Contents

Evaluation

Completion of requirements

The project has met each of the sets of the objectives initially set.

Objective set	Evaluation
1- Login	This objective was completed first, as the login flow will always be the initial entry point for the application and the authentication token is required when implementing further objectives. The login flow is always shown if an authentication is not found, as this check is performed in the base `Activity`. The initial objective of allowing the user to sign in was exceeded by ensuring that the action that the user was attempting to complete is continued once they have successfully logged in.
2- Users	This objective was met next, as the first `Activity` shown once the user has logged in on first launch. The first five sub-objectives in this section were implemented as separate `Fragments` within a `ViewPager`. This structure allows all of the information available via the desktop website to be shown on a mobile device while maintaining a suitable information density for a mobile device.

Objective set	Evaluation
3- Repositories	This objective was met in a similar manner to objective 2. The set of information available on the GitHub website was split into multiple `Fragments`, each with their own set of objectives to complete. This objective required the implementation of a custom `WebView` to interface with JavaScript and when dealing with scroll events. It also required the implementation of a second `Activity` to display the file tree and files for a particular repository. This was achieved using a single `RecyclerView` and a tree data structure which was loaded asynchronously to ensure an optimal user experience with as little waiting as possible.
4- Issues	This objective dealt with the display and editing of issues. The first `Fragment` within the `IssueActivity` dealt with displaying information about the issue, and manipulating the stream of events which have occurred in order to create a merged list of different events.
5- Commits	The implementation of the commit objectives was similar to that of the issues objective, using two `Fragments` to display information about the commit and the comments made on the commit. The key difference was the information displayed after the main text body, as each commit may contain a highly variable length of changes which cannot all be shown immediately on a mobile device. This problem was solved by listing the information in a collapsed state showing only the file and allowing it to be expanded if the user wanted to list all of the changes made to a particular file. The display of statuses was also handled in a similar manner, displaying the overall status first, before listing each of the individual statuses making up the larger status.
6- Projects	Displaying projects was one of the more difficult objectives to implement as it dealt with multiple different types of data as well as numerous user interface features.

Objective set	Evaluation
	Displaying of issues was handled using the same utility method as used when displaying them elsewhere, maintaining a consistent style. The asynchronous loading of issues and parsing of their content significantly improves the user experience by ensuring that the UI thread is not locked. Implementing drag and drop was one of the more challenging parts of this objective as it required implementing a set of listeners on the views which might be dragged over as well as managing scrolling in both x and y based on the dimensions and positions of Views which were often relative to themselves rather than the screen itself. Fuzzy string searching was successfully implemented allowing the user to search large projects by the content of the cards displayed within the Activity and to jump to selected items.
7- Link handling	Link handling met the objectives set for it, opening usernames, repositories, and direct items, as well as rejecting links which cannot be handled by the app. Link handling could have been improved by checking by adding a blacklist for top level paths on the GitHub website such as the trending, integrations, and showcasing pages. It would also have been useful to add support for opening repository file paths directly, rather than loading the root of the file tree.
8- Notifications	The notification service was successfully implemented and handled notifications quickly without becoming a wakelock as update events can be packaged with other events.
9- Markdown	Markdown parsing was successfully implemented, as was demonstrated in the testing section. The spans used for large code blocks and tables allowed content to be displayed without disrupting the ability of the user to read the

Objective set	Evaluation
	surrounding text or requiring constant checks to override scrolling in the x direction.
10- Markdown editing	The markdown editor allowed for the insertion of formatting characters without requiring the user to switch their keyboard layout each time. The structure of the base `EditorActivity` allowed easy adaptation for other types of content to be edited while handling input from the adapter.

Feedback and reflection

User feedback

Evaluation from another student, Ben Sheffield:

The application has clearly met each of its objectives.

The primary activity displays information about my repositories with each page showing a different page from the GitHub website without having to navigate backwards and forwards to view the content.

The ability to pin repositories was particularly useful as it allowed me to manage the repositories which I use the most, without having to scroll to each one.

When viewing a repository the README view was helpful as it does not require navigating to another page as on the GitHub website.

The ability to filter issues was helpful as it is missing from the mobile website allows searching through the filtered issues.

The markdown editor was invaluable for formatting messages without having to switch the keyboard layout and look for symbols, and image uploading through the app was much quicker than uploading an image normally which would

require moving to another app, copying the link, and then inserting it into the editor.

The notification service allowed me to be informed of updates to my repositories without having to check the website periodically.

Some improvements would be to add a view for all of the notification events and issues which have happened across my repositories, rather than having to check individual repositories.

While the notification system showed notifications quickly it didn't always provide much information about the event that had happened, requiring it to be opened to view the content.
It also wasn't clear that notifications had been dismissed, and it would be more useful to have buttons on the notification to mark it as read.

Other improvements

Given the opportunity to rewrite this product I would have structured it differently, allowing me to implement features much more easily.

The largest change would be writing the project in Kotlin rather than Java.

Kotlin is a language with runs on the Java Virtual Machine and brings various improvements over Java while maintaining performance.
Kotlin has an increased advantage on Android as Android does not support language features past Java 7 meaning that it does not include support for:

- The `forEach` call in the `Iterable` interface.
- Default and static methods in interfaces
- Lambda expressions for anonymous interfaces and classes.
- The stream API:
    - The stream API brings filter, map, and reduce style operations to collections
    - It also allows parallel execution
- The Time API which replaces the broken `Date` system
- Various other improvements

Kotlin brings these features as well as many more useful ones:

- Null safety:
    - All types are non-nullable by default
    - The compiler enforces null checks before accessing nullable variables

    - This can be done with the chainable safe call operator `nullable?.methodCall()` which performs a call if the object is non-null
- Extension functions
    - Rather than defining static utility methods, these methods can be directly added to types
    - An extension function can be added to a type without directly extending it, and this function can then be called as if it were a member function
- Higher-order functions and lambdas:
    - Rather than having to define an interface, and pass an instance of an implementation of that interface higher order functions can be used
    - Functions can be stored for later use, or created within a function
- Data classes:
    - Data classes can be used to generate all getters and setters in a single line
    - The compiler will also generate members for equality, string conversion, and object copying
- Coroutines:
    - Coroutines enable operations to be executed without blocking a thread
    - Coroutines can be suspended without blocking a thread and without any context switching
- Type aliases:
    - Type aliases allow simplifying code by removing the need for repeatedly stating types

Particularly useful for this project are data classes and coroutines.

Data classes

Suppose I need a model to hold a simple set of information about a contact. This contact model must contain the following information:

- Name
- Number
- Email address
- Age

In Java this class would be constructed as follows:

```
class Contact {

    private String name;
    private String number;
    private String email;
    private int age;

    Contact(String name, String number, String email, int age) {
        this.name = name;
        this.number = number;
        this.email = email;
        this.age = age;
    }

    String getName() {
        return name;
    }

    String getNumber() {
        return number;
    }

    String getEmail() {
        return email;
    }

    int getAge() {
        int age;
    }

    void setName(String name) {
        this.name = name;
    }

    void setNumber(String number) {
        this.number = number;
    }

    void setEmail(String email) {
        this.email = email;
    }

    void setAge(int age) {
        this.age = age;
    }

    @Override
    public boolean equals(Object o) {
        if(this == o) return true;
        if(o == null || getClass() != o.getClass()) return false;

        final Contact c = (Contact) o;
        return (name != null  && name.equals(c.name)) &&
            (number != null && number.equals(c.number)) &&
            (email != null && email.equals(c.email)) &&
            (age == c.age);
    }

    @Override
```

```
    public int hashCode() {
        int result = name == null ? 0 : name.hashCode();
        result = 31 * result + (number == null ? 0 : number.hashCode());
        result = 31 * result + (email == null ? 0 : email.hashCode());
        return 31 * result + age;
    }

    @Override
    public String toString() {
        return "Contact{" +
            "name='" + name + "\'" +
            ", number='" + number + "\'" +
            ", email='" + email + "\'" +
            ", age=" + age "\'" +
            "}";
    }
}
```

75 lines for an object holding 4 values.
On Android the parcelable interface would also be required in most cases, adding another 2 lines per variable, 3 functions and a static nested class.

In Kotlin this entire class would be declared in 1 line:

```
data class Contact(val name: String, val number: String, val email: String,
val age: int)
```

The `hashCode`, `equals`, and `toString` methods are automatically generated along with `copy`.
Variables can be made final by writing val rather than var, and access modifiers can be used.

The model can also be treated as a list.

Suppose we have a list of `Contact` data models.
The items can be iterated as follows:

```
for((name, number, email, age) in listOfContacts) {
    print(name)
}
```

Data classes would have greatly increased development speed, and along with Kotlin's null safety reduced most model classes to simple declarations.

Co-routines

While some markdown processing was offloaded to a separate thread, the solution was not ideal.
The thread did remove work which would normally run on the UI thread and cause stutter, but a single thread is not an optimal solution for highly parallel work such as parsing each of the cards in a project.

Coroutines are ideal for this workload.
On a desktop, millions of simple coroutines can be run at once.
On a mobile device this number is smaller, but still orders of magnitudes larger

than the number required to allow processing each section of markdown in its own coroutine.

Moving parsing to coroutines within the `MarkdownTextView` and `MarkdownEditText` classes would ensure that no parsing is run on the UI thread, eliminating any latency before animations which is sometimes triggered when an issue or commit with a large body is parsed while the transition animation for the `Activity` is running.

Analysis of possible improvements

The first suggestion for improvement was to implement a combined view of events and issues relevant to the authenticated user.
Much of the structure for this is already in place:

- There are bindings for a list of issues
- There are bindings for some forms of events

These feeds could be added as `Fragments` only displayed for the authenticated user.
The events feed is an area in which Kotlin data classes would be especially useful, as there are 34 different event types with a wide range of data.

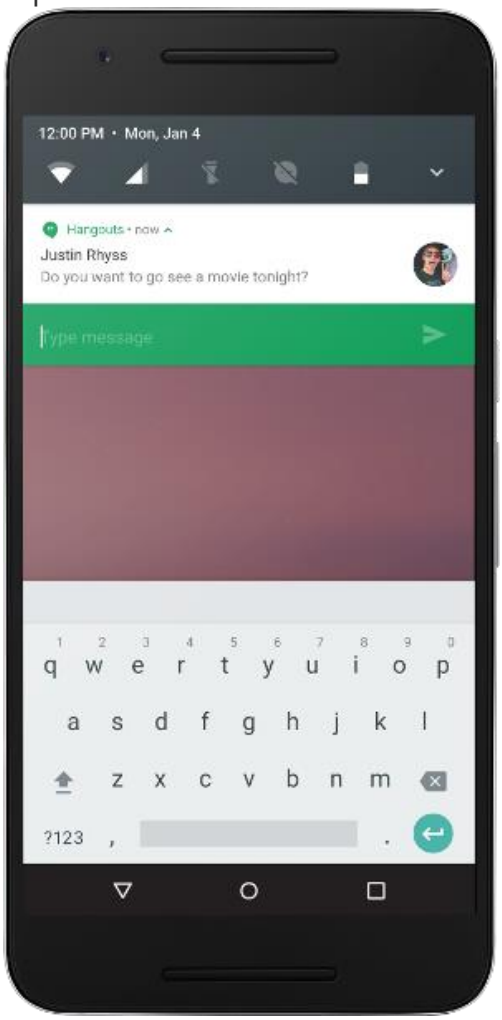
The second suggestion for improvement was to add further information to notifications.

There is very little information given via the notifications API, which is expected as it is designed for polling.
In order to add more information, extra requests would need to be made.

- The user that mentioned the authenticated user in a mentioned event can be found by loading the comments and searching them for the last mention of the authenticated user
- The latest comments on a comment thread from a comment event can be displayed in a notification in the same way that many email clients display short snippets of emails in their notifications
- The assign event issue can be loaded by loading the issues for the authenticated user and checking for the event repository, a snippet of the issue content can then be displayed in the notification

The user also mentioned making the notifications more clearly actionable.
This can be quite easily achieved by using the notification builder to add inline buttons, and adding `PendingIntents` to perform the read actions.

These buttons could also be specific to an event type, for example allowing an issue to be dismissed from the notification.
Other than just displaying extra content, Android N brought support for quick replies in notifications.



This would allow the user to reply to an event on an issue or commit without having to leave their current application.