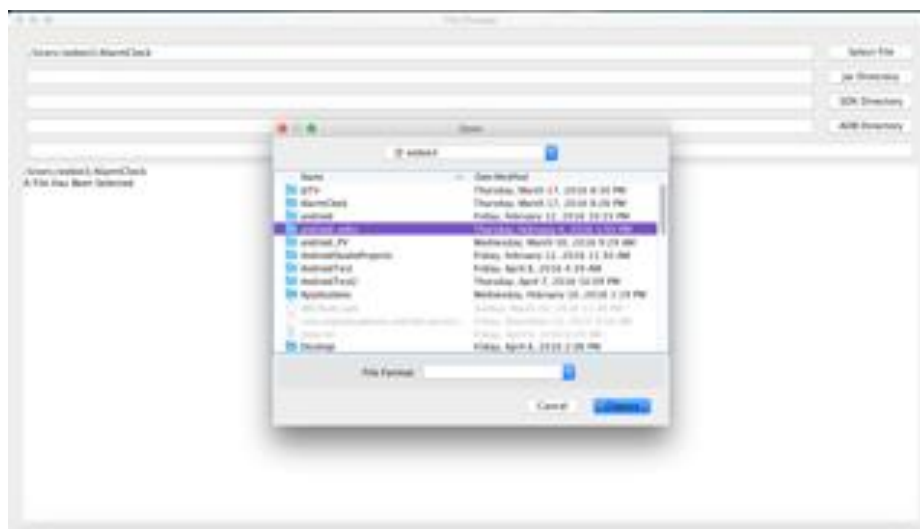


Fig 1) File Explorer in choosing the apk or source code

Once the desired file has been selected, a SDK version and any external jar files that are required. Then it will begin the automatic parsing process once pushing “parse” as shown in Figure 2. Java Applet Parser will pull the package name, methods, parent classes, parameters, etc. for each method and will output that into a source file to be parsed by ModuleBuilder.

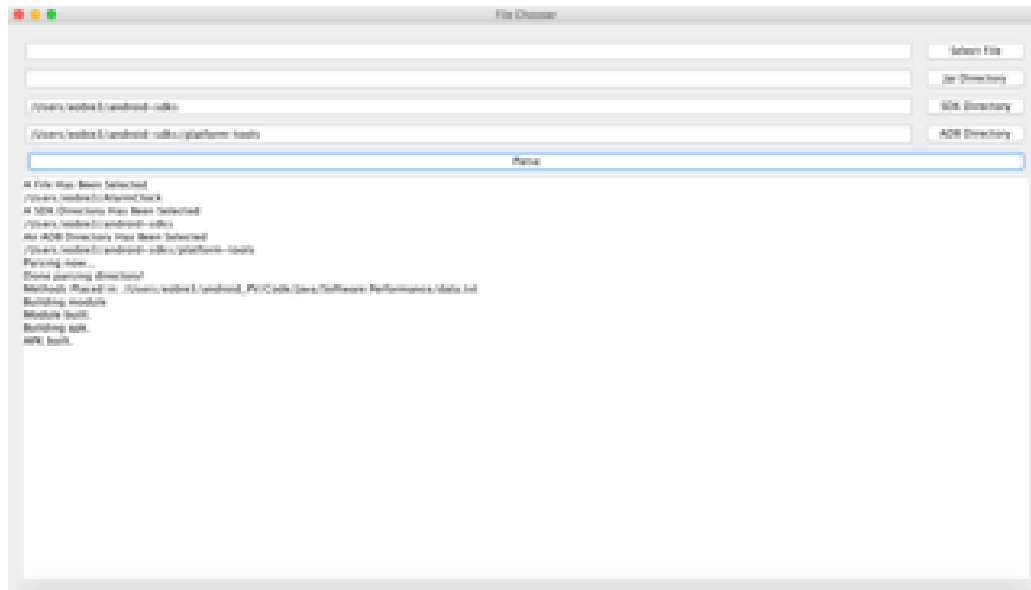


Fig 2) Parsing demonstration with output from the terminal

Android Components

The ModuleBuilder from the Java Applet Parser will be then generate an Xposed module as shown in Figure 3.

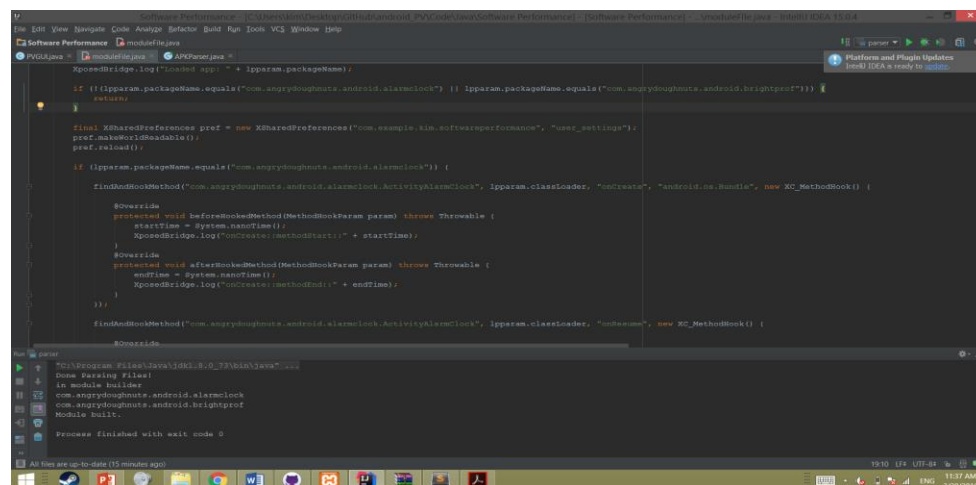


Figure 3) Xposed Module code

The Xposed module hooks methods based on the app you are analyzing. When a hooked method is executed, Xposed will record the start and exit time of the method in order to calculate the full execution time. Thankfully, the user will not have to touch the code, as it is automatically installed in an existing machine. Then the app will be analyzed to collect data. Once the user is done with collecting data, there will be a button on the java applet where the user can press to “Send Data”. The user then will be able to decide whether to cancel and scrap the trace data, or push it onto the database. These two options can be seen as seen in Figure 3.5. The user just needs to input the application name and their username again to get it set and working.

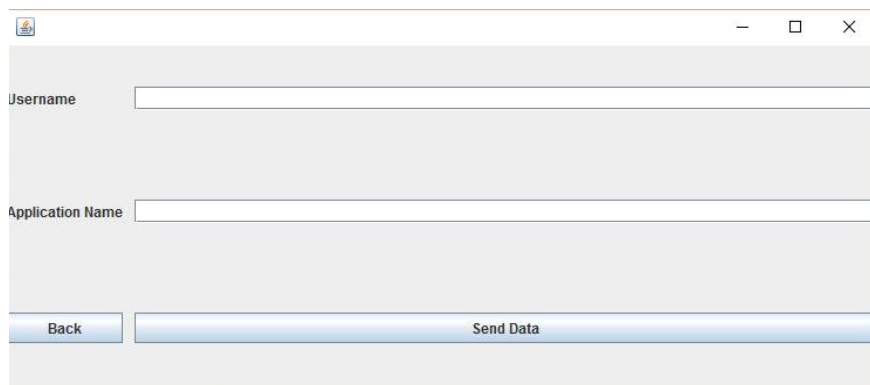


Figure 3) Data Sending window

From this point on, the user must log into the webapp in order to access and visualize the data.

Web Application Components

The dataBaseListener, upon upload of this output, will add it to our database to be represented on the website, and organize it to the required tables. From this point, the user will move onto the website application. First, the user will be prompted to login to authenticate the user’s application and traces as shown in Figure 4.

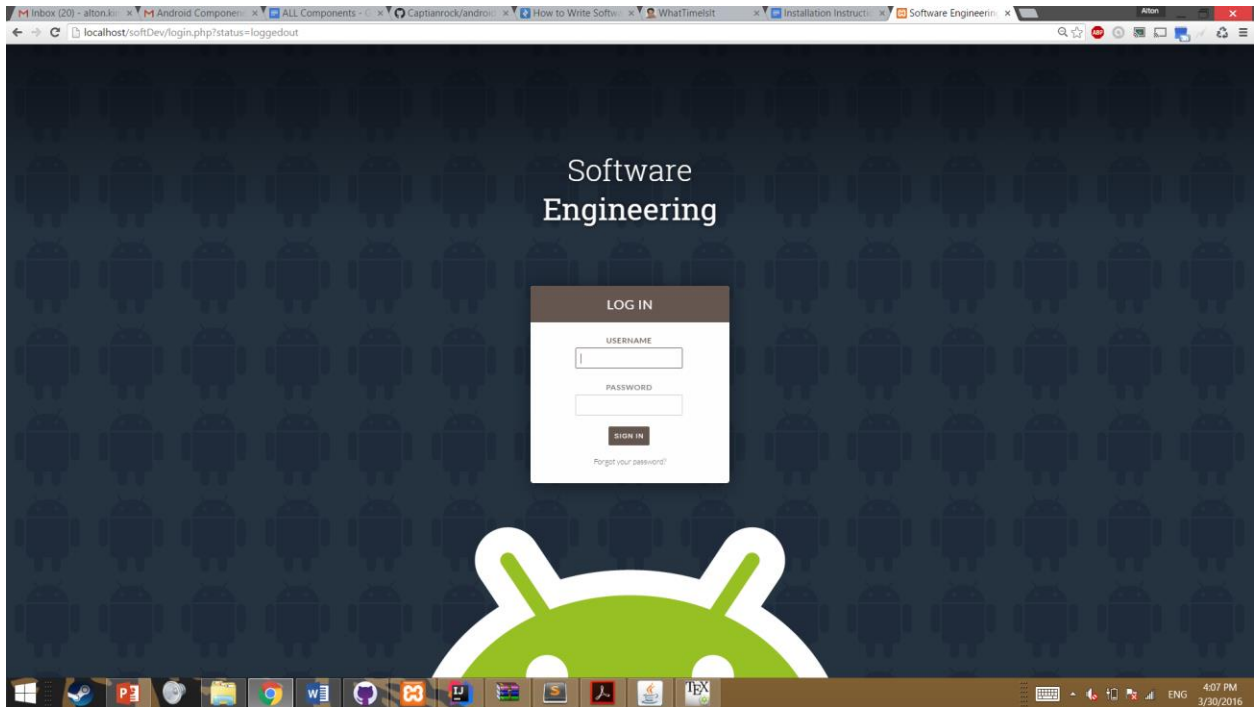


Figure 4) Login Screen for the User

This will guide the user to the homepage screen where they have the option to navigate through their projects that they have current traces for as shown in Figure 5.

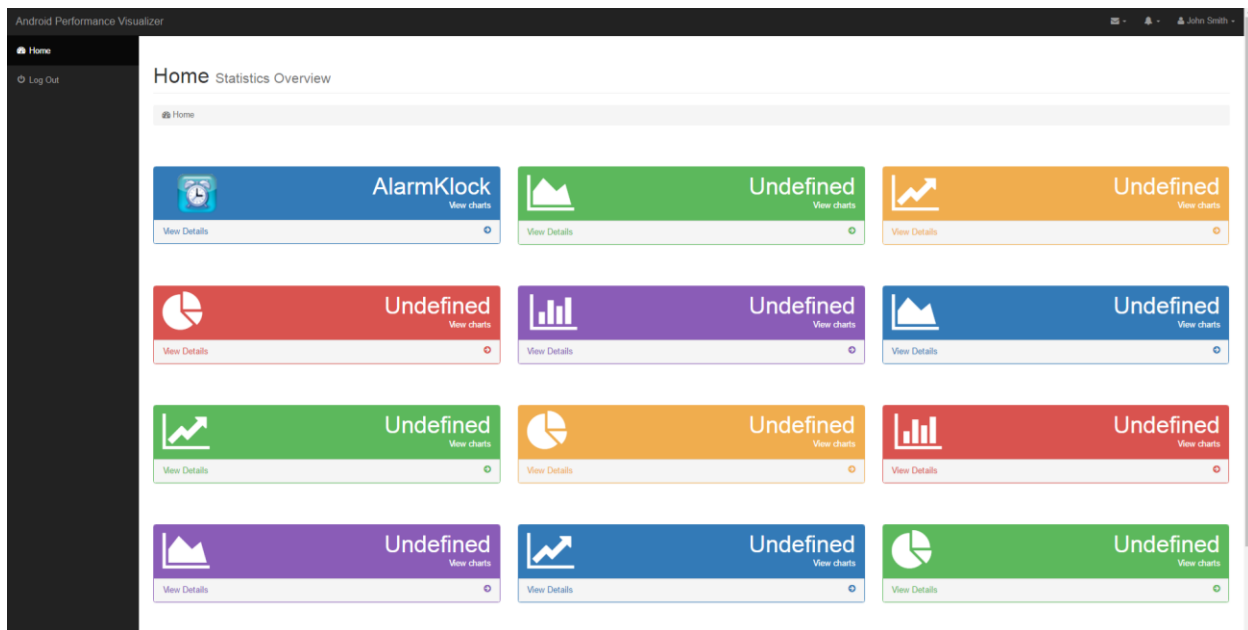


Figure 5) Home page with options for the user.

When choosing the option, the user will be able to select from multiple traces that they may have on the database. Once choosing a specific trace, the user will be shown the data for their trace as shown in the highcharts below.

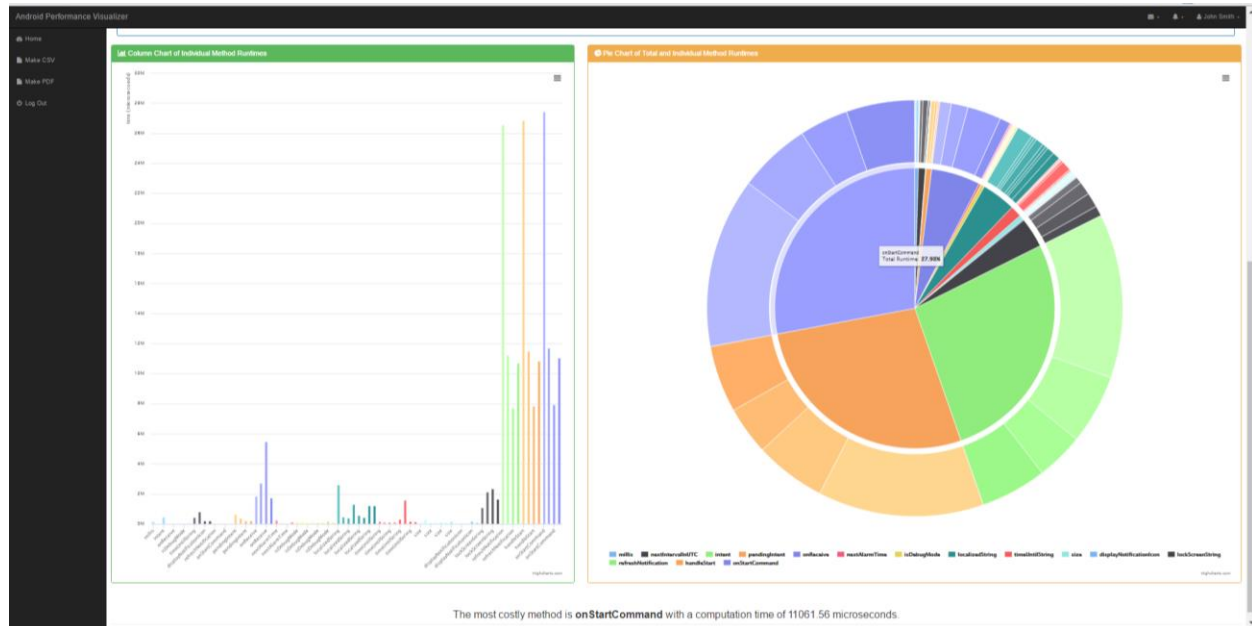
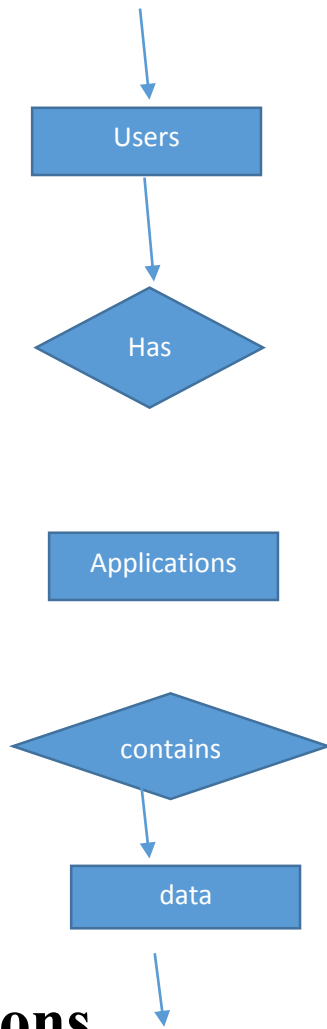


Figure 6) Example of Data Visualization

Database Model

Here is the current planned layout of the database. Each user logged in will be have their own set of applications and traces that are accessible to only them.





Entites:

Membership(username, id, password)

Users(username, application_name)

Applications(application name,
traceld)

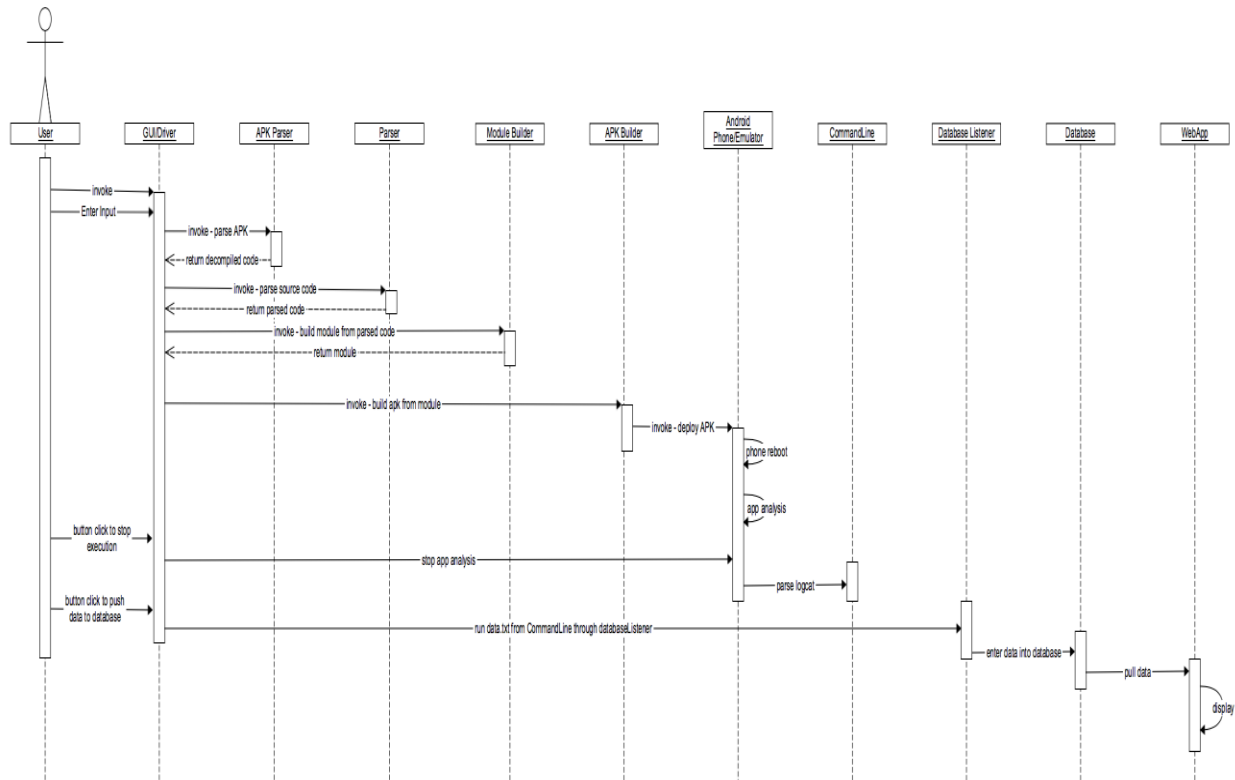
Data(traceld, methodName,
startTime,endTime)

Database Definitions

Entity Name	Entity Description	Column Name	Column Description	Data Type	Length	Primary Key	Nullable	Unique
Application	An application is the app the user is analyzing.	id	Auto-incrementing primary key.	integer	11	TRUE	FALSE	TRUE
		username	User's username.	varchar	40	FALSE	FALSE	FALSE
		application	The application a given user has analyzed.	varchar	40	FALSE	TRUE	FALSE
Data	Data contains information from the app analysis.	id	Auto-incrementing primary key.	integer	11	TRUE	FALSE	TRUE
		traceld	Indicates the trace number, the user, and the application	varchar	40	FALSE	FALSE	FALSE

		methodName	The name of the method.	varchar	40	FALSE	FALSE	FALSE
		timeStart	The time at which the method began execution.	varchar	40	FALSE	FALSE	FALSE
		timeEnd	The time at which the method ended execution.	varchar	40	FALSE	FALSE	FALSE
Traces	A trace is an analysis of a given app. There can be multiple traces for each app with respect to a given user.	id	Auto-incrementing primary key.	integer	11	TRUE	FALSE	TRUE
		username	The user owning the trace.	varchar	40	FALSE	FALSE	FALSE
		application	The application the user analyzed.	varchar	40	FALSE	TRUE	FALSE
		traceId	unique id for the a trace	varchar	40	FALSE	TRUE	FALSE
Users	A user has analyzed one or more apps one or more times and signs in to view the data collected from his/her analysis.	id	Auto-incrementing primary key.	integer	11	TRUE	FALSE	FALSE
		username	User's username.	varchar	15	FALSE	FALSE	FALSE
		password	User's encrypted password.	varchar	41	FALSE	FALSE	FALSE

Sequence Diagram

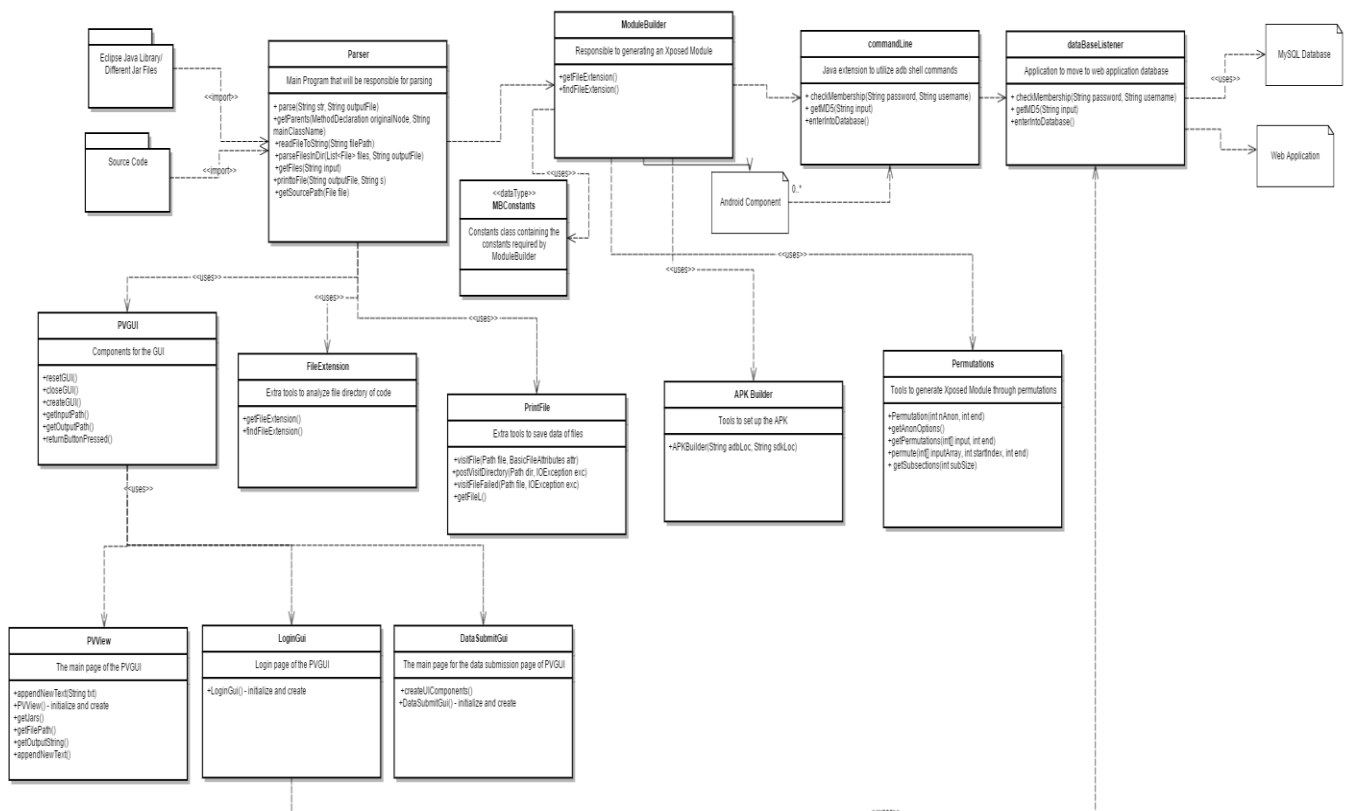


Above is the sequence diagram that shows a more intricate flow of data for our application. It is a very one-directional flow of data, as there is no need for the android phone to communicate back with the java applet until the data needs to be collected. It can be seen that much of the application can be split into separate independent components that rely on each other's output, but are not specifically dependent on the components themselves. Merely zoom in to the png to see more details or go to <https://www.gliffy.com/go/publish/image/10396331/L.png> for a more detailed image.

Class Diagrams

Java Component

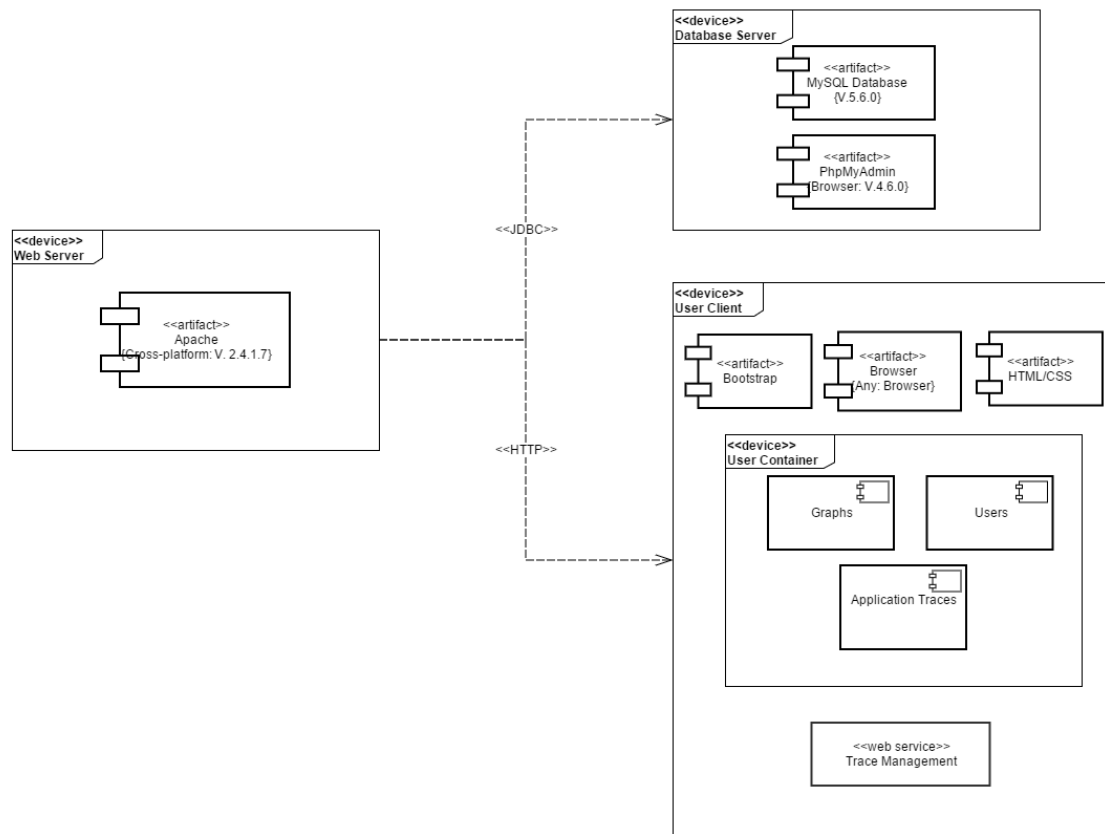
The Android component of the class diagram is also integrated as it is very closely tied to the flow of data of the java parser, since the java parser is the one to automatically upload it into the emulator. The main driving program is the Parser class, with it using and extending multiple other classes to successfully parse through source code, generate a valid Xposed module, and automatically deploying it onto the android phone connected. This includes the GUI and the user input.



We show the following connection between all the classes in the class diagram for the java applet as shown above. Merely zoom in to the png to see more details or go to <https://www.gliffy.com/go/publish/image/10384529/L.png> for a more detailed image.

Web Application Deployment Diagram

Although a class diagram may show the more intricate details of the web application, there is too many libraries and files that it would be too difficult to put on this document. However, the deployment diagram for the web application should cover the basic structure we followed in organizing the web application. A larger version is available at <https://www.gliffy.com/go/publish/image/10398685/L.png>



As seen through the deployment diagram, much of the web application is focused on the user client side. This allows us to create more personalized webpages for each different users. This may be moved to the server side if this causes any slowdowns on the client's side.