CACHAT

Cross-Platform Mobile Application

Index

1. Introduction 3
2. System Overview 13
   1. Dev-Ops 13
   2. Flutter 21
3. Design Configuration 26
   1. Requirements
      1. Functional Requirements
      2. Non – Functional Requirements
   2. Software Design 30
      1. Dataflow diagram
      2. Context diagram
      3. Login Sequence diagram
      4. ER diagram
4. Web-Services
   1. Firebase
      1. Firebase\_core
      2. Firebase\_storage
      3. Firebase\_auth
      4. Firebase\_analytics
   2. Google\_sign\_in
   3. Flutter\_facebook\_auth
   4. Cloud\_firestore
   5. Provider
   6. Get\_it
   7. Image\_picker
   8. File\_picker
   9. Flutter\_spinkit:
   10. Timeago
   11. Flutter\_keyboard\_visibility
   12. Flutter\_profile\_picture
   13. Google\_sign\_in\_web
5. Testing
   1. Software testing Goals
   2. Validation and Verification
   3. Types of test
   4. Development testing
   5. Test-driven development

Introduction**:**

A decade ago, social media sites were promotion channels that served as a path between users and the poster’s site. The borders between different sites were fluid — people would discover content on Facebook, Twitter, and LinkedIn, then click through to content (usually hosted on another site).

Today, it's no longer enough to create a piece of content for your own site, then schedule promotion across channels that point back to that content.

Algorithm changes and shifts in the way people choose to communicate mean that marketers are starting to see the need to invest in multi-channel marketing. For many, this means adopting messaging apps.

**What Is a Messaging App?**

A messaging app is a chat application or platform that enables users to instant message and connect with each other through their computers or mobile devices. Think Facebook Messenger, WhatsApp, WeChat, or Slack (just to name a few).

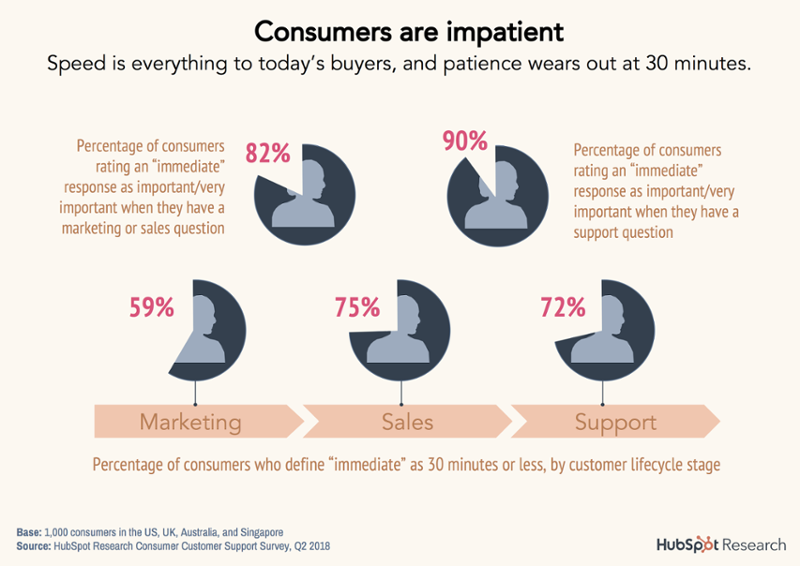
**The Good**

In 2015, sources such as Time began reporting that the attention span of the average online consumer had shrunk to less than that of a goldfish.

While the science behind this claim has largely been debunked, the theory itself is compelling.

In a device-dependent age, where consumers are bombarded with endless information, distraction, and content, it seems intuitive that time is one of our most valuable resources.

And like all resources, time is something people don't like to waste it.



<https://blog.hubspot.com/customers/messaging-apps-good-bad-ugly#:~:text=A%20messaging%20app%20is%20a,just%20to%20name%20a%20few>).

**Speed**

Messaging apps allow you to connect and engage your customers in real time. In some cases, messaging apps even help you respond to your potential leads, prospects, and customers instantaneously.

The speed of this communication channel will almost always outpace that seen on other channels, such as email or phone.

**Familiarity**

Businesses began to send emails to prospects and customers after people had adopted it as a channel to connect with their friends and family. Likewise, businesses only adopted GIFs and emojis in their social posts, emails, and on their websites after their audiences had started using GIFs and emojis as a way to communicate with one another.

Nowadays, people send messages to one another to discuss just about everything. The audience on Facebook is significant and there are already 70 million business pages currently on Facebook. And yet, only 20 million businesses send Facebook messages. This is a huge gap in the total addressable market, and it shows that most businesses just haven’t arrived yet. When they do, we’ll see incredible momentum in adoption because of consumers’ pre-existing familiarity with the channel.

**Convenience**

Messaging apps can be seamlessly blended into your everyday workflow. You can reach out or answer an inquiry from any device, close the app, and return to whatever you were doing. Then, whenever the other party is available to respond, they will  —  and then you can take your turn.

The one-to-one nature of the channel is extremely convenient for both businesses and consumers, as it allows both parties to have complete context into the conversation and continue the thread as they see fit.

**Industry** **Forces**

Two changes are happening in the market that will increase business’ adoption of messaging apps like Facebook Messenger in 2018. The first is the push away from traditional marketing channels like email or social media.

The Gmail tab has made it difficult for marketing emails to surface in your contact’s inboxes. Regulations like GDPR, CAN-SPAM, and CASL make the email space even more difficult to navigate.

On the flip side, organic reach has declined year-over-year. The average organic reach for posts on Facebook is about 2%, and that window of opportunity is expected to continue to decline even further.

The second force at work is a general pull for businesses to adopt messaging apps. Facebook continues to invest heavily in Messenger, for example. Slack recently released integrations with companies like HubSpot (learn more here) to help businesses communicate more effectively internally and externally.

As messaging apps become easier for businesses to adopt and incorporate, the steadier the growth and migration to the channels will be over time.

**The** **Bad**

Messaging apps enable you to communicate with your website visitors in live time, but it comes with a catch.

* How much automation will you use?

A good bot takes time to build. Live chat communication requires your team to be available for your customers. And not all platforms are suited to fit your needs. The short and simple is that all messaging apps require a certain degree of your resources, and you may need to weigh the pros and cons of each to vet which one fits the exact needs of you, your business, and your audience.

Start by asking questions like:

* How much budget do I have for a messaging app?

Free services will usually require that you show the messaging app's branding. You may also only have access to a limited amount of features. With this in mind, it's important to compare what each service offers in their pricing packages and how that fits your requirements as a company.

* What segment of visitors would this be the most impactful for?

Having conversations for the sake of having conversations is not the most effective use of your company resources. Instead, try to think of the visitors you could have the most impact with if you were able talk face-to-face. These are the people you should be building relationships and initiating conversations with.

* How many people need to monitor this channel? Would a bot be more effective for what I'm trying to accomplish?

In a recent HubSpot study, 80% of respondents said they’d stopped doing business with a company because of a poor customer experience. If your customers are dissatisfied, they can — and will — switch to another provider.

It can be frustrating when a company sets expectations and isn't able to live up to that promise. For this reason, you want to make sure you have the proper number of people responding the messages.

If your team is multi-tasking, consider using automatic follow messages that set clear expectations about when the user will hear back. If you are using a messaging app and notice that the same questions or interactions keep popping up, consider investing in a simple bot. This can help delight visitors by offering quick responses to their FAQs and segmenting more sensitive or unique questions to the correct member of your team.

How will we communicate our brand voice and tone over this channel?

Gifs and emoji are great for quickly and informally communicating ideas or emotions. But brand voice and tone go beyond effective use of visuals or quippy copy.

Training your team members to chat in a manner that aligns with your company's vision, mission, and impact is key. This creates alignment between the copy on your website and the experience your team is providing through messaging. The same goes for if you choose to incorporate a chatbot to your messaging app to add a layer of automation.

System Overview:

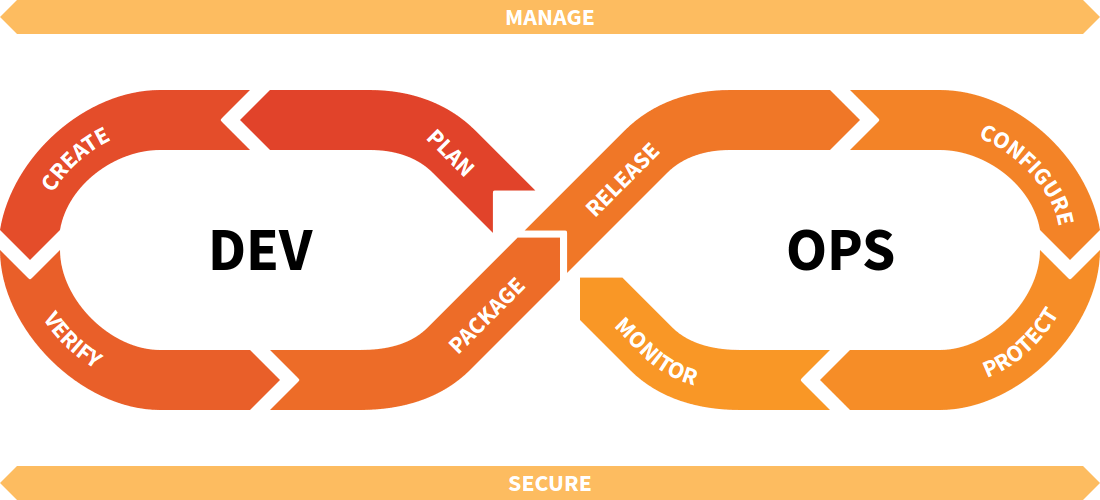
Is a Cross-Platform chatting Application targeting to allow users to use Real-Time chatting. Cachat is originally CAT-Chat. Since Chat is pronounced as ( cHat ) and we have a (Cat ) so we removed the T to be CaChat.

This project is Aiming to build a Co-Working Team to get ready for the Graduation project. First, we are going to test the DEV-OPS in order to work in an infinite loop between developing and testing.

**Dev-ops**



DevOps combines development and operations to increase the efficiency, speed, and security of software development and delivery compared to traditional processes. A more nimble software development lifecycle results in a competitive advantage for businesses and their customers.



**DevOps explained:**

DevOps can be best explained as people working together to conceive, build and deliver secure software at top speed. DevOps practices enable software developers (devs) and operations (ops) teams to accelerate delivery through automation, collaboration, fast feedback, and iterative improvement.

Stemming from an Agile approach to software development, a DevOps delivery process expands on the cross-functional approach of building and shipping applications in a faster and more iterative manner. In adopting a DevOps development process, you are making a decision to improve the flow and value delivery of your application by encouraging a more collaborative environment at all stages of the development cycle.

DevOps represents a change in mindset for IT culture. In building on top of Agile, lean practices, and systems theory, DevOps focuses on incremental development and rapid delivery of software. Success relies on the ability to create a culture of accountability, improved collaboration, empathy, and joint responsibility for business outcomes.

**Core DevOps principles**

The DevOps methodology comprises [four key principles](https://about.gitlab.com/blog/2022/02/11/4-must-know-devops-principles/) that guide the effectiveness and efficiency of application development and deployment. These principles, listed below, center on the best aspects of modern software development.

1. Automation of the software development lifecycle
2. Collaboration and communication
3. Continuous improvement and minimization of waste
4. Hyperfocus on user needs with short feedback loops

By adopting these principles, organizations can improve code quality, achieve a faster time to market, and engage in better application planning.

The four phases of DevOps

As DevOps has evolved, so has its complexity. This complexity is driven by two factors:

* Organizations are moving from monolithic architectures to [microservices architectures](https://about.gitlab.com/topics/microservices/). As DevOps matures, organizations need more and more DevOps tools per project.
* The result of more projects and more tools per project has been an exponential increase in the number of project-tool integrations. This necessitated a change in the way organizations adopted DevOps tools.

This evolution took place in following four phases:

Phase 1: Bring Your Own DevOps

In the Bring Your Own DevOps phase, each team selected its own tools. This approach caused problems when teams attempted to work together because they were not familiar with the tools of other teams.

Phase 2: Best-in-class DevOps

To address the challenges of using disparate tools, organizations moved to the second phase, Best-in-class DevOps. In this phase, organizations standardized on the same set of tools, with one preferred tool for each stage of the DevOps lifecycle. It helped teams collaborate with one another, but the problem then became moving software changes through the tools for each stage.

Phase 3: Do-it-yourself DevOps

To remedy this problem, organizations adopted Do-it-yourself (DIY) DevOps, building on top of and between their tools. They performed a lot of custom work to integrate their DevOps point solutions together. However, since these tools were developed independently without integration in mind, they never fit quite right. For many organizations, maintaining DIY DevOps was a significant effort and resulted in higher costs, with engineers maintaining tooling integration rather than working on their core software product.

Phase 4: DevOps Platform

A single-application platform approach improves the team experience and business efficiency. GitLab, The DevOps Platform, replaces DIY DevOps, allowing visibility throughout and control over all stages of the DevOps lifecycle.

By empowering all teams – Software, Operations, IT, Security, and Business – to collaboratively plan, build, secure, and deploy software across an end-to-end unified system, GitLab represents a fundamental step-change in realizing the full potential of DevOps. The DevOps Platform is a single application powered by a cohesive user interface, agnostic of self-managed or SaaS deployment. It is built on a single codebase with a unified data store, that allows organizations to resolve the inefficiencies and vulnerabilities of an unreliable DIY toolchain.

As we look ahead to software-led organizations becoming even more distributed and agile, every company will need a DevOps platform to modernize software development and delivery. By making it easier and trusted to adopt the next generation of cloud-native technologies – from microservices to serverless and eventually edge architecture – all companies will be empowered to ship software faster, at maximum efficiency, with security embedded across their end-to-end software supply chain.

**Flutter**

This year, mobile applications continued to become more and more popular. Fortunately there are many programming tools available to developers who want to create them. Among these tools there is Flutter, which has distinguished itself lately.

**What is Flutter?**

Flutter is a free and open-source mobile UI framework created by Google and released in May 2017. In a few words, it allows you to create a native mobile application with only one codebase. This means that you can use one programming language and one codebase to create two different apps (for iOS and Android).

Flutter consists of two important parts:

* An SDK (Software Development Kit): A collection of tools that are going to help you develop your applications. This includes tools to compile your code into native machine code (code for iOS and Android).
* A Framework (UI Library based on widgets): A collection of reusable UI elements (buttons, text inputs, sliders, and so on) that you can personalize for your own needs.

To develop with Flutter, you will use a programming language called Dart. The language was created by Google in October 2011, but it has improved a lot over these past years.

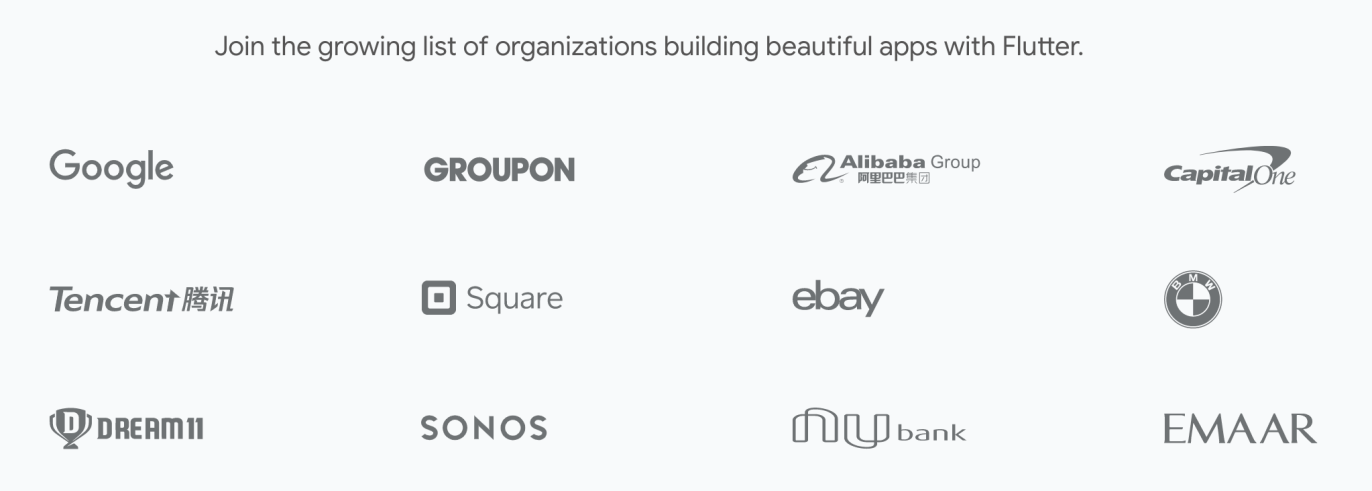
Dart focuses on front-end development, and you can use it to create mobile and web applications.

If you know a bit of programming, Dart is a typed object programming language. You can compare Dart's syntax to JavaScript.

“Flutter is Google’s UI toolkit for building beautiful, natively compiled applications for mobile, web, and desktop from a single codebase.” - Google, [flutter.dev](https://flutter.dev/)

**Why you should learn Flutter?**

I selected some of the reasons why I like Flutter and why I want to use it next year. I will give you details and my feedback below.

[Companies using Flutter](https://flutter.dev/showcase)

**Simple to learn and use**

Flutter is a modern framework, and you can feel it! It’s way simpler to create mobile applications with it. If you have used Java, Swift, or React Native, you'll notice how Flutter is different.

I personally never liked mobile application development before I started using Flutter.

What I love about Flutter is that you can create a real native application without a bunch of code.

**Quick compilation: maximum productivity**

Thanks to Flutter, you can change your code and see the results in real-time. It’s called Hot-Reload. It only takes a short amount of time after you save to update the application itself.

Significant modifications force you to reload the app. But if you do work like design, for example, and change the size of an element, it’s in real-time!

**Ideal for startup MVPs**

If you want to show your product to investors as soon as possible, Flutter is a good choice.

Here are my top 4 reasons to use it for your MVP:

* It’s cheaper to develop a mobile application with Flutter because you don’t need to create and maintain two mobile apps (one for iOS and one for Android).
* One developer is all you need to create your MVP.
* It’s performant – you won't notice the difference between a native application and a Flutter app.
* It’s beautiful – you can easily use widgets provided by Flutter and personalize it to create a valuable UI for your customers (you can find examples of applications made with Flutter below).

Design Considerations:

**Requirements**

* Functional :
* Authentication
* Recent conversation with users or groups.
* Activity status (online - Last seen).
* Search for Users.
* Initiate conversation with user or group.
* Real time sending & receiving text messages.
* Ability to create new user with name, Profile picture, Email and password.
* Non-Functional Requirements:

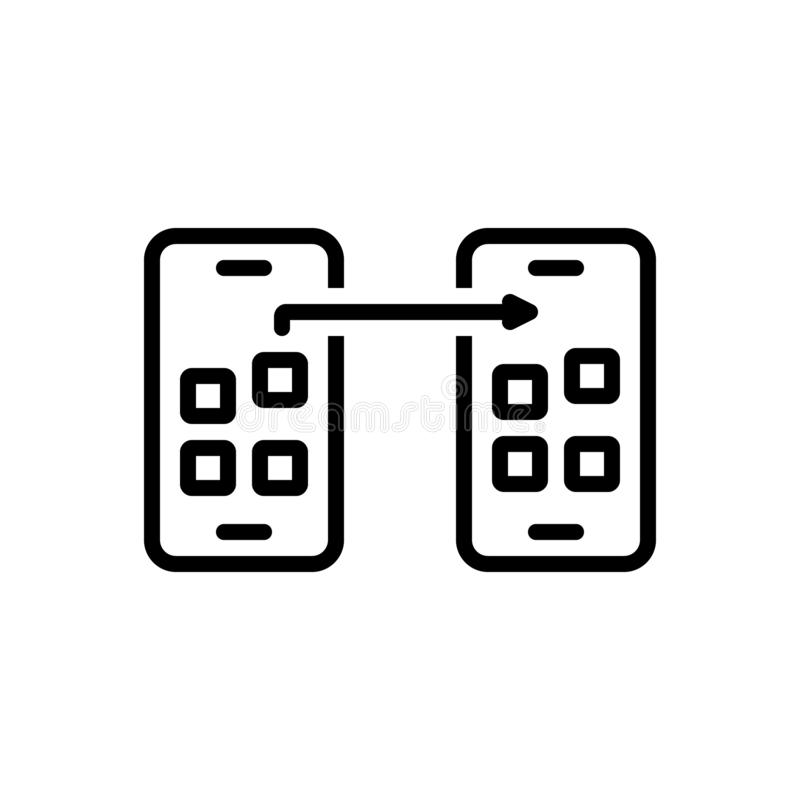
1. Security: 

application require users to create accounts to access the applications features .

1. Usability: 

the application should be easy to use for all type of users.

1. Portability:



the application should be portable

on many mobile devices..

1. Compatibility:



the application should be fully compatible on both iOS devices and android devices.

1. Reliability:



application should be reliable and performing consistently well.

1. Performance:



application should be lightweight and send messages instantly.

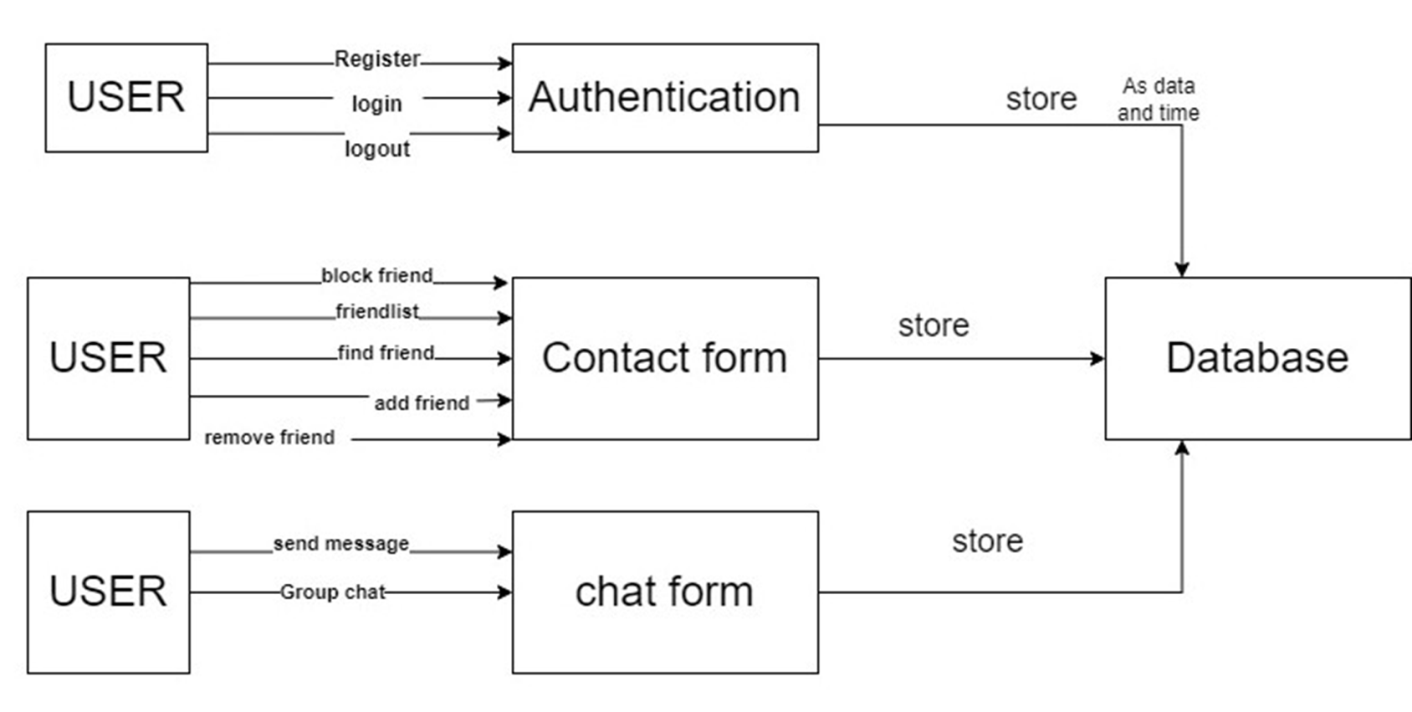
1. Efficiency:



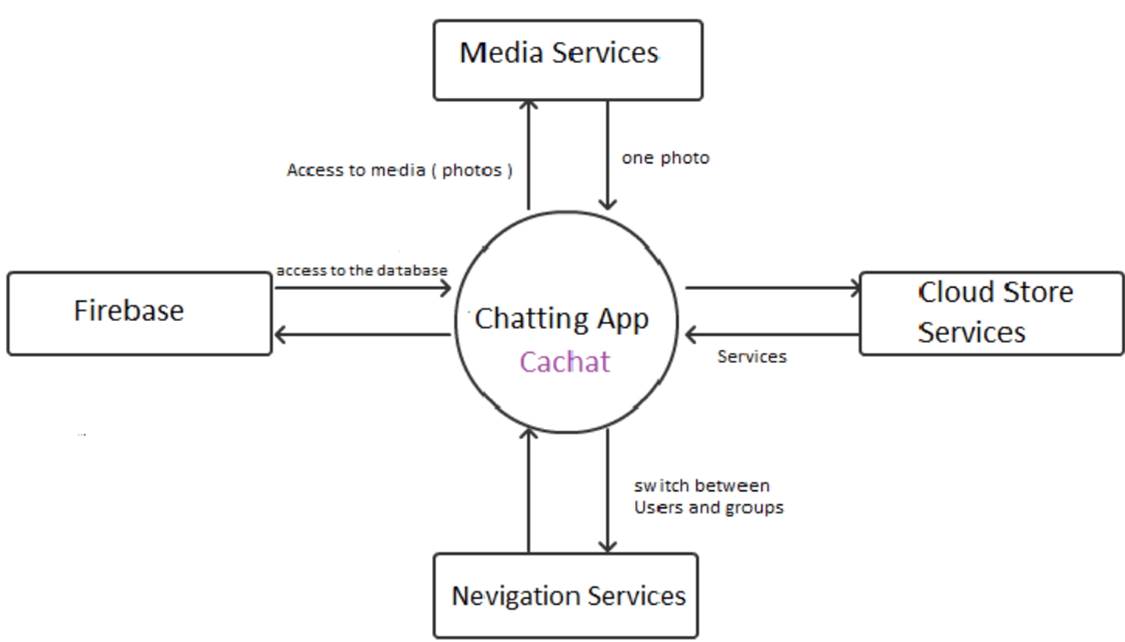
application should use minimum resources.

**Software design:**

Data flow diagram :

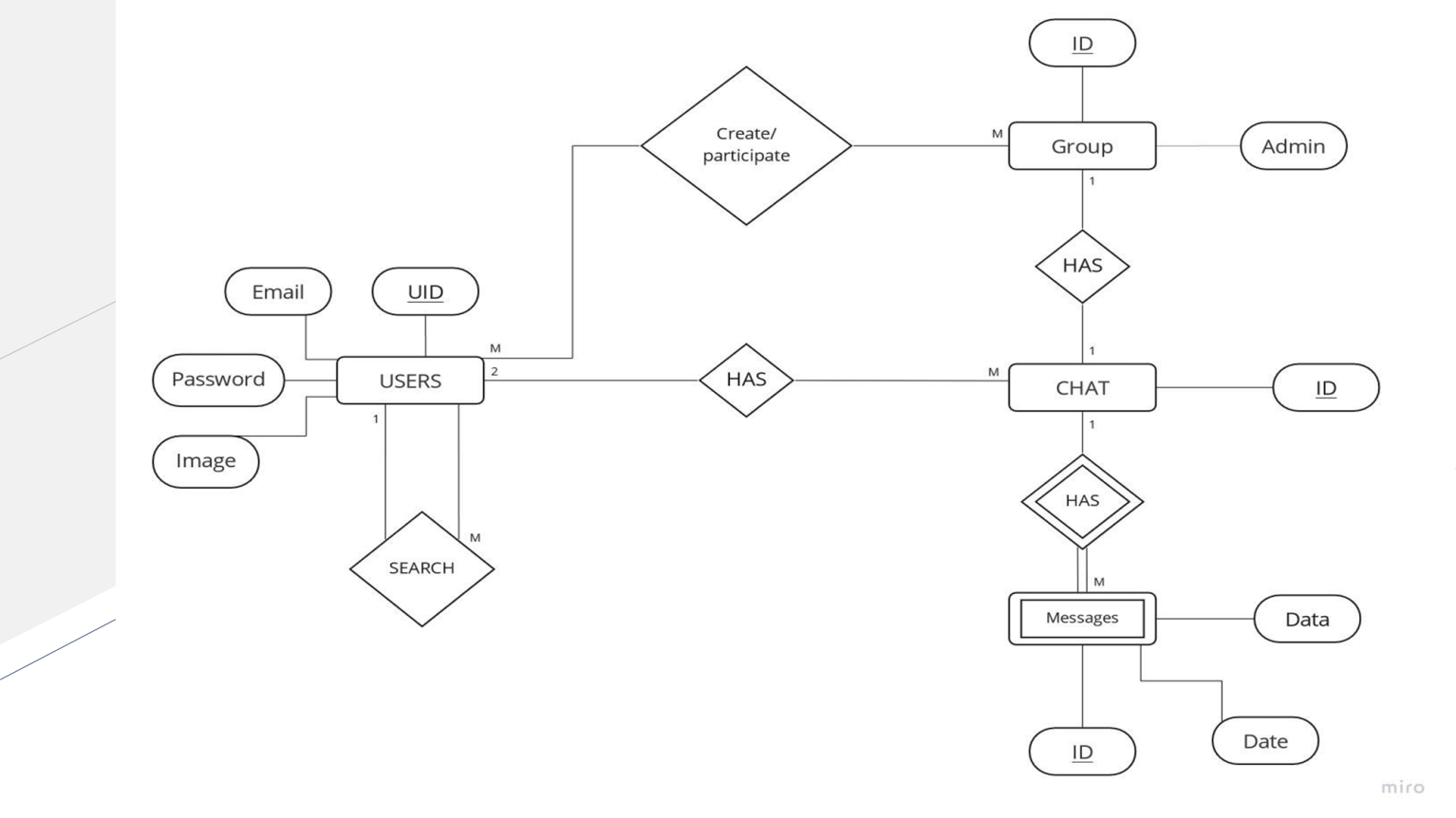


Context diagram :



## 

ER diagram :



# Firebase Core for Flutter

A Flutter plugin to use the Firebase Core API, which enables connecting to multiple Firebase apps.

To learn more about Firebase, please visit the [Firebase website](https://firebase.google.com)

## Getting Started

To get started with FlutterFire, please [see the documentation](https://firebase.flutter.dev/docs/overview).

## Usage

To use this plugin, please visit the [Core Usage documentation](https://firebase.flutter.dev/docs/core/usage)

## Issues and feedback

Please file FlutterFire specific issues, bugs, or feature requests in our [issue tracker](https://github.com/FirebaseExtended/flutterfire/issues/new).

Plugin issues that are not specific to FlutterFire can be filed in the [Flutter issue tracker](https://github.com/flutter/flutter/issues/new).

To contribute a change to this plugin, please review our [contribution guide](https://github.com/FirebaseExtended/flutterfire/blob/master/CONTRIBUTING.md) and open a [pull request](https://github.com/FirebaseExtended/flutterfire/pulls).Cloud Storage for Flutter

A Flutter plugin to use the [Firebase Cloud Storage API](https://firebase.google.com/docs/storage/).

To learn more about Storage, please visit the [Firebase website](https://firebase.google.com/products/storage)

## Getting Started

To get started with Cloud Storage for Flutter, please [see the documentation](https://firebase.flutter.dev/docs/storage/overview).

## Usage

To use this plugin, please visit the [Storage Usage documentation](https://firebase.flutter.dev/docs/storage/usage)

## Issues and feedback

Please file FlutterFire specific issues, bugs, or feature requests in our [issue tracker](https://github.com/FirebaseExtended/flutterfire/issues/new).

Plugin issues that are not specific to FlutterFire can be filed in the [Flutter issue tracker](https://github.com/flutter/flutter/issues/new).

To contribute a change to this plugin, please review our [contribution guide](https://github.com/FirebaseExtended/flutterfire/blob/master/CONTRIBUTING.md) and open a [pull request](https://github.com/FirebaseExtended/flutterfire/pulls).

Firebase Auth for Flutter [#](https://pub.dev/packages/firebase_auth#firebase-auth-for-flutter)

A Flutter plugin to use the [Firebase Authentication API](https://firebase.google.com/products/auth/).

To learn more about Firebase Auth, please visit the [Firebase website](https://firebase.google.com/products/auth)

## Getting Started

To get started with Firebase Auth for Flutter, please [see the documentation](https://firebase.flutter.dev/docs/auth/overview).

## Usage

To use this plugin, please visit the [Authentication Usage documentation](https://firebase.flutter.dev/docs/auth/usage)

## Issues and feedback

Please file FlutterFire specific issues, bugs, or feature requests in our [issue tracker](https://github.com/FirebaseExtended/flutterfire/issues/new).

Plugin issues that are not specific to FlutterFire can be filed in the [Flutter issue tracker](https://github.com/flutter/flutter/issues/new).

To contribute a change to this plugin, please review our [contribution guide](https://github.com/FirebaseExtended/flutterfire/blob/master/CONTRIBUTING.md) and open a [pull request](https://github.com/FirebaseExtended/flutterfire/pulls).

# Firebase Analytics Plugin for Flutter

A Flutter plugin to use the [Firebase Analytics API](https://firebase.google.com/docs/analytics/).

To learn more about Firebase Analytics, please visit the [Firebase website](https://firebase.google.com/products/analytics)

## Getting Started

To get started with Firebase Analytics for Flutter, please [see the documentation](https://firebase.flutter.dev/docs/analytics/overview).

## Usage

To use this plugin, please visit the [Analytics Usage documentation](https://firebase.flutter.dev/docs/analytics/usage)

## Issues and feedback

Please file FlutterFire specific issues, bugs, or feature requests in our [issue tracker](https://github.com/FirebaseExtended/flutterfire/issues/new).

Plugin issues that are not specific to FlutterFire can be filed in the [Flutter issue tracker](https://github.com/flutter/flutter/issues/new).

To contribute a change to this plugin, please review our [contribution guide](https://github.com/FirebaseExtended/flutterfire/blob/master/CONTRIBUTING.md) and open a [pull request](https://github.com/FirebaseExtended/flutterfire/pulls).

# Cloud Firestore Plugin for Flutter

A Flutter plugin to use the [Cloud Firestore API](https://firebase.google.com/docs/firestore/).

To learn more about Firebase Cloud Firestore, please visit the [Firebase website](https://firebase.google.com/products/firestore)

## Getting Started

To get started with Cloud Firestore for Flutter, please [see the documentation](https://firebase.flutter.dev/docs/firestore/overview).

## Usage

To use this plugin, please visit the [Firestore Usage documentation](https://firebase.flutter.dev/docs/firestore/usage)

## Issues and feedback

Please file FlutterFire specific issues, bugs, or feature requests in our [issue tracker](https://github.com/FirebaseExtended/flutterfire/issues/new).

Plugin issues that are not specific to FlutterFire can be filed in the [Flutter issue tracker](https://github.com/flutter/flutter/issues/new).

To contribute a change to this plugin, please review our [contribution guide](https://github.com/FirebaseExtended/flutterfire/blob/master/CONTRIBUTING.md) and open a [pull request](https://github.com/FirebaseExtended/flutterfire/pulls).

A wrapper around [InheritedWidget](https://api.flutter.dev/flutter/widgets/InheritedWidget-class.html) to make them easier to use and more reusable.

By using provider instead of manually writing [InheritedWidget](https://api.flutter.dev/flutter/widgets/InheritedWidget-class.html), you get:

* simplified allocation/disposal of resources
* lazy-loading
* a vastly reduced boilerplate over making a new class every time
* devtool friendly – using Provider, the state of your application will be visible in the Flutter devtool
* a common way to consume these [InheritedWidget](https://api.flutter.dev/flutter/widgets/InheritedWidget-class.html)s (See [Provider.of](https://pub.dev/documentation/provider/latest/provider/Provider/of.html)/[Consumer](https://pub.dev/documentation/provider/latest/provider/Consumer-class.html)/[Selector](https://pub.dev/documentation/provider/latest/provider/Selector-class.html))
* increased scalability for classes with a listening mechanism that grows exponentially in complexity (such as [ChangeNotifier](https://api.flutter.dev/flutter/foundation/ChangeNotifier-class.html), which is O(N) for dispatching notifications).

To read more about a provider, see its [documentation](https://pub.dev/documentation/provider/latest/provider/provider-library.html).

See also:

* [The official Flutter state management documentation](https://flutter.dev/docs/development/data-and-backend/state-mgmt/simple), which showcases how to use provider + [ChangeNotifier](https://api.flutter.dev/flutter/foundation/ChangeNotifier-class.html)
* [flutter architecture sample](https://github.com/brianegan/flutter_architecture_samples/tree/master/change_notifier_provider), which contains an implementation of that app using provider + [ChangeNotifier](https://api.flutter.dev/flutter/foundation/ChangeNotifier-class.html)
* [flutter\_bloc](https://github.com/felangel/bloc) and [Mobx](https://github.com/mobxjs/mobx.dart), which uses a provider in their architecture

# get\_it

This is a simple **Service Locator** for Dart and Flutter projects with some additional goodies highly inspired by [Splat](https://github.com/reactiveui/splat). It can be used instead of InheritedWidget or Provider to access objects e.g. from your UI.

Typical usage:

* Accessing service objects like REST API clients or databases so that they easily can be mocked.
* Accessing View/AppModels/Managers/BLoCs from Flutter Views

**V7.0 has some breaking changes** Check please check the release notes to see what's new.

## Why GetIt

As your App grows, at some point you will need to put your app's logic in classes that are separated from your Widgets. Keeping your widgets from having direct dependencies makes your code better organized and easier to test and maintain. But now you need a way to access these objects from your UI code. When I came to Flutter from the .Net world, the only way to do this was the use of InheritedWidgets. I found the way to use them by wrapping them in a StatefulWidget; quite cumbersome and has problems working consistently. Also:

* I missed the ability to easily switch the implementation for a mocked version without changing the UI.
* The fact that you need a BuildContext to access your objects made it inaccessible from the Business layer.

Accessing an object from anywhere in an App can be done by other ways, but:

* If you use a Singleton you can't easily switch the implementation out for a mock version in tests
* IoC containers for Dependency Injections offers similar functionality, but with the cost of slow start-up time and less readability because you don't know where the magically injected object come from. Most IoC libs rely on reflection they cannot be ported to Flutter.

As I was used to use the Service Locator Splat from .Net, I decided to port it to Dart. Since then, more features have been added.

If you are not familiar with the concept of Service Locators, it's a way to decouple the interface (abstract base class) from a concrete implementation, and at the same time allows to access the concrete implementation from everywhere in your App over the interface. I can only highly recommend to read this classic article by from Martin Fowler [Inversion of Control Containers and the Dependency Injection pattern](https://martinfowler.com/articles/injection.html).

GetIt is:

* Extremely fast (O(1))
* Easy to learn/use
* Doesn't clutter your UI tree with special Widgets to access your data like provider or Redux does.

### The get\_it\_mixin

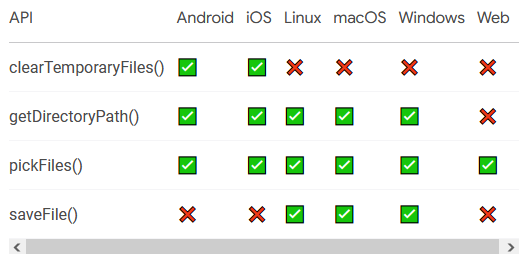
GetIt isn't a state management solution! It's a locator for your objects so you need some other way to notify your UI about changes like Streams or ValueNotifiers. But together with the [get\_it\_mixin](https://pub.dev/packages/get_it_mixin) it gets a full featured easy state management solution that integrates with the Objects registered in get\_it

# File Picker

A package that allows you to use the native file explorer to pick single or multiple files, with extensions filtering support.

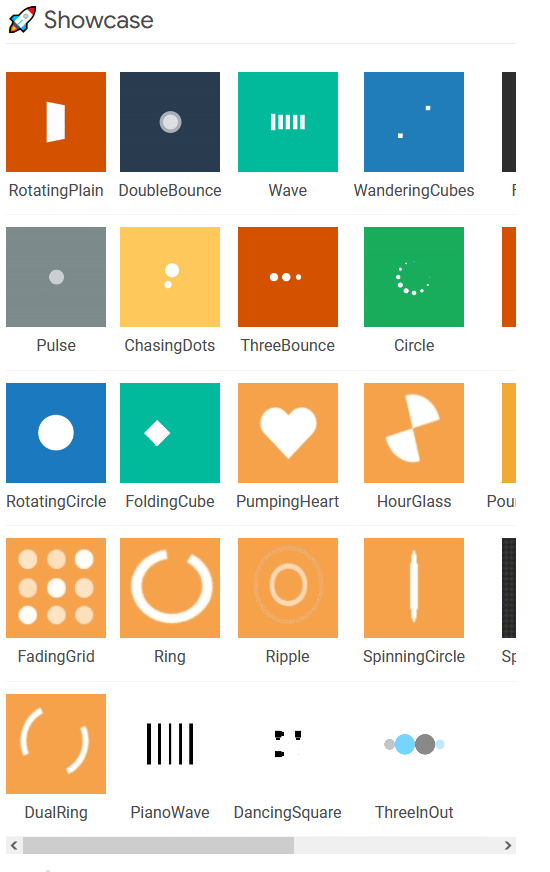
## Currently supported features

* Uses OS default native pickers
* Supports multiple platforms (Mobile, Web, Desktop and Flutter GO)
* Pick files using **custom format** filtering — you can provide a list of file extensions (pdf, svg, zip, etc.)
* Pick files from **cloud files** (GDrive, Dropbox, iCloud)
* Single or multiple file picks
* Different default type filtering (media, image, video, audio or any)
* Picking directories
* Load file data immediately into memory (Uint8List) if needed;
* Open a save-file / save-as dialog (a dialog that lets the user specify the drive, directory, and name of a file to save)



# Flutter Spinkit

A collection of loading indicators animated with flutter. Heavily inspired by [@tobiasahlin](https://github.com/tobiasahlin)'s [SpinKit](https://github.com/tobiasahlin/SpinKit).



# timeago

timeago is a dart library that converts a date into a humanized text. Instead of showing a date 2020-12-12 18:30 with timeago you can display something like "now", "an hour ago", "~1y", etc

## Scope

While there are many request for adding more complex functionality I want keep this library as simple as possible to allow minimal maintenance.

The focus of this library should be

1. Provide a single format function that transforms a date to a humanized value
2. Give the abstractions for users to add their own languages or overriding them as they please
3. Provide languages contributed by the community so users can add them as they need we should not add all languages by default.
4. Library should not depend on any dependency

# timeago\_flutter widgets

* Timeago
* TimerRefresh
* TimerRefreshWidget

# Flutter Keyboard Visibility

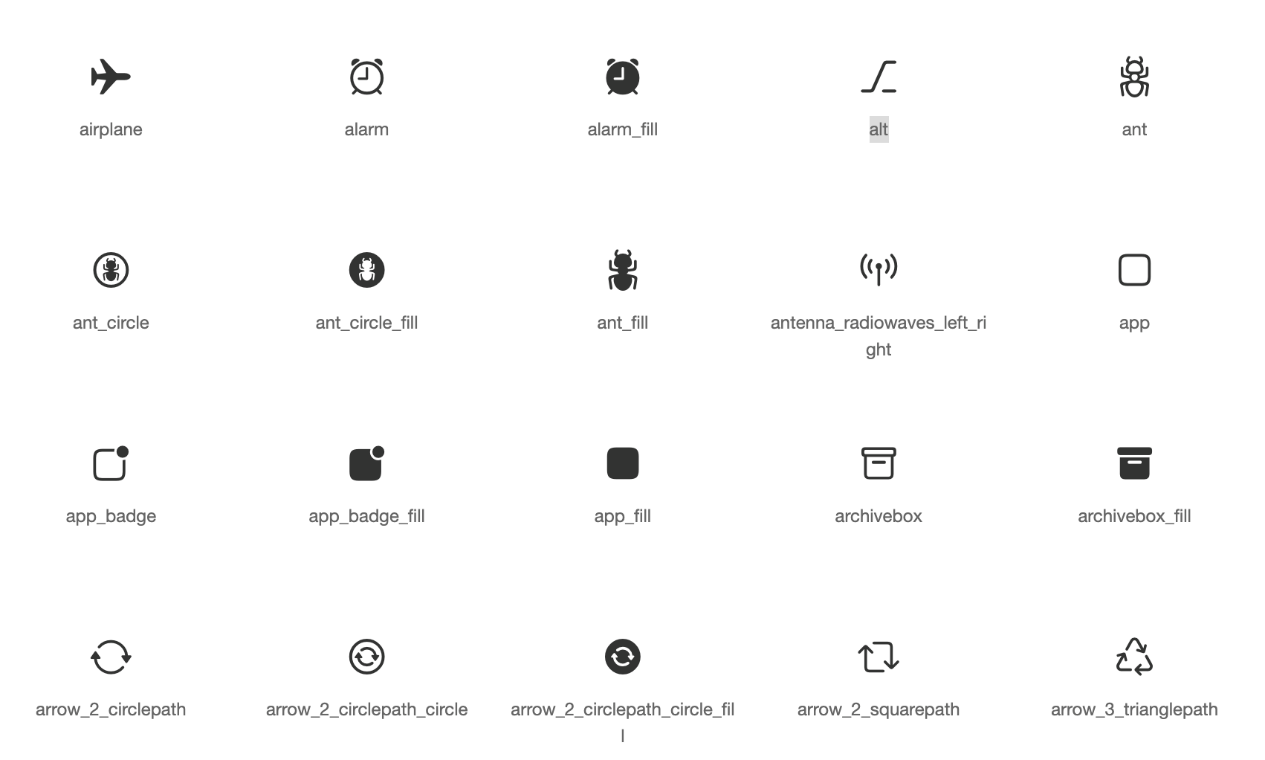
React to keyboard visibility changes.

### Note about Flutter Web support

Web support is an open issue [here](https://github.com/MisterJimson/flutter_keyboard_visibility/issues/10). Currently this library will just return false for keyboard visibility on web.

يا ريمااااس دي باكدج تانية هااااا

# Cupertino Icons

This is an asset repo containing the default set of icon assets used by Flutter's [Cupertino widgets](https://github.com/flutter/flutter/tree/master/packages/flutter/lib/src/cupertino).

## **REFERENCES:**

<https://blog.hubspot.com/customers/messaging-apps-good-bad-ugly#:~:text=A%20messaging%20app%20is%20a,just%20to%20name%20a%20few>

<https://about.gitlab.com/topics/devops/>