Somnio
SOFTWARE

Flutter Technical Playbook

Our Flutter journey: best practices, processes and tools

Love at first sight

As a Software Development Company, we focus on choosing the right technology for every project. Our mission is to achieve first-class products with modern and high-quality technology.

We have worked with several technologies before **we fell in love with Flutter, but since we met it has definitely changed the way we develop apps.** It was a before-and-after moment in our developers' lives.

Flutter is our solution for productive, high-quality, and modern app development. It saves us the trouble of having multiple codes for different platforms and different working teams. **This means Flutter improves efficiency while reducing costs and still achieving outstanding apps with native performance**.

Development

- Flutter
- Serverless
- Firebase
- Node.js
- MongoDB
- Docker
- SQLite
- Algolia
- Hive

CI/CD

- Fastlane
- Github actions
- Codemagic
- Firebase App Distribution
- Testflight
- Jenkins

Infrastucture

- Amazon Web Services
- Google Cloud

A little bit about Flutter:

What is Flutter?

Flutter is a software development kit created by **Google** for building beautiful, natively compiled, and high-performance applications for mobile, web, and desktop from a single codebase.

✓ Fast programming and time-to-market

Flutter has server tools that speed up processes and skip time-consuming steps and allows faster code development such as a hot reload feature, quick experimentation, and UI building.

✓ Native and smooth performance

Flutter's performance is indistinguishable from a native app. Contrary to most cross-platform frameworks, Flutter not only has a native look but it provides a native performance.

✓ Less Testing

The Quality Assurance process is faster because developers write automated tests only once since it's the same codebase for multiple platforms.

Why choose Flutter?

The Google technology offers several **benefits** of the development of software products, some of them are:

✓ Cross-platform

A single codebase. One working team. Multiple Platforms. The useof Flutter saves time, effort, and costs while still achieving high-quality results.

✓ Better compatibility

Flutter offers a wide variety of ready-made widgets that solve the most common problems when developing user interfaces. Flutter supports accessible widgets and allows customization.

✓ Expressive and flexible UI

The UI process is more flexible and versatile since it provides the possibility of customizing anything you see on screen. With Flutter it's simple and adjustable regardless of the complexity of the components.

Main tools we use to guarantee quality processes and results

Communication	Management	Productivity	Tracking and documentation
slack	Jira Software	Clockify	Confluence
zoom	Trello	TickTick	LastPass
	ClickUp	GitHub	N
	G Suite		

Flutter Team Highlights

- Team working with Flutter since its production release on 2018
- Team members: **15+ Flutter Developers**
- **20+ projects** on Flutter
- App that reached higher in the ranking: **Top News Apps #4 in App Store**
- **5 stars rating** in Flutter projects on Clutch and Fiverr
- Participants of the **Flutter Community** with multiple **Open Source libraries** published on Github and information in our **blog**
- **Flutter Testing:** 100% Coverage in some projects
- **Partners** with big companies and referents of Flutter

Some Flutter Integrations and tools we work with

Cloud Infrastructure/Services
- Amazon Web Services
- Google Cloud Platform
- 12+ Firebase Services
- Serverless Platforms
- Authentication
- Google APIs

Database and Search
- Algolia
- Redis
- SQLite
- Shared Preferences
- Secure Storage

Custom App Features
- Custom Animations
- Always-On Background Service
- Native Caller ID
- Custom Canvas Drawing
- Native code using Kotlin, Java, Swift and Objective C

Hardware
- Face ID and Touch ID
- NFC
- Bluetooth
- GPS
- Camera
- QR Code

Messaging/Notifications
- Push Notifications
- Rabbit MQ
- Websockets

Payments
- In-app purchase
- Apple Pay
- Google Pay
- Stripe

Audio/Video/AR
- Unity AR
- Agora Real Time Voice and Video
- Podcast

Analytics
- Appsflyer
- Firebase Analytics

Tools
- VS Code
- Android Studio
- Xcode

Quality is our primary focus

Software Design

- We focus on correctly creating **scalable, maintainable and testable architecture**
- We design every software solution before implementing them
- We do Unit testing, Integration testing, and Smoke testing with Flutter
- We apply good practices around **Clean Code**
- We use **scalable software designs** like for the Main State Management used: BLoC and Provider
- **Reactive Programming** for live and efficient updates on the UI
- Error management and error logging
- We have a set of **pre built Flutter modules** for using to speed up the development

Process

- We use **SCRUM methodology** to manage and measure scalable teams
- We constantly **review and improve processes** from Flutter consulting to deploy and enhance apps
- Pull request, Linter, Refactor
- Branching model depending on the project GitFlow with GitLab flow or GitHub flow
- **CI/CD:** Testing, Fastlane, Github actions, Codemagic, Firebase App Distribution, Test flight
- **Code review:** every code written by a Developer is reviewed by at least 2 other Devs
- We use services to **measure performance, analytics, bugs, UX experience**, and much more

Communication & Study

- Daily Scrum Meetings in **English** & English Day on Thursdays to practice
- At least 1-2hr per week dedicated to **training and learning** something new
- **Pair programming** when needed
- Constant **retrospectives** to analyze and understand which areas need improvement
- **Documentation of the code and project to guide our work** and maintain all the relevant parts of the project

Flutter for the Web

As soon as Flutter web was officially released on stable with Flutter 2, we started immersing ourselves in this new possibility and creating web projects with this technology.

We are able to **reuse over 95% of the code** used for each project into the web platform. As a result, we practically have the web of the project without additional cost.

When to use it?

- Progressive Web Applications
- Single Page Application
- Existing mobile applications

Layout

- Responsive
- Adaptive

How to use it? (Web renderers types)

- HTML
- Canvas Kit

Serverless Infrastructure

- Firebase hosting
- Google Cloud Hosting
- GitHub pages

Firebase Hosting

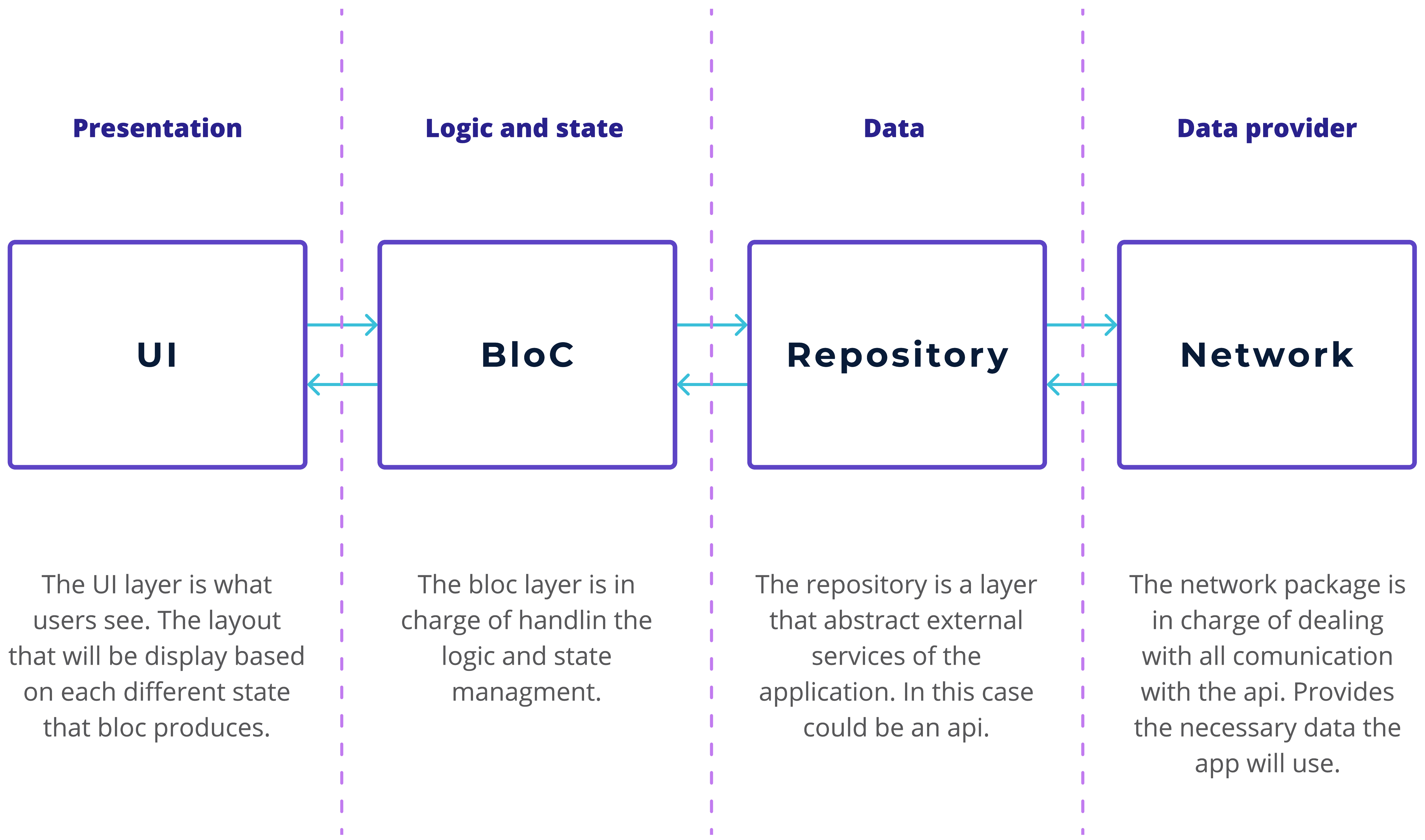
GitHub Pages

Flutter architecture

The software architecture we used is a combination of all the best practices we have acquired and worked with during our Flutter journey. We always focus on implementing **clean architecture methodologies** and continue learning and improving it.

As a result, this approach has a proven record with our success cases that allows our clients' businesses to **easily scale** with no sweat and without compromising quality.

We focused on defining the following structure on a high level. It is simple enough yet really **powerful** when scaling an app on Flutter.



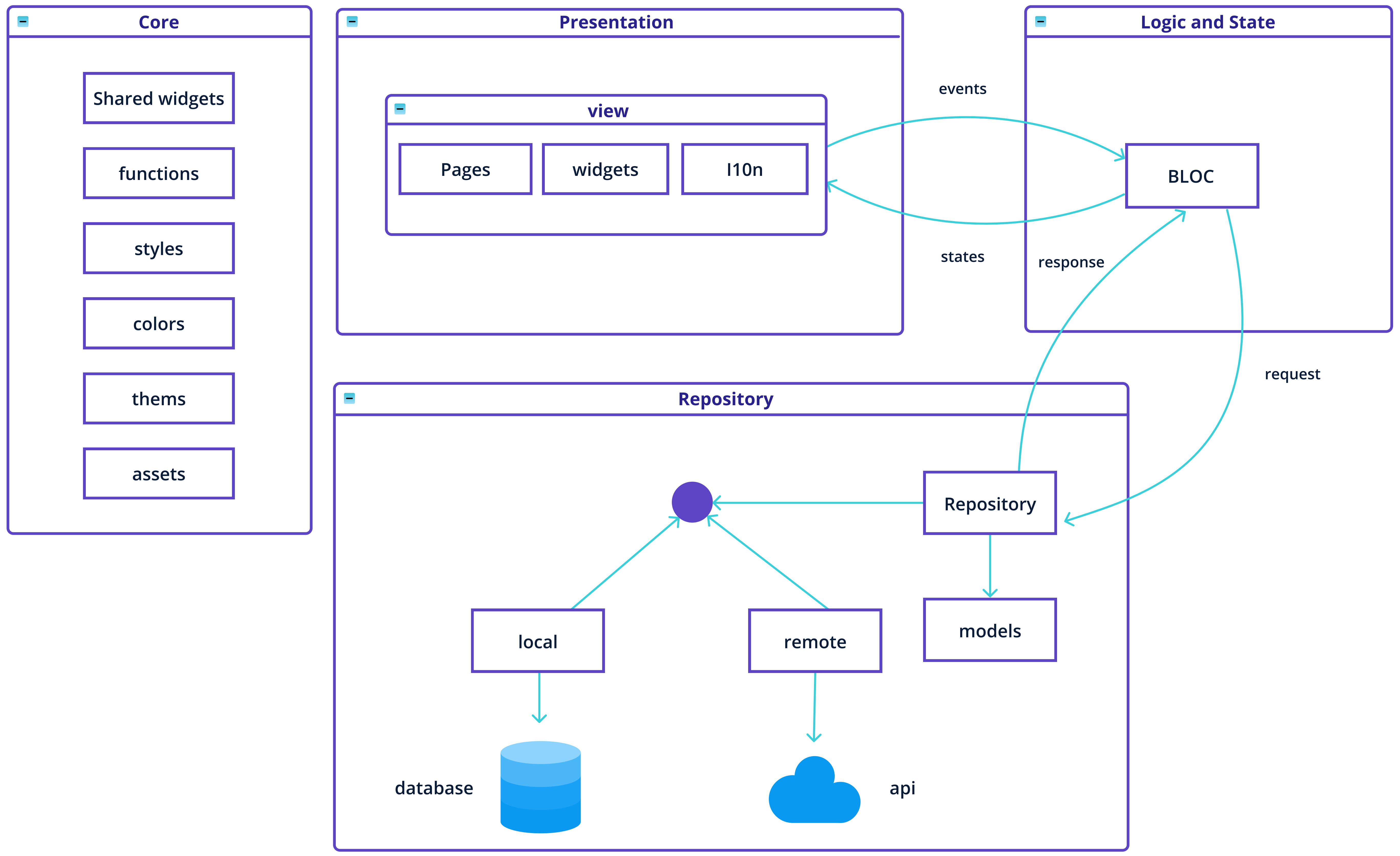
The UI layer is what users see. The layout that will be display based on each different state that bloc produces.

The bloc layer is in charge of handlin the logic and state managment.

The repository is a layer that abstract external services of the application. In this case could be an api.

The network package is in charge of dealing with all comunication with the api. Provides the necessary data the app will use.

Flutter Architecture in depth: Demystifying our architecture

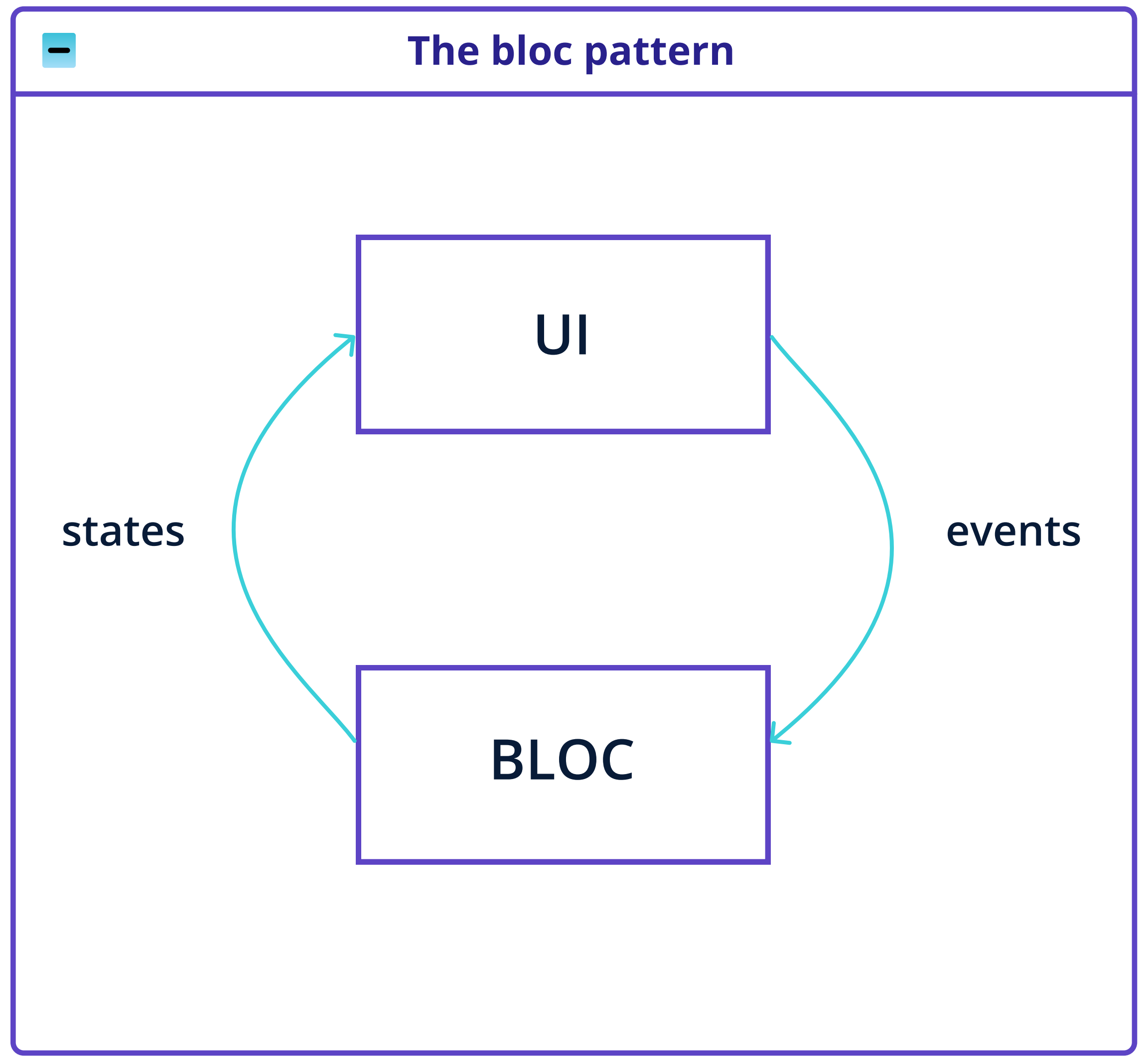


State management: The BLoC Pattern

BLoC (Business Logic Component) allows us to **manage the state within the application**. In this way, all the things that change within the screens can be handled from one component, rather than through the app in various locations.

It has provided the team the confidence to scale a production-ready challenging application. Its short-term bureaucracy a.k.a "boilerplate" has highly contributed to **long-term efficiency**. Using the BLoC pattern also enables our team to accomplish reactive programming without the complexity of managing traditional reactive libraries like rxdart.

We have studied and experimented with different solutions such as an inherited widget, provider, GetX, Riverpod, MobX, and have decided to choose BLoC because of its efficiency with scalable projects.

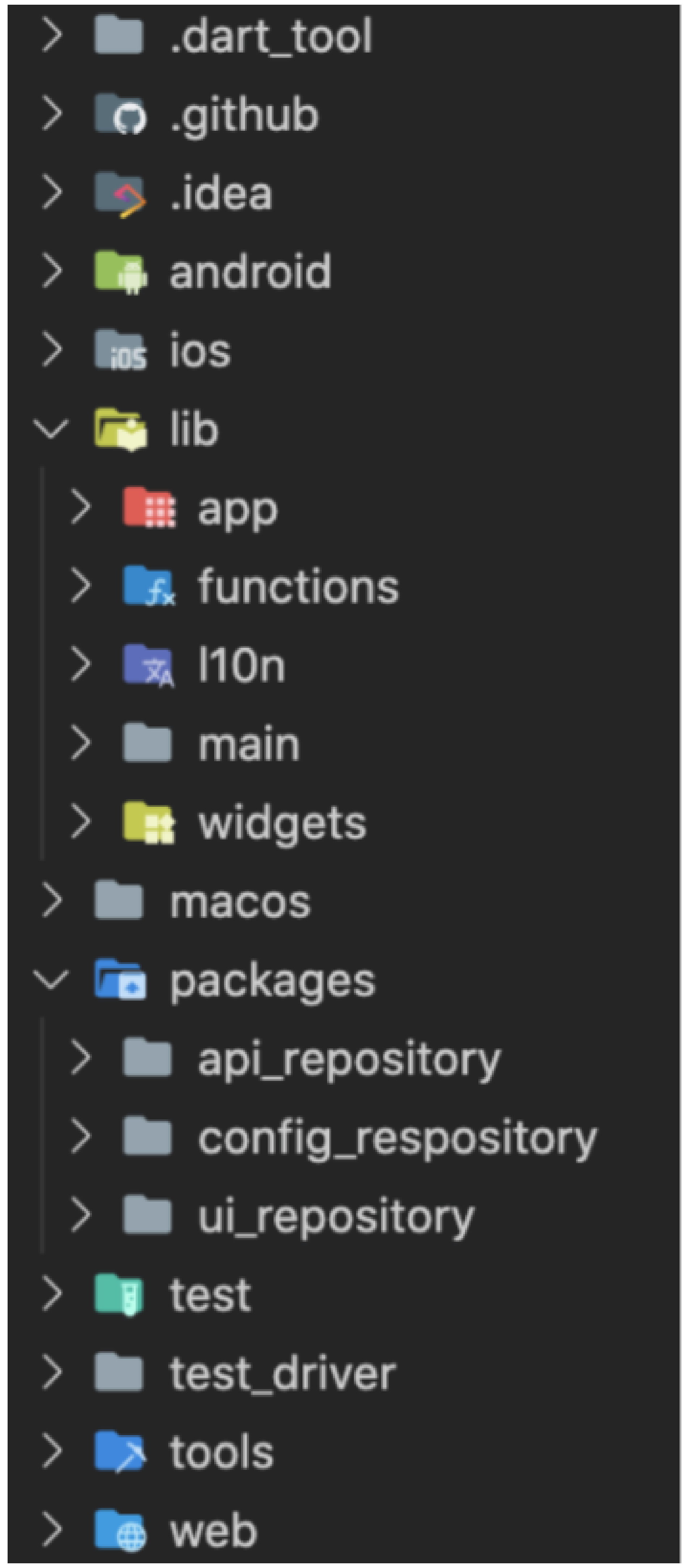


UI = *f*(state)

bloc

ReactiveX

Our folders’ structure and modules



```
> .dart_tool
> .github
> .idea
> android
> ios
v lib
    > app
    > functions
    > l10n
    > main
    > widgets
> macos
v packages
    > api_repository
    > config_respository
    > ui_repository
> test
> test_driver
> tools
> web
```

We always define with clarity all the responsibilities from the beginning of the project. This way developers can **easily navigate** throughout the project structure and correctly **follow the architecture.**

Every library, package, component, and widget has its corresponding place. By increasing the level of abstraction on how we code we can tackle the hardest problems and at the same time we are able to find patterns that **make solutions simpler**.

github/workflows

CI/CD scripts for automation.

lib

Inside lib we typically define a folder for each feature. We follow a feature-driven approach. Also we can have helper functions that we use across the app and internationalization to support languages. Finally we can have a widgets folder where we keep purely UI components.

tools

Here we define all necessary scripts related to tools that we use, so developers can easily access. For instance, testing, automation, coverage, code generation.

packages

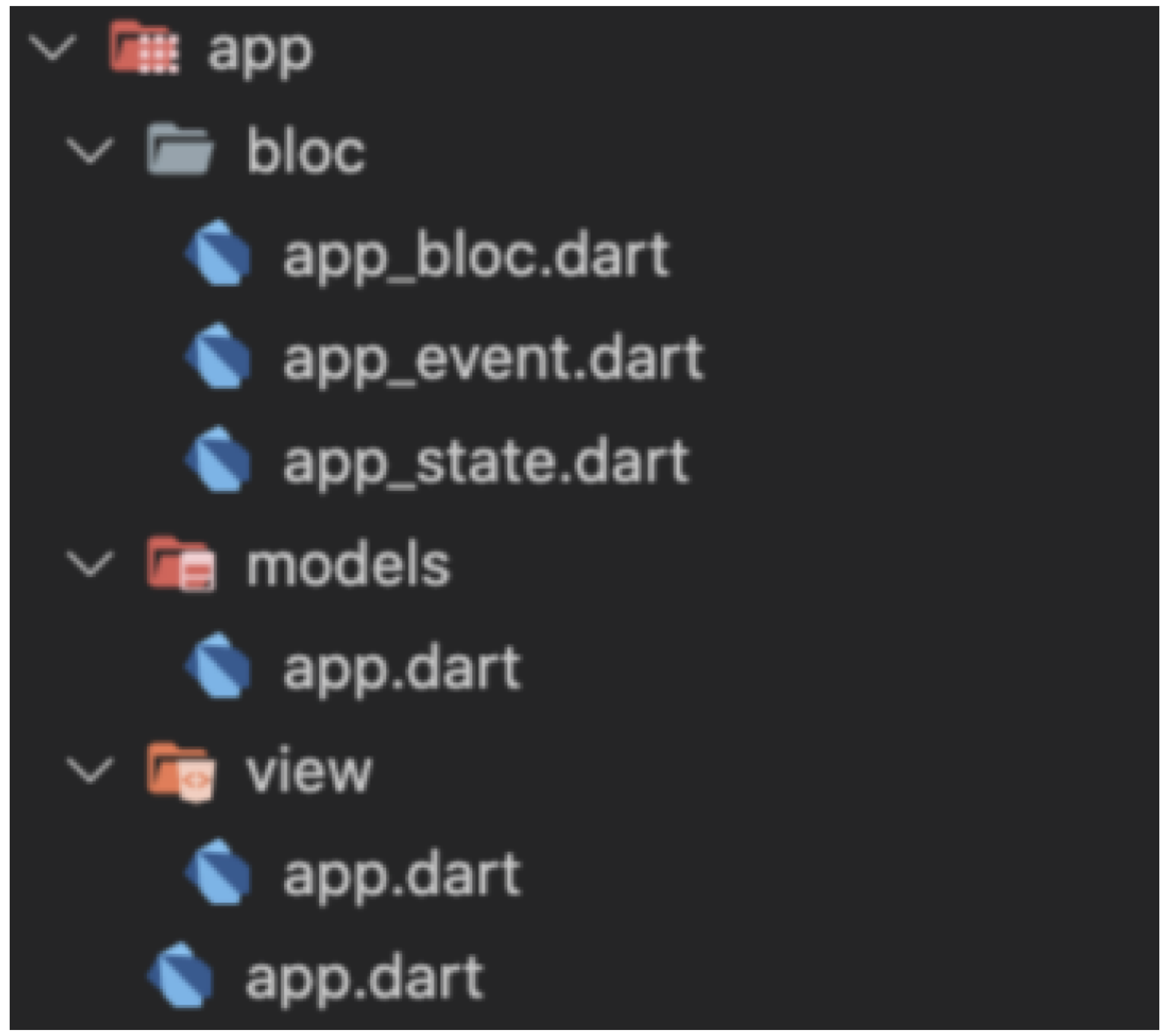
We define a package for each repository that is necessary. For instance we could have an api repository that communicates with an external api, third-party services that we use across the app, like persistence, notifications or a UI repository where we have files related to strings, font weights, colors, themes and assets.

test

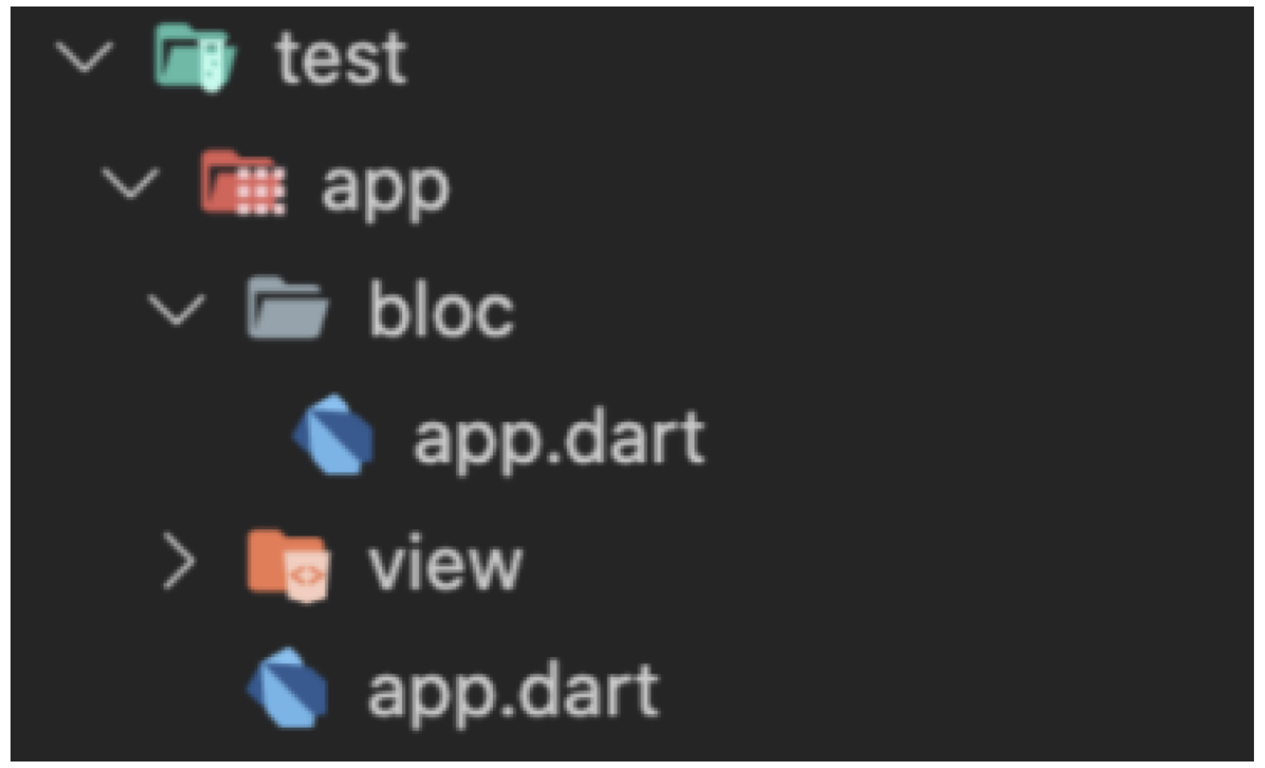
Unit & widget testing. Here we define a folder for each future that we defined in lib folder.

test_driver

Integration testing.



```
⌄ app
  ⌄ bloc
      app_bloc.dart
      app_event.dart
      app_state.dart
  ⌄ models
      app.dart
  ⌄ view
      app.dart
    app.dart
```



```
⌄ test
  ⌄ app
    ⌄ bloc
        app.dart
    › view
      app.dart
```

Feature-driven development

In our experience, organizing a project using a feature-driven approach achieves **better results** than using a layered approach. One major advantage is that developers **can focus on a single feature** without it affecting other features and potentially reduce merging conflicts. This happens because the feature is more isolated.

view

Here we put the screen of the feature with the necessary widgets. This folder is responsible to implement the screen (UI) that users are going to interact with. It can make use of several widgets and it will be a function of the state that will continuously listen for changes and re-render itself.

bloc

This is where we handle the current state of the App. The view communicates with the bloc dispatching events and listening to changes in the state. The bloc will be in charge of processing those events and recreating the state based on the old state. Typically it will communicate with a repository to exchange information.

models

Here you can find abstractions that define entities of the real world or entities that will help complete a feature in an easier way. By defining classes we can program with a higher level of abstraction.

Testing

Testing is at the core of our process. We focus on correctly evaluating the app to **assure high quality**. This way we can **prevent undesired bugs** and provide the **perfect user experience.**

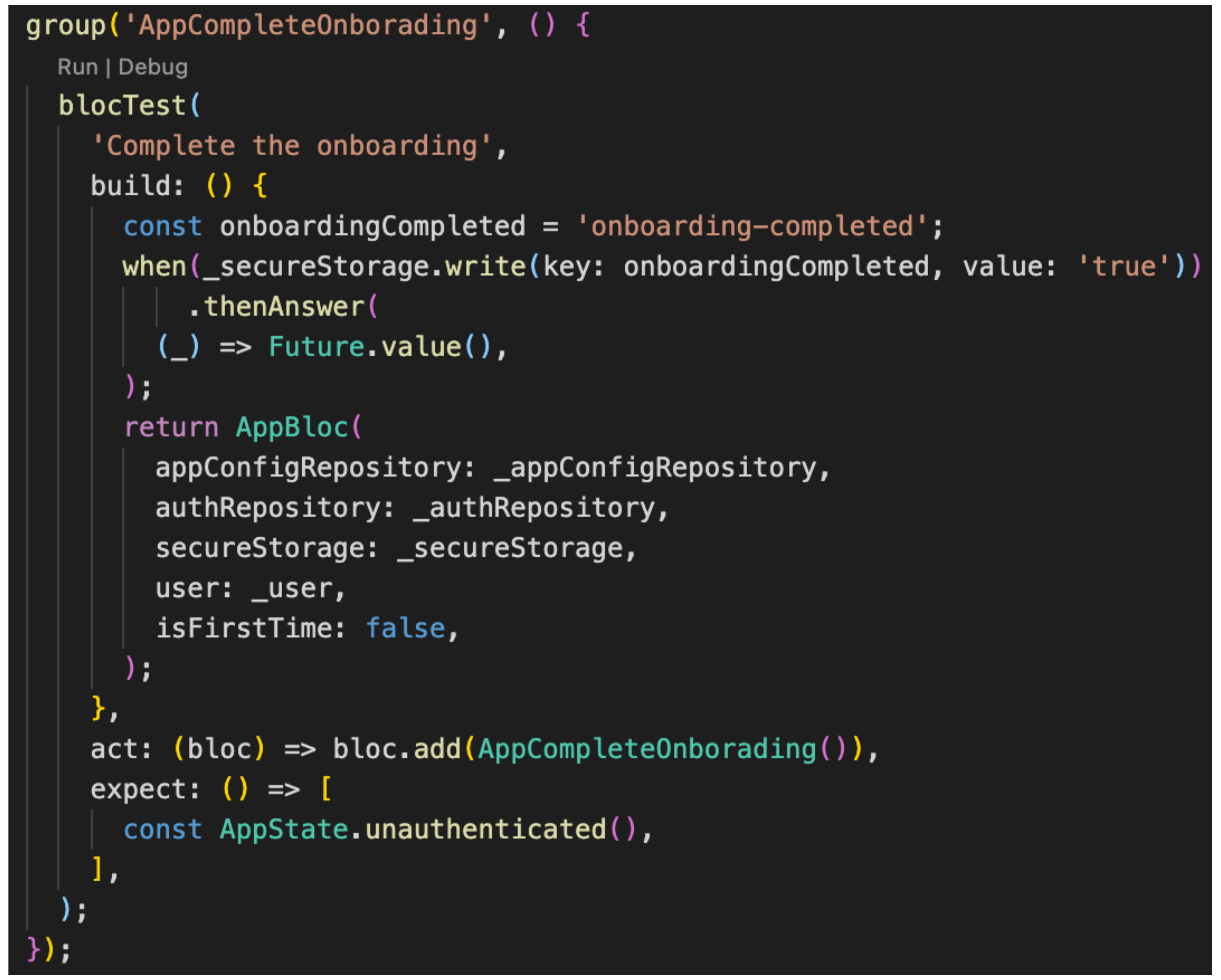
We carry out different kinds of testing, such as:

- Unit testing
- Widget Testing
- Integration Testing
- Golden testing

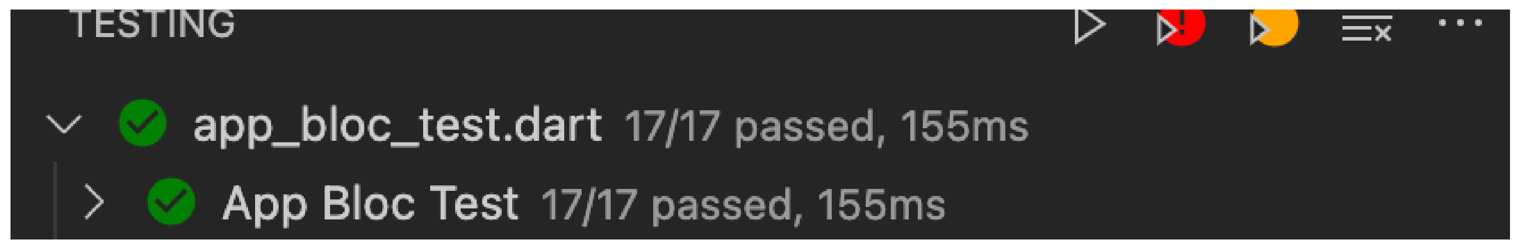
The beauty of Flutter also relies on the possibility of **integrating everything with this technology**. That's why we also perform **Quality Assurance processes with Flutter** in order to improve efficiency and quality testing.

We have **high-standard metrics** in our QA process to achieve the highest quality. One of our key metrics is **code coverage**: testing every line of code our developers write.

We also use other tools such as Icov to know exactly what **percentage of the app has been tested**. This allows us to clearly visualize in percentages whats lines of code have been tested and the ones who still need another test. We normally establish the threshold percentage we want to accomplish and then define what scenarios must be prioritized based on the **core interests of our client's business**.



```
group('AppCompleteOnborading', () {
  Run | Debug
  blocTest(
    'Complete the onboarding',
    build: () {
      const onboardingCompleted = 'onboarding-completed';
      when(_secureStorage.write(key: onboardingCompleted, value: 'true'))
          .thenAnswer(
        (_) => Future.value(),
      );
      return AppBloc(
        appConfigRepository: _appConfigRepository,
        authRepository: _authRepository,
        secureStorage: _secureStorage,
        user: _user,
        isFirstTime: false,
      );
    },
    act: (bloc) => bloc.add(AppCompleteOnborading()),
    expect: () => [
      const AppState.unauthenticated(),
    ],
  );
});
```



```
TESTING
∨ ✓ app_bloc_test.dart 17/17 passed, 155ms
  > ✓ App Bloc Test 17/17 passed, 155ms
```

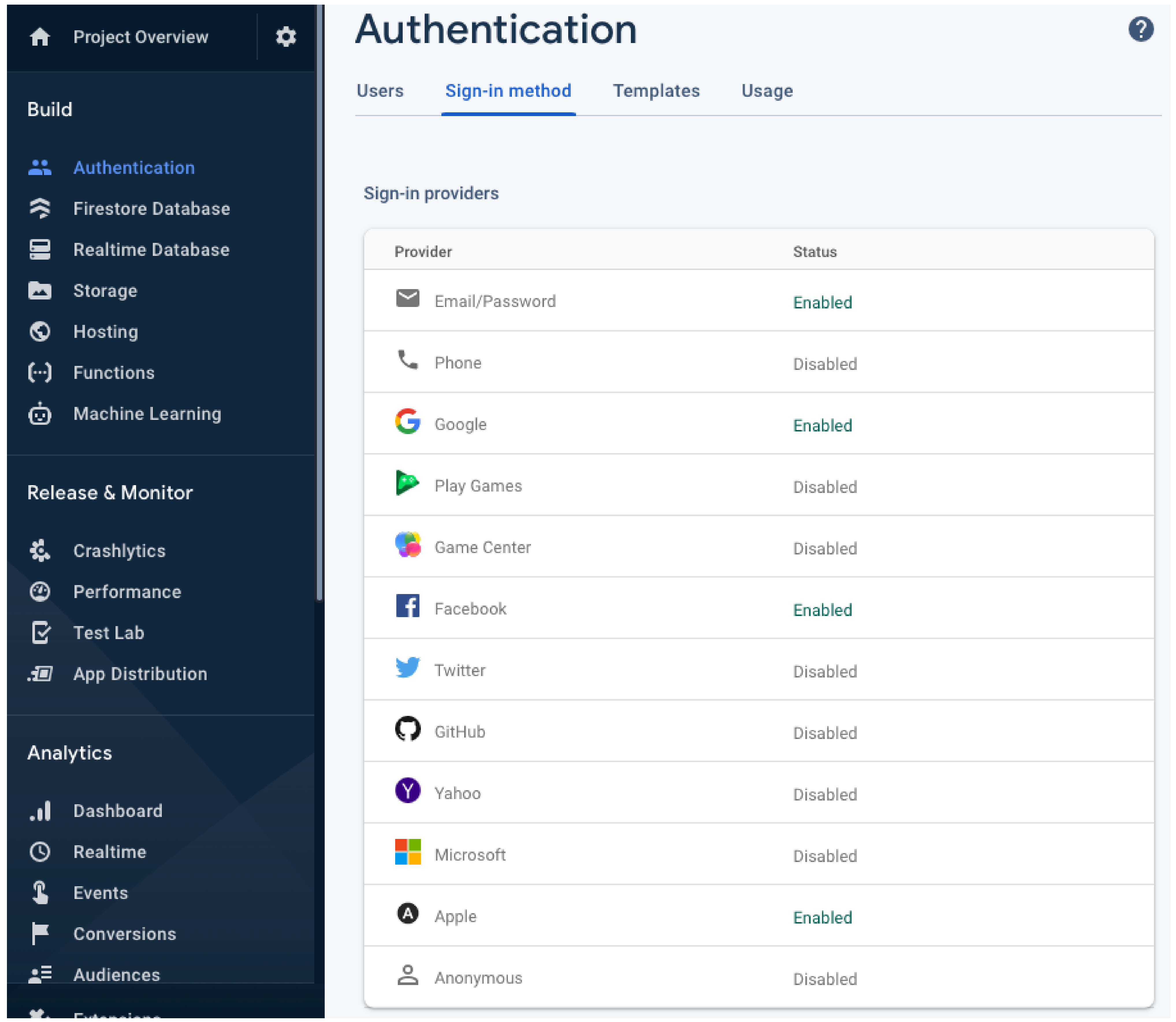
Back-End

Option 1. Flutter + Firebase

Firebase is a great **Backend-as-a-service (BaaS)** that provides hosted backend services such as a real-time database, cloud storage, authentication, crash reporting, machine learning, remote configuration, and hosting for your static files.

We especially use it for MVP and medium size apps because it allows us to have a **scalable database** and **cloud functions** that help the business times and needs.

Flutter + Firebase is a great combination to start a project. We use all the services it provides, each one providing a specific use for each client case. They are organized under the following categories: “Build”, “Release & monitor “, “Analytics”, and “Engage”.



Sign-in providers

Provider	Status
Email/Password	Enabled
Phone	Disabled
Google	Enabled
Play Games	Disabled
Game Center	Disabled
Facebook	Enabled
Twitter	Disabled
GitHub	Disabled
Yahoo	Disabled
Microsoft	Disabled
Apple	Enabled
Anonymous	Disabled

Opcion 2. Flutter + Serverless

Serverless is a method of providing Back-End services without the hassle of worrying about the underlying infrastructure. This way we can **focus more on writing business logic code rather than maintaining the server**.

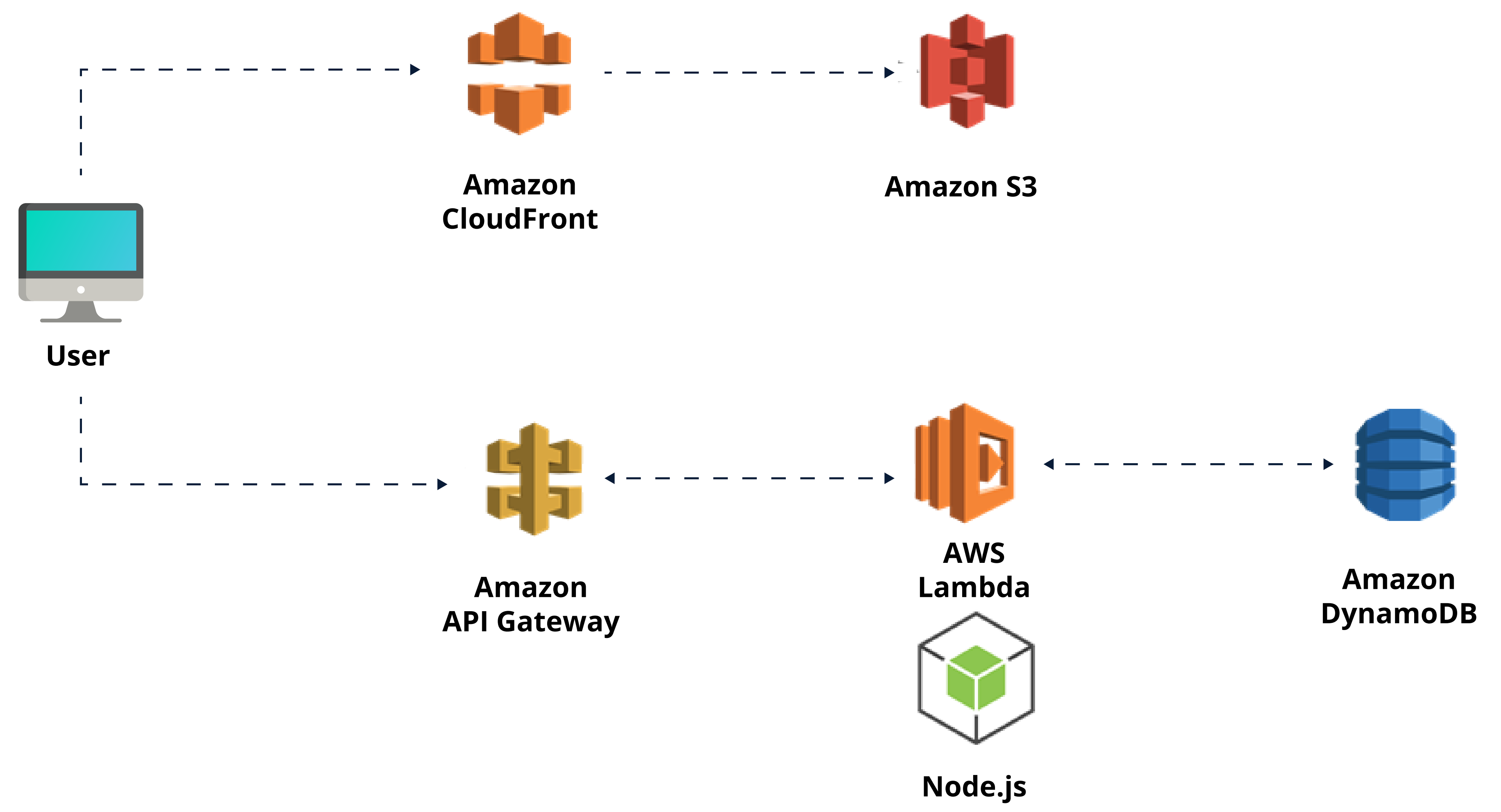
We suggest using this option when you want **more control over all the business logic and needs of the app**. The combination Flutter + Serverless provides more flexibility and the possibility for developers to focus on developing more features. As a result, this modern technology saves time, effort as well as cost.

Even though when using a BaaS sometimes you have to deal with some restrictions, when moving to Serverless it doesn't interfere since you can **define any business logic you prefer**.

In addition, developers can write server- side code and deploy it to the cloud in a quick and interactive way that **increases their efficiency**. As a matter of fact, clients can also save cost because they have the possibility of only paying for what they use, meaning the total number of hours a particular function runs.

Our favorite stack is a combination of Node.js (typescript) and Lambda functions, either G Cloud or AWS.

serverless



Opcion 3. Microservices and Containers

Using a **single microservice** will implement some business logic and might also expose an API so it can i**ntercommunicate with all of the other microservices**. By having this as a separate service, we won't have a big monolith and the **code will be much simpler and maintainable**. Each developer is able to select the best technology or programming language that fits better and we can easily re-write a microservice when we think it's becoming obsolete.

We suggest using this option in big projects when the client already knows the app counts or will count with a big number of users since day 1.

We choose to use a microservice architecture for many reasons:

- Possibility to independently scale each service
- Programming language heterogeneous
- Simple to maintain
- Highly reusability
- Reduced downtime through fault isolation

Even though our favorite stack is a combination of Node.js, Docker, and AWS, **each scenario will be different** depending on the selected software architecture. We feel very comfortable working with modern architecture like microservices yet we know there is not a perfect architecture rather is an architecture that fits a problem and teams must be prepared and know why they are choosing the architecture.

Continuous integration & Continuous Deployment

Continuous Integration is the practice of merging all of the developers' work to a shared mainline often. This gives them the confidence to include new features in the current codebase, knowing it will be fully tested.

Github Actions is our best option for CI since we already use GitHub for SCM. This way we can have instant feedback on each Pull Request.

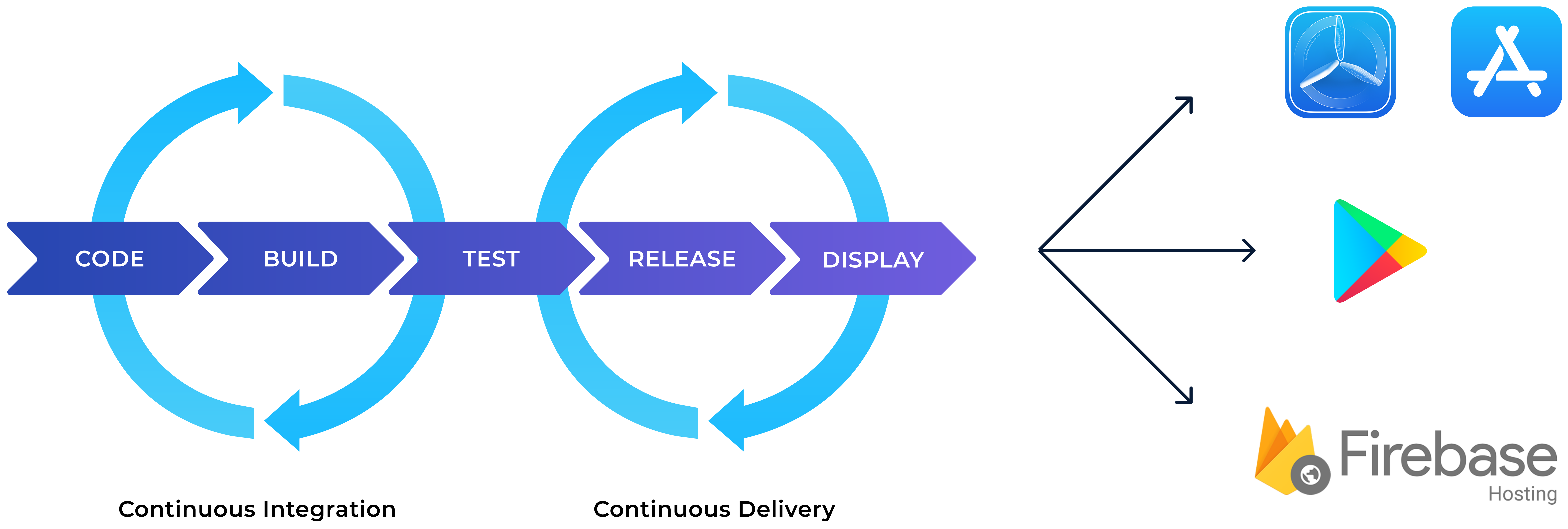
In addition, **Continuous Deployment** allows us to ship versions of the app more often. In order to not carry out repetitive tasks and risk skipping a step, we set up a CI/CD from day one so we can **automate** these tasks.

Codemagic is a great option for CD since it focuses on mobile apps and has great support for Flutter. It allows us to continuously deliver the latest version of the applications to our clients

We also integrate other **tools** into our workflows such as Slack, Jira, Firebase App Distribution, Fastlane, TestFlight, Google Play Console, and App Store Connect. Through automating all this work, we can get notifications, send emails with new builds, update Jira columns, and focus on continually developing new features.

codemagic
by Nevercode

CI/CD example diagram



Community

- Multiple **Libraries**
- Multiple **Reported Issues**
- Owner of unique **Meetup Flutter Uruguay Group** that belongs to Flutter Meetup Pro Network.
- Multiple **Stack Overflow answers.**
- Our own **Tech Blog** with the latest Flutter News.
- **Somnio Academy** (Coming soon! Flutter and programming academy on Uruguay).

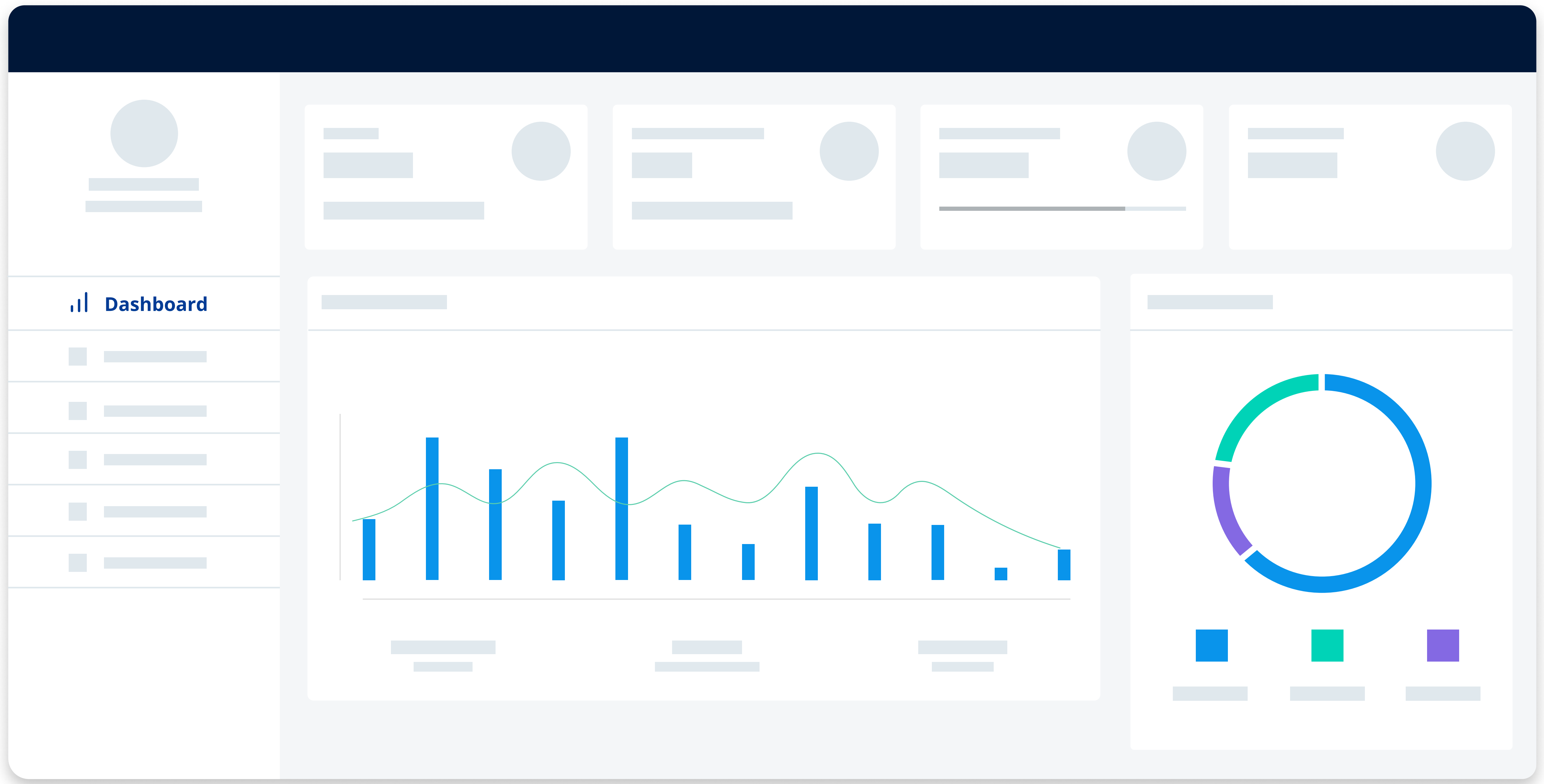
Community: Flutter Web Back Office

This open-source project consists of **rapidly creating and spinning up a Flutter Web Back Office with Firebase integration**. Out of the box and very quickly you will be able to **set up an admin dashboard panel** with authentication, CI/CD, and data visualization with tables and charts.

Some highlights of the project:

- CI/CD with GitHub Actions
- Firebase hosting
- Responsive design
- Role-based authentication
- Data visualization with tables and charts
- Advanced routing

Learn more about the project:
https://github.com/somnio-software/flutter-web-backoffice



Community: Flutter Firebase Starter

This open-source project integrates almost all **Firebase services that a production-ready app will need**. We've decided to focus on **documenting how we work** with all the services for a single project in order to be able to **reuse them** inside of other projects in the future. We've also decided to open source the project so we can continue growing and **contributing to the community**.

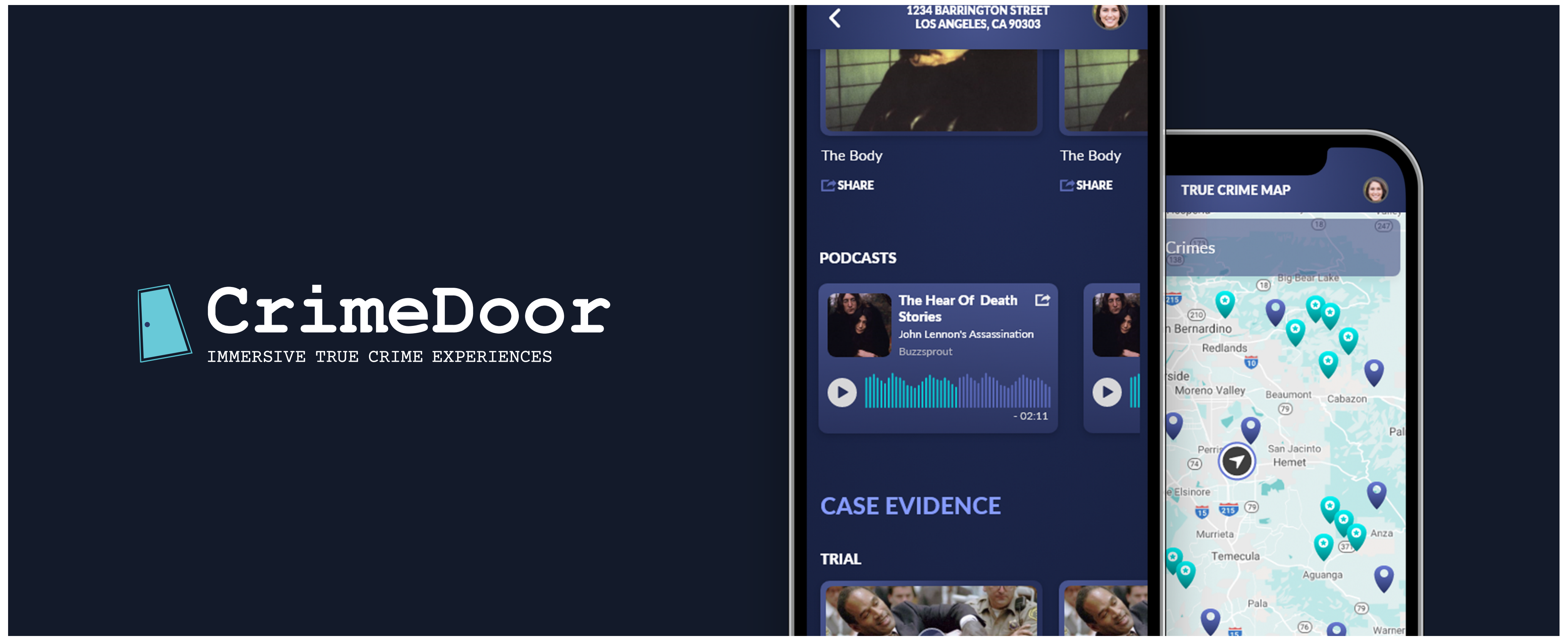
The main reason to do it is that we feel there wasn't a project completely focusing on this. We believe we can **share our knowledge and experience with Firebase**, which we use in a lot of our client's projects, to help everyone in the community.

Some highlights:

- Firebase features out of the box
- Scalable project structure
- Flavors
- Custom design
- Reusable services

Learn more about the project:
https://github.com/somnio-software/flutter-firebase-starter

Our Work: CrimeDoor


CrimeDoor
IMMERSIVE TRUE CRIME EXPERIENCES
1234 BARRINGTON STREET LOS ANGELES, CA 90303
The Body
SHARE
The Body
SHARE
PODCASTS
The Hear Of Death Stories
John Lennon's Assassination
Buzzsprout
CASE EVIDENCE
TRIAL
TRUE CRIME MAP
Crimes

CrimeDoor is a **crime database** for the most notorious unsolved mysteries. The app provides case information in several ways, such as podcasts, images, police reports, recreated crime scenes through **augmented reality**, and so much more.

Some highlights

- **#4 in App Store**
- **4.5 star rating** in App Store
- **+100** thousand downloads
- Client from Hollywood, LA, California
- 7 Somnio team members
- 8 months of development

Technologies we used

Flutter

We use in-app purchases to monetize the app and provide different modalities for users to choose, such as single purchases, subscription model (freemium), and pay up front.

Firebase

We use Firebase as a BaaS. Firebase also helped us integrate a lot of services in order to measure and provide a better experience in the app, such as push notifications, analytics, crashlytics, dynamic links, testlab, among others.

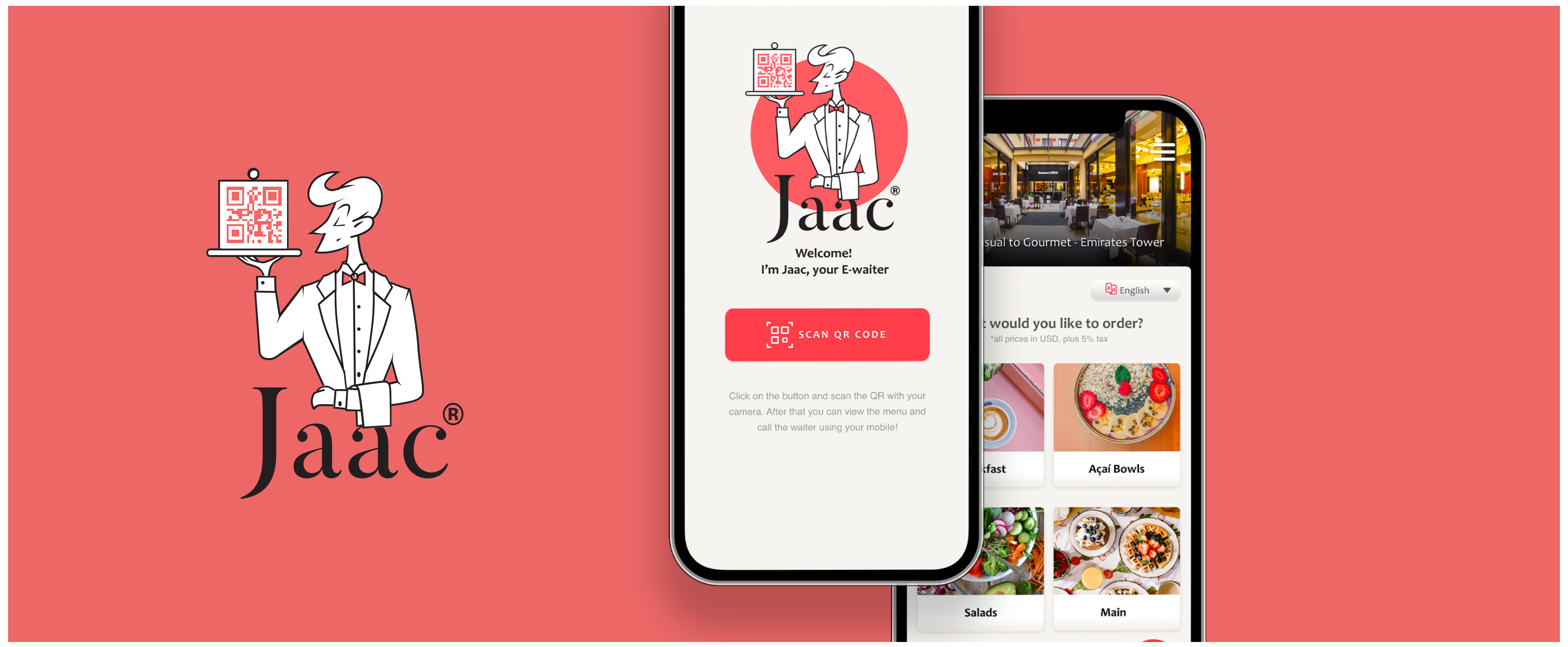
unity

We use Flutter's Unity Widget to enable bidirectional communication between Flutter and Unity so we can add Augmented Reality features directly embedded into Flutter.

algolia

We integrate Firebase with Algolia in order to have an engine that handles fast searches. We also provide Cloud functions to keep Algolia up to date with Firebase.

Our Work: Jaac



Jaac is the result of an innovative idea to revolutionize the restaurant industry. We built a platform that **improves customer service for restaurants** and at the same time helps them manage it better.

Some highlights

- **+100** restaurants using Jaac successfully
- Restaurants in the US, Germany, Dubai, Egypt, Spain, and many more
- Platform with heterogeneous technologies
- 1 and a half years of development
- 95% of users rate **5 stars**
- **2 platforms:** one Guest App and one Staff App

Technologies we used

Flutter, PWA, node, aws, Firebase, Jenkins, stripe, Pay, docker, PayTabs, PayPal, appium, serverless, Google Pay

This is just a quick overview of how we use **Flutter**.
Truth is, our everyday lives' sum up would be: eat, Flutter, sleep, and repeat this order.
Also truth: we love it.

Thank you :)

Schedule a meeting and let's talk about your project with Flutter!

Get in touch with us:

hello@somniosoftware.com
(+598) 98 168 142

Somnio
SOFTWARE