Team Anyalgorithms - Firefox Focus Architecture

Deliverable 1

Ahmed Sirajuddin

Jon An

Kia Naderi

Ricky Chen

Robert Peng

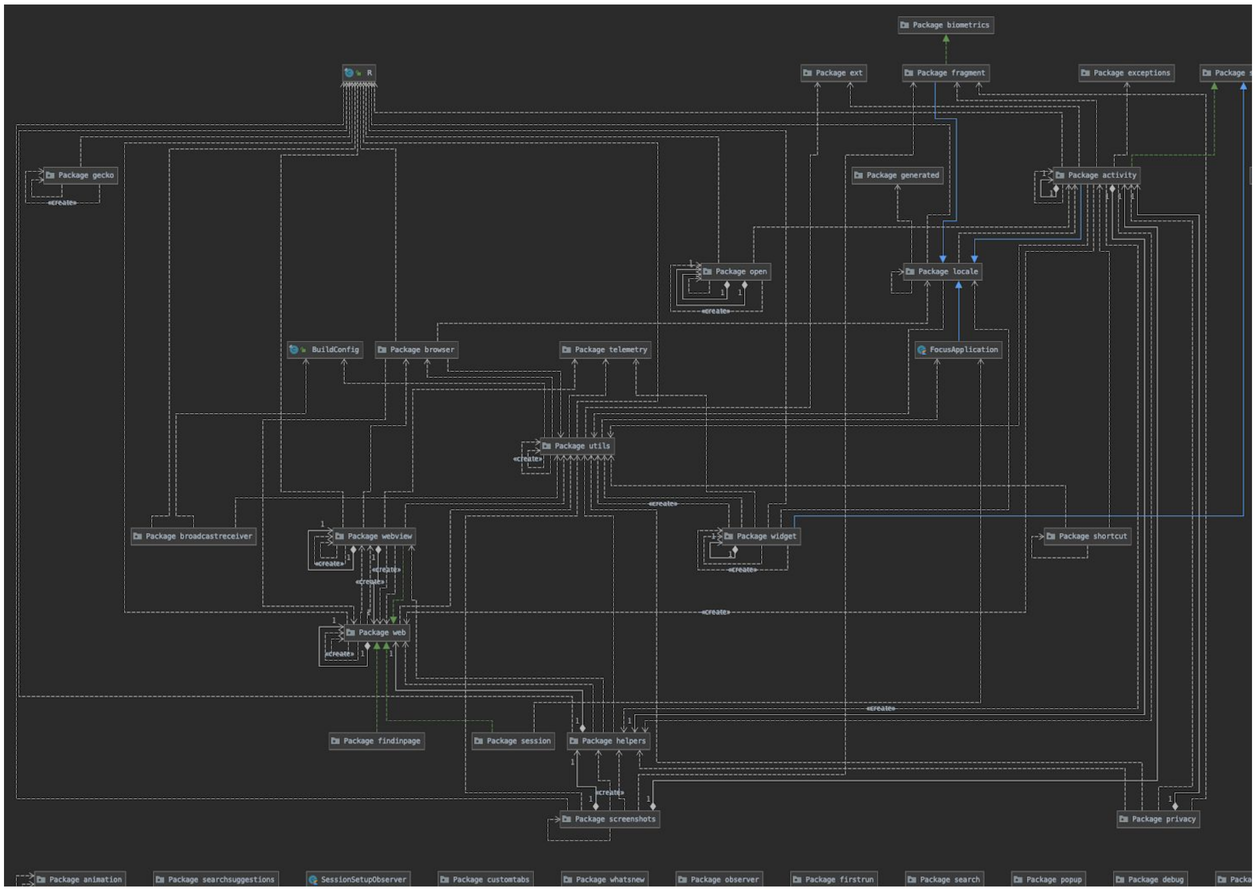
Table of Contents

Overall Architecture

From looking through the source code, generating UMLs (fig 1) by using reverse engineering tools, and reading documentation, our group found that Firefox Focus for Android has an open, multilayered architecture. We found that there is low coupling (fig 2; packages (ignore the tests) are seperated) and in most cases, high cohesion. Our group decided that the architecture could be improved by increasing cohesion in some cases (breaking down components that have multiple tasks so that they only have one specific task instead). We found the overall architecture interesting, as it is such a large architecture but does a good job having low coupling and high cohesion for the most part (the only time we found there to be low cohesion was a common instance, where a `resources` and `utils` folder holds resources shared between multiple different components).

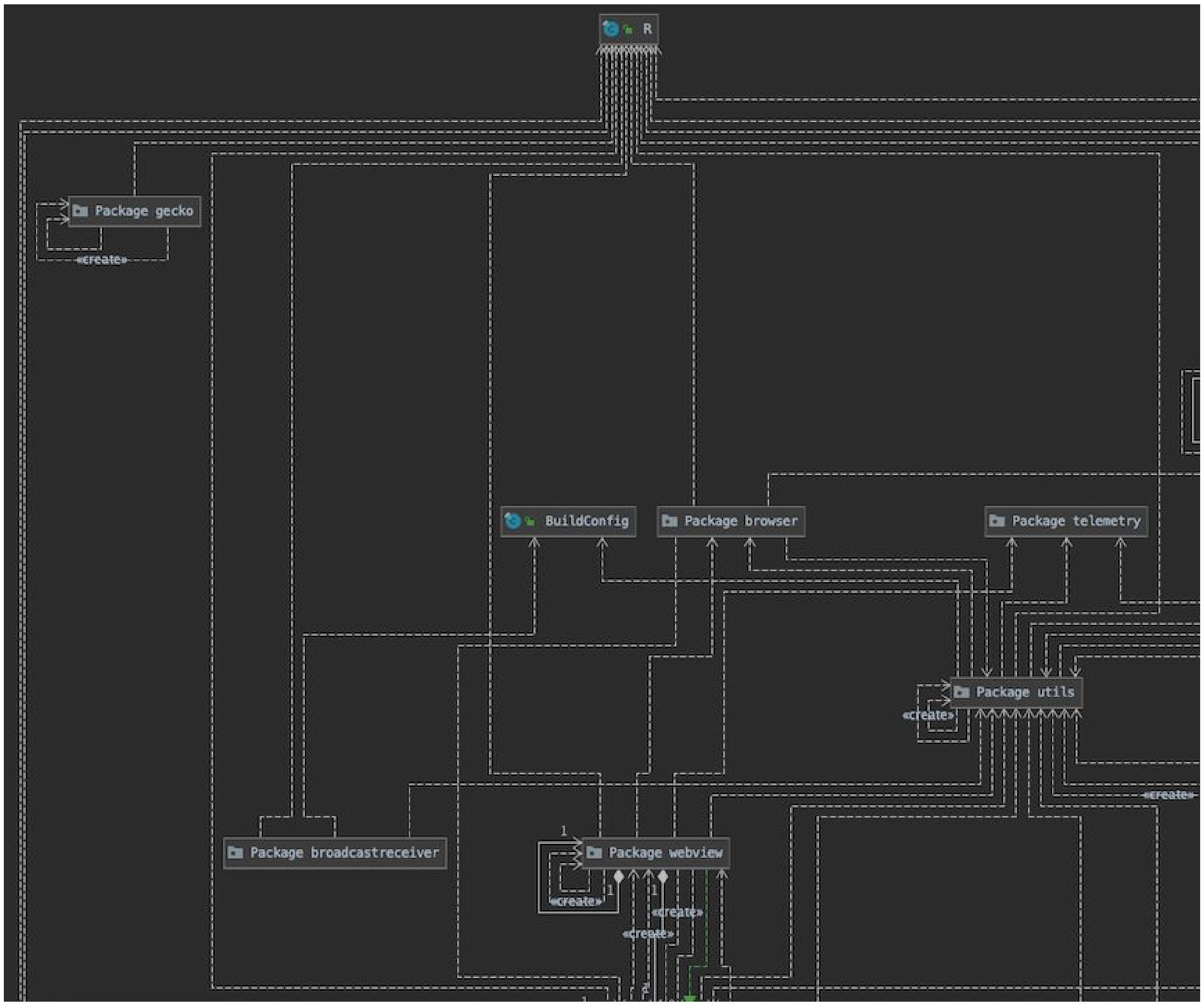
Our group found that Firefox Focus is a **3 layered** (multilayered) application. There is a **Resources layer**. This layer contains all resources (PNGs, global strings, shared resources like icons and animations). There is also, as expected, a **Front-end layer**, that focuses on how Firefox Focus looks (the UI of Firefox Focus itself). Lastly, there is the **Functionality layer**, that determines how Firefox Focus functions. This layer also determines how the websites are rendered (the HTML / CSS). Our group determined Firefox Focus is an open layered architecture, as layers and packages aren't necessarily "hierarchical". In other words, layers and packages don't need to "go through" the layer right below it. For example, the package `utils` in the Functionality layer has a dependency to the Resources layer, and the package `activity` in the Front-end layer also has a dependency to the Resources layer.

A large overview diagram of Firefox Focus is here (fig 1):

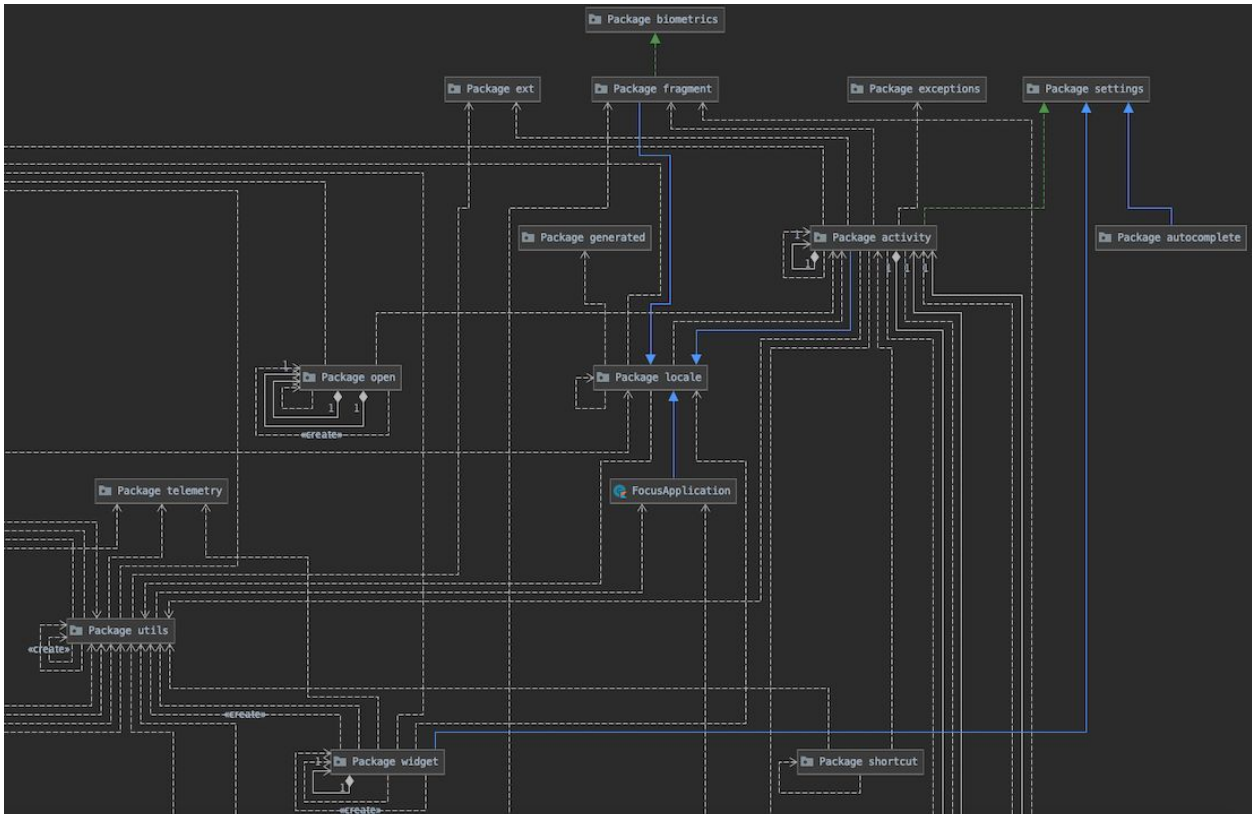


For a more clearer look at this large diagram, we broke it down to 4 sections (top left, top right, bottom left and bottom right):

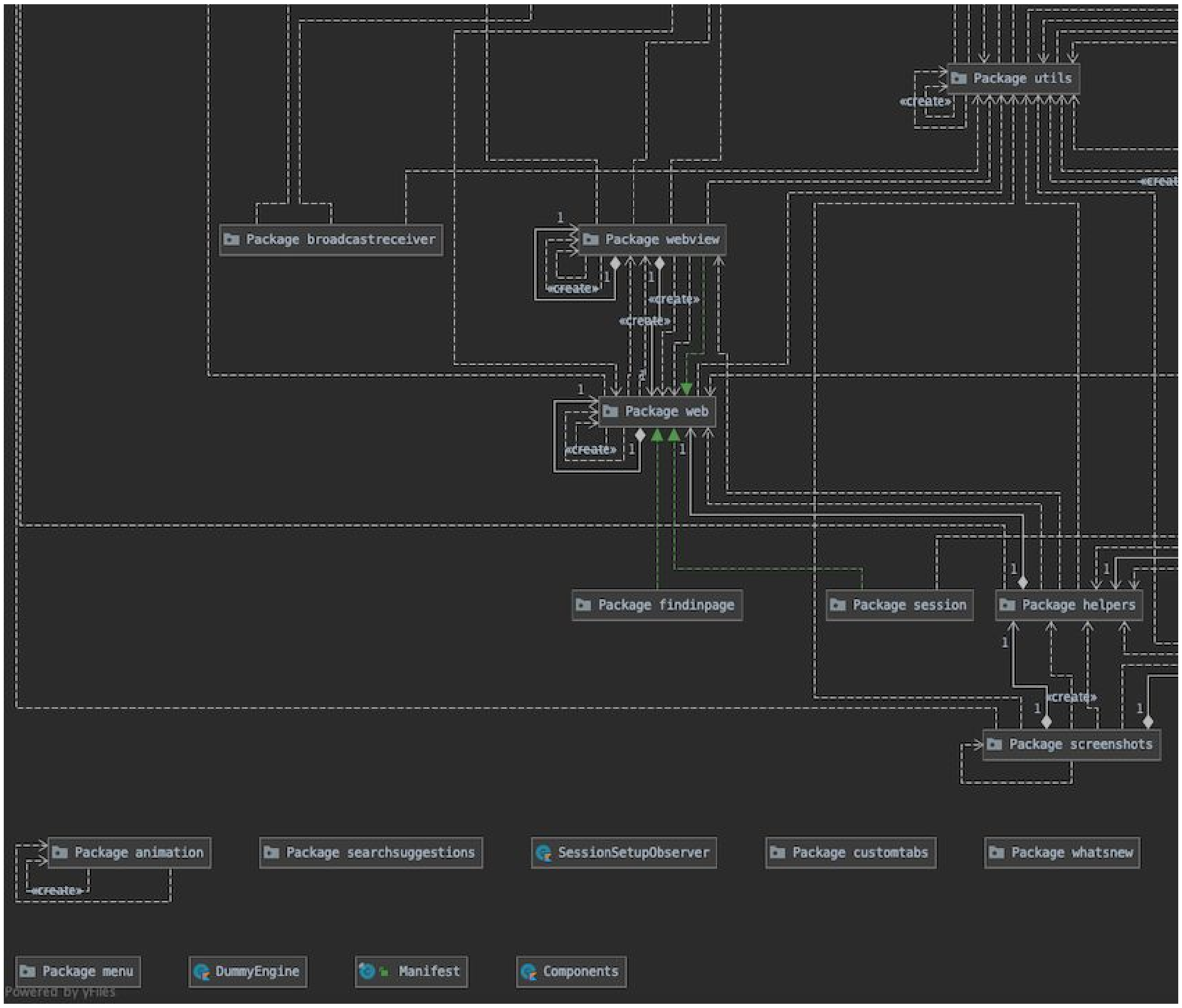
Top Left:


R
Package gecko
«create»
BuildConfig
Package browser
Package telemetry
Package utils
«create»
Package broadcastreceiver
Package webview
«create»
«create»
«create»
«create»
1
1
1

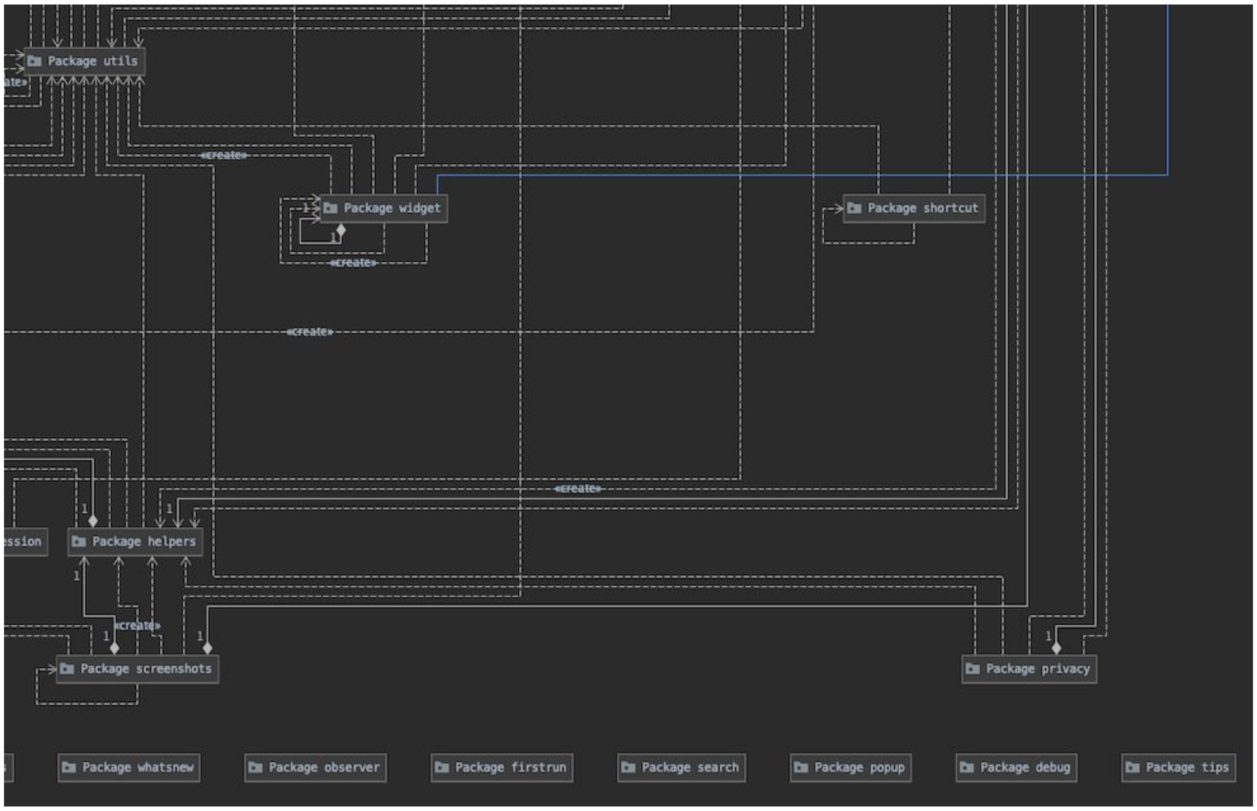
Top right:



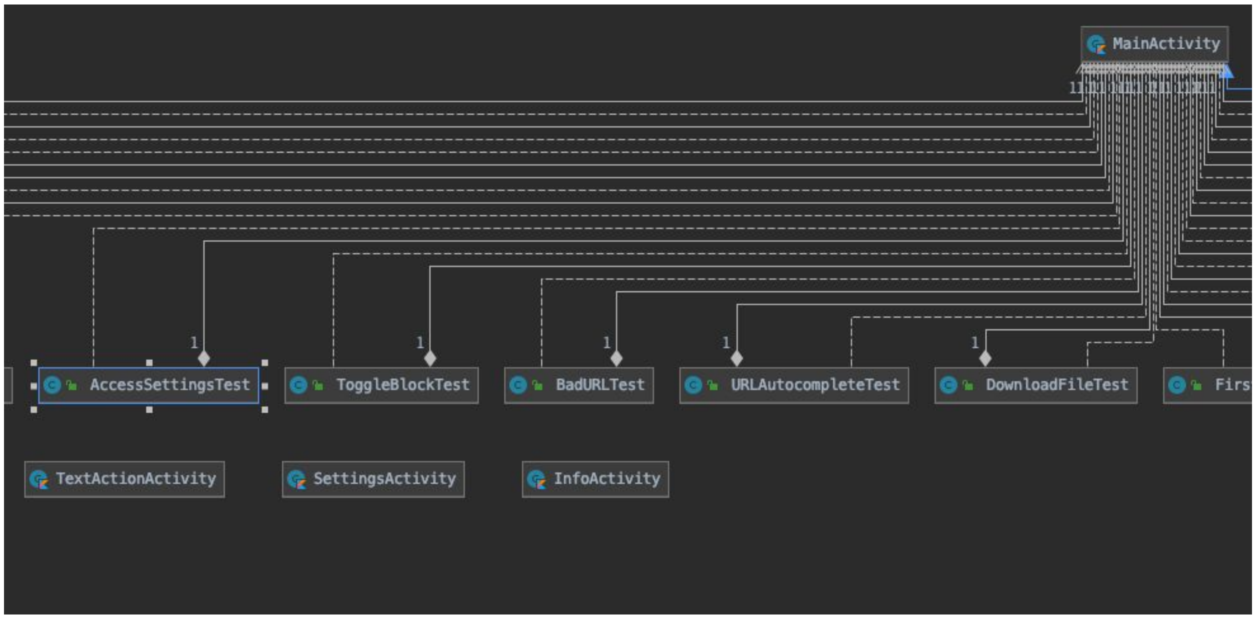
Bottom left:



Bottom right:

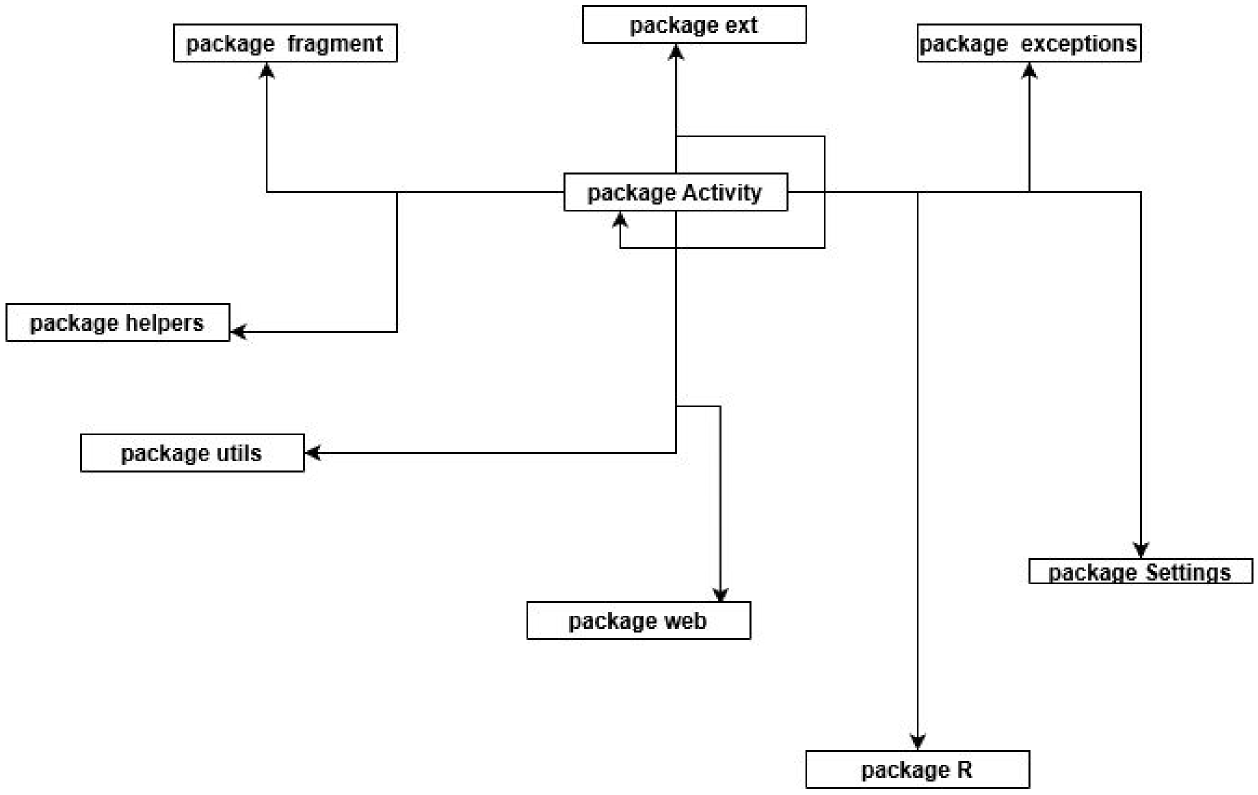


An example of low coupling, using the Activity (Front-end) package (fig 2):



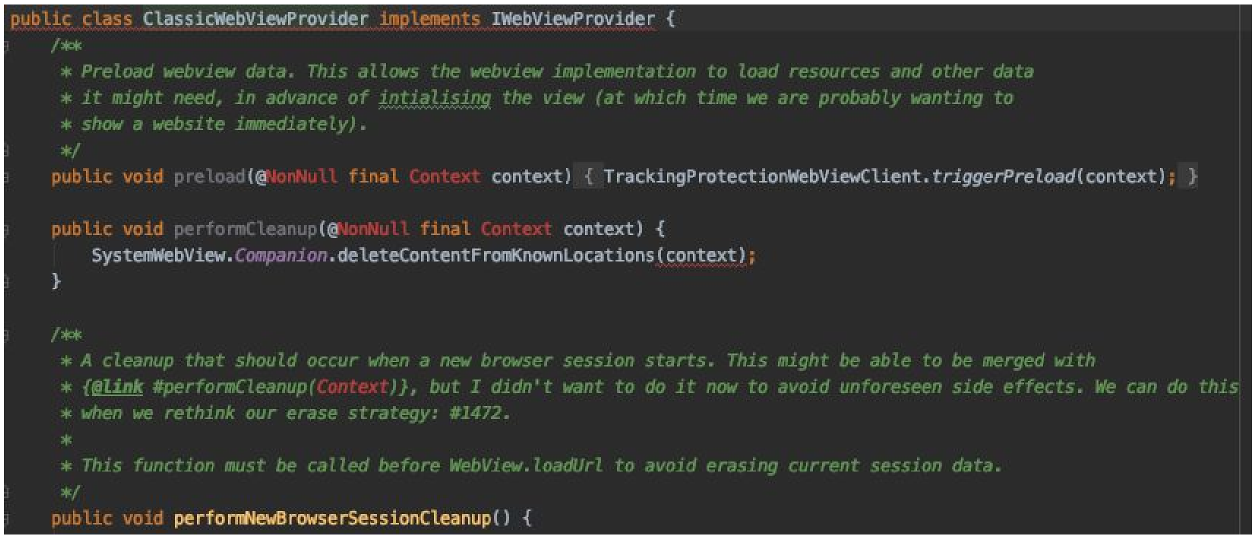
Front-end layer

The Front-end layer focuses on how Firefox Focus looks (the UI of Firefox Focus itself). It consists of the `activity` package and the `layouts` package. The `activity` package has different screens (activities) and code that handles the logic of the front-end. Amongst other screens, it has the `MainActivity`, `SettingsActivity` and `InfoActivity`. The `layout` package has the actual layout of the screens (the `xml`). The `layout` package is within the `res` (resources) folder, so the front-end uses the resources folder (Resources layer). This is an example of low cohesion, as the Functionality layer as we will soon see also uses the `res` folder. Here is a high level diagram for the Front-end layer (packing `R` is `res` / Resources):



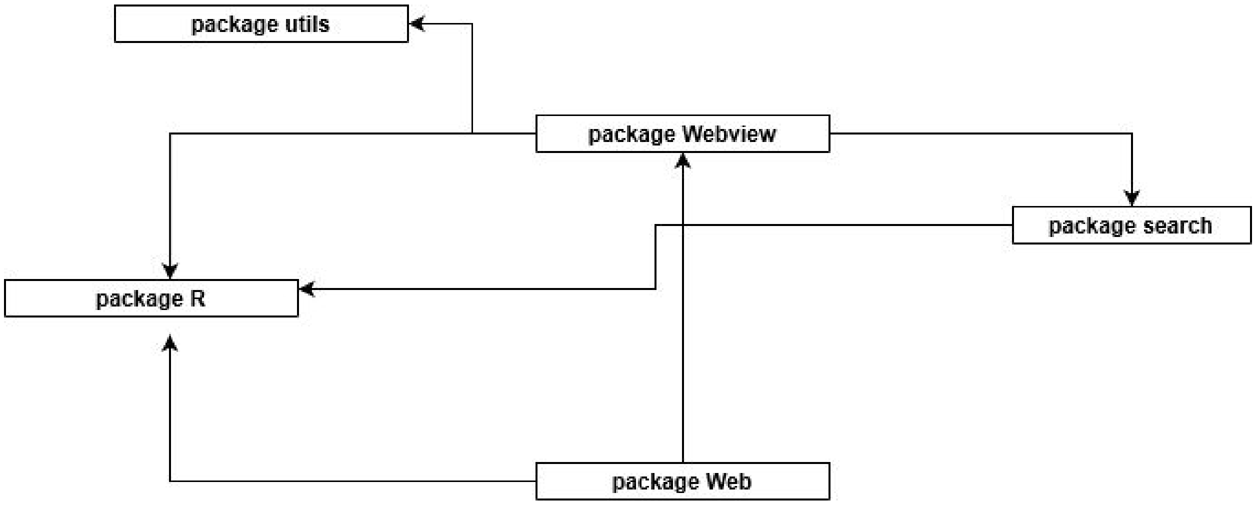
Functionality layer

The Functionality layer handles all the functionality. For example, it consists of the `webview` and `web` component. The files within these layers help render the web and HTML / CSS. This is different from the front-end layer (which is the front-end / UI of Firefox Focus itself). For example, inside `web/ClassicWebViewProvider.java`, we can see based on the comments that the render of the web and HTML / CSS is handled here:



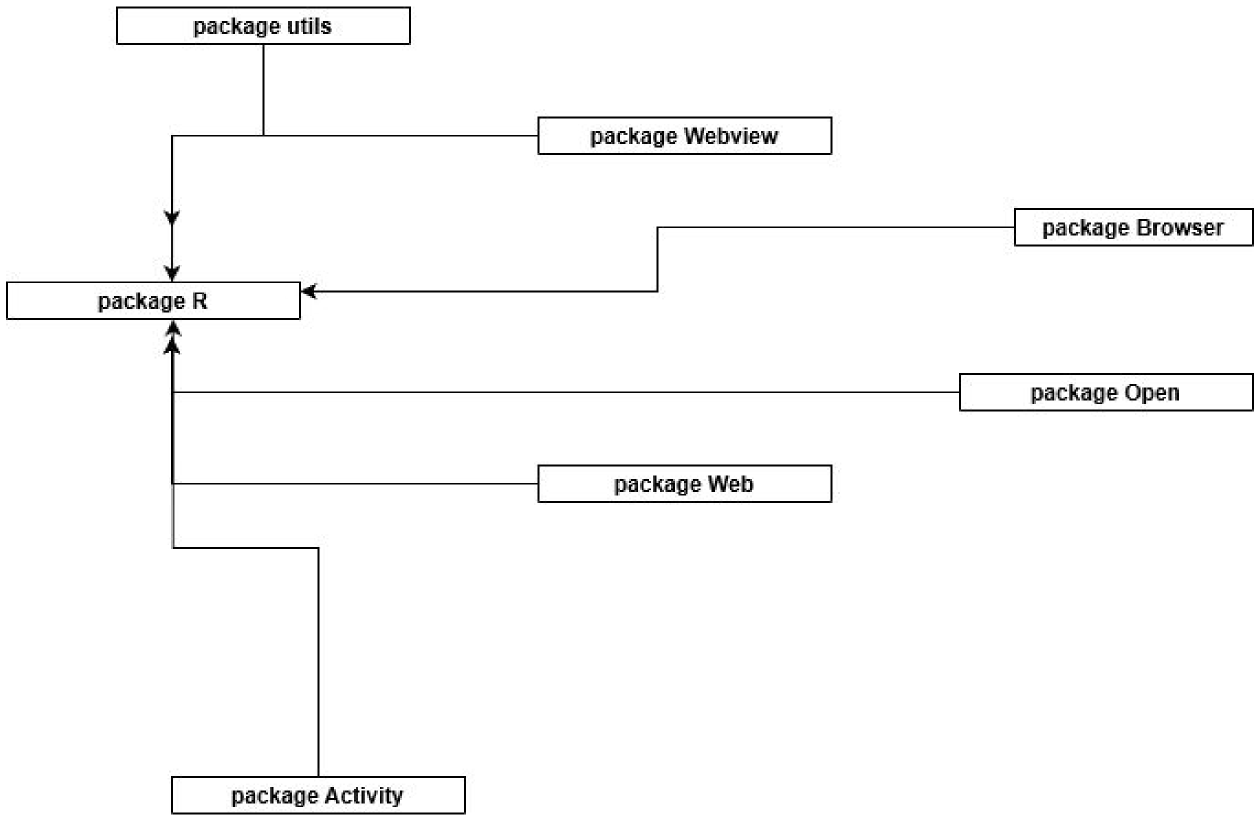
```java
public class ClassicWebViewProvider implements IWebViewProvider {
    /**
     * Preload webview data. This allows the webview implementation to load resources and other data
     * it might need, in advance of intialising the view (at which time we are probably wanting to
     * show a website immediately).
     */
    public void preload(@NonNull final Context context) { TrackingProtectionWebViewClient.triggerPreload(context); }

    public void performCleanup(@NonNull final Context context) {
        SystemWebView.Companion.deleteContentFromKnownLocations(context);
    }

    /**
     * A cleanup that should occur when a new browser session starts. This might be able to be merged with
     * {@link #performCleanup(Context)}, but I didn't want to do it now to avoid unforeseen side effects. We can do this
     * when we rethink our erase strategy: #1472.
     *
     * This function must be called before WebView.loadUrl to avoid erasing current session data.
     */
    public void performNewBrowserSessionCleanup() {
```

It also depends on `utils` and `res` (resources), but is seperate from the Front-end layer. Other packages in this layer include the `search`. A high level diagram of this layer can be seen here:



Resources layer

The Resources layer has all the resources for Firefox Focus (PNGs, shared icons and animations, global strings, etc). This layer has resources that both the Front-end and Functionality layers use (`activity`, `web` and `utils` all depend on it, amongst other packages). As we can see from the high level diagram below, it is used by a lot of different packages. For this reason, our group believes this is an example of low cohesion (the task the `res` folder performs is not one particular task, it is used by different layers for different reasons). Here is a high level diagram of the Resources layer and the `R` / `res` folder:



Suggested Improvements

Our group suggests that Firefox Focus can improve their architecture design by increasing cohesion. As seen above, packages and layers sometimes perform different, unrelated tasks. For example, The `activity` package (Front-end) and `webview` package (Functionality layer) both depend on the `utils` package. Another example is that the Resources layer (`res` folder) has `layout` (which has to do with the Front-end), but the Functionality layer (overal app logic that isn't specific to the UI) also depends on Resources. Although coupling is low and we can make changes to different parts of Firefox Focus without affecting unrelated components, cohesion can be increased by making the Resources layer more independent (and so that both the Functionality and Front-end layers do not depend on it).

Overall, we believe that since this application is new, the architecture is well thought out and clean. The folders are organized intuitively with the Activities folder containing the activities that use the functionality and aside from the Activities folder, the other folders are the different modules for the functionality of the browser.

Software Development Process

With respect to the choice of software process, it must be best suited to the nature of this project which is characterized in the following manner. Our team consists of five members. The purpose of this project is to fix three bugs present in the system we chose, and the length of time in which we will be directly contributing to Firefox Focus is a relatively short period of about one month. As well, while there is documentation for the system, the approach to documentation has been to record a brief summary of a given decision made in designing the architecture; this allows stakeholders to be able to quickly become familiar with the context behind any given decision and be informed on whether they should accept the decision or take another course. As well, because documents are fairly short, it is easier to update them.

Here we will briefly mention processes that were more readily rejected. Boehm's Spiral model is not feasible for this project as its central focus is the minimization of risk and the risk analysis process is a specialized skill none of the team members possess. Similarly, the Rational Unified Process is not suitable for our project as none of us are familiar with this particular process.

There are some advantages in the more heavily plan-based software processes such as Waterfall in our project. A way in which Waterfall is compatible is that we know the specifications from the beginning because we will be fixing bugs; we are given how an existing functionality should be behaving and how it is currently broken. We are not involved in developing new features so there will be no situation where specifications will change. Because of this, it is possible to more thoroughly plan for the entire software process and have an easy way to measure our progress. However, Waterfall does not fit well with the scope of this project; it is more well suited for large systems with multiple teams. While it is the case that Firefox Focus has many contributors and can be said to have many teams in that sense, progressing through our project involves only the five members of our team and the bug reporters; the coordination of tasks is not inhibited by the size of our team. The rigid nature of the heavily plan-based software process is not only unnecessary, it would prevent us from taking advantage of the inherent strengths of smaller teams. Overall, it does not seem that the Waterfall software process is appropriate.

Agile software processes like Scrum and XP appear to be more closely compatible to the nature of this project. Agile is well suited to the short timeframe of this project as the emphasis is made on rapid development and feedback of the software. As well, because coordination will not be inhibited by our team size, we can be flexible in re-prioritize certain tasks if our estimations were initially off. One other consideration is that documentation for Firefox Focus is generally brief instead of being very specific in nature. But while Agile methods may be advantageous for us in many ways, they are not anticipated to be optimal. A major consideration is that we will be focusing on fixing bugs in the system rather than implementing new features. While various tasks involved in bug fixing may be re-prioritized, this is a decision more appropriately made by the team than the client. Whether a bug is considered to be fixed implies that some specification of the system is already defined in advance; the exploratory process in finding new specifications according to changing client needs is not present within the scope of this project. So the involvement of the client is only essential to the extent that the expected behaviour of the system is properly described.

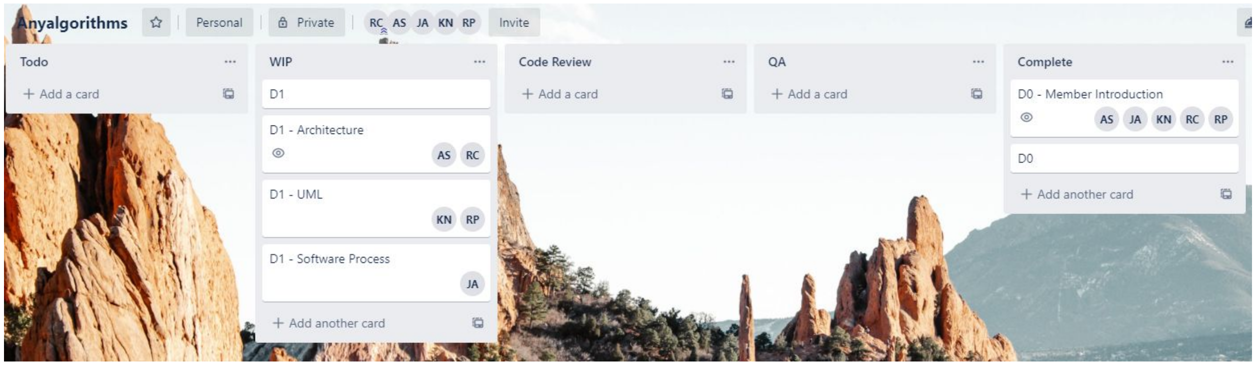
With these considerations, we believe Kanban will be the most appropriate software process for us to follow. One of the considerations is that the limitation on work-in-progress tasks will enable us to identify what part of the project requires the most amount of attention. As well, because we are not implementing new features, the iterative process in the Agile form is not as relevant as the client is not intended to make new specifications and the validation process is more limited in scope; determining which sub-task in fixing a bug should be done in a sprint would be somewhat arbitrary as a bug is not considered resolved if it is only partially fixed. From the start, we already know how the system ought to behave; the incremental nature of Kanban without any iteration cycle is sufficient for our purpose. We will be following the Kanban process through the use of Trello.

Our Development Process in Depth

In more depth, our group narrowed down **two Kanban workflows**. The **first** workflow is as follows: During our groups weekly meetings on Monday, we would decide what task each group member will do. For example, if there are three bugs to fix (bug 1, bug 2, bug 3), we would divide it so that all members have an equal workload and a bug to work on. We would then create 3 tasks (cards) for each bug on the Kanban Trello board. These three tasks are: Bug Fix, Bug Fix Code Review, Bug Fix Code Test. The Bug Fix card will be assigned to the member working on the bug. The Bug Fix Code Review card will be assigned to the member who is reviewing the code for the particular bug, after the fix is complete. The Bug Fix Code Test card will be assigned to the member who has to test the code after the fix and the code review is complete. The Bug Fix Code Review card should be moved to the “In Progress” column only once the Bug Fix is completed by the assigned member (to ensure that the code being reviewed is the completed code). Once the code review is done and the Bug Fix Code Review card is placed in the “Complete” column of the Kanban Trello board, the member assigned to test the code will move the Bug Fix Code Test card to the In Progress column, and start testing the code. This Kanban approach makes it very easy for our entire group to visualize the progress of the deliverable, and see if there are any bottlenecks.

Our group also considered a **second** way of setting up the Kanban Trello board. The second way is having two extra columns: the Code Review column and the QA column. So instead of creating multiple cards for each bug as suggested above (a Bug Fix Code Review card and a Bug Fix Code Test card), there will be only one card for each Bug, and the card will be moved through the columns in the following order (From Todo → WIP → Code Review → QA → Complete).

Our group decided that the second way of setting up the Kanban Trello board is more efficient and easy to visualize, so we decided to use the second approach. An example of our Kanban Trello board is here:



This process allows us to easily see where blockers are, and let us drag cards backwards if problems arise.