12/26/2022

Assignment 4

**FA19-BSE-(085-093-109-114)**

**Submitted to: Dr. Fatima Sabir**

# Introduction:

Snapchat is an American multimedia instant messaging app and service developed by Snap Inc., originally Snapchat Inc. One of the principal features of Snapchat is that pictures and messages are usually only available for a short time before they become inaccessible to their recipients. The app has evolved from originally focusing on person-to-person photo sharing to presently featuring users' "Stories" of 24 hours of chronological content, along with "Discover", letting brands show ad-supported short-form content. It also allows users to store photos in a password-protected area called "my eyes only". It has also reportedly incorporated limited use of end-to-end encryption, with plans to broaden its use in the future.

# History Of Snapchat:

When did Snapchat start?

Snapchat is released in the App Store in the living room of Evan Spiegel's father's home in July 2011.

However, Snapchat was originally named “Picaboo” until the founders were sent a cease-and-desist from another company who had already trademarked the Picaboo name.

**October 29, 2012**

Snapchat launches their app in the Android Store

**October 2013**

Snapchat launches 'Stories'

**November 2013**

Around two years after Snapchat launched, Mark Zuckerberg attempted to buy Snapchat for 3 billion dollars in November of 2013.. but his offer was refused.

**August 2014**

Snapchat first introduced Geofilters to the public in August 2014.

**End of August of 2014**

40% of adults in the United States were using Snapchat on a daily basis.. and the rest is history.



# What refactoring Strategies they use?

Rewriting code is fun, and old code often looks ugly and hard to comprehend. In the process, it’s important to not get carried away into rebuilding more than what you need. This, more than anything, is what we thought could add risk to our timeline and introduce scope creep. We wanted our new app to be much better, but we did not need to solve every single problem at once.

After experimenting with several ideas and brainstorming, it became clear that we could cut down the scope by postponing work in many areas:

* We would not add new features. With small exceptions, our Android app feature set was frozen for 6 months during the rewrite.
* We would not change the app UI. This turned our rewrite into a pure engineering problem, and also allowed us to do an apples-to-apples comparison.
* We would not make any changes to the client-server protocol, unless explicitly needed.
* We would not rewrite components of our application that were already isolated and of good quality simply to adopt new languages or libraries.
* We would not change our build systems, CI, QA or release processes.

As an example, at the time of the rewrite our app heavily relied on JSON to make network requests. We knew JSON was inefficient and expensive to parse, and wanted to move to a more modern solution. However, doing so would take longer as we needed to change our client-server protocols and endpoints and do a careful migration. Instead, we adopted an intermediary solution where we introduced a centralized network manager API, which hid the usage of JSON as an implementation detail. This pattern of centralizing areas of future improvement behind APIs became widely used.

# How they apply Re-write.

Rewrites can have positive results, but they are challenging. In order to develop the strategy, the app’s performance thoroughly researched before starting. Once they began, a tremendous amount of coordination and support was needed to keep the project on track, and they used and tested the new app from early on to keep the quality high

## The Problem:

The main motivation to rewrite was to improve app performance. Snapchat is a camera app, optimized for capturing fleeting moments. If the app takes too long to load, the moment can be lost. Like many apps that evolved at a fast pace, Snapchat grew in complexity into a new problem space that it wasn’t originally designed and built for.

The features in the code base were tightly coupled in a way that hurt flexibility. Particularly troublesome was the lack of a structured way to initialize and schedule work. A lot of code and data was loaded immediately at startup by features that were not part of the startup path, in a way that was hard to reason about or unwind. So much work was done at startup that it took the app 30-60s to settle down, leaving a large memory footprint to be carried throughout the app session.

When thinking about the future, we wanted an app that would:

* Be performant: The app would load instantly and feel fast.
* Allow quick iteration: The app should be easy to develop on.
* Be sustainable: As we added new features, the app would remain fast to use and build.

# Deciding to Rewrite

When considering rewriting an app, there’s some fairly common wisdom across the industry: don’t do it! The idea that could overhaul 5 years of work and re-implement it better in a fast timeline seemed optimistic. They would inevitably introduce new bugs, which could take a long time to address. Even if they’d pulled it off, who is to say it wouldn’t regress back to the old state after? There would be a lot of pent up demand for launching new features after the rewrite, and launching features fast is what created our issues in the first place.

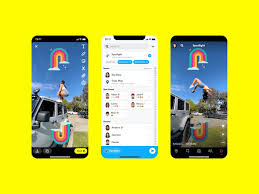
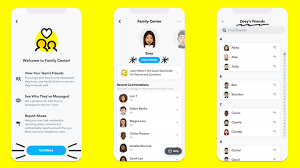
They pushed ahead with a rewrite because we believed it would be faster and less risky than an incremental refactor. The existing app was complex and interconnected. They believed issues introduced by the rewrite would be easier to find and fix than the ones caused by the current, complex system. Their leadership fully supported this decision, giving us the latitude to plan for a successful rewrite.

To address some of the coming challenges, we adopted the following strategies:

## The Ground Rules:

Snapchat consists of many small apps rolled into one, including camera, chat, memories, photo editing, content consumption, and a map. It opens to the camera, which is resource intensive in both memory and CPU, and includes AR lenses and a lot of heavy media content. Combining these features together into a single app makes for an engaging user experience, but presents a hard engineering challenge.

In the Android operating system, users can have many different apps installed on their phone, and each app is able to load fast and perform smoothly in isolation. We started seeing our Android app as a mini OS, and our features as mini apps running inside of that OS. If each mini app could be made to load fast in isolation, it should be possible to combine them while keeping performance high, without the need to preload features at startup.

Graphical user interface, application

Description automatically generated

## Focus:

After experimenting with several ideas and brainstorming, it became clear that They could cut down the scope by postponing work in many areas:

We would not add new features. With small exceptions, our Android app feature set was frozen for 6 months during the rewrite.

We would not change the app UI. This turned our rewrite into a pure engineering problem, and also allowed us to do an apples-to-apples comparison.

They would not make any changes to the client-server protocol, unless explicitly needed.

They would not rewrite components of our application that were already isolated and of good quality simply to adopt new languages or libraries.

They would not change our build systems, CI, QA or release processes.

As an example, at the time of the rewrite our app heavily relied on JSON to make network requests. They knew JSON was inefficient and expensive to parse, and wanted to move to a more modern solution. However, doing so would take longer as we needed to change their intermediary solution where we introduced a centralized network manager API, which hid the usage of JSON as an implementation detail. This pattern of centralizing areas of future improvement behind APIs became widely used.

It also kept them honest and allowed us to spend our energy solving the crucial performance problem that motivated the rewrite to begin with. Finally, it built confidence within the company that the rewrite would ship, which was important giving how many people were dedicated to it.

## MVP Mentality:

From early in our development process, they made a point of having an app that would be stable around a core set of MVP features: communicating with friends, viewing stories, and using the camera. Less than 2 months into rewrite, many Snap employees were dogfooding the new app, switching to the old one only to access missing features. Any bugs or performance regressions introduced into new app would be treated as if they were production bugs, and addressed immediately.

The first version of the app we used only allowed sending photos and videos with simple text captions and sending/receiving simple text messages without effects such as stickers. This allowed us to later catch many performance regressions that were introduced as we added extra functionality to the communication flow. Catching these regressions early on gave us the time to dive deep into the root causes and invest in better testing strategies and design patterns to make similar problems less likely to occur. If we had waited until features were fully implemented to start using the app, it may have been much more difficult to address them.

## Launching It:

Once we had enough of our feature set ready, we started testing the app on new users. We listened closely to customers, and learned a lot about which features they valued the most in our app. The learnings allowed us to adjust our rewrite plan as we went.

Their new app is much more performant. We were able to reduce slow cold starts and ANR rates by 60%, frozen frames by 45%, and APK size by 25MB. It also laid a good foundation for continued performance improvement. Each fixed bug is now seen as an opportunity to add a new test, a new metric, or a new performance test to make it less likely to regress. By starting from scratch and adopting a new mindset, the experience of launching our rewritten app helped solidify a new engineering culture internally which values craftsmanship, performance, and great engineering practices.

# What is not change in the project?

1. **Old Features:**

* Messages viewed once cannot be seen again.
* Snaps couldn’t be replayed.
* Snaps could not be saved
* There exist no subscriptions
* No games were introduced.
* Location of a person was not accessible

1. **Newly added features:**

* Snapchat+ is here where user can access the application in web.
* New filters are added after every update.
* Messages could be saved in the chat up to 24 hrs. and could be saved by tapping the message.
* Subscriptions are available for creator accounts
* Location of a friend is accessible.
* User could customize their emojis
* Users can maintain streak by sending at least sending a snap in 24 hrs.

1. **Changes for each version:**
2. **Current State:**

Currently snapchat is one of the popular social media app that is used worldwide. It gives users the access to following features:

* Make snaps with different filters
* Chatting
* Audio and Video calls (users can use different filters on video call)
* One message of a person could also be sent as a snap to the same person.
* Messages could last in chat upto 24 hrs and could be saved afterwards.
* Location could be accessed of different friends in the list
* Users can subscribe to different public profiles of other users that aren’t even in their friend list
* User can play games in snapchat
* User can set their bitmojis using the bitmoji link app
* Users can send sticker emojis while chatting
* Users can also pin the chat of a specific person which wasn’t available before
* Users can even find friends using the snapcode of any other person.
* New friends could also be added by using ***all contacts***