The Four Pillars of Building a Great Mobile App

Embrace

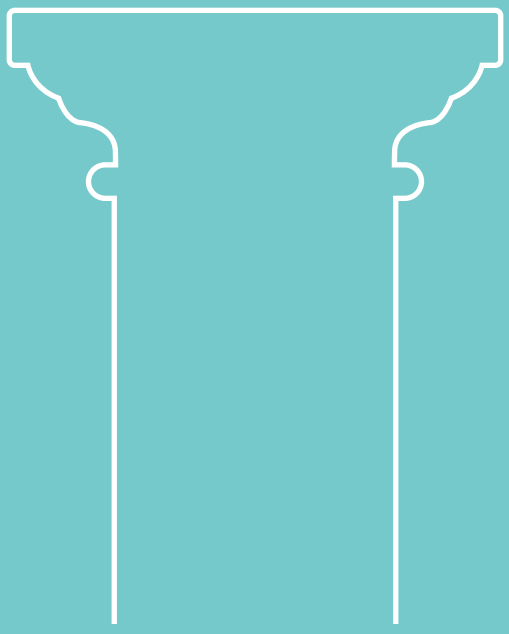
Introduction

As you build mobile apps, you should be future-proofing. This means being proactive about planning for and addressing the pains associated with growth. The development process should be as seamless as possible as you grow your feature set, user base, geographic footprint, and team size. If you focus on best practices for building, measuring, and monitoring your app from Day 1, you'll eliminate friction and innovate rapidly.

In this eBook, we'll show you why the following four pillars are important to building a great mobile app:

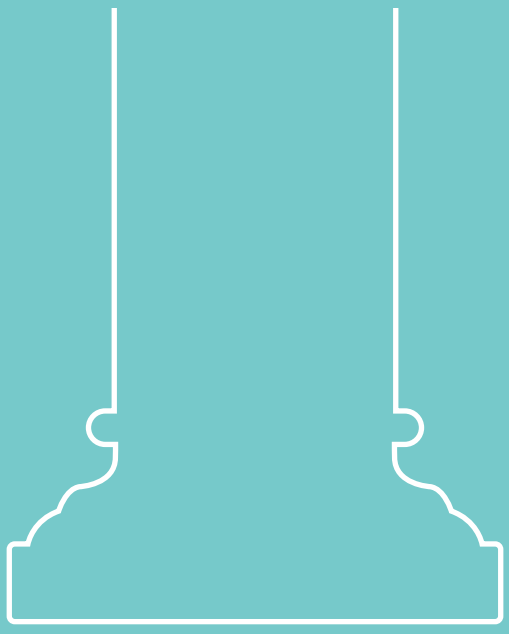
1. Design a Modular Architecture
2. Create a Structure for Features and Releases
3. Measure Stability, Performance, and Impact
4. Monitor Early

As you read, it will become apparent how closely related each of these pillars are and that shortcomings in one will affect the success of the others. For example, streamlining feature building, testing, and releasing requires a modular architecture. And, knowing what to monitor depends on how your users experience your app in production, which requires measuring its performance and stability. We've laid out the eBook in four distinct pillars, but we know you understand that designing and delivering mobile apps is part of an interconnected ecosystem.



PILLAR 1

Design a Modular Architecture



In general, designing a modular architecture is the best way to write software. This is particularly true in mobile, with its fast-paced, constant release cycles.

But what, exactly, do we mean by modular? Modularity means you can add or remove something without breaking the coherence of the whole system. In software terms, a modular architecture allows for features or individual pieces of functionality to be added or removed easily. The best example of the power of modular code is microservices, where you can incorporate services and software into your applications and systems with minimal effort. These integrations are frequently just a few lines of code.

Instead of approaching your mobile app as one monolithic piece of software, you'll be able to iterate on it faster if you design and build it in discrete, separate components. You'll quickly know which pieces of code would be affected when you want to implement a change. And, with the speed of feature production so critical to app success, you need to give your teams the power to innovate as rapidly as possible.

Best Practices for Designing a Modular Architecture

Apps without modular architectures suffer from reduced feature velocity, resulting in slower growth and scale. Your team is not building a static piece of code. They're building a foundation for your app – and business – that you'll adjust frequently moving forward. The following are common changes that growing applications experience, and our recommendations for how best to anticipate them. Making smart decisions now will save developer time later, when important changes need to be made. By keeping these principles in mind, you'll build the foundations of success right from the beginning. You'll grow, maintain, and scale your mobile app with minimal tech debt.

1 Write Code That Can Easily Be Rewritten

You know your code will be replaced over time, whether in small increments or through entire component refactors. The ability to safely and rapidly release new code is reduced if multiple feature teams have to touch the same code when

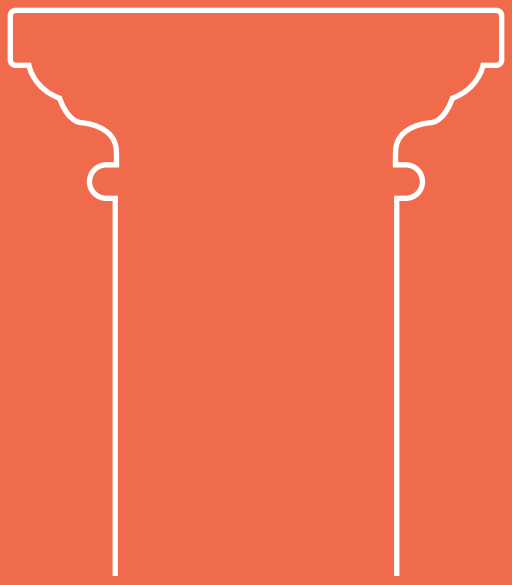
working on separate features. It's difficult to get the right people in the room, and it's almost impossible to ensure everyone knows all the effects of the code. If your code base is not modular, it's also likely you'll have to do extensive code rewrites for simple updates. To maintain engineering velocity, you need to be able to move fast and with minimal friction. To achieve this, we encourage you to design your app components so they can be easily iterated on and can be ripped and replaced rapidly. How? Decouple components, especially from a user interface (UI) standpoint. And, decouple logic where possible so that you can take advantage of and string together microservices quickly.

2 Create a UI That Can Account for Future Design and Feature Changes

It's inevitable that your app will undergo frequent design refreshes, so you should make decisions now that will simplify this process later. For example, put colors, font sizes, and font families into a theme file so you can make adjustments and changes quickly and easily. When Apple released Dark Mode in 2019, many app developers scrambled to redesign their apps. In contrast, developers who had committed to good modular design architecture from the beginning, like employing theming libraries, adapted quickly.

3 Build for Internationalization Early

You may initially launch in a single country or region, but it's important to plan for future expansion and internationalization. An excellent best practice is to build for internationalization (i18n) from the beginning, rather than trying to recode for it later at a significant tech cost. Take into account latency, intermittent connectivity, time zones, date formats, languages, and currencies. For example, you can store and transmit dates in International Organization for Standardization (ISO) formats, wrap dates and numbers in function calls to display the right format for a particular locale, or wrap user-facing static strings.



PILLAR 2

Create a Structure for Features and Releases

As your mobile app grows, so will the number of features and releases. Processes that were sufficient when the app was smaller and fewer developers worked on it will no longer work when several teams are building new features separately and concurrently. As the number of contributors to the code increases, it becomes more important to implement a structure for building, testing, and releasing new features. Otherwise, incompatibilities will be introduced by separate teams. The result? Your release velocity will grind to a halt.

Best Practices for Features and Releases

Without a strong structure for features and releases, deprecating a feature can affect the functionality of a newly released one. Or, releasing code for one feature might break functionality in another feature. It can also be difficult to determine whether performance issues in a new release are the result of one feature or another. With the innovation speed that growing apps require, your developers need to put careful systems into place to minimize problems stemming from feature management at scale. Here are a few of our top recommendations.

1 Make Version and Deprecation Decisions Ahead of Time

Before you deploy, think about how you'll support old versions, and when and how you'll deprecate them. Many users don't update regularly, so you'll need to plan

The Magic Laptop

Make sure you don't have a single point of failure. We had a partner whose entire deployment process became dependent on a single laptop after a developer left the company. More companies than you realize have a similar "magic laptop" or server. Don't let yourself become vulnerable like this! There should never be only one person who knows how to build, deploy, or configure your app. Automate and share knowledge from the very beginning.

ahead for how to enforce upgrades when the time comes. For example, you can add a script that checks the app version. If it's a version you no longer support, the script can display a clean UI notifying the user they're on an outdated version and must upgrade to continue.

2 Consider Feature Gating

Feature gating is the practice of building new features in the production branch and selectively controlling whether they are turned on or off. This approach is beneficial for several reasons. You can more easily test features during development or beta, and you have more control over how those features are released. Feature gating also simplifies the process of merging finished features into production, especially features that require several iterations.

3 Choose Appropriate Tools

As your app grows, your users will have different experiences that stem from unique combinations of several factors, including device, OS, region, network conditions, and version. As an example, you may have rapid adoption in India when your previous users were largely in America. This can surface a number of performance shortcomings that weren't previously apparent, including poor network conditions causing app crashes, or the OS killing the app on older devices due to the CPU being pegged.

To troubleshoot issues beyond code exceptions, you need information beyond crash reports. You need tools that provide the context to solve unexpected issues

Work Smarter, Not Harder

We encourage you to build your apps cross-platform with modern tools like React Native, Expo, and Flutter. For many types of apps, the performance costs are negligible, and with fewer developers, you can build a great app and achieve higher velocity.

as they arise. It's also important to keep your signal-to-noise ratio clean. This ensures you're not drowning your engineering, support, and DevOps teams in notifications and reporting. And, we encourage you to enable your users to submit feedback. You can save time by asking users what they were trying to accomplish rather than trying to recreate it from logs.

Platforms and tools purpose-built for the needs of mobile can save time and alleviate the pain of solving particularly challenging bugs.

4 Use Testing Resources Wisely

Take time to consider your testing ratio to maximize your limited resources. Unit tests are easy to write, tightly coupled to the code, and fairly robust. They're easy wins. Next, a healthy number of integration tests should be largely focused on fixing regressions. And finally, reserve scarce resources for end-to-end testing. These eat up the most developer and compute resources and become outdated frequently.

5 Automate Wherever Possible – Especially Deployment

We encourage you to automate whatever and wherever you can to make your developers' lives easier and to accelerate development and deployment. If you come across common mistakes that are easily made and easily automated, create a rule! Automation can help your team avoid those mistakes and reduce the cognitive load wherever possible. For example, don't make your developers think about where to insert semicolons. Instead, automate formatting and typechecking.

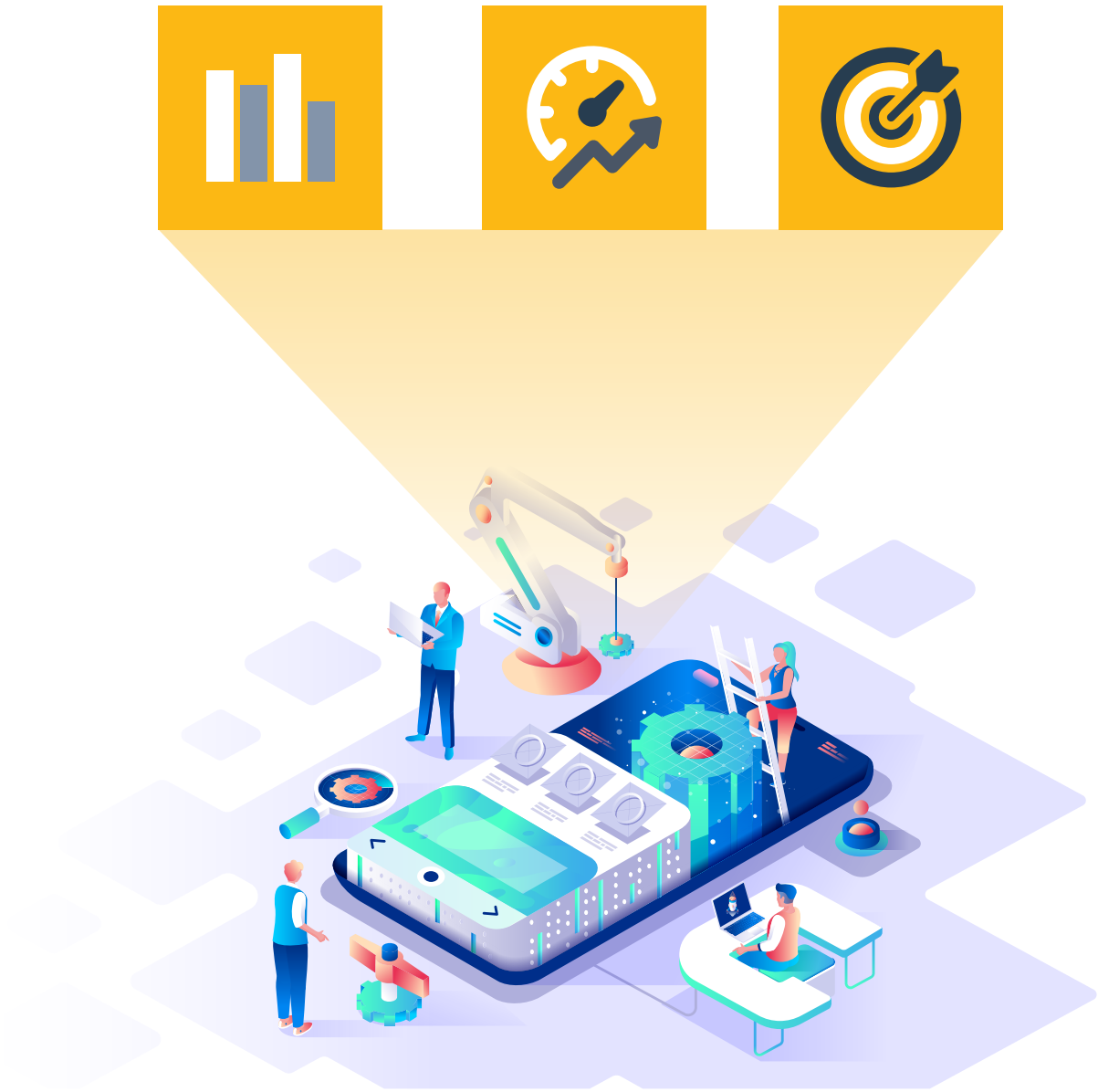
PILLAR 3

Measure Stability, Performance, and Impact

In mobile, there are too many variables at play to predict everything that might happen. Instrumenting and measuring wherever possible is the key to surfacing issues and trends as soon as they appear. To successfully instrument and measure your app, you need to start with a baseline. This enables you to immediately recognize improvements, regressions, and trends — across releases, operating systems, user segments, and regions — and take rapid action in response.

Broadly speaking, there are three key measurements:

1. **Stability**
2. **Performance**
3. **Impact**



Stability

Stability encompasses anything that causes app failures, whether it stems from the user, the app, or the OS itself. And, stability reaches far beyond simple app crashes. Anything that affects a user's ability to experience the app without disruption is a stability issue, and users are extremely sensitive to these disruptions. The majority of users will abandon an app after only three app failures. Getting stability right is critical to your commercial success.

Best Practices for Measuring App Stability

Stability issues can feel the same as a crash to a user, even if they do not directly cause a foreground session failure. For example, memory leaks can lead to out of memory exceptions (OOMs) which can cause an eventual app failure in the background. Users may reopen the app as a cold start, as if it had crashed. These leaks are often unreported in a crash reporter, leaving you with no stack trace to investigate what went wrong.

Slow network calls can block the main thread, resulting in a frozen screen or ANR (app not responsive). This experience can cause users to rage tap or force quit the app out of frustration. And, broken web views can become blank screens. To a user in the middle of a purchase flow, these provide the same experience as a crash. The end result is the same for you: poor user experience and the associated impact to revenue or engagement.

To avoid these situations, here are some key stability indicators you should always measure.

1 Percentage of Cold Start Sessions

Cold starts are a great metric to track because they can indicate the presence of other underlying issues. For example, the OS will kill your app if it's using too much memory or CPU or if it's hanging the main thread. If an app freezes, users will frequently force quit it, triggering a subsequent cold start. If you see cold start increases release over release, you need to dig in further.

2 Percentage of Sessions with Low Memory Warning

The OS will notify you if you have a memory warning. These warnings can indicate possible memory leaks that are affecting performance and possibly lead to the OS killing your app. If you see increases in memory warnings across releases, you may have an issue in your app.

3 Percentage of Sessions in Low Power Mode

It's important this number stays stable release over release. If you see a sudden increase, you may have introduced a feature that is draining the battery and producing a poor user experience.

Performance

Performance is a measure of how fast the app runs and is a key differentiator in the competition for users. Your performance KPIs are specific to the type of app you're building. For example, photo upload time is important for social media apps, while add to cart, checkout, and purchase times are critical for e-commerce apps. We encourage you to identify, baseline, and monitor the critical KPIs for your app.

Best Practices for Measuring App Performance

Here are some KPIs to measure and keep top of mind. Whatever performance metrics you're measuring, it's critical to record your baseline and track it over time. That's the only way you'll understand how changes are affecting the user experience.

1 Startup Time

For all apps, startup time is critical and directly impacts sales conversions and engagement. If you don't measure and optimize startup time, it's likely users will abandon your app before experiencing the key value proposition. Do you know how long your startup time is? Do you know your competitors' startup time?

Optimize Startup Time

Why is it so important to optimize startup time? For Amazon, reducing app startup time by 100ms resulted in a 1% increase in sales. The cost of latency can be staggering for large players, and in a fiercely competitive app marketplace, you can't afford to give up any market share.

2 Login Successes/Failures

Do you measure login time and the number of logins? If you see a sudden spike, your app may be logging people out.

3 Timing for First Party Networking Calls

Slow network calls cause user experience slowdowns that lead to app uninstalls or abandons and failures during checkout and purchase flows. For apps where users upload heavy assets like photos and videos, bad network connections can lead to frozen screens or the OS killing the app in the background. Slow network calls can be caused by several things, including poor network connection, misconfigured payloads, or an overloaded backend. You can reduce unnecessary friction in key user flows by working closely with your backend team. They can implement solutions like caching to speed up networking times.

4 Success/Failure Rates for Key User Flows

Success and failure rates differ depending on the app, but it's critical to ensure that the experiences that directly impact engagement or revenue do not suffer slowdowns.

5 Test Under Actual User Conditions

Consider where your users are and what devices and operating systems they are using. Replicate these environments in your testing to understand the true user experience when it comes to performance.

Impact

Your resources are limited, and it can be difficult to decide where to prioritize them. If you have several unsolved crashes affecting the same percentage of users, you need to make a decision on which crash to address first. The best way to evaluate issues is by measuring how they impact both user experience and business value.

As a social media app, Facebook cares deeply about user engagement. A bug affecting how long it takes users to create posts may be more urgent than one that affects how long it takes to upload a new profile picture, because users create and comment on posts far more often than they change their profile pictures. In contrast, e-commerce apps are focused on maximizing revenue. Bugs affecting the checkout or purchase process have far more impact than ones that create a slow user experience. But what if the slow user experience leads customers to abandon the app completely? It depends on how much both issues ultimately impact revenue.

Make sure that you always put what you're measuring into context and evaluate the impact. For example, you may choose to measure 4xx or 5xx errors for a certain network path. If most of your users with these errors don't end the app session or delete the app, fixing the errors may not be urgent. Understanding the user experience and revenue impact of what your team is working on is the most critical priority. The bottom line: don't spend a lot of time and resources on things that don't make a measurable difference.

Best Practices for Measuring App Impact

Social media apps measure impact by engagement, and a setback in the user experience might result in a decrease in median session duration and total time in-app each day. Failure to measure metrics like these will slow your ability to identify possible causes, such as a bug that is logging users out. E-commerce companies measure impact by revenue. Failing to measure metrics like average cart value at checkout or average purchase value can make it hard to quickly identify issues that affect your bottom line.

Here are some key impact metrics to measure, regardless of the type of app you're building.

1 Measure Performance Across Regions

If you're expanding to a new region, you can compare performance in the new region to your existing regions and ensure users in both locations have the same experience. Differing network conditions might surface underperforming user flows.

2 Measure by Personas

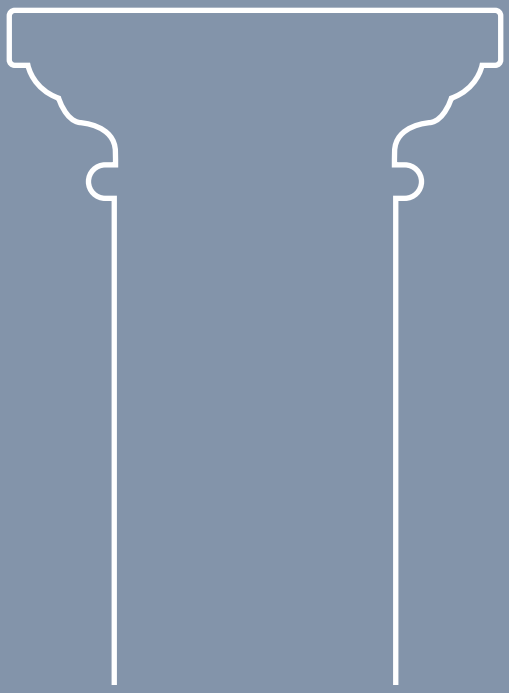
Identify your high-value users to understand their experience and how those experiences result in that particular impact. Similarly, you can monitor and measure your new user experience. Identify where you're retaining and/or losing new users. Do they have to download too many files in the sign-up process? That may be contributing to attrition.

3 Measure High-Level Indicators

Track indicators that will quickly alert you to underlying issues. Select metrics that should remain relatively stable, so that you can easily spot disruptions. Examples include average cart value or median session duration.

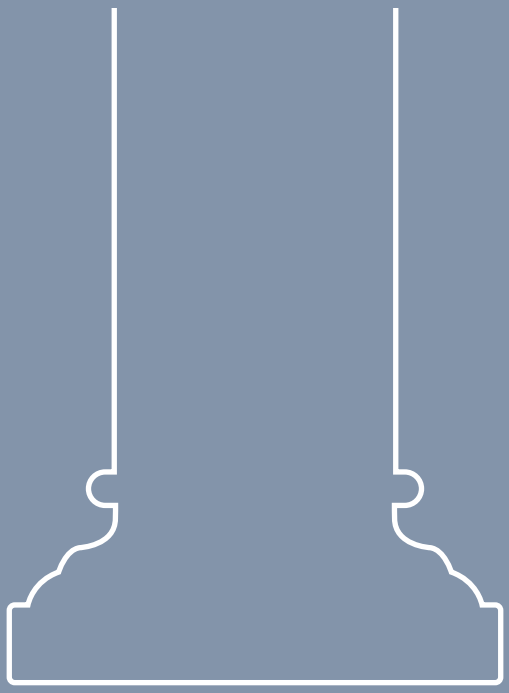
Measure *and* Monitor

It's important to regularly monitor the things you're measuring, so you get a regular baseline as well as regular updates on what's changing. This will help you understand how to interpret these measurements and what decisions to make based on that data.



PILLAR 4

Monitor Early



We work with mobile teams every day, from mobile-first and mobile-only to companies just building and deploying mobile for the first time. One of the biggest differentiators of a mature mobile team is how much effort they put into actively monitoring their apps in production. It's important to note that we're referring to mobile (or device-side) monitoring, not to server-side monitoring. In order to improve your app's user experience, it's vital that you understand how the app performs from your end-user perspective. Relying on server-side monitoring alone will not give you the context needed to debug many of mobile's toughest issues.

Best Practices for App Monitoring

The earlier you monitor, and the more you monitor specifically for mobile, the more successful you'll be. You'll be able to troubleshoot and get ahead of issues before users are affected. Here are some of our suggestions for how to do this the most effectively.

1 Monitor Your Own API

Mobile teams often assume that the monitoring being done by the backend team is sufficient. This assumption can result in a lack of coverage on device-side issues. For example, a server may send a 200 response for a network call that the backend registers as a success. However, the phone may be in an environment that causes a timeout. If the mobile developers didn't code for this possibility, the user could be experiencing a frozen screen or a crash. The server-side monitoring doesn't reveal how long the actual network call takes from the perspective of the phone, or what your end-user is actually experiencing. The device may have 20 concurrent network calls, which means it will be a long time before the device can process the call.

As the network call example shows, the way the user experiences an app can be different from what the server says. Teams should optimize for the user experience. Monitor all the way to the device so you can take detailed information to the backend team when you find issues.

2 Use Tools Designed Specifically for Mobile

It's tempting to adopt tools already in use by your backend team or to use product tools like Mixpanel to help determine the path a user took when they hit an issue. But mobile has so many environmental nuances and possibilities that aren't accounted for with tools that were created for web. Using tooling not created specifically for mobile leaves you with gaps and trying to predict what might go wrong before it happens. With mobile-specific tooling, you can access specific information - like full user session context, logging, network calls, user actions, device information like CPU and memory, and views and activities of your users - that will help you troubleshoot issues more effectively than using server-side tools..

3 Focus Equally on Feature Development and Monitoring

We understand that your focus is initially on features so you can build a strong user base. We encourage you to make sure you don't leave monitoring stability and performance as an afterthought. That's a sure way to lose users when things start to go wrong, and you try to fix issues with a brittle code base. Your best path to long-term success is to start with an equal focus on feature health and overall app health.

4 Create a Streamlined Monitor-and-Response Process Early on

Set up your monitoring process so issues and alerts sent to the team are triaged based on severity and impact on your users. The earlier this is in place, the sooner your team will develop best practices for problem solving. As your app scales, or when big issues pop up, your team will be ready to handle it.

5 Monitor a Broad Range of Metrics

Focus on more than crashes. Look at the app as a whole and make sure that you have adequate coverage for stability, performance, and user impact. All of these factors affect the user experience and the long-term success of your app.

6 Track Success and Failures for Key Flows

One of the most difficult issues you'll face is how to resolve a problem when you don't have all the information. One of our e-commerce partners began seeing 1% of purchases fail without a corresponding set of errors. This made it extremely difficult to troubleshoot an urgent problem that was affecting revenue and brand reputation. Ultimately, the company realized that the API call for the purchase was not being made. By tracking successes and failures on key flows like this, the team was able to see the API call that wasn't happening in the subset of purchases that weren't completing successfully. These problems are hard to resolve because the solution lies in an absence of information. Tracking both successes and failures will help you surface the answer.

Conclusion

Adopting these four pillars will help you create a robust mobile app that is responsive to even the most unexpected scenarios. The ability to identify, measure, and pivot toward trends, easily roll out features, and automate as much as possible in a highly competitive app marketplace is critical. Regardless of the size of your team, following these best practices will help you build and scale a successful mobile app.

Check out our webinar on the 4 Pillars at https://hug.army/4-pillars.

About Embrace

Embrace is a comprehensive observability, monitoring, and developer analytics platform built for mobile. Identify, prioritize, and solve issues faster with full access to unsampled user sessions. The world's best mobile-first companies use Embrace to ship better apps.

Try Embrace for free:
Learn more at embrace.io

Request a demo:
embrace.io/request-demo