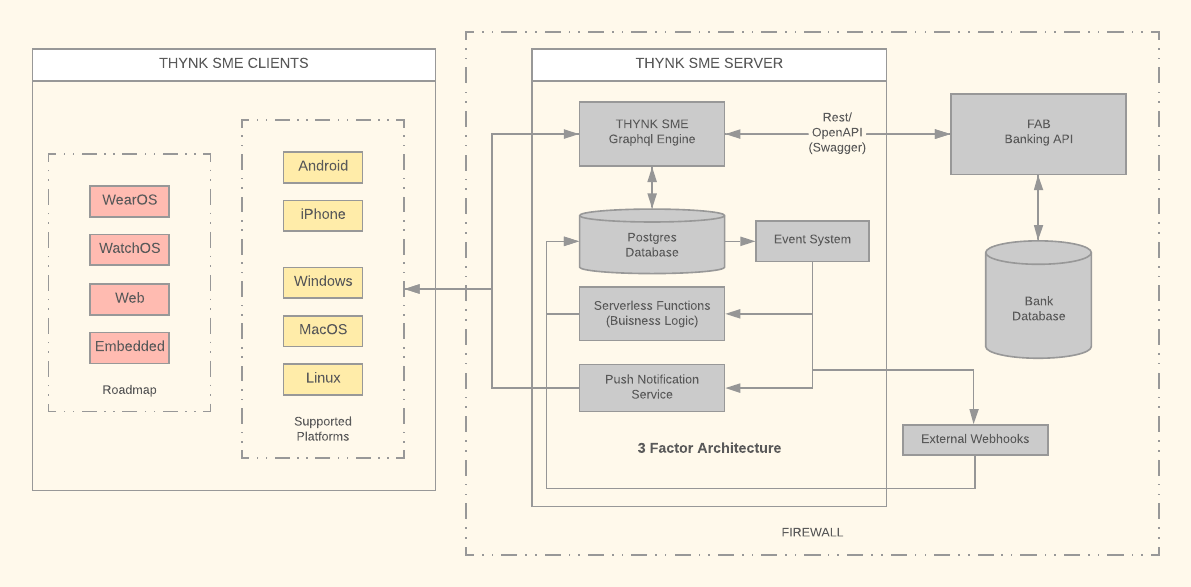
**THYNK SME Architecture**

****

Overview

Thynk SME architecture consists of realtime graphql database with mobile and desktop clients. We also have web, watch and embedded client support on our roadmap.

Our server stack uses Facebook’s GraphQL protocol (<https://graphql.org/>); unlike REST, GraphQL allows for requesting specific data that a client needs, departing from the fixed data structure approach. Additionally it allows advanced features like live queries, not possible with traditional REST. [Goto server architecture for more detail.](#o7lg8potz10u)

On client side we have chosen Google’s Flutter (<https://flutter.dev/>) as our layout engine technology. By virtue of this choice we have consolidated all out client layout logic in a single code base. Their is still android or ios specific code to interact with platform sensors etc but 90% of the code is centralized via this approach. This results in consistent layout across all platforms for which we can write tests in single place as well. Flutter seamlessly run on android, ios and across desktop alike since it brings its own native rendering engine called Skia (<https://skia.org/>). It’s designed have gorgeous 120 frames/sec experience. Soon it will also support web and embedded systems. [Goto client architecture section for more detail.](#z7k842hqvgbb)

**All the software libraries in our stack are open source.** Below is the list of libraries currently being currently used.

Libraries

Server

* Graphql Engine - <https://github.com/hasura/graphql-engine>
* Database - <https://github.com/postgres/postgres>
* Business Logic - <https://github.com/serverless/serverless>
* Serverless Functions - <https://github.com/serverless/serverless>
* Devops - <https://github.com/kubernetes/kubernetes>

Our devops is deployed on Microsoft's Azure services, though the stack used is portable to be deployed anywhere.

Client

Layout engine - Flutter - <https://github.com/flutter/flutter>

Android – Kotlin

Ios - Swift

Desktop app - Golang using GLFW- <https://github.com/golang/go>

Roadmap

* Tablet optimized layouts
* Offline support using encrypted client side database
* Admin App
* Web support
* Virtual keyboard for better experience and security
* Embedded support
* WatchOS and WearOS support
* Smart Legal Contracts, arbitration and escrow services for SME’s

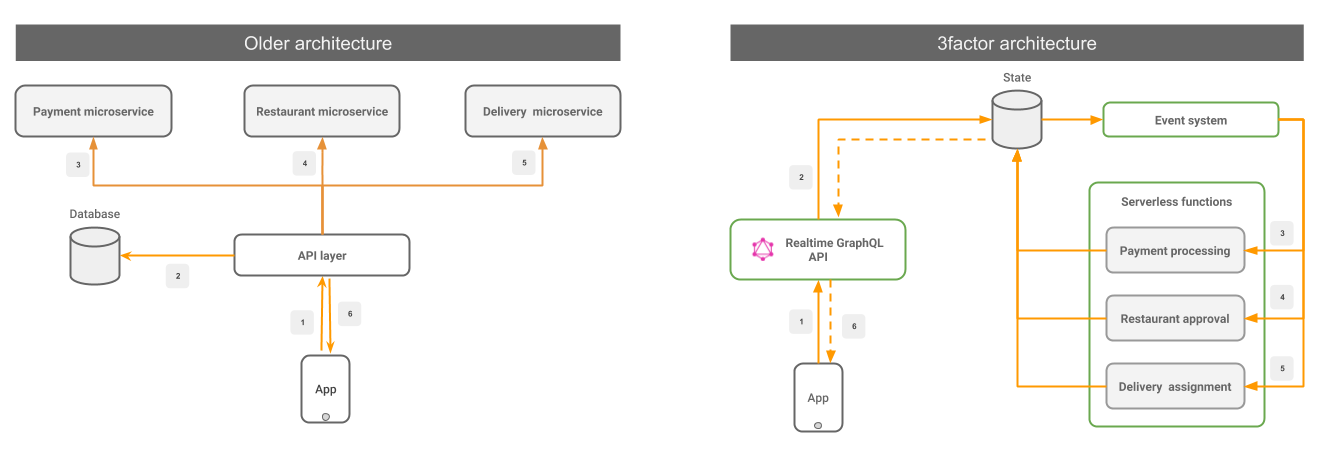
**Server Architecture**

3factor app is an architecture pattern for modern full-stack apps. Today, it is possible to build apps that have high feature velocity and scalability from the get go. Jump to reference implementation.

We propose an architecture pattern which is composed of 3 factors:

1. Realtime GraphQL
2. Reliable eventing
3. Async serverless

Consider a traditional food ordering application which moves to a 3factor architecture:



## Factor #1: Realtime GraphQL

Use GraphQL for a very simple and flexible frontend developer workflow. GraphQL is a crucial component for delivering high feature velocity. Your GraphQL layer should also support the following 2 properties:

* **Low-latency**: An end-user should see instant feedback of an action and not have to wait long on an API (<100ms ideal, upto 1 second at worst).
* **Support subscriptions**: Consume information “realtime” from the backend via GraphQL Subscriptions. Avoid the use of continuous polling (thereby reducing resource consumption).

## Factor #2: Reliable eventing

Remove in-memory state manipulation in your backend APIs and persist them as atomic events instead. Having an immutable event log helps in crash recovery, replayability and observability among others. Your event system should have the following 2 properties:

* **Atomic**: Mutations to the application state should atomically create event(s).
* **Reliable**: Events once emitted should be delivered (to any consumer) at least once.

## Factor #3: Async serverless

Write business logic as event handlers. Deploy these event handlers on serverless compute. Serverless minimizes backend ops and gives free scalability while being cost-efficient. The serverless backends should follow few best-practices:

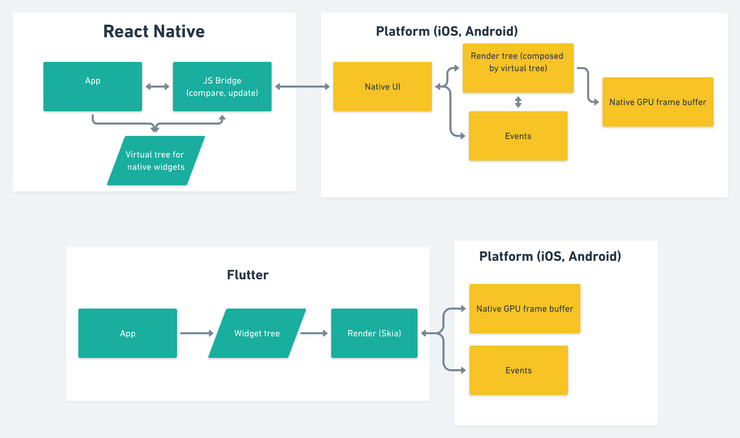
* **Idempotent**: The code should be prepared for at least once (for same event) delivery of events.
* **Out-of-order**: Events may not be guaranteed to be received in the order of creation. The code should not depend on any expected sequence of events.

In short, a 3factor app requires you to remove state from your code and put it in your datastore and/or event queues as much as possible. Cloud vendors make it easy to scale and replicate your datastore, event-queues and compute backend. Consuming asynchronous information and performing sync actions in the frontend requires a high-performant realtime GraphQL API.

An interesting sidenote: A 3factor app’s architecture is analogous to the redux dataflow model on a react app, but applied to the fullstack.

**Client Architecture**

Flutter is Google’s UI toolkit for building beautiful, natively compiled applications for mobile, web, and desktop from a single codebase.



Pros of Flutter

* Cross - Platform
  + Support Android, Ios
  + Works with desktop windows, mac, linux, embedded device (like kiosks, touch screen) (Soon official support)
  + Web (under development)
* Fast performance 120 fps skia engine
* Consistent Open - source skia engine – single place for writing UI testing code
* Precise and flexible for design development (Control over pixel)
* Fast Development cycle due to instant hot reloads
* Opensource; security auditable
* Fastest growing jobs on linkedin (recruitment)

Skia is an open source 2D graphics library which provides common APIs that work across a variety of hardware and software platforms. It serves as the graphics engine for Google Chrome and Chrome OS, Android, Mozilla Firefox and Firefox OS, and many other products.

Skia is sponsored and managed by Google, but is available for use by anyone under the BSD Free Software License. While engineering of the core components is done by the Skia development team, we consider contributions from any source.