Implementing a Design System for Mobile Cross Platform Development

Mark Ryan

2021 Laurea

Laurea University of Applied Sciences

Implementing a Design System for Mobile Cross Platform Development

Mark Ryan
Degree Programme in BIT
Bachelor's Thesis
March, 2021

Laurea University of Applied Sciences **Abstract**

Degree Programme in Business Information Technology

Bachelor's Thesis

Ryan, Mark

Implementing a Design System for Mobile Cross Platform Development

Year 2021 Number of pages 41

The purpose of this thesis project was to implement a design system to increase the speed of front-end development at the client company. The beneficiary is Meetch, a start-up business offering a cross platform application known also as Meetch, in both the Apple App store and Google Play store. The purpose of the design system is to create a uniform look and feel on 2 distinct platforms: iOS and Android.

The project followed a series of steps before the design system was implemented. These included performing a visual audit, benchmarking industry standards and making observations based on the results of the benchmark. Implementing the design system consisted of creating fundamental, components and patterns based on the observations, these form the designs system that translates business requirements into engaging customer experiences.

Keywords: Mobile Development, Brand experience, Service Design, Flutter,

Contents

1 Introduction

Digital services have become increasingly complex, both in usage and programming paradigms especially in now during 2020 and the COVID-19 situation, (European Centre for Disease Prevention, 2021). “The Speed of Change” (outsystems, 2021) reports on how the urgency and need to adapt is “expected to happen to every kind of organization at unprecedented speed”. The users of these services are no longer limited to a single platform or device. Instead, services need to be ubiquitous across multiple platforms with more complex use cases. Development teams responsible for these services are required to have extensive knowledge of multiple tools, languages, and standards to implement the business requirements. Alternate to a having highly skilled team. Is to have multiple teams. Each with a focus on a single platform. For the purposes of this thesis, we will focus on the service Meetch will provide. We will focus only on Apple iOS and Google Android platforms for mobile as they make up approximately 99,22% of the market share as seen in Figure 1.

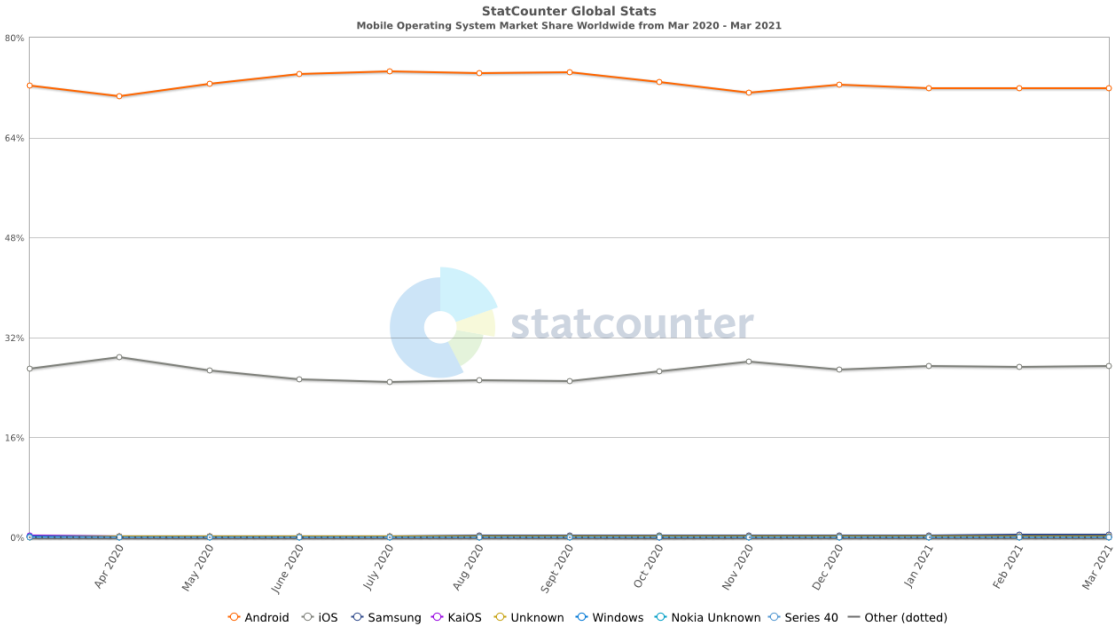

Figure 1: Mobile operating system market share worldwide

How these platforms relate to web implementations will be mentioned for reference but not in-depth analysis. Mobile platforms after approximately 12 years have firmly embedded in our lives Apple (2018). As applications are developed, they are subject to change. Responding to this change and adapting the plan as part agile software development allows us to create meaningful iterations (2003).

1.1 Client

Meetch, a start-up company based in Helsinki, engaged in mobile development on both iOS and Android platforms to deliver the Meetch (2021) application. The focus of the development is to build and maintain a psychological type matching application based on Socionics the theory of intertype relationships. The work by Victor Gulenko (2019) in “Psychological types: Why are people so different?” heavily influences the service Meetch provides.

1.2 Project background

Due to the size of the development team in Meetch speed, adaptability and the least complex tooling is of the utmost importance. Learning multiple platforms and the programming languages that come with them for iOS and Android is unfeasible for Meetch. Native development will require considerably more time than a cross-platform approach. The iOS and Android platform have become increasingly complex. Supporting new secondary languages that are replacing the well-established Objective-C and Java for iOS and Android respectively. For these reasons a cross platform tool (Flutter 2021) has been accepted by Meetch. Using Flutter and a simplified Kanban framework to manage development. Making use of both incremental and iterative agile methods is expected to increase speed, adaptability, and allow a simplified development experience.

1.3 Problem Statement

Meetch needs a concise way to translate business requirements into engaging customer experiences. The codebase used must be as reusable across iOS and Android as possible to save momentum during development. Flutter (2021) is expected to deliver this cross-platform experience as it is a declarative UI framework. Flutter applications can be published on both Apple App Store and Google Play Store. With only the publishing process for each respective artefact being different.

To accelerate the speed of development the major design tokens, assets and components need to be developed in a reusable design system that developers can use to rapidly scaffold views that deliver experiences that Meetch expects.

Experiences developed in this cross-platform approach will lead to a coherent brand recognition that a customer can easily intuit. Developers will benefit from not having to create custom implementations. Avoiding regression that would be created if developing two separate applications with separate tooling, codebases, design rules and most importantly asset formats.

The design system is expected to help keep the install size herein referred to as bundle, of the mobile application to a minimum. A known issue in Flutter is that these sizes can be large if not implemented well.

The initial start date of thesis work with Meetch is March 8th, 2021. However, the discovery and review of the problem had begun prior to March. The below timeline is intended to give an overview of the activities carried out during this thesis and the expected conclusion of these activities.

Activity	March				April				May			
Problem definition												
Literature review												
Platform Analysis												
Designing framework												
Developing prototype												
Evaluating framework												
Documentation												
Proofreading												
Submission of thesis												

Table 1: Overview of project timeline

1.4 Objectives

The objectives of this project are:

- To define, develop and implement a set of standards in a design system that will increase speed of development and reduce bundle size.
- Create a uniform look and feel on two distinct platforms taking advantage of Flutter declarative development.
- Show the increase in development speed and reduction in bundle size from using the design system.

These objectives do not need to be carried out in order. It is beneficial that they are done in parallel when possible. To take advantage of the agile framework development process.

1.5 Requirements

Work on the design system should not impact the already implemented features. When implementing the design system, it should be retroactively applied. This is expected to reduce the bundle size of the release candidate Meetch on both iOS and Android platforms. The design system will also allow general thematic changes from a central library. Without

the retroactive application older features will not change without direct code changes in each feature.

To track the increase in speed and reduction in regression. Metadata of the Git source control system and automated build pipelines will be required. It is not possible to look at full source code. instead, the tracking will be done on branches of similar size containing equivalent amounts of committed changes.

If no similar size branches in source control can be found. Then views will be re-created using the design system and not using the design system to collect the required data for benchmarking.

The delta between bundle size before design system implementation and after design system implementation will be used to show file size differences.

1.6 Limitations

In the development process we will not be considering any other development than the user interface layouts and transitions. User interfaces or UI refers to the interface screens of mobile platforms that render the design elements to the user. Background service tasks, such as fetching data from restful application programming interfaces is irrelevant unless somehow requiring rendered error, success and loading messages.

2 Design theories and tools

2.1 Tools

2.1.1 Google Android

Android is the mobile operating system offered by Google. It was incorporated as a Google offering in 2005. It is based on the Linux kernel architecture (Android Architecture, 2021) and each update to the operating system is free, fully functional, and open source. Android as an operating system fully endorses open source. Allowing users and manufacturers alike to make changes to the operating system without licensing (Android, 2021). Currently 11 versions of android have been released and it is currently the leading OS in terms of market adoption as

shown (Appbrain, 2021) in Figure 2.

Android OS market share

Android OS version	Market share	Change in the last 30 days
10 (Android 10)	35.4 %	↓ 2%
9.0 (Pie)	18.8 %	↓ 2%
8.0-8.1 (Oreo)	14.6 %	No change
11 (Android 11)	11.3 %	↑ 25%
7.0-7.1 (Nougat)	7.8 %	↓ 3%
6.0 (Marshmallow)	5.5 %	↓ 4%
5.0-5.1 (Lollipop)	4.4 %	No change
4.4 (KitKat)	1.6 %	↓ 4%
4.1-4.3 (Jelly Bean)	0.6 %	↓ 6%
4.0 (Ice Cream Sandwich)	0.1 %	↓ 8%

Figure 2: Android OS market share

Manufacturers also benefit from the open-source nature of Android by being able to fully customize the builds that are installed on their handsets. For example, Samsung customize Android by using applications developed by them to replace the standard Google Mobile Services suite (Samsung, 2021). This leads to difficulty in maintaining version control over the selection of devices Android can be installed on. The most popular versions of Android as May 2021 is Jelly Bean 4.1 according to Figure 3.

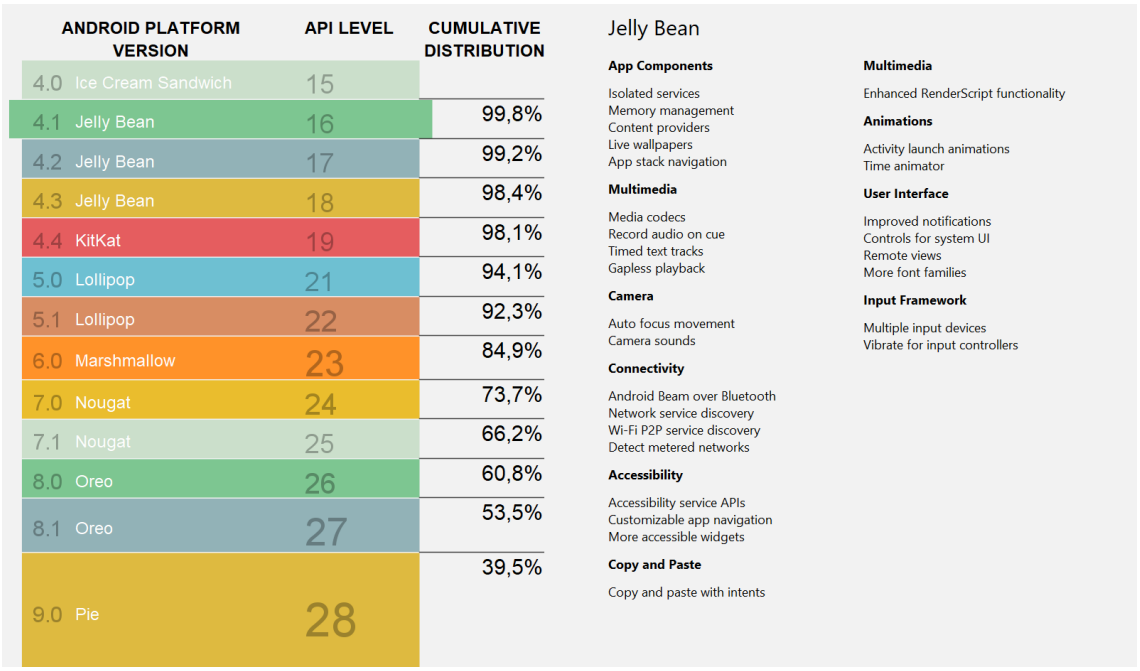

Figure 3: Android Version Coverage

These are not the most modern versions which is to be expected in Android as new smart devices are propagated on the market it takes time for newer versions to be adopted. Android devices also cannot always update to the latest version which requires the purchase of a new device to gain access to the newest software experiences.

2.1.2 Apple iOS

Apple's iOS unlike Android is not open source and does not run-on 3rd party smart devices. It is tightly controlled by Apple. Since both hardware and software are controlled by Apple. The integration between the two is better than any other platform. For Apple, the device family of iOS is made up of iPhone, iPad and iPod. Applications developed for Apple devices are subject to strict design standards. This is to cultivate a sense intuitiveness regardless of the application. These are not the same standards that are expected for Flutter framework use in Meetch development.

Apple updates unlike Android updates are often sought out by the users as they bring new features that are sought after. This coupled with Apples rigid control of the platform and OS leads to much less versions in use at any given time (Figure 4).

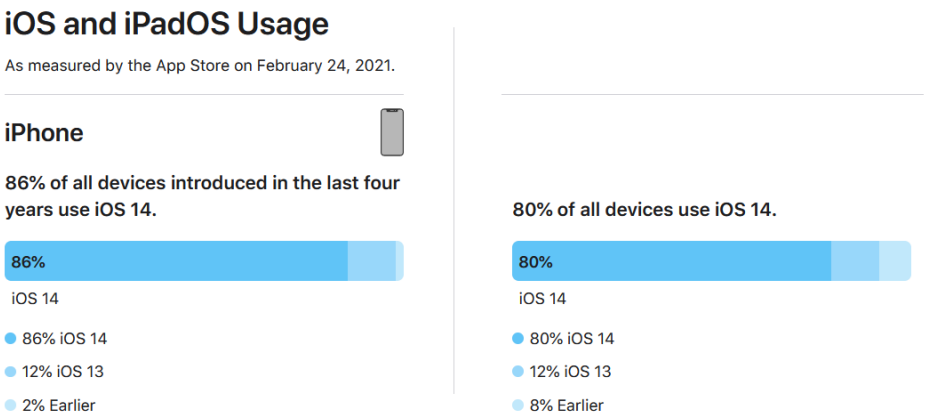

Figure 4: App Store Version Usage

2.1.3 Flutter

Flutter is a cross-platform development framework built using the Dart language from Google. Taking inspiration from another successful framework such as React. It is declarative in nature. User interfaces are built using widgets. Widgets can be considered immutable blueprints of how the UI should be rendered. Everything in Flutter can be its own self-contained widget or a widget of other widgets.

This allows a large amount of customisation. Widgets are comparable to components in web frameworks in how they encapsulate functionality. A widget can but used to describe how it looks in a specific state configuration. When this state is changed or interacted with widget will rebuild itself according to the new description then compare itself to the old description in order deliver the minimum viable render.

As it is stated in Flutter's documentation that the main parts of the tool are:

- **Fast development** using stateful reloading that allows paint using a customizable pre-set of widgets in milliseconds or less.
- **Expressive & flexible user interface** using architecture that allows full customization based on requirements that will be defined by Meetch.
- **Native equivalent performance** on the selected platforms Meetch has selected (iOS and Android). All interactions like scrolling, icons and call to actions will be familiar feeling to the user.

Flutter has an active development community. This community publishes collections of widgets known as packages to https://pub.dev/, here we can see the iOS icon library published for flutter as flutter_sfsymbols (Figure 5).

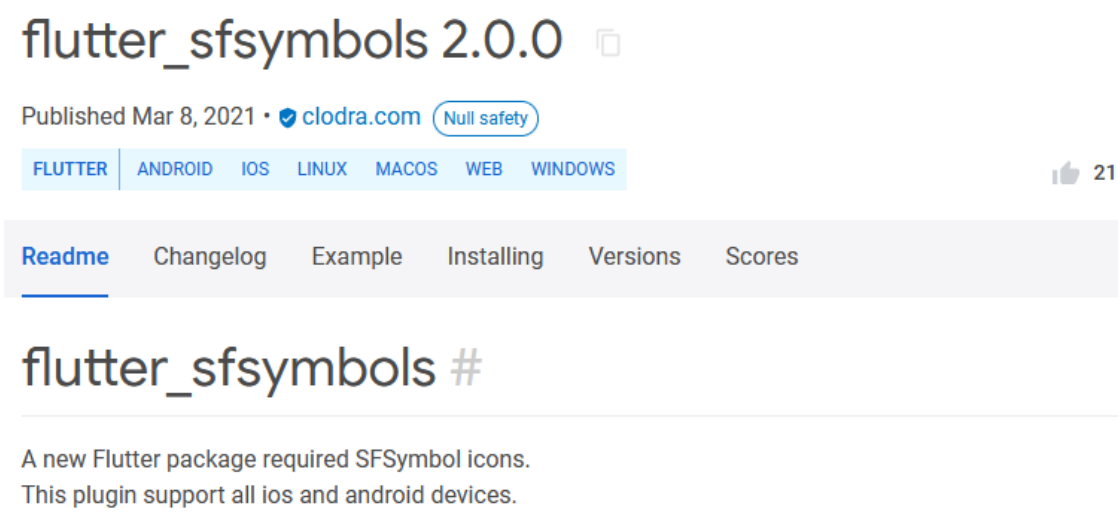

Figure 5: Flutter package of iOS icons

These packages can be reused in Meetch development. Common user interface items such as navigation, authentication and sign-in will be used from packages found here. These will allow user experiences to match the choice of platform the user has selected instead of forcing design choices on them that they may want to avoid from the other platform.

The most important part of a Flutter application is the pubspec file it is a written in YAML format which is a human readable format that describes the application. For the purposes of this thesis, we will make use of the pubspec when importing assets to the application that are to be deployed in both iOS and Android.

In Figure 6 we can see an example of Flutter application structure.

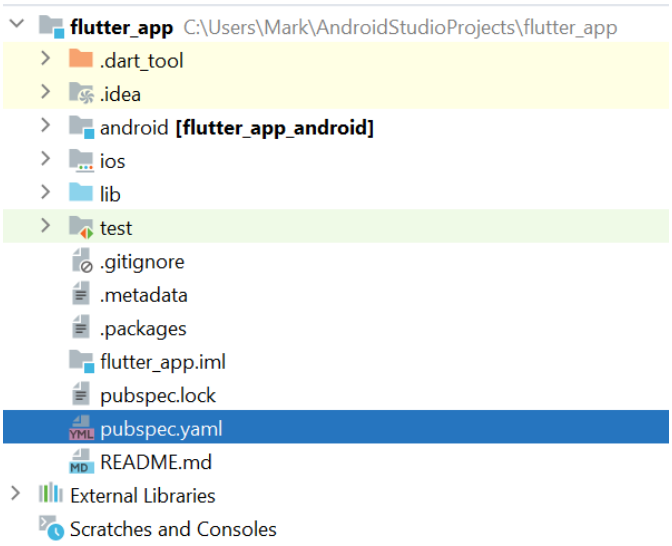

Figure 6: Example Flutter app structure

2.2 Design theories

2.2.1 Human Interface Guidelines

The Human Interface Guidelines (2021) known as HIG, are a set of UI resources as well as in-depth implementation documentation that describes exactly how to integrate with Apple platforms. It is the desired way Apple wants user interfaces created for use on their range of hardware products. The device families included in this are:

- MacOS
- iOS & iPadOS
- watchOS
- tvOs

These device families make use of the same assets and thus create an intuitive experience for the users of these platforms. Going against these standards in any way can lead to extremely complicated implementations that may not be supported by Apple in the future and is ill advised. We will not be covering any guidelines from HIG on MacOS, watchOS and tvOS device families as they are not in scope of the development.

iOS applications are developed rigidly based on 3 core themes from Apple. Based on these there are high expectations in both functionality and design quality that are required before any application is admitted to Apple App Store.

- Clarity
- Deference
- Depth

Clarity can be related directly to accessibility which has become a right of access in the European Union as part of GDPR (2021) and as such is considered a mandatory aspect of application development. Fortunately, Apple also has a heavy focus on this need for accessibility and conveying the content and how it should be interacted with. Clarity will focus on things related to typography legibility, colour, graphics including pictures and illustration, interface elements and negative space.

Deference, which is used to categorise anything involving animation, blurring and shadows as well as any form of gradient in colours. These rules are used to keep content free of any unnecessary adornment and have a clean and crisp interface.

Depth is the focus on distinct layers of the user interface. These can be the navigation patterns and the ease at which you can intuit functionality. The focus for depth is to not lose context of what, where and how you will interact with the user interface.

Alongside the core themes in Human Interface Guidelines. There are also design principals that can be utilised in all the themes. These are aesthetic integrity, consistency, direct manipulation, and feedback.

Consistency and aesthetic integrity are often mistakenly considered the same. Being consistent in the case of an iOS application means making sure that throughout the application a user will feel a similar experience from colours, navigation, font, and behaviour. Aesthetic integrity is a focus on how well the actual functionality works with the appearance and category of the application. For example, if you are developing a note taking application it may requires subtle notifications and graphical content the is not obstructive. Apple takes the details of what your application intent is and the features you wish to make use of on a user's iOS device very seriously. Often rejecting applications that they consider out of context.

Direct manipulation and feedback are more to do with how a user interacts with your application via touch, rotation, gestures, or any other interaction method that is possible with an iOS handset. These interactions can be complex but must be fluid. Conversely to user's direct manipulation the feedback principal is about ensuring that feedback from your

application via haptics, notifications, acknowledgement, or any response to user interaction is fit for purpose and not in any way breaking the described dos and don’ts in HIG.

Apple through its design resources creates a high-quality standard that must be reached to be viable for distribution in its App Store. Implementing the rules contained withing HIG can lead to applications containing large amount of negative space and a somewhat bland feel to the applications if not branded correctly. The issues with how brand identity are implemented alongside user experience is oft not clear. Apple’s development community has many cases of refactoring due to cryptic reasons of rejection at the final steps of publication to Apples App Store.

2.2.2 Material Design

Material Design was first released in 2014 at Google I/O. It was and still is Googles design language containing a set of principles and guidelines to help govern application development on Android devices to give a more uniform interaction pattern for users.

Material design along with its design system recommendations comes with components which are interactive building blocks that are used to scaffold applications quickly. This is beneficial for applications that want to follow the Material design regardless of platform.

There are Material design implementations for both iOS, Flutter and Android native. To save time in Meetch development it could be argued to use one of these libraries to follow Material design on both iOS and Android. This was consciously avoided as it is believed using one of these libraries will negate the Apple user’s choice of using the iOS platform style of interaction and aesthetic integrity. It may be acceptable by Apple standards but not by Meetch as user experience is considered paramount and the user’s choice must be catered for fully.

Alongside the principals and components of Material design there is also theming. Material theming allows the incorporation of brands in a custom version of Material design. This is referred to as a Material theme. There is support for this in both web and Android platforms. Material design has a heavy focus on motion and animation. The user interfaces created with Material design can seem more cluttered than the iOS equivalents. An example of these differences can be seen in Figure 7

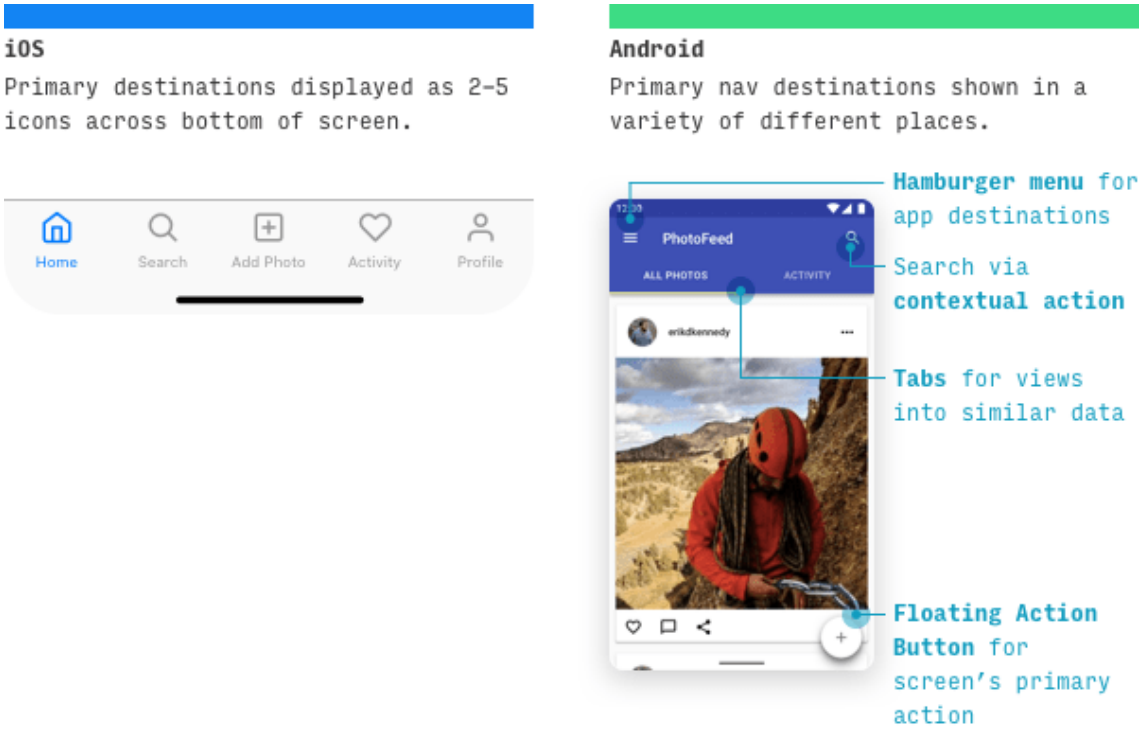

Figure 7: Primary Navigation Differences

In Apple the icons in the menu are consistent and firmly placed in a clear place. Android however has hamburger menu as part of Material design with contextual menus and controls that are animated.

To build Material designs, components or themes. There is a large amount of tooling provided by Google. That accelerates development quickly as it can be auto generated.

- Color Palette Creator
- Color Interface Tool
- Type Scale Creator
- Shape Customization Tool
- Google Fonts

The tools for Material design encompass multiple platforms. With this wide range of tools and support for many scenarios of application use together with comprehensive documentation. It makes Material design less of a design system and more like a design environment. If supporting a wide number of devices on multiple platforms including web. Material design is considered by many in the development community the easiest approach. Yet with all the clearly defined implementation guides of Material design. It does show immediately. It also has a strong connection to Google as it is used throughout its many applications. This can be

hard for your own brand to show through. In the case of Meetch this was the case for not fully utilizing Material design.

2.2.3 Atomic Design

The Atomic design is a methodology based on 5 specific principals. These principals are ordered in levels as seen in Figure 8. At the beginning of design systems, a lot of effort is put on the more traditional colour, typography, margin, padding, texture, and graphics. These are typically the fundamentals that are needed in order to begin building an Atomic design system.

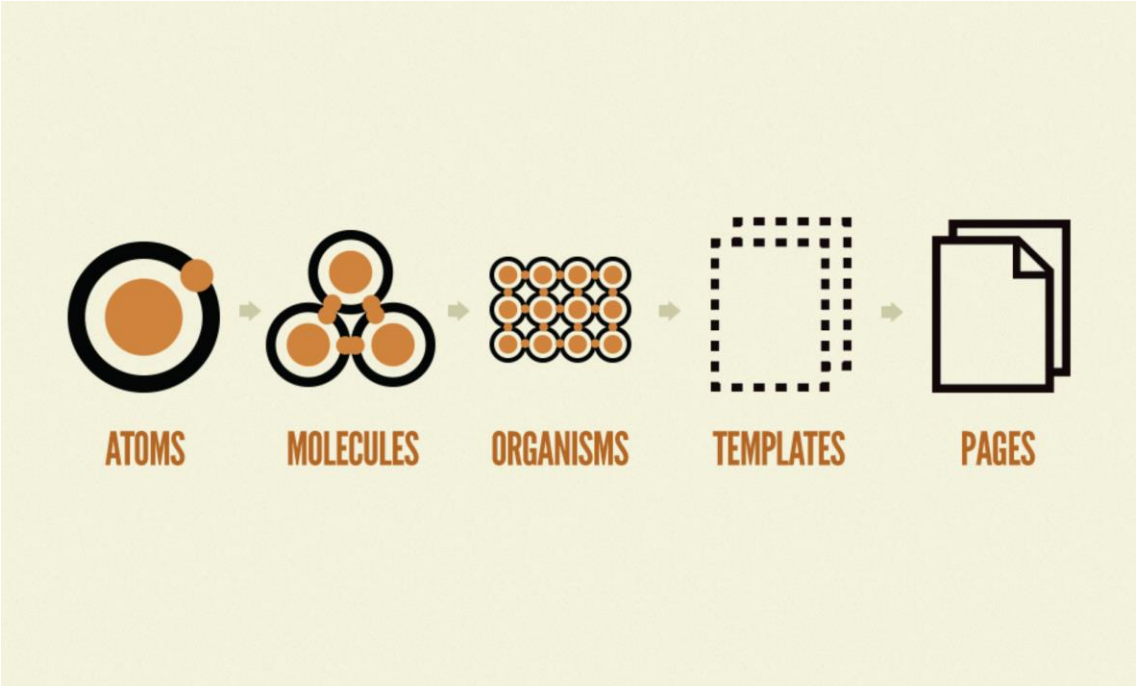

Figure 8: Atomic Design Principals

Each principal of Atomic design feeds the next as a constituent part. For example, an atom of design could be considered any smaller base part of the design. A button, icon, label, and input field could be considered atoms. These items in either design or code have little meaning beyond their atomic function as-is.

These atoms together however can make a more complex and meaningful patterns that can be clearly understood. The same atoms we mentioned before that are a button, icon, label and input when viewed individually are of the same use as their namesake but grouped together and given a context they become a meaningful design pattern that is clearly recognisable.

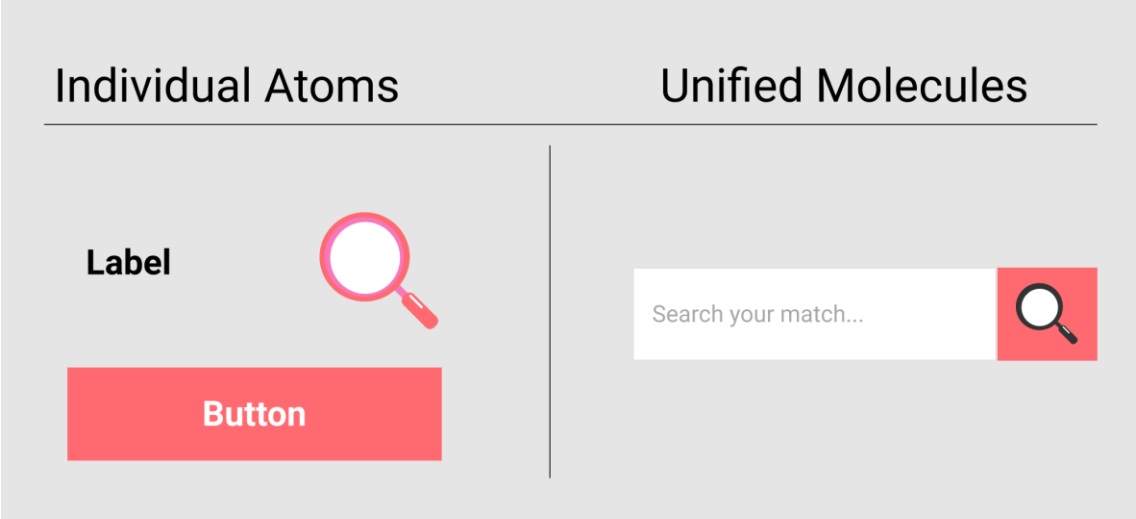

Figure 9: Atoms grouped to Molecules

As you can see in Figure 9 the individual atoms on their own are meaningless apart from their own context as a label, button, icon. Although it can be argued an button is a molecule but in this case it is a button without function which makes it meaningless.

Grouped together with context it is clear this is a search box that has a label, input field for search terms, button with an icon indicating search with a magnifying icon.

For developers it is easy to see similarities in atomic design to the way we create classes in object-oriented programming.

Organisms in this context could be specific parts of the application like navigation menu, messages view or login menu. These are all groupings of molecules that together bring a higher function.

Templates and pages are the highest fidelity form of Atomic design. Templates can be considered the raw template of the design without finished content or context. It would just be placeholders and the layout of organisms. When final content such as graphics are added to the template and there is a context given to it then it is considered a page. An example of context on an Atomic design page could be visualizing message received on chat window. Another context could be visualising the same chat received window, but 1000 messages received instead. This form of design is perfectly suited to the throwaway prototyping that is practiced in Meetch as only the accepted designs will progress and these assets will be turned into reusable code representing atoms, molecules and organisms which will then be in turn used to create templates and pages.

This is not a new way of working in a development environment as code reusability is always a focus in good software engineering. This form of Atomic design coupled with Flutter and the

adapted rapid development methods allow a more precise decision on the priorities of design and development.

3 Development methods

3.1 Adapted Rapid Development Framework

Meetch manages its work practices through a hybrid setup of Agile software development processes, Rapid Application Development and Kanban practices (2021). The use of throwaway prototyping is especially prevalent. Which according to Martin, J (1991) is the practice of exploring the factors critical to the systems success and then throwing that code away. The prototyping does not necessarily need to use the same programming languages and or tools as the official implementation. The rapid application development methodology can

be seen here in Figure 10.

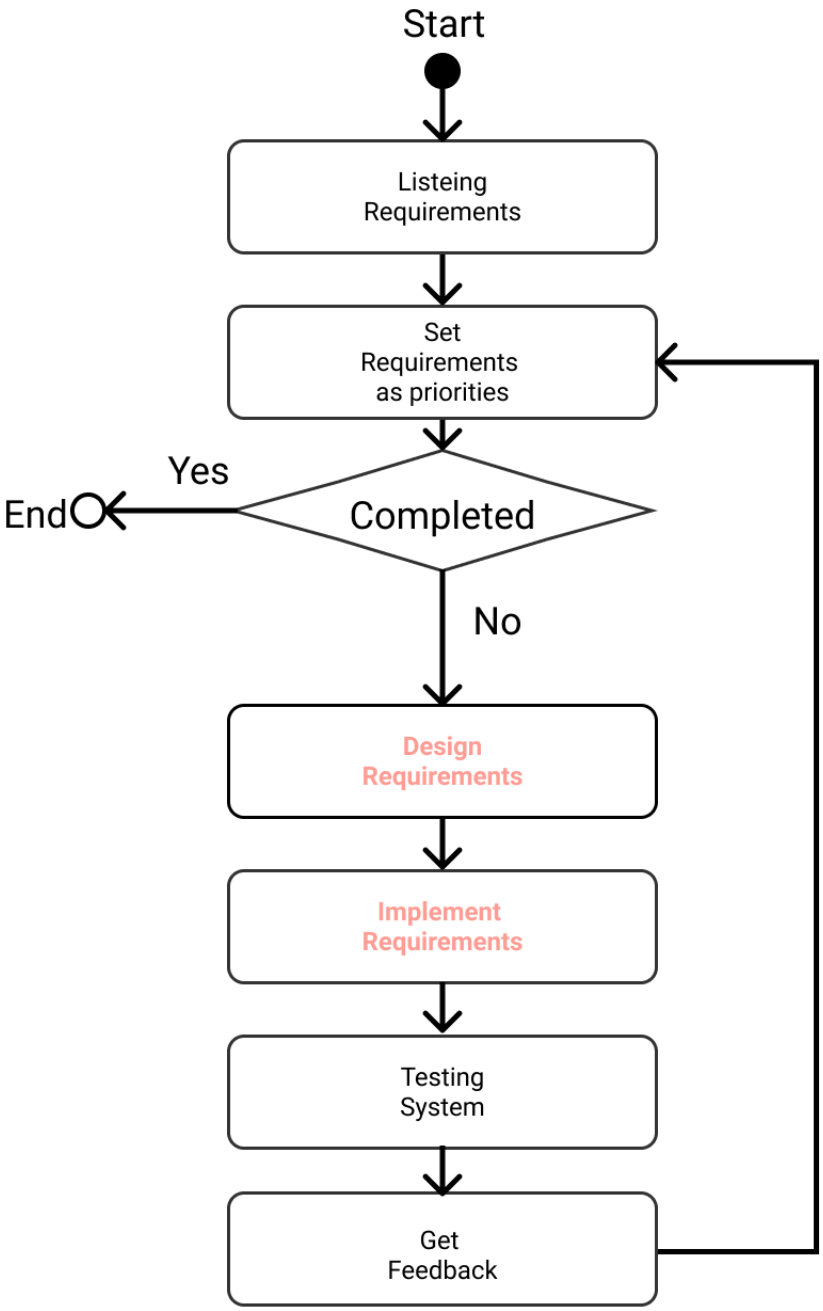

Figure 10: Rapid Application Development Methodology

User interfaces especially are prototyped far more than other parts of the system. This has allowed the listing requirements quickly which are in turn planned as priorities for implementation. This rapid development is contextually based on singular platforms. It does not account for the contextual need of a multi-platform service required by users in 2021. This is also a factor in the decision to use Flutter.

Meetch seeks to enhance the rapid application development method even further by focusing on managing the throwaway prototyping in such a way that the defined assets for design of the user interface can be re-used in a design system library. This will avoid in single platform context mentioned earlier which is considered detrimental to speed of implementation in Meetch development.

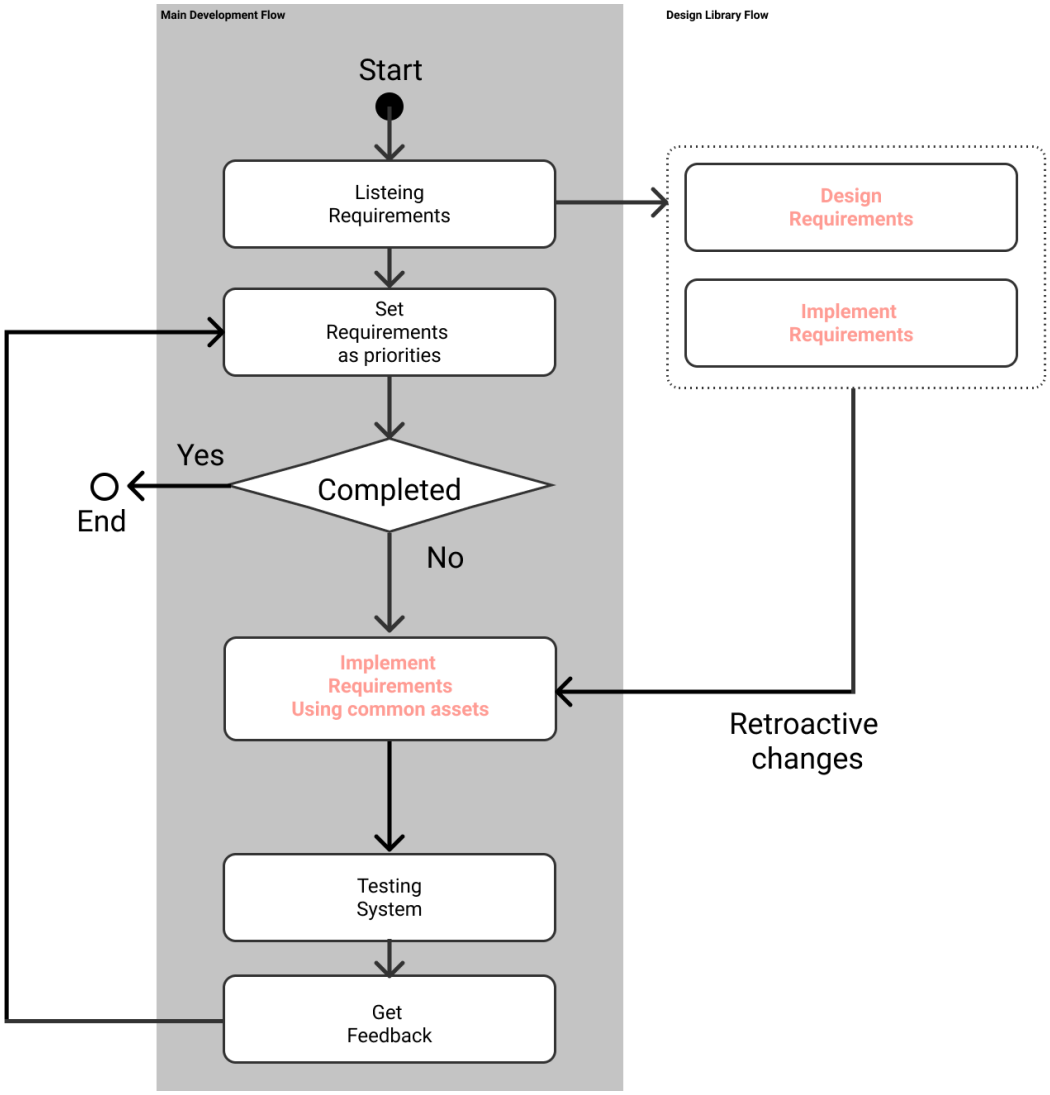

Figure 11: Adapted Rapid Development Methodology

As you can see in Figure 11 there are two flows that are running concurrently as part of the development lifecycle. The main development workflow is concerned with getting the critical functionality of the application running. The design is created during this process but is not delivered in a definition of done. It is instead created as a placeholder and uses design tokens, assets and widgets which are a Flutter specific term for any enclosed functionality. Everything in Flutter can be a widget. The design library flow is where these placeholders are

populated with the results of listed requirements, design requirements and results from throwaway prototyping.

The value of this methodology is that when coupled with Flutter as a development framework. This does not require monolithic advancement of tasks. Each flow can work independently of each other and during respective iterations assets can be used from the design system library or planned to be implemented from listing requirements work.

3.1.1 Kanban

To manage, prioritise and track requirements through the development life cycle. Kanban practices like those listed by Atlassian (2021) were adopted. Specifically, the restrictions of work in progress items (WIP) that are used in Kanban. In Figure 12 an example Kanban board can be seen with Max 3 items in progress.

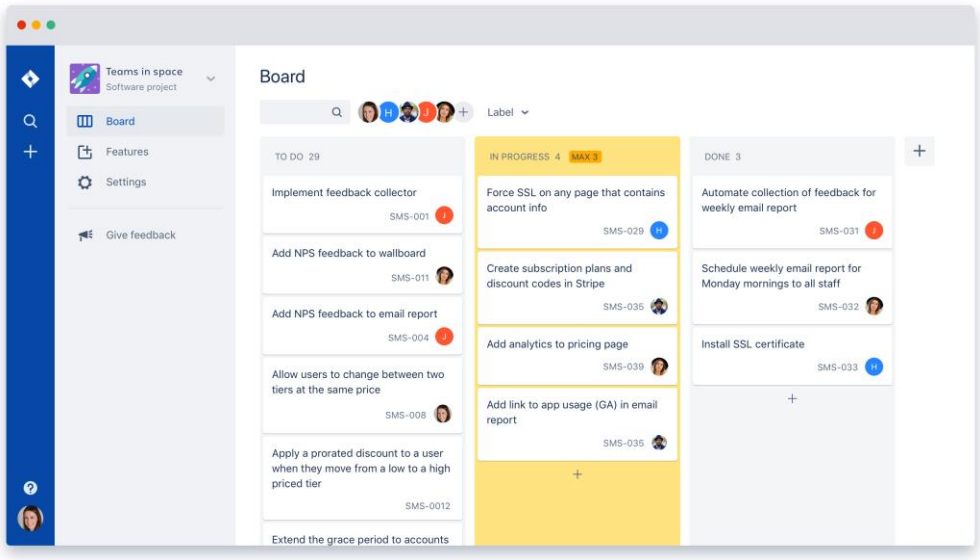

Figure 12: Example Kanban board

In Jira, like the example seen in Figure 12. Exceeding the WIP items is possible with a warning. This is strictly not allowed within the Meetch workflow.

Requirements that have been listed and then prioritised are known as features in this Kanban way of working. These features can be broken down into smaller user-stories that may or may not also need their own sub tasks. This depends entirely on complexity and is subject to review at any point during the iteration period. The iteration period is no more than 2 weeks. It is also directly mapped to the start and end points shown in Figure 11.

The current limit during this thesis is 4 work items in progress at any given time. The reasoning for this approach is to keep rigid focus on the actual value of an item of work. Work is completed or returned to the backlog with clear reasoning of why it is returned. Thus, allowing accurate mapping of where work progress is at any given time.

4 Implementation of project

4.1 Visual audit

The visual audit of Meetch was done on minimal design fundamentals and all subsequent iterations to the final design were done through throwaway prototyping. Working with the Meetch designer each atom was individually reviewed for its potential use. Instead of styling all possible system components on Android and iOS. A focus was given to what is the bare minimum we can use to meet the needs of the listed requirements. This practice allowed only the things that worked for Meetch and its users to progress further. The tool used for this collaborative approach was Figma as it allows easy sharing of the assets.

Our goal for the visual audit was not to create atoms, molecules or organisms as seen in Atomic design. It was to get the fundamental properties of colour, spacing, icons and typography defined (Figure 13).

Figure 13: Meetch fundamental design assets

Meetch design was created to be inviting, harmonious, natural, and relaxing. As such a simple set of colours was selected to embody these choices (Figure 14).

Figure 14: Meetch colour palette

Rather than using SF Symbols or Material icons which distinctly identify as patterns from Apple and Google respectively (Figure 15) a custom set of icons was created for use by Meetch seen in Figure 16.

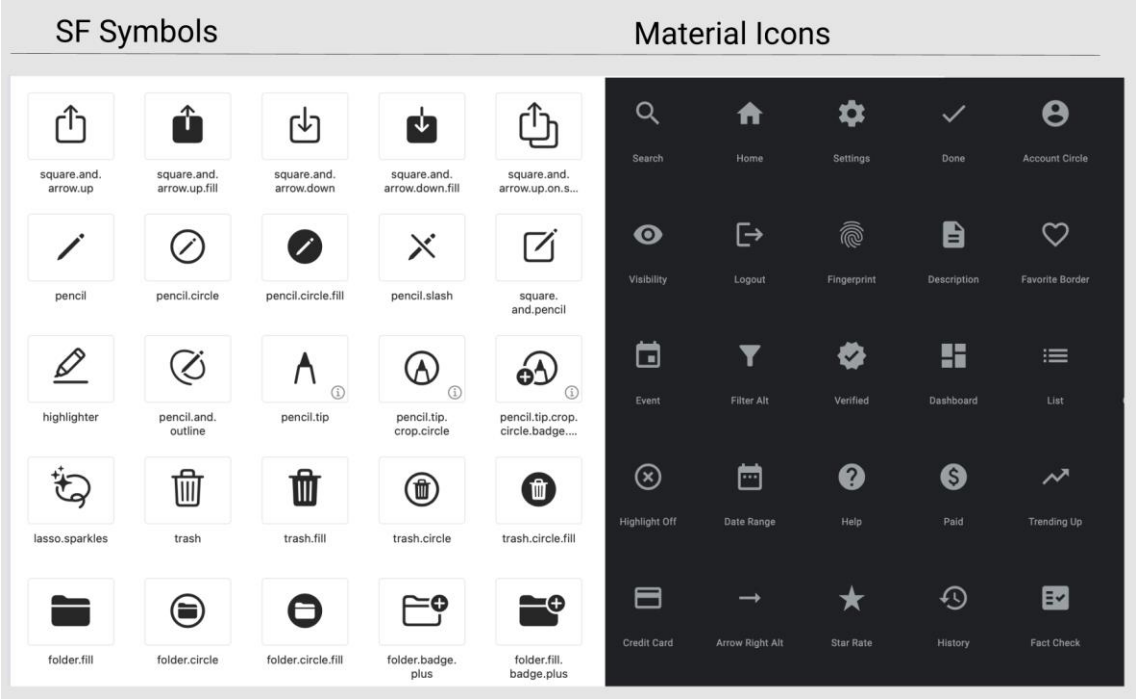

Figure 15: SF Symbols and Material icons

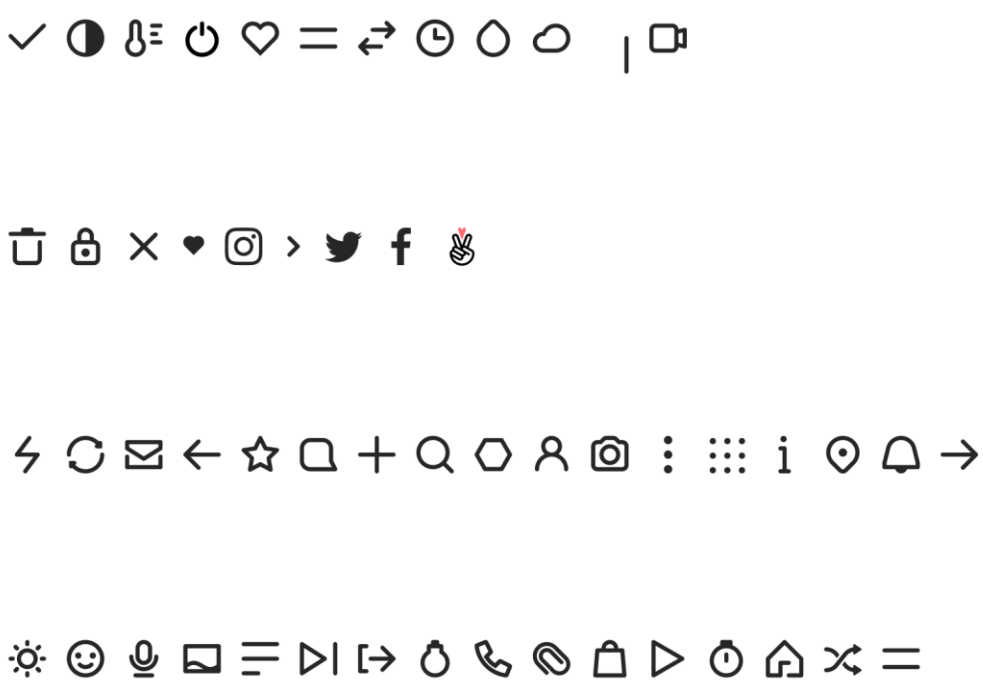

Figure 16: Meetch icon set

The one element that has been taken from Material design is Roboto font typography (Google Fonts, 2021). These fonts are distinctly legible by design and is also the default font type on Android. Meetch accepted Roboto as the chosen font due to its popularity across multiple platforms and the legibility it provides on mobile devices. In Figure 17 you can see some of the styles that were considered.

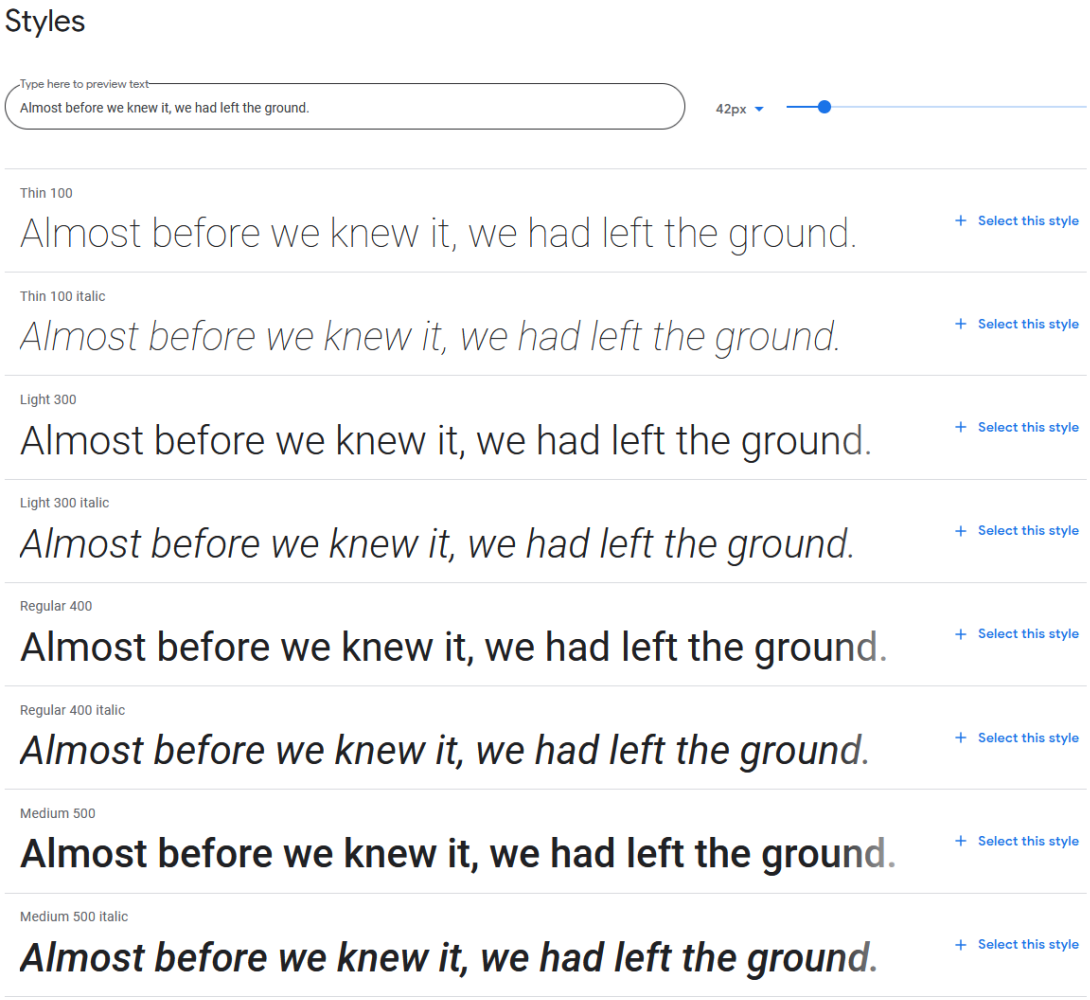

Figure 17: Roboto font styles

The combinations of Meetch brand colours, custom icons and Roboto typography is used to build the unique style of Meetch. The design assets compared with other leading brands can seem sparse. Especially when compared with a large ecosystem like Material design.

With Meetch focusing on speed the usage of these minimal artefacts during implementation gives more room to focus on user experience. When developing the prototypes, it became apparent more needed to be done on delighting the user. Functionality of the application on multiple platforms had been solved using Flutter packages. Negative space and a bland application user experience were the results only the fundamentals being used.

4.2 Benchmarking

The initial page designs, although compliant with Apple's Human Interface Guidelines and Material design do not lead to an engaging user experience (Figure 18). To test if introducing

motion design with animations and video content through multiple prototypes were generated with picture, video, and illustration content.

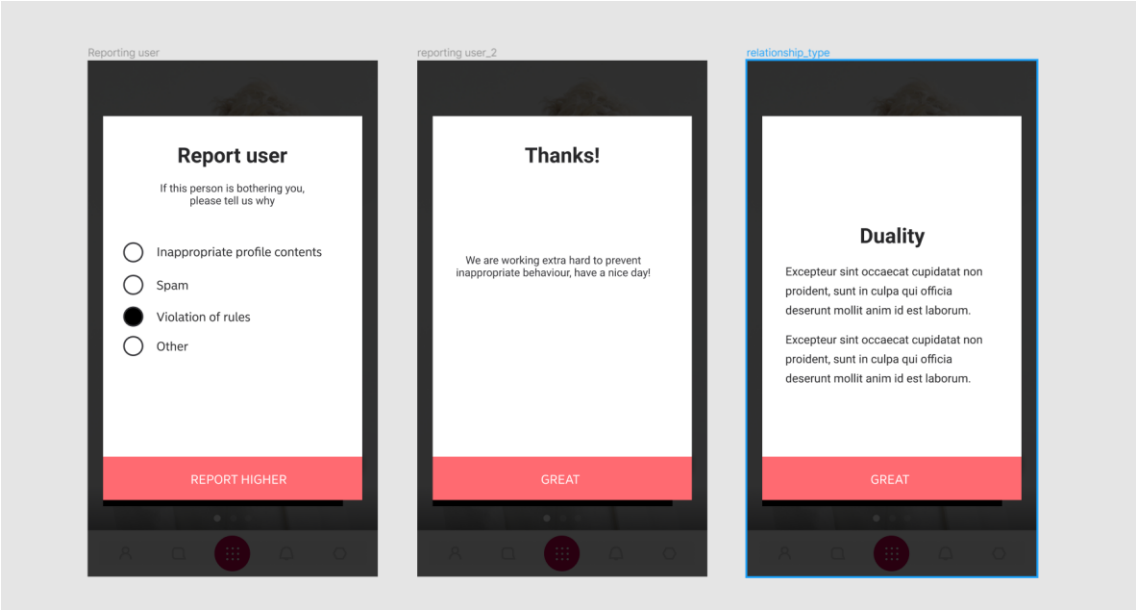

Figure 18: Bland initial atomic pages

There were only 2 specific checks when reviewing any of the prototypes.

- Does this require maintenance?
- Does it break the user experience on their chosen platform?

When reviewing based on maintenance it is crucial that we do not have individual ways of maintaining UI on Android or iOS. All due effort is given to find an approach that allows a single implementation usable on both platforms.

Flutter provides widgets that are used for creating material applications with flutter. It is based on Material design but respects the navigation patterns of iOS. We use these widgets when implementing the fundamentals of the design system that will be form the building blocks of more complex atomic items.

Reviewing based on user experience of chosen platform is difficult as there may be cases when a decision is made that is favouring one platform. An example of this type of decision is the use of Roboto font. It is clearly identifiable as part of Material design. However, it was an accepted decision due to the legibility this font provides.

4.3 Observations based on benchmarking

According to Vyrazu Labs (2021) Material design enhancements from Google focus more on transitional effects through fading user interface elements. For Google, the principals of motion are aimed at being:

- Informative: Highlighting clear connections between user interface elements.
- Focused: Not creating unnecessary distraction from key contextual information
- Expressive: Providing clear and understandable feedback to the user on interaction choices or results of these choices.

Meetch aligns fully with the "Focused" principal and avoid unnecessary motion in the user interface. The aim is to focus the user experience on the interesting content and not the animated user interface itself. Another reason is that the motion system from Material design is not applicable in iOS (Material Design, 2021).

The negative space from observing Human Interface Guidelines and the choice not to use motion according to Material design created the alpha release of Meetch application. It is considered working but not engaging. This left us with enhancing the fundamentals of the application. Using emoji and illustration based on the fundamental design fulfilled the criteria setup in the visual audit.

Using emoji and graphical illustrations

- Does this require maintenance?
    - No, it is based on Scalable Vector Graphics which is easily supported
- Does it break the user experience on their chosen platform?
    - No, as it is based on the brand identity. User choices are still respected.

Using video in the application

- Does this require maintenance?
    - Yes, depending on format and screen size we may need to maintain higher test standards. Video playback is also different on each platform.
- Does it break the user experience on their chosen platform?
    - Yes, is you have a smaller screen size you do not get the same user experience.

Using motion animated UI

- Does this require maintenance?
    - Partly, iOS does not fully support motion in the UI as Android does.

- Does it break the user experience on their chosen platform?
    - Partly, if iOS makes changes in the future that does not support this UI we will have a breaking change.

Here we can see the difference small but purposeful additions based on a clear criterion have had for the UI in just some of the screens (Figure 19).

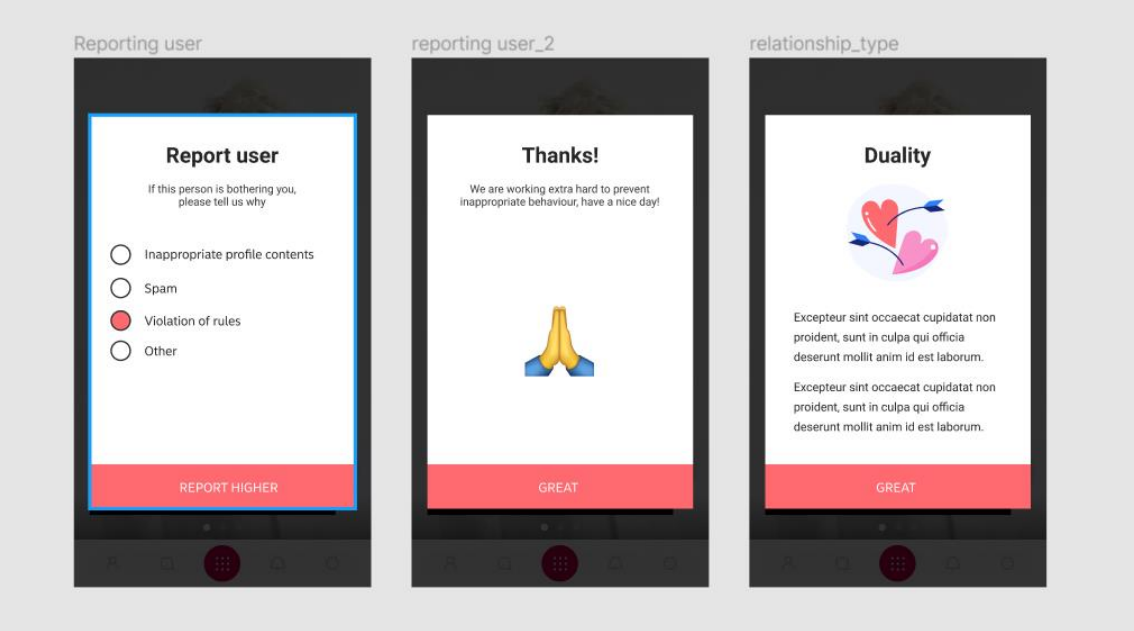

Figure 19: Illustrations and emoji added

4.4 Implementing design rules in Meetch

4.4.1 Fundamentals

To implement the fundamental design each design artefact was created as its own reusable class in the Flutter code base. An example of such is how we handle the colours (Figure 20).

```
import 'package:flutter/material.dart';

Color mytypeRed = Color(0xFFFF6A71);
Color mytypeLightGray = Color(0xFFD0D0D0);
Color mytypeMediumGray = Color(0xFF949494);
Color mytypeLightGrayBG = Color(0xFFF6F6F6);
Color mytypeDarkGray = Color(0xFF262628);
Color mytypeBlue = Color(0xFF88A4F5);
Color mytypeLightBlue = Color(0xFFE9F1FF);
Color mytypePurple = Color(0xFF915FB5);

const MaterialColor white = MaterialColor(
  0xFFFFFFFF,
  <int, Color>{
    50: Color(0xFFFFFFFF),
    100: Color(0xFFFFFFFF),
    200: Color(0xFFFFFFFF),
    300: Color(0xFFFFFFFF),
    400: Color(0xFFFFFFFF),
    500: Color(0xFFFFFFFF),
    600: Color(0xFFFFFFFF),
    700: Color(0xFFFFFFFF),
    800: Color(0xFFFFFFFF),
    900: Color(0xFFFFFFFF),
  },
);
```

Figure 20: Colour definitions in Meetch

mytype was the development prototype name before it became Meetch officially. We can see an import of a material library from flutter. This import refers to the use of “MaterialApp” in Meetch. Now that these colours are implemented in the code repository, they can be used in the atoms of design. The following fundamental artefacts have also been implemented in a similar approach.

- Spacing and sizes
- Typography

These are now easily accessible for use during the main development flow. Graphical content is implemented through the assets folder and naming convention. Since the decision was to make use of illustrations and icons to build the brand identity. They need to be easily accessed. In dynamic application that make use of a RESTful application programming interface. Image assets are normally fetched from an endpoint which means they can be changed at the endpoint and on the next fetch the update asset can be provided.

In Meetch we have made the decision to embed these image assets so that offline interaction is also an enjoyable experience that does not break the brand identity. The image assets are

made in a scalable vector format which is a small footprint and can be scaled to any size. The pubspec file of a Flutter application is used to reference directories that are needed to be accessed by the application.

Declaring the files in the following directories allows us to use the graphical content within as assets in the application (Figure 21).

```
# To add assets to your application, add an assets section, like this:
assets:
  - assets/animations/
  - assets/icons/
  - assets/images/
  - assets/images/relations/
  - assets/images/types/
```

Figure 21: Image asset declaration

Any file that is located inside the asset folders can be referenced by its name. You do not need the full source file path, only the asset/folder/name is required.

Fonts are also structured in the same way. The folder needs only to be listed in the pubspec which will then expose the targeted font to the application as an asset (Figure 22).

```
# To add custom fonts to your application, add a fonts section here,
# in this "flutter" section. Each entry in this list should have a
# "family" key with the font family name, and a "fonts" key with a
# list giving the asset and other descriptors for the font. For
# example:
fonts:
  - family: Raleway
    fonts:
      - asset: fonts/Raleway-Light.ttf
      - asset: fonts/Raleway-Regular.ttf

  - family: Roboto-Bold
    fonts:
      - asset: fonts/Roboto-Bold.ttf

  - family: Roboto-Light
    fonts:
      - asset: fonts/Roboto-Light.ttf

  - family: Roboto-Regular
    fonts:
      - asset: fonts/Roboto-Regular.ttf

  - family: OpenSans-Regular
    fonts:
      - asset: fonts/OpenSans-Regular.ttf

  - family: MeetchIcons
    fonts:
      - asset: fonts/MeetchIcons.ttf
```

Figure 22: Font declaration

We can see for the typography implementation as seen in Figure 23 that we are also importing the colors.dart file. This is so we can create more complex typography for use in molecules and organisms of our user interface.

```
import 'package:flutter/material.dart';

import 'package:bmflutter/widgets/styles/colors.dart';

var heading = TextStyle(
    fontSize: 24, fontWeight: FontWeight.bold, fontFamily: 'Roboto-Bold');
var subHeading = TextStyle(
    fontSize: 18, fontWeight: FontWeight.bold, fontFamily: 'Roboto-Bold');
var smallBold = TextStyle(
    fontSize: 16, fontWeight: FontWeight.bold, fontFamily: 'Roboto-Bold');
```

Figure 23: Typography implementation

4.4.2 Component level iteration

When creating components, we focus on reusing the definitions for colour, sizing and spacing that have been implemented into the application. At no point must we manually declare these values. Instead, we must use their tokenised names. For example, declaring the colour red in a Flutter widget can be done with:

- Color(0xFFFF6A71)
- mytypeRed

The first is a hardcoded value. The second is a reference that we have given a hardcoded value. If we hardcode any values into a component, then we will break the ability to have design library flow work concurrently which will in turn cause regression and loss of speed to Meetch's development cadence. The ability to change the entire design from the design library is also lost if values are hardcoded into components.

We should however create components that accept these tokens. In Figure 24 we can see that it accepts 3 variables.

- buttonText – The text that will be shown on the button.
- onClick – An action to be carried out triggered by clicking.
- buttonColor – Colour variable.

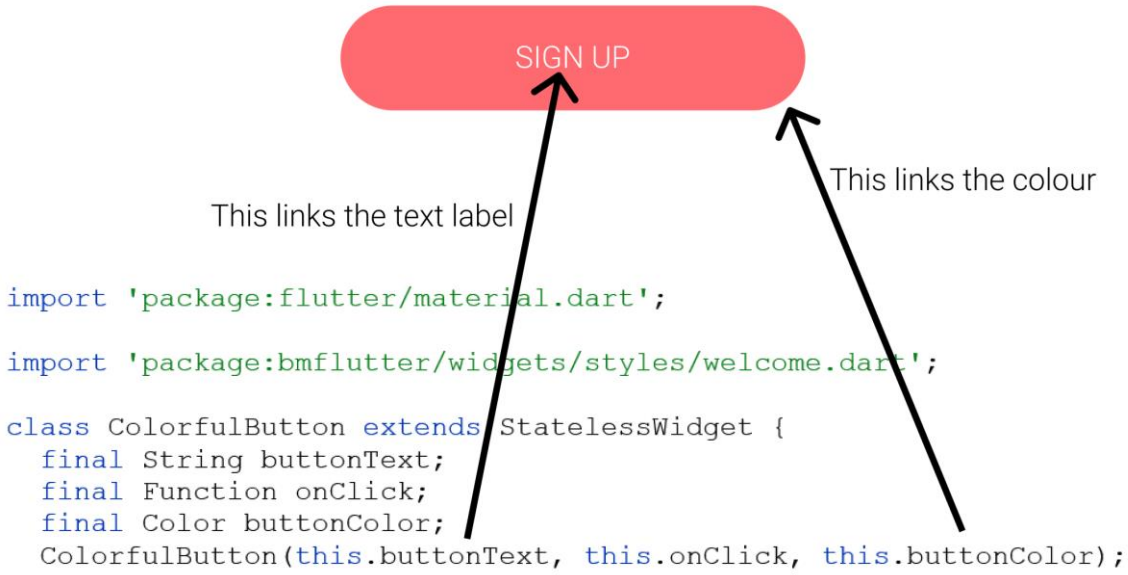

Figure 24: ColorfulButton widget properties

Components are created with the ability to accept fundamental values instead of being hard coded values. This allows us to make use of these as molecules reusable fashion. Depending

on where this component is used its context may change. In the login screen (Figure 25) of an iOS build and Android build of the application we can see the same button used.

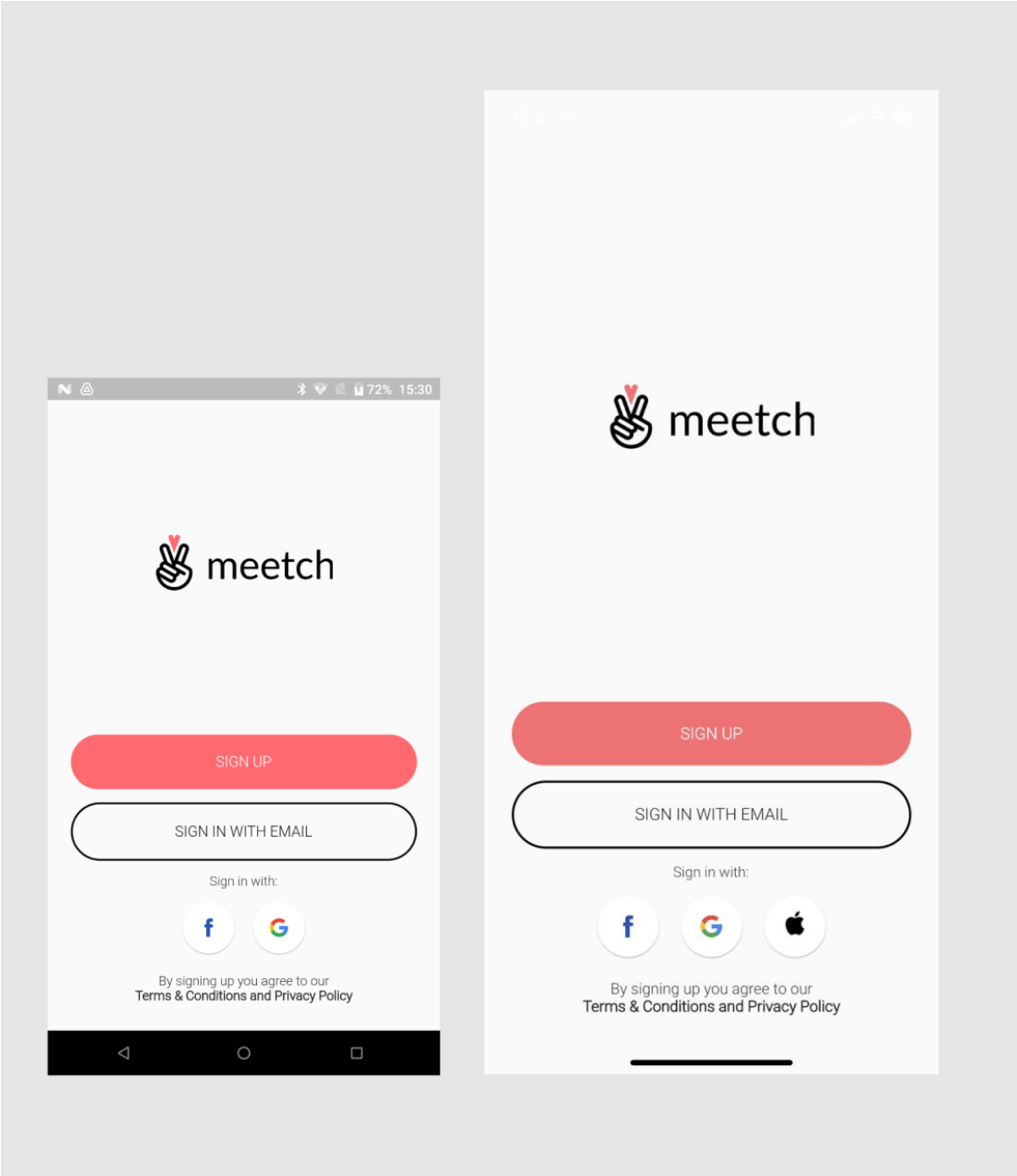

Figure 25: Android and iOS Login screen of Meetch

These are all the same widget that has been reused. Thus meaning we don't have 2 types of button, a primary and secondary for example. We also do not have an iOS implementation and an Android implementation. This can be viewed as 4x less the effort for implementation. This may not be true in all instances of reuse. In the conclusion we will explore some of the challenges faced.

4.4.3 Patterns of Atomic Templates and Pages

Patterns based of components are based on the 4 navigation items in the bottom tab bar (Figure 26). These pages are fulfilling the requirements initially set out by Meetch. The design is not the same as was anticipated before implementation of the design artefacts in the code repository. A clear set of fundamental assets was created. Due to the very minimal approach that was taken and by rigidly throwing away things that did not fulfil the criteria of working on both iOS and Android this allowed a better brand identity to be created.

While working on the using pictures as part of the visual identity, even though it was finally rejected. That work influenced the large full screen portrait view of a user's profile picture in the feed page.

This has become a unique selling point of the feed and something that users now actively enjoy. Without the adapted process of rapid development, it is thought this may not have been developed.

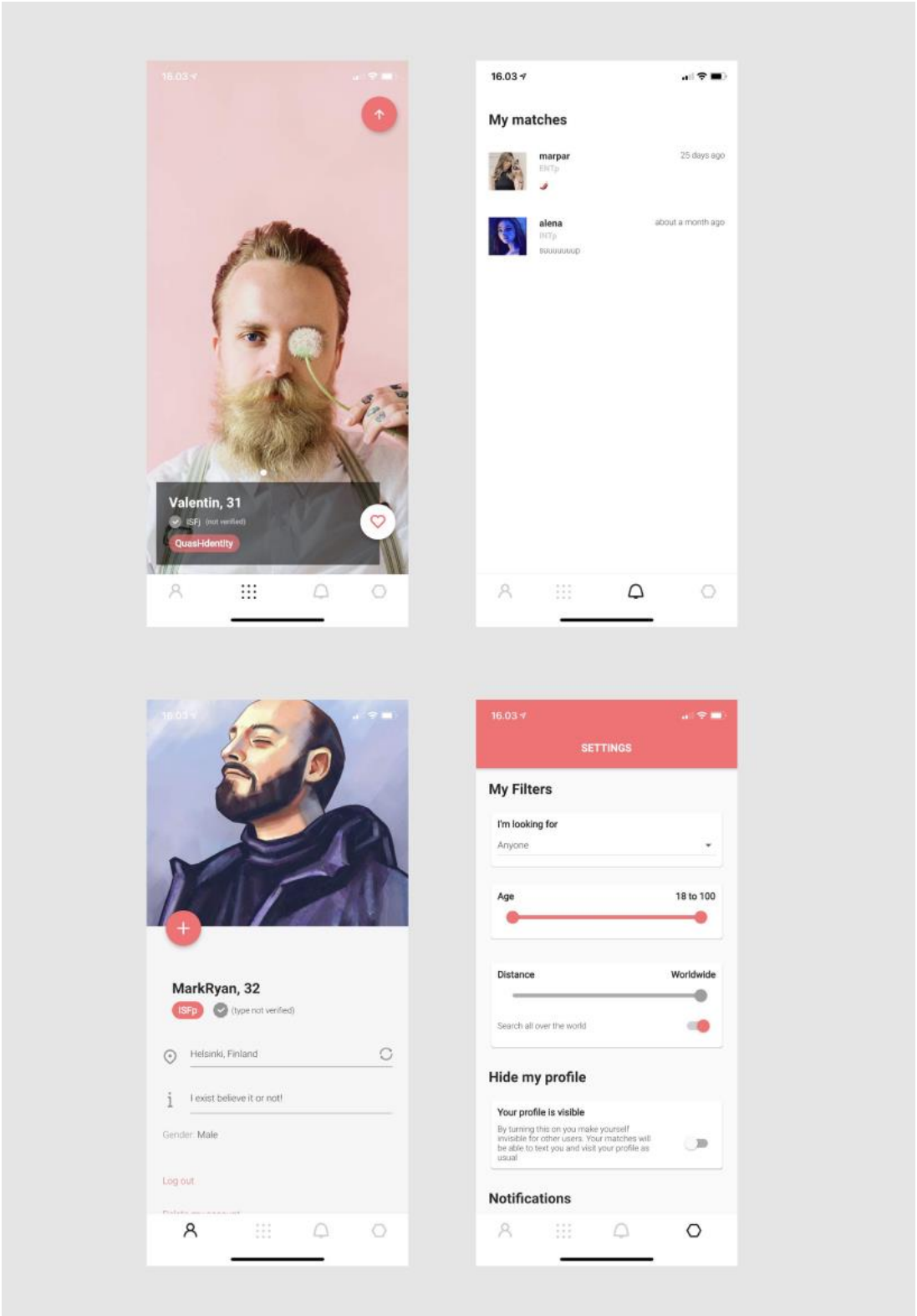

Figure 26: Atomic Pages in Meetch

5 Results

The focus of this thesis was on speed of iteration. Based on the bitbucket commits of recent development, the cadence is thought to have had some effect but not noticeable enough to be considered beneficial to speed.

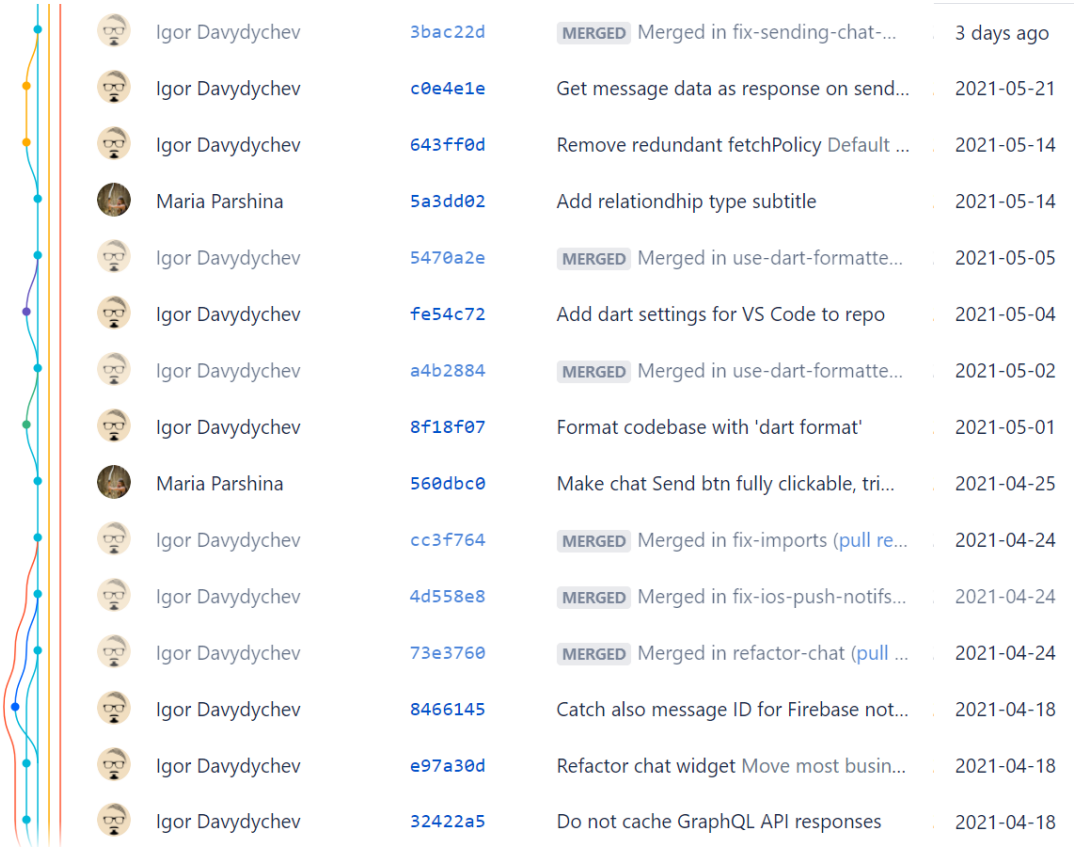

Figure 27: Snapshot of Bitbucket commits.

We know that on average the time to merge new changes into the codebase after branching is approximately 7 days. This is misrepresented due to incomplete branches in the codebase and the final value being during a team members holiday. The team size is 5 person including the thesis author.

Commit	Branched	Merged	Merge Commit	Days taken
32422a5	18/04/2021	24/04/2021	73e3760	6
e97a30d	18/04/2021	24/04/2021	4d558e8	6
560dbc0	24/04/2021	02/05/2021	a4b2884	8
fe54c72	04/05/2021	05/05/2021	5470a2e	1
5a3dd02	14/05/2021	28/05/2021	3bac22d	14
			Average Feature delivery	7

Table 2: Merge commit avg

6 Conclusions

Larger sample data is not needed as the conclusion is already accepted that the speed has not changed since the creation of the design library. Meetch did however find more value in the rapid development process and use of Kanban WIP limits. These 2 items are thought to be responsible for uncovering unique iterations of Meetch design to deliver better user experiences. These practices will continue within Meetch in the future. Throwaway prototyping is also considered a successful process that helps frame the requirements of work items that need to be prioritised.

The Meetch application is now released and can be found in both iOS and Google play stores. The iteration of new requirements will continue with a focus being but on business-to-business service next.

The current user base is above 1000 users and is expected to grow as marketing makes use of the design artefacts to market it to target audiences.

References

Printed

Gulenko, V. Psychological Types: Why Are People So Different?: 64 Portraits in Socionics. How each of the 16 Jungian types varies in 4 main ways. 2019

Martin, J. Rapid Application Development. 1991

Martin, Robert C. Agile software development: principles, patterns and practices. 2003

McConnel, Steve. Rapid Development. 1996

Electronic

Android Developer, 2021. Android Version Usage. Accessed 25 May 2021 https://developer.android.com/about/versions/android-4.1.html

Android, 2021. Android Architecture. Accessed 22 May 2021 https://source.android.com/devices/architecture

Android, 2021. Kernel Fragmentation. Accessed 24 May 2021 https://source.android.com/devices/architecture/kernel/generic-kernel-image

Android, 2021. What is Android. Accessed 22 May 2021 https://www.android.com/what-is-android/

Appbrain, 2010. Top Android OS Versions. Accessed 28 May 2021 https://www.appbrain.com/stats/top-android-sdk-versions

Apple Developer, 2021. App Store Version Usage. Accessed 27 March 2021 https://developer.apple.com/support/app-store/

Apple, 2018. The App Store turns 10. Accessed 27 March 2021 https://www.apple.com/newsroom/2018/07/app-store-turns-10/

Apple, 2021. Human Interface Guidelines. Accessed 27 March 2021 https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/

Atlassian, Kanban Practices. Accessed 16 March 2021 https://www.atlassian.com/agile/kanban

European Centre for Disease Prevention, 2021. COVID-19 Situation Dashboard. Accessed 1 April 2021. https://qap.ecdc.europa.eu/public/extensions/COVID-19/COVID-19.html#global-overview-tab

Flutter, 2021. Accessed 9 March 2021 https://flutter.dev/

GDPR, 2021. General Data Protection Regulation. Accessed 29 March 2021 https://gdpr.eu/data-privacy/

Google, Google Fonts, 2021, Accessed 24 May 2021 https://fonts.google.com/specimen/Roboto

Google, Material Design, 2021. Accessed 15 April 2021 https://material.io/design

Meetch, 2021. Accessed 1 March 2021 https://www.meetch.app/

Samsung, 2021. One Ui. Accessed 22 April 2021 https://www.samsung.com/us/apps/one-ui/

Vyrazu Labs, 2021. Accessed 13 May 2021 https://www.vyrazu.com/material-design-vs-human-interface-guidelines/

outsystems, 2021. The Speed of Change. Accessed 24 March 2021 https://www.outsystems.com/1/speed-change-app-dev/

Figures

Tables