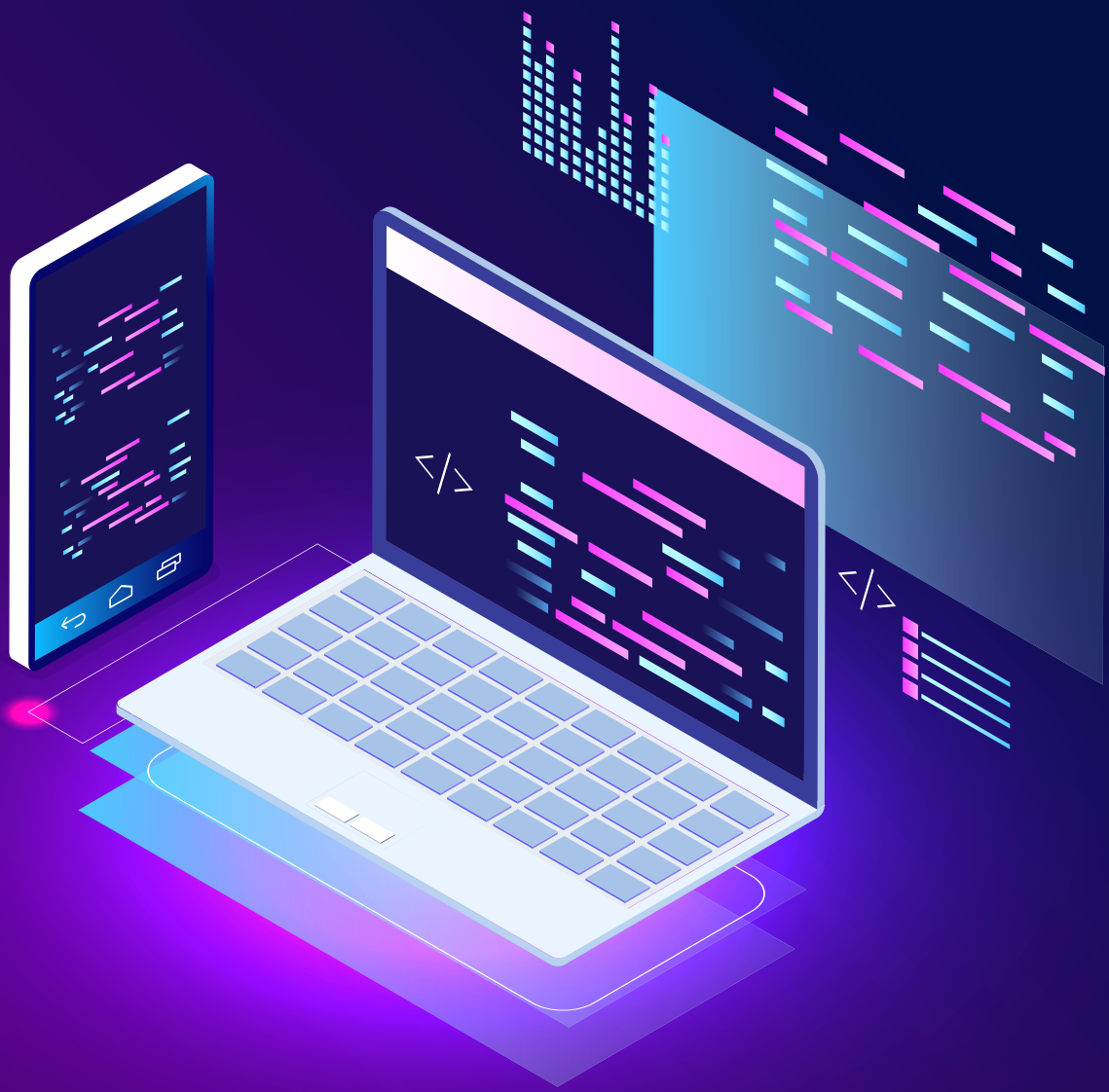
Progress® Kinvey™

Best Practices for Modern Application Architectures

An examination of today's best practices for software development crucial to building modern, scalable front-to-backend apps.

E-BOOK



Contents

The Rise of Digital Disruption

Companies are using rapidly evolving technology to disrupt traditional business models and create new ones—everything from Amazon and WeWork, to Uber and Lyft.

Since the web changed the world in the mid-1990s, we've seen mobile, conversational UX with chatbots and digital assistants like Alexa and Siri, AR, VR are emerging, and we have functional machine learning and AI in the marketplace.

That phenomenon helps bring into focus the two facets that define "Digital Disruption":

- The pace of technological innovation is accelerating
- Markets are segmenting into two classes of companies: disruptors and the disrupted

Customers expect modern, digital experiences and the companies that deliver—the disruptors—are the winners.

Applications are the tools disruptors use to:

- Reach customers and deliver better value
- Improve business processes and cut costs
- Accelerate past the competition

Today's Application Architecture Challenges

Meeting the demand for modern digital experiences calls for mastery of four primary challenges—multichannel, fast iteration, elastic scalability and adaptability.

Multichannels

Customers expect fast, consumer-grade digital experiences, whenever and wherever. So your challenge isn't just building "an app"—it's creating a unified experience across channels including web, Android, iOS and chat. Further, the architecture we choose affects our ability to address platforms that we may not yet know we need to accommodate.

Rapid Iteration

Thanks to the rapid pace of change, gone are the days when it was adequate to release a piece of software once per year, or even once per quarter. Today's customer expects rapid improvements—once per month, once per week—sometimes even daily.

Elastic Scalability

Scalability is a must for today's apps. Customers demand full functionality and availability at all times, however it's difficult to plan for those events where you might need instant scalability. For instance, imagine the very realistic scenario of a marketing campaign that is more successful than anticipated and suddenly tens of thousands of people are trying to download your app simultaneously.

Adaptability

Due to the rapid pace of innovation and the willingness of companies to try new things, apps need to be built in such a modular way that they can be easily adapted to suddenly changing circumstances.

Data Connectivity

A modern application needs to securely connect in real time to legacy data wherever it's located.

Building Modern Digital Apps Isn't Easy

Common obstacles include resources and legacy systems and architectures.

Resources

Many organizations lack the development resources and processes to continuously deliver native application experiences across web, mobile and chat at scale.

Legacy Systems

"Out with the old and in with the new" doesn't apply when ramping up to move at digital speed. New applications must be integrated with multiple enterprise and legacy systems—and the valuable data therein—while delivering modern performance. For example, a legacy ERP system might be old and slow, but it's loaded with rich data that you need to leverage while delivering a fast, modern mobile experience to your customers.

Legacy Architectures

Legacy architectures can't meet today's speed and flexibility requirements. The challenge is that implementing a modern cloud architecture is a daunting task when you consider ensuring security, scalability, performance and a host of other considerations.

The answer is a modern application architecture—one that allows for the speed and agility mandatory for digital transformation.

Tools That Can Help

What options do you have to help deliver these modern apps?

The Right Language

There are numerous programming languages to choose from and determining which language to use to build your next modern app is a critical decision.

That's not to suggest that there is one language with an overmastering advantage above all others; however, depending on your project goals, one language might be more appropriate than another.

For example, if your goal is to build an app that relies on machine learning, you might choose R or Python. If you're delivering a highly scalable web service, your choice would likely be Node.js and JavaScript. In any case, you'll be looking for a language

with wide adoption and developer affinity. Avoid niche languages because that makes it hard to attract and keep talent.

The Value of Open Source

It's advantageous to leverage open source as much as possible. There are numerous tasks that are common across many different types of applications. For example, Node.js users are aware that Express is a common framework for delivering web APIs. You could spend inordinate amounts of time writing your own web framework, but why? Someone else has already solved those problems.

There's a dynamic, vibrant open source community filled with rich expertise. Leveraging that expertise (and contributing your own to the community in turn) relieves you of the Sisyphean task of mastering every aspect of software development. Capitalizing on the advantages of open source frees developers to focus on what adds value and makes the application unique, rather than solving common problems.

Productivity Enhancers

Productivity enhancers exist to help you do common, repeatable tasks that add to your development time and slow down your time to market. For instance, most software engineers use an IDE because it helps them speed development, debug their code and produce better results.

UX Builders

It's certainly possible to write code to develop a good UI. Let's say you've done that, and part of your design is a button. Upon reflection, you decide you don't like the button's placement and want to move it down a little. If you're working just in code, you have to edit an XML or code file, change the number of pixels relative to the position, which in turn moves the button down. After that, you need to review your work and decide if the placement is ideal this time. Repeating that process multiple times is tedious and time-consuming, especially considering today's rapid development cycles.

UX builders provide an easy way to work in a WYSIWYG environment and move things to match your vision for the app while saving time.

Integration Tools

It's likely you'll have existing data that you'll need to make available to your modern apps. Yet traditional enterprise data sources are not optimized for modern application

architectures and multichannel experiences, and don't meet user performance expectations. Integration tools allow you to easily integrate your modern apps into any existing data source, transform the data to fit multichannel experiences, and deliver existing enterprise data with cloud-native performance.

SDKs

A good SDK or library will take common tasks that might take hundreds of lines of hand-coding and give you something that you can use by writing perhaps just a few lines of code.

Cross-Platform Frameworks

As discussed earlier, when developing a modern app you need to contend with multiple platforms and their differing programming languages and frameworks. iOS uses Objective C or Swift, Android uses Java or Kotlin, chat uses a variety of API formats and many other frameworks. Cross-platform frameworks allow you to write code for all the different channels you want to make use of.

The advantages include better utilization of development resources by writing once and deploying on many platforms. For instance, if you wanted to refine a bit of your Android UX, you could do that while still being able to share the bulk of your code.

Development Practices—The Best of Agile and Lean

An agile mindset and staying lean doesn't mean rigid adherence to any one methodology.

Software development should be done with an "agile mindset." Agile is touted as the way to write software, and it means different things to different people. In this context, we're not talking about agile methodology, but being and staying agile. Core principles of an agile mindset include:

- Build and iterate fast
- Get continuous feedback from your colleagues, sales teams and users

- Incorporate that feedback into iterations
- Build more with less

When development teams adopt Agile, there are generally three patterns they follow:

1. Bolt Agile onto their existing processes—which typically fails.
2. Adopt a specific Agile methodology, whether it's Scrum or Kanban or extreme programming, and replace their legacy methods entirely, even if it doesn't always make sense.
3. Look at what other teams have done and try to incorporate that to fit how their team works and communicates.

No one methodology is appropriate for every team or organization. Find one that is the closest fit and makes sense, and then iterate on that methodology. Review each completed project, see what worked and what didn't, and change processes accordingly.

Stay Lean

The most critical thing to remember from Lean is "seek feedback early and often," from focus groups, customers, prospects and the sales team, and continually adjust your plan based on that feedback. Doing so will help tremendously toward avoiding false starts and dead ends. Measure your results to determine if you are indeed on the right course.

Next, utilize MVP effectively, focusing on the key words "minimum" and "viable." You want to produce the minimum unit possible that will deliver a viable result that can be tested, so you can get that all-important feedback.

Embracing Continuous Delivery

Continuous delivery has two subcategories. Embracing both results in continuous value.

Continuous Integration

The point of continuous integration is to continually integrate any code you write into the mainline branch of your source repository, under the principal that the mainline branch should be deployable at any time should that become necessary. You should also rely heavily on automated testing.

Continuous Deployment

In its strictest sense, continuous deployment means that as soon as you merge something into your master branch, it gets deployed immediately into production or is released to your customer. While this is a solid deployment practice, it isn't always practical. What's important is to deploy on a regular basis.

At Progress Kinvey, we deploy changes to our production service multiple times per week—approximately 40 per month. The SDK team deploys a new version every two weeks.

When you deploy code that frequently, it doesn't mean that everything is generally available when you deploy. But using tools such as feature flags, A/B testing and early adopter/beta programs lets you get the code out to users for validation, without making it GA.

The Point of Both is Continuous Value

You engage in continuous integration and deployment to deliver continuous value to the customer. A partial feature that delivers value is better than waiting until the entire feature vision is complete. You also reduce the risk of failures in the complete feature set.

Platforming for Success

Leveraging the cloud and serverless.

The Cloud

The cloud offers several unique benefits to developers and in turn, customers:

- Provision only what you need

- Elastic scale based on utilization removes uncertainty from capacity planning and offers protection from unanticipated spikes in demand

- If you architect your app to take advantage of the cloud's redundancy by default through multiple regions and multiple availability zones, you get built-in redundancy and failover

- Thorny problems, such as security, are entrusted to the experts at your cloud partner, freeing you to focus on application value

Serverless

With the cloud, the base unit is a virtual service—you provision a virtual machine, deploy your code there, set up your service, get your OS, etc. Serverless is a platform for you to implement application logic and focus on the code, not the infrastructure.

Serverless is not the same thing as "Function as a Service" (FaaS). That is certainly one model of serverless, but not the only one. Serverless is any paradigm that allows you to focus on delivering value and not on hardware or resources.

Application Platform

This is where everything comes together. At the platform level, the app frontend is connected to the various backend services, supports Agile and continuous workflows and includes productivity enhancers to automate important but repeatable tasks, while the developer focuses on the unique value of the application.

Application Architecture

Two key architectural patterns are decoupled development and layers of abstraction.

Decoupled Development

When developing a modern app, it's important to separate the project into components so team members can focus on specific areas to increase velocity and reduce risk. For instance, if you're developing an app that needs to connect to an Oracle database, ordinarily you'd need to wait for IT to provision an Oracle server. In a decoupled development model with the frontend separated from the backend, the developer can design the frontend working against mock data, while the backend

developer works on the integration with the Oracle database. At the end they have a middle layer that helps join the two with little effort.

Layers of Abstraction

Layers of abstraction provide frontend developers with a single interface. The idea here is to enable the developer to access many different data sources or APIs, with middleware acting as a façade for backend functionality. This eliminates the frontend developer from having to master multiple backend systems.

Functions as a Service

With FaaS you write the smallest amount of code possible and invoke that from your client app, with the key benefit being that each function can scale independently based on its unique needs. For instance, a hashing function that is used for authenticating users and is CPU-intense would have a different scaling profile than something that is making an external call and spends most of its time at idle waiting for I/O to complete.

Microservices

These small, lightweight, single-purpose services should be designed to handle a distinct subset of the overarching application's functionality. The thing to remember about developing microservices is the word "micro"—microservices can become hard to maintain if you try to give an individual microservice the functionality of several.

Cloud Services

These no-code services implement some backend app need, such as cloud data store, file store, user management and integrations with external data sources and SSO providers. This is another tool to help you avoid becoming bogged down with details that don't add value to the app.

About Progress Kinvey

A modern platform for rapidly building complex enterprise apps and scalable consumer app experiences.

The approaches discussed in this paper can all be applied with Progress Kinvey. Kinvey breaks the mold of traditional approaches to innovation by being purpose-built to deliver enterprise-grade multichannel experiences running on a modern serverless cloud platform.

With Kinvey, use a single language to write once for native apps across multiple platforms. Use web-based development and a rich collection of UI components to build truly native apps with NativeScript and your choice of JavaScript, TypeScript, Angular or Vue.js.

Easily build a modern serverless backend that will auto-scale to the highest levels, have the flexibility to support frequent changes, and integrate easily into Systems of Record and authentication using a configuration-based low-code approach.

About the Author

Michael Salinger is Sr. Director of Software Engineering for Progress Kinvey, where he leads the team responsible for the development of the Kinvey Serverless Cloud. Michael has extensive experience in cloud, serverless technology, web, mobile and backend systems.

Explore Kinvey

About Progress

Progress (NASDAQ: PRGS) offers the leading platform for developing and deploying strategic business applications. We enable customers and partners to deliver modern, high-impact digital experiences with a fraction of the effort, time and cost. Progress offers powerful tools for easily building adaptive user experiences across any type of device or touchpoint, the flexibility of a cloud-native app dev platform to deliver modern apps, leading data connectivity technology, web content management, business rules, secure file transfer, network monitoring, plus award-winning machine learning that enables cognitive capabilities to be a part of any application. Over 1,700 independent software vendors, 100,000 enterprise customers, and two million developers rely on Progress to power their applications. Learn about Progress at www.progress.com or +1-800-477-6473.

2019 | RITM0055419

Progress®