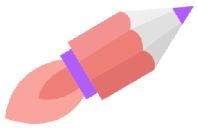


crowdbotics

The Non-Technical Founder's Guide to Building an App

A beginner's guide to building an app—for non-technical founders who want to build an app but don't know where to start or how to build/manage a team.

Table Of Contents

1. A Simple Guide to Native, Hybrid, and Web Apps 4

2. How to Choose the Best Architecture for Your Web Application 10

3. Key Factors To Consider When Hiring Contract Engineers 20

4. What Every Non-Technical Founder Should Know About DevOps 26

5. How Teams Can Get to Market 4x Faster with Automated Design-to-Development 32

6. Build Your App with Crowdbotics 41

Hey there,

If you downloaded this ebook, it's because you're someone who wants to build an app. We think that's pretty great!

Building an app for other people to use and enjoy can be one of the most exciting and rewarding things you can do as a founder...but it's not always easy, especially when you don't have the technical background and skills that other founders used to build their apps (think Mark Zuckerberg and Jack Dorsey).

Not having a technical background can deter some people from ever bringing their ideas to life.

But it's not going to deter *you*, right?

We know it won't, because by downloading and opening this ebook, you're already one step closer to making your dream a reality.

We hope the content you read throughout this ebook provides you with the knowledge, motivation, and resources needed to keep taking more steps.

To your success!

crowdbotics

Ch. 1

A Simple Guide to Native, Hybrid, and Web Apps

When developing an application, it's important to think about the different platforms that it will eventually need to run on. Nowadays, there are many different platforms through which your customers may want to access your application, and it is important to understand the benefits, drawbacks, and use cases of each one in order to determine which platforms you should support.

When it comes to accessibility on a mobile device, there are three main types of applications that you must consider: native apps, hybrid apps, and web apps. Each of these app types can be leveraged in different ways depending on how suitable they are for your product. In this post we will discuss the features of each in order to help you best understand which one to choose.

APP TYPE	PERFORMANCE	FEATURES & API ACCESS	ACCESSIBILITY	COST & TIME	DEVICE STORAGE	USER EXPERIENCE
Native	Fastest speed and efficiency as it is directly built on OS's platform	Access to all the devices technologies and features and full API integrations	Can be directly accessed on the platform's app store	High time and cost of development as each platform must be created individually	Must be downloaded onto device	Best UX since app is specifically tailored to the platform
Hybrid	Slower than native apps as they are not usually optimized for their platform	Access to all the devices technologies and features and full API integrations	Can be directly accessed on the platform's app store	Single codebase allowing for faster and cheaper development	Must be downloaded onto device	Decent UX, but some features that were generalized across platforms may not work as intended
Web	Can be decently fast, but speed is entirely dependant on internet connection	Limited access to device's features, and many basic functionalities like push notifications may not be possible	Must be loaded as a webpage every time	Single codebase allowing for faster and cheaper development	Takes up no storage as it's loaded every time it's opened	Poor UX, as app is not specifically optimized for the device it is running on

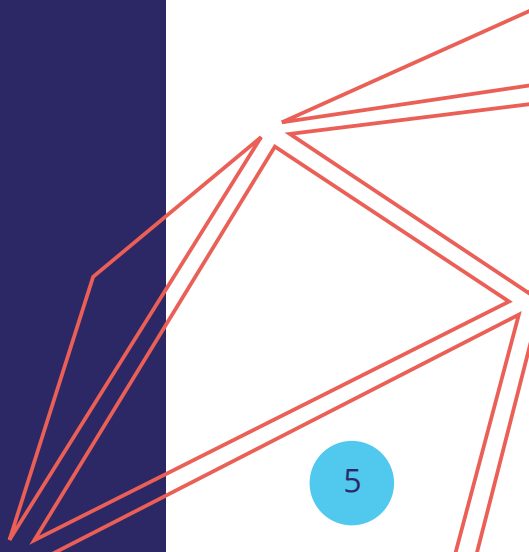
A quick comparison of the different app types

Native Applications

Native applications are specifically and uniquely developed for the platform that they are running on. This means that if you want your app to run on iOS devices, you will develop the app on Apple's platform (using Objective-C or Swift in Xcode), release it on the App Store, and have it only be accessible to iOS (iPhone) users. In order to release the same app on Android, you would have to redevelop the app from scratch on Android's platform (using Java or Kotlin in Android Studio) and release it on the Google Play Store. This app would only be accessible to Android users.

This process brings up an obvious con of native application development: high development costs and time. You will need to redevelop the app for every single platform that you want it to be supported on, and in turn, also need to independently provide support, maintenance, and updates for each of these platforms.

These costs do, however, come at a benefit. Native apps generally have the highest performance and best user experience of any of the app types. Since these apps are developed directly on the platform, they provide access to the widest range of the device's features, allowing you to easily access functions like the GPS, Camera, or Push Notifications.



They will look the best on their devices, too, as they will most likely have design elements that match and stay consistent with the platforms' design choices.
They will also have access to the widest and most clearly defined set of APIs, and be directly accessible to a targeted and defined customer base through the platform's app store.

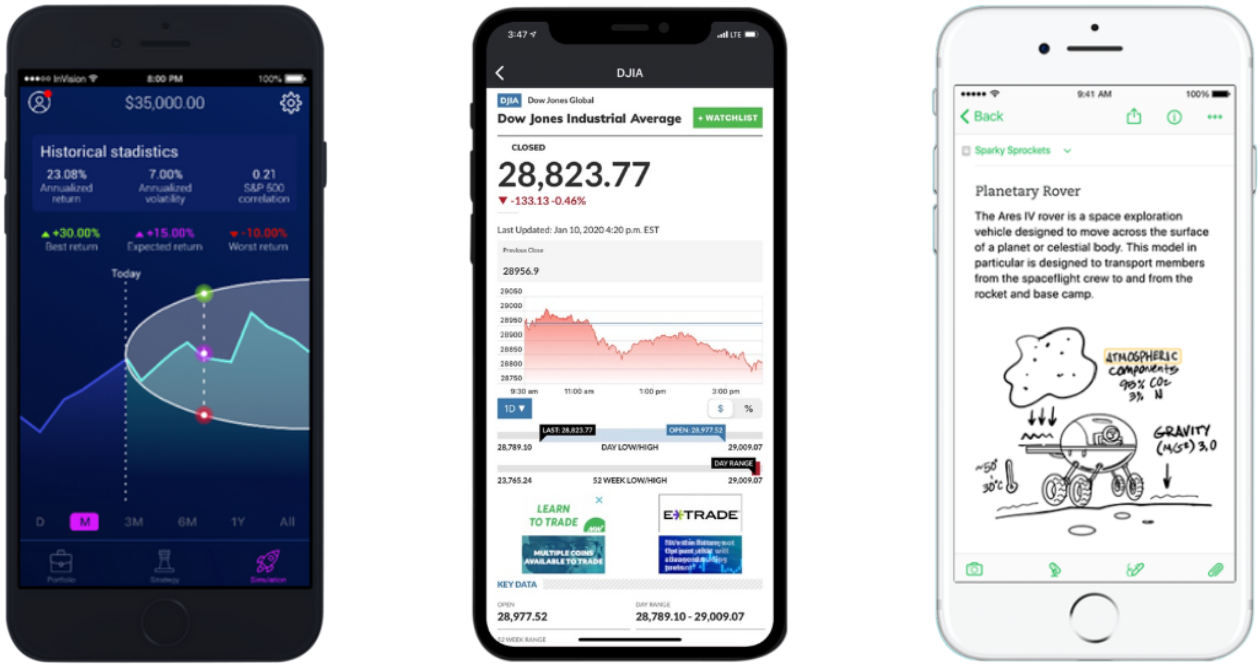
Apps like Aura (built with Crowdbotics), Twitter, and Waze are all natively built on their platforms, allowing them to easily access key features like Apple Health, the camera, and GPS

Hybrid Applications

Hybrid apps can be misleading, since you initially install them like a native app through the platform's app store. However, once installed, they are actually running a web app that is disguised by some wrapper to look like a a mobile app. Rather than being developed directly on the platform that you want to release the app into, hybrid apps are developed like web apps, using Javascript, HTML5, and CSS.

The biggest advantage to hybrid apps over native ones is the simplified codebase. You only need to keep track of one set of code for all your platforms, making maintenance and updates much simpler, and keeping your overall costs and development times lower. This single codebase means that your app will also be easier to scale, as you can release it to all platforms at once with minimal customizations needed for each. Additionally, most hybrid apps can usually still access the device's main features.

There is a cost to this flexibility, however. The biggest downside of hybrid apps is that they usually do not perform as well compared to native apps. Since they are not written using the platform's default language, they may not be as efficient as they can be on that platform. It may also be hard to ensure consistency across platforms, as differences in the different devices may cause some generalized features to not function as intended.



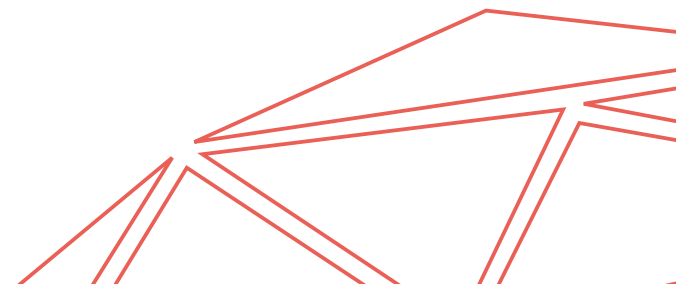
Carl (built with Crowdbotics), MarketWatch, and Evernote all use hybrid platforms, simplifying the development and maintenance process for their teams, but still giving their users a fully functioning mobile app

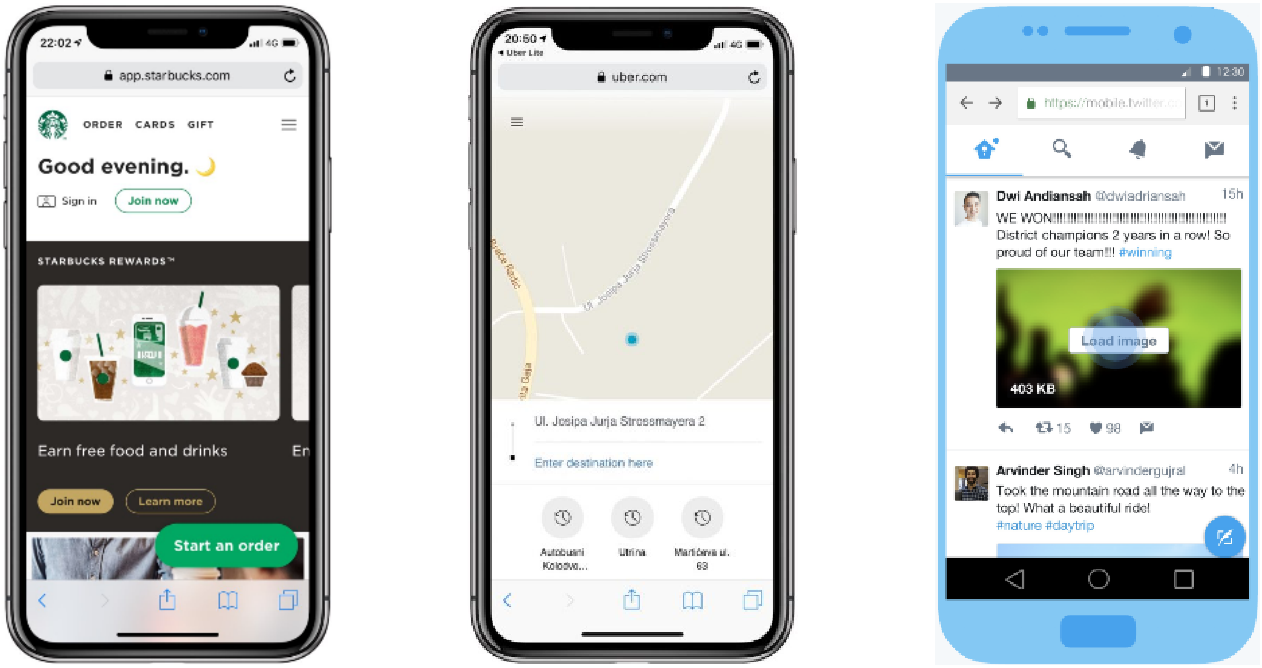
Web Applications

Web apps are just websites that are optimized for mobile devices. Like a website, they are accessed through a browser rather than installed as an app, but they do provide greater functionality and interactivity than a website. Similarly to hybrid apps, they are developed using web technology tools, like Javascript, HTML5, and CSS. However, unlike hybrid apps, they are not packaged in a wrapper that allows them to be downloaded like a normal app, and must always be accessed via URL in a browser.

Similar to a hybrid app, a major advantage of web apps is that they only require a single codebase in order to be accessed on all devices, again saving a lot of time and money in development. Since they are loaded in browsers, this opens up the potential opportunity to have your app be accessible on computer browsers as well. Additionally, since web apps continuously access all their data from servers, they do not have to be downloaded onto the device and can save a lot of storage space for users.

Web apps do, however, come with many drawbacks. The main one is the lack of available features and functionalities. Since the apps are not loaded onto the OS of the device, they may lack access to a lot of features that traditional apps use. For example, a web app will not be able to send push notifications to its user, which can be crucial for user retention. Additionally, since they are spawned from a single codebase, like hybrid apps, web apps may not perform as consistently across platforms. Lastly, since they need to be reloaded in the browser every time they are opened, they will also be inaccessible if the user has a poor internet connection.





Many traditional apps, like Starbucks, Uber, and Twitter, also have web apps for users who may need quick access to some basic features without having to install the entire app

Conclusion

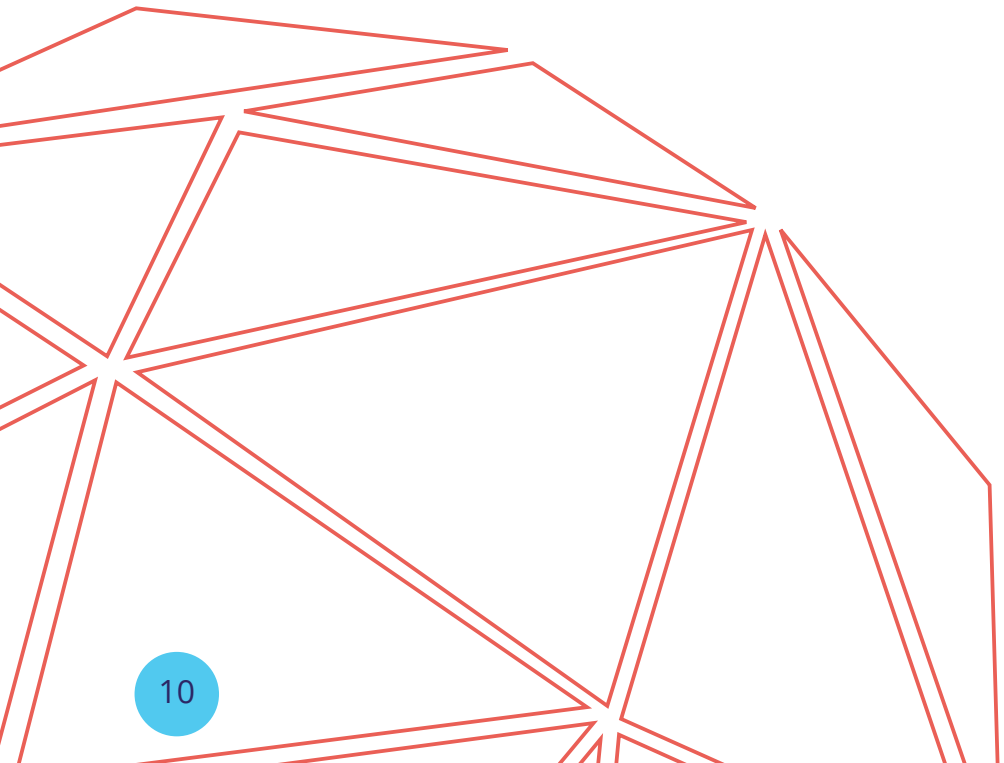
Overall, all three app types have their benefits and drawbacks, and it will come down to your specific needs when deciding which one to move forward with. Native apps will without a doubt have the best performance; however, if time and cost are a factor, your ability to develop them may be hindered. Web apps are usually the simplest to make, but this simplicity comes at the cost of functionality. Hybrid apps offer a good medium between a more efficient development cycle and abundant feature accessibility, but this compromise may not be necessary if costs or advanced features are not important to you. In the end, you will need to consider the pros and cons of each before making your final decision.

Ch. 2

How to Choose the Best Architecture for Your Web Application

Web application architecture describes the relationship between web app components (i.e. - user interfaces, databases, and middleware systems) and the way they interact with each other.

In other words, it provides a structure for how the client and server are connected. Properly designed web application architecture ensures that all of your components interact properly and serves as a strong foundation for expanding the app in later rounds of development.



How Does Web Application Architecture Work?

There are two subprograms, or sets of code, that are common for any web application: client-side code and server-side code. These programs run separately, yet concurrently, with the shared goal of delivering a seamless web experience for users.

- **Client-side code is the code that resides in the browser and responds to user inputs**

- **Server-side code is the code that resides in the server and responds to HTTP requests**

When developing a web application, the developer is responsible for deciding which code should go on the server and what they should do in relation to the client-side code. Any code that is capable of responding to HTTP requests can run on a server, and languages like PHP, Java, Python, C#, and Ruby on Rails are widely used for server-side coding. The server-side code is also responsible for creating any page requested by users as well as storing various types of data like user-profiles and inputs.

On the other hand, the client-side code communicates through HTTP requests exclusively and cannot read server files directly. Instead, it is parsed by the web browser and reacts to user inputs. In contrast to server-side code, client-side code can be viewed and modified by the user, and a combination of HTML, CSS, and JavaScript is utilized to build pages and content.

Types Of Web Application Architecture

We classify the different types of web application architectures based on the way the app logic is distributed between the client and server. The most common types of web architectures, along with examples of each, can be found below:

1. Single page applications (SPAs) –
Single Page Applications are increasingly popular due to their minimalist layout and architectural structure. SPA architecture is organized in such a way that only part of the page content gets updated when a user moves on to a new page, so there's no need to reload the same components. This alone makes it extremely convenient for both developers and users alike! Utilizing popular JavaScript frameworks like Angular and React, developers use SPA architecture to deliver a unique and interactive user experience by allowing single-page apps. **Gmail**, **Google Maps**, and **Facebook** are just a few great examples of Single Page Applications.

2. Multi-page applications –
Multi-Page applications are very common on the web since all web applications used MPA architecture in the past. This architecture type reloads web pages for sending from/to the server through the user's browser, and developers generally opt for MPA architecture if the application is pretty large. **Amazon** and **eBay** are two of the most well-known multi-page applications around today.

3. Microservices –
Microservices are a kind of service-oriented architecture (SOA) used to build distributed software systems. With this style of architecture, developers build web apps using a collection of loosely coupled services that can be independently deployed. In turn, this functionality fragmentation makes it easier to build, expand, and scale an application.
Some well-known projects with Microservices architecture are **Netflix, Uber, Spotify,** and **PayPal**.

4. Serverless architectures –
With this type of application architecture, developers no longer have to configure or manage servers using server management software. However, that does not signify a complete lack of servers—third-party cloud providers like **Amazon** and **Microsoft** offer virtual servers that dynamically handle the allocation of machine resources.

5. RAD stack –
RAD stack is a relatively new player and is a combination of React Native, APIs, and Django. It allows developers to assemble applications of any size quickly and deploy them in critical contexts. Moreover, it's the default stack of the **Crowdbotics App Builder**! Here are a few case studies featuring applications built with Crowdbotics: **Prehab 101, Solace,** and **Aura.**

Strengths and Weaknesses of Different App Architectures

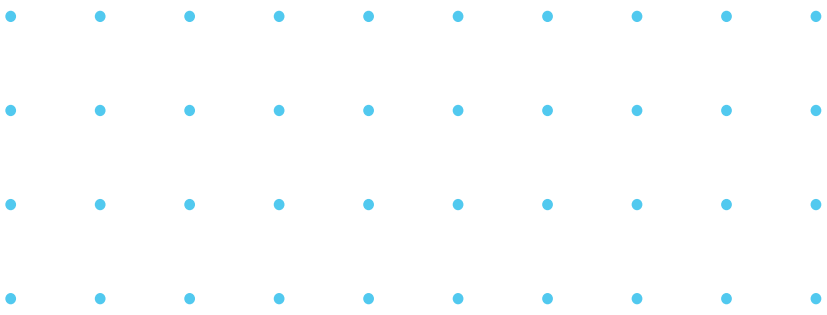
1. Single Page Applications (SPA)

Strengths

- Super fast performance compared to traditional architectures
- Excellent functionality on both desktop and mobile devices
- More flexibility and responsiveness since there's no need to reload or re-render web pages
- Simplified and optimized development

Weaknesses

- Heavy browser workload
- They require high data protection due to their use of cross-site scripting (XSS), which makes it easier for hackers to get access to the client-side code and potentially add harmful scripts



Strengths and Weaknesses of Different App Architectures (continued)

2. Multi-Page Applications (MPA)

Strengths

- Rich functionality, as MPAs allow for integrating a lot of features while maintaining an intuitive interface
- High SEO optimization can be achieved through multi-page apps since they enable distributing various keywords among different pages–unlike SPAs, which can end up cluttering everything on a single page
- Better analytics, as MPAs can be easily tracked and monitored by analytics tools like Google Analytics

Weaknesses

- A lot of complexity in backend development. MPAs heavily depend on server-side code, which means that developers have to spend more time on backend development
- Low performance and speed. MPAs are a lot bulkier than SPAs, which leads to reduced loading speed and less responsiveness
- Complicated debugging as developers need to check the GUI (Graphical User Interface) and relations of each page to ensure there are no broken requests

3. Microservices

Strengths

- A clear division of the application by modules helps to easily understand how any part of the code works and to add new features conveniently
- High scalability. Developers can add new services at any stage of the development process without altering the entire architecture
- High availability. The application will continue to work even if non-critical services break
- Ability to select a variety of tools and technologies when developing each service
- Ease of deployment compared to other architectures since services are independent of each other

Weaknesses

- The complexity of the development
- Difficulties in support since each microservice needs to be maintained separately, which requires constant or automated monitoring

4. Serverless Architectures

Strengths

- Lower cloud costs
- Reduction of development costs
- High scalability
- Faster releases
- Integrated registration and control mechanisms

Weaknesses

- Limitation of resources like runtime, memory, throughput, and CPU usage
- Security issues due to many applications running on a common platform
- Limited options for monitoring and debugging

5. RAD Stack

Strengths

- Highly flexible and adaptable to change
- Thanks to code generators and code reuse, there's less need for manual coding
- The use of code generators and reusable code means that you won't require as large of a team to accomplish tasks. This also means that you and/or your team can focus on more valuable work
- Incredibly useful when you're looking to reduce your overall project risk
- The use of scripts, intermediate codes, and high-level abstractions makes it easier to transmit deliverables

Weaknesses

- RAD projects can fail if developers are not committed to delivering software on deadline
- Requires highly skilled designers or developers

How to Choose the Right Architecture For Your Project

Choosing the appropriate application architecture sets the foundation for your entire application's development, so it's essential to consider the whole development process and its future expansion when selecting it. After all, the application architecture is not something that can be easily changed later! It's definitely worth your time to do a little research in order to determine that you're making the right decision for your needs.

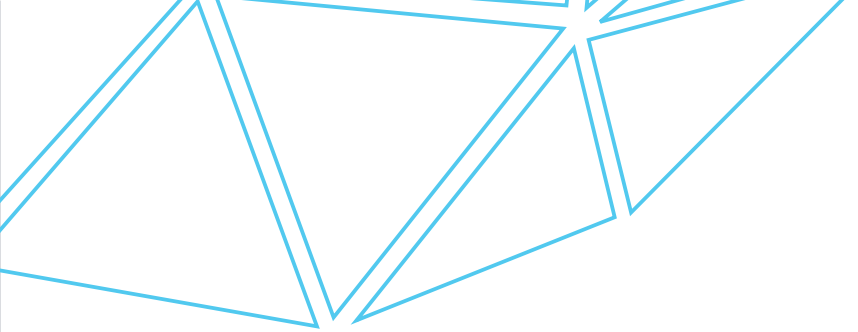
Multi-page applications are a strong option if you have to deliver a lot of content. They may not be well-suited for real-time responsive applications, but they work well as an enterprise application architecture. Large companies with a wide range of services and products will benefit more from using the traditional MPA structure. Online stores, company websites, catalog sites, and marketplaces are a few examples of large businesses that should consider taking this route.

In contrast, single-page applications are well matched for dynamic applications with small volumes of data. They are also a great option if you are planning to create a mobile app down the road. While the main downside of this architecture is SEO, it is well suited for Software-as-a-Service (SaaS) platforms, social networks, and closed communities since they don't need to be search engine optimized.

Microservices architecture is suitable for large and complex projects, as each service can be modified without having a detrimental effect on any other existing blocks or modules. For instance, if you have to update payment logic, there's no need to shut down the whole website during that time. However, if you need a fast solution like a prototype, small application, or an app with a tight deadline, a microservices architecture may not be the right solution for you.

If you don't want to manage or support the servers or hardware infrastructure required for the application, a serverless architecture is going to be your best bet.

If you want maximum scalability and performance across all platforms, RAD stack is the way to go. Rather than being specialized for a few niche use cases, RAD is built to support a huge range of business types and app requirements.



Trends and Best Practices In Web Application Architecture

As technology evolves at a rapid pace, so does web application architecture. One popular trend you'll see is the use of service-oriented architecture. With this architecture type, most of the code for the application remains as services, and each service has its own HTTP API. This allows one segment of code to make requests to another segment of code that may be running on a different server.

Another major trend that we've already touched on is single-page applications. The UI of this web application is delivered via a rich JavaScript application, and it resides in the user's browser over various interactions. It also uses AJAX or web sockets to make asynchronous/synchronous requests to the web server, eliminating the need to refresh the pages. This offers a more robust experience for users thanks to limited page loads and interruptions.

With the above two trends, web applications have become more sophisticated and capable of proving an optimal view on multiple platforms and devices. While most of the code for the application remains the same, it can still be viewed comfortably on smaller screens.

Best Practices For Web Application Architecture

In order to provide users with a great web experience, you should go beyond just having a functional web application. Here are some best practices that you should keep in mind:

Consistency –
The architecture you select should provide a uniform approach for solving all of your development problems, and you should analyze the application requirements to pick a solution that covers most of your development goals.

Fast performance –
t's better to keep the architecture as lightweight and responsive as possible. Analyze some of the best web apps in the industry—measure their page speed and responsiveness in order to set up standards for your product.

Simplicity –
Your digital transformation strategy may force you to face tradeoffs like temporary disruptions in business processes, higher costs due to investments in technology and a greater number of employees, and impacted business continuity. Therefore, it's essential to devise a proper plan to avoid or minimize these effects.

Mileposts and deadlines –
If you can build your app with a minimalistic architecture, choose the simplest option possible. While it's essential to consider the possible scaling options, there's no need to overcomplicate things in advance.

Self-maintenance –
The application architecture should be able to identify issues and repair them on its own.

Automation –
Try to automate as much development, testing, and deployment as possible. This will be useful when you decide to scale your app!

Convenient and error-free data management –
Consider your data storing and processing practices and pick the easiest ways to manage databases while avoiding unnecessary costs.

Application architecture is the foundation of all web application development. Whatever application architecture you choose determines all of the following logic for developing the application, the interaction between its elements, and the functionality. Therefore, it is critical to identify the peculiarities of each architecture type and select the right one for you prior to developing your application.

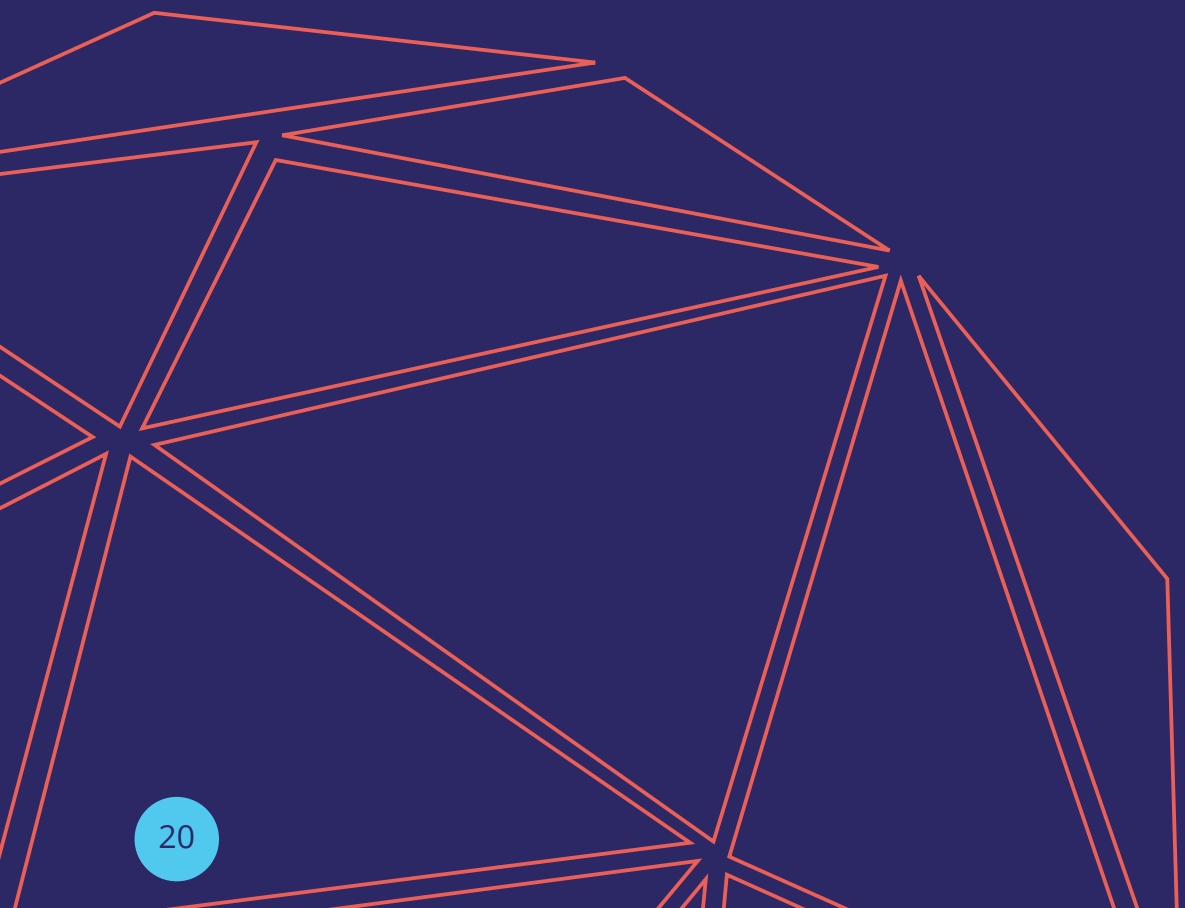
Ch. 3

Key Factors to Consider When Hiring Contract Engineers

When you are developing enterprise applications, there are numerous factors to think about, such as time, features, testing, user experience, and functionality. Each aspect of an application needs to work correctly.

To produce the most efficient and powerful software, it's important to hire experienced and competent software engineers. But going through the hiring process can be difficult. In addition to sorting through a large pool of candidates, a dilemma that every employer faces is whether they should hire a contractor or a full-time engineer.

Let's look at some factors one should consider before hiring contract engineers.



Job Experience

When you have to choose between two candidates, one with experience and one without, it is obviously prudent to opt for the former, especially if you are on a shoestring budget and have no time to train new employees. Experience can be a differentiating factor to consider when you are hiring contract engineers. If the candidates have a proven track record in similar jobs, they'll probably be able to replicate that success at your company, too.
But experience is not the sole criterion. While you make sure to consider experience, don't prioritize it over all the other factors.

Hard Skills

Hard skills reflect the skills acquired by the applicant either at their educational institution or at past jobs. They are measurable and easy-to-define skills that will enable you to judge the applicant. It's hard to overlook them since they are indispensable to the job that they are being hired for.In the absence of the right skills, special training has to be arranged for the candidate. For instance, you may need to teach the applicant a specific software language, framework, or tool, which is usually not worth the investment with a short-term contractor.

Soft Skills

Soft skills reflect the personality of the candidate. Though hard to measure, they are often reflective of the applicant's behavior. Soft skills are less critical when hiring a contract employee than when hiring a full-time employee. However, they become more important depending on the urgency of the development project. An employee with poor soft skills will have a much more negative effect on the team's productivity during crunch time than they will have on a build with a longer timeline.

Potential

An experienced interviewer often spots candidates who may not have adequate experience, but possess good educational credibility and potential. For such promising candidates, sometimes it pays to take a chance by hiring them as they might grow into top performers.

What Are the Advantages and Disadvantages Of Contract vs. In-House Developers?

As with most things, there are pros and cons associated with each choice. Depending on the needs of your company and expectations from a new employee, their role will shift. Factors like autonomy, company loyalty, and working hours differ between contractors and full-time employees. Nevertheless, the biggest difference between the two is that while a full-time employee can be managed more closely, a contractor exists as a separate and autonomous entity.

Advantages Of Hiring Contract Engineers

Perform or perish –
Contractors are generally more active and quicker, as they are aware that once they stop delivering value, they are gone.

More productive –
Contract employees often telecommute and are exempted from office meetings and other such time-consuming events. Furthermore, these employees are seldom affected by interoffice politics or corporate red tape. This amounts to higher productivity.

Hire and fire –
Contract employees are employed temporarily to execute a task. Once the job is completed, the contract comes to an end. Thus, the organization has no liability toward the contract employee and is not obligated to provide health insurance, paid vacation, sick leave, or any other benefits.

Less expensive in the long run –
If your organization is looking for assistance on a short-term project, hiring a contractor will turn out cheaper. Similarly, it pays to hire contractors in the case of small businesses that need some boosting.

Diverse portfolio –
On average, a contract employee works on multiple projects, is adapted to many more teams and organizational cultures and works for a set of stakeholders that run the gamut.

Technical knowledge –
Contractors are likely to have a much broader base of expertise and experience than average in-house developers.

Faster execution of work –
Due to their autonomous state, contractors independently plan their schedules at their own pace, which makes their work proceed more smoothly.

Low investment –
If the company is assigning a task to a contract engineer, then the company doesn't have to invest in equipment, training, annual raises, or other supplemental costs.

Disadvantages Of Hiring Contract Engineers

More expensive per hour –
As contractors have to handle their taxes, health insurance, benefits, and other marketing costs, one must be prepared to shell out more money to hire contractors, as compared to regular employees, over the long run.

Lack of supervision –
Due to the independent working methodology of contractors, the employer might find it difficult to supervise the project and make amendments. They have to wait till the project is submitted and then make changes if any.

Lack of loyalty –
As the contract employee doesn't need to be a "culture fit" for an organization, no loyalty should be expected from such an employee.

No flexibility –
Due to the temporary nature of work, there is no flexibility in working hours or in assigning any work.

Ch. 3 - *Key Factors To Consider When Hiring Contract Engineers*

Advantages Of Hiring In-House Employees

Scope of flexibility –
Having a full-time employee provides you more freedom to plan. There is a scope of flexibility in allocating and assigning the work.

Loyalty –
Over time, some full-time employees might develop a sense of responsibility and belonging. They can turn out to be great assets to an organization and develop a sense of loyalty.

More adaptive –
Being permanent employees, they are aware of the needs of the company and can adapt to them faster than a contractor.
Culture Fit: Full-time employees can be tailored to a specific working style to suit the needs of an organization.

Disadvantages Of In-House Employees

Lengthy onboarding process –
Hiring a full-time employee is a tedious and tough process. Finding the right person for a job is as tough as it is for that person to find the right job. There can be a lot of delay in executing a project when there is a vacancy to be filled for a given position.

More investment –
The advantage of having in-house developers comes at a prohibitive cost. Having in-house developers means buying equipment, training, and maintaining employees. The employer is also saddled with the responsibility of bearing the expenses of employee insurance, health care, annual raises, and other incidental costs.

Lack of exposure –
Having been confined to the routine of office work, an in-house employee may lack the vision to execute a task dynamically. Due to limited exposure, in-house employees may also not be acquainted with trending technologies.

Work at ease –
As there is a flexibility of working hours, manner of work, and interdependence between co-workers, there is often delay in executing projects.

What Types Of Projects Are Geared Towards Contractors?

Many organizations set up a huge infrastructure and carry out all IT services in-house. They can have much higher research, development, and implementation time, and all of this increases the operational cost. But if the same task is assigned to contractors of a quality IT services organization (such as Crowdbotics), the project could start immediately. Handling the same project in-house might involve weeks or months to hire the right people, train them, and provide the support they need, thereby escalating the cost and duration.

Another advantage in handling projects off to contractors is that you stay focused on your core business without getting distracted by complex IT decisions that can be managed by experienced contractors. Most small and upcoming companies cannot afford to match the in-house support services enjoyed by big companies. By assigning some tasks or entire projects to contract engineers, small companies get an opportunity to act “big” by getting access to technology and expertise similar to larger competitors.

One should examine carefully all the pros and cons before assigning a project to contractors or in-house developers, and make sure that the benefits outweigh the risks.

Advantages of Hiring Crowdbotics for Managed App Development

As there are always pros and cons to each choice, there isn’t always an easy solution. For this reason, it’s prudent to consider choosing a path that integrates aspects from both contract and full-time employment. Companies like Crowdbotics provide contracted, high-quality skilled engineers with proven track records. Crowdbotics focuses on hiring remote resources that operate within the client’s time zone, possess verified language skills, and are compatible with the client’s work culture. Crowdbotics utilizes a flexible, single contract that enables clients to build virtual teams that adapt to every need in the project life cycle.

Ch. 4

What Every Non-Technical Founder Should Know About DevOps

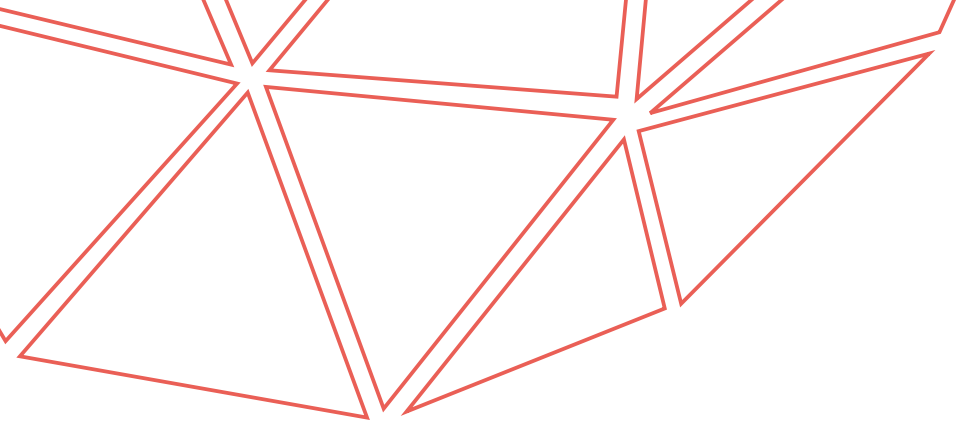
A non-technical founder can run a successful IT or software development company without much programming knowledge or in-depth IT expertise. However, a founder still needs to understand the important jargon and details so that they can keep track of what their more technical employees are building.

In this chapter, we will help non-technical entrepreneurs by detailing what DevOps at a startup entails and how the different elements of DevOps combine in a software-based business product. Additionally, we will cover other tech knowledge that non-technical founders need to understand to run their business more efficiently.

Overview of DevOps at a Startup

DevOps is the convergence of development processes with operations. The advantage is that there's no need for separate teams to communicate back and forth if a product requires changes. In a DevOps team, members with different strengths and skillsets collaborate to build a product.

DevOps also facilitates automation, as many tasks in a startup require repetition. These jobs include data collection, approvals, updates, and more. The DevOps team can use computer algorithms to automate redundant processes to eliminate human mistakes and enable the IT team to focus on more important work. DevOps needs are different for each company, and every firm needs to figure out what they require.



Advantages of DevOps for a Startup

Startups can gain the following benefits from DevOps:

- Fewer unexpected problems – Proper DevOps implementation can reduce problems and improve workflow.
- Better understanding – You can better comprehend the processes and tasks required to create the software product.
- Consumer knowledge – DevOps can help you gain information about the needs of your customers so that you can produce relevant solutions for them.
- Improved customer satisfaction – By meeting user expectations, you can boost customer satisfaction and loyalty.
- Enhanced team agility – Your development and operations teams can function in an agile manner using the newest and most effective methodologies.
- Better communication – DevOps can improve communication and collaboration between several teams.
- Happier employees – Your staff can work efficiently to improve productivity and gain recognition for their performance.
- Automation – You can automate routine and repetitive tasks to save time and effort.

How the Different Components of DevOps Tie Together in a Software-Based Business Product

DevOps has three main elements, which are people, technology, and process. People involve employees with the right soft skills and hard skills for efficient DevOps implementation. Technology allows you to automate everything so your team can function in a hands-off manner. Process is the use of relevant and effective methods for your business and employees.

There's no need to build all your applications manually or to repeat simple tasks every time. Automate routine tasks in the marketing, customer support, financial, and administrative departments to ensure consistency in all processes. Development, production, and staging require a consistent system in which all stakeholders can work efficiently.

Developers understand the needs of operations teams and provide them assistance. They can enable rapid and continuous deployments to facilitate the speedy rollout of features and fixes. Operations and IT developers can focus on collaborating effectively using physical environments and tools such as Heroku, Slack, Docker, Jenkins, Nagios, and others.

You can also develop and use your internal apps intelligently. The DevOps emphasis on tooling can help to maintain code quality and create a better software product while reducing the time and effort required to fix bugs later.

DevOps enables the identification and correction of glitches before the release of your software product or mobile app. This will almost always lead to higher user satisfaction scores. Continuous delivery can help your developers release customer-facing apps more rapidly.

Other Tech Knowledge that Non-Technical Founders Should Understand

Though you don't need coding knowledge to run a technology startup, you still need to be aware of basic, non-DevOps tech jargon and language to understand what's going on. Here's a more general list of IT and software terminology every non-technical founder should know:

The Cloud –
The cloud is a set of multiple servers linked to the web that are used to store information and data. Cloud computing offers the ability to access and use data on any internet-ready device, anytime or anywhere.

Database –
You can use a database to store all your app's information. A SQL database functions as a sequence of columns and rows, much like Excel files. Each database table is like a single Excel sheet. On the other hand, a NoSQL database like MongoDB can be compared to a Word file. This database type is more challenging to manage.

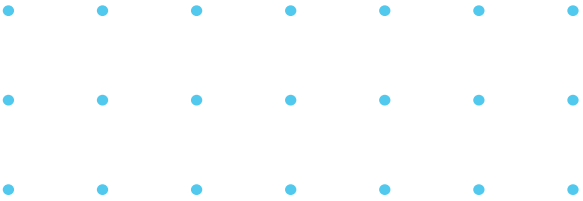
Servers –
Servers are linked to the web and serve data or files. Whenever you visit a website, you request files from an internet server. Cloud applications like Amazon Web Services (AWS), Microsoft Azure, and Google Cloud have made it easier and cheaper to set up servers remotely. Previously, IT professionals had to physically set up and install servers in data centers.

Frontend/Backend –
Each app has a frontend and backend. The backend accepts user entries, stores them, and gets data from the database. The frontend is the interface with which the end-user interacts. Frontend development involves design techniques, while backend development is focused on performance, algorithms, and data.

UI and UX Design –
UI is short for user interface. UI design dictates how your app looks. Thus, the focus is on animations, colors, style, typography, and other elements. UI design helps to make your software product look engaging and beautiful. UX is short for user experience. UX design concentrates on the ease of use of a product or app. It helps answer questions like does the app layout information in a sensible manner? Does it expectedly react to user input? Does it help or hinder them in solving their problem? UX design also involves automatic emails, customer support, copywriting, and more.

Web App –
A web application is hosted on the internet and not downloadable on your computer or mobile device.

Native App –
A native mobile app is created in the device's native coding language. For iOS, this is Swift or Objective-C. For Android, it is Java or Kotlin. This specialization helps native apps perform better. Each native system has common elements and different paradigms. Developers should look to customize their app to the platform it is run on, to produce a cohesive user experience.



Hybrid Application –
A hybrid mobile app is created using web tools and is later compiled into two apps. There is a single code base for hybrid apps, which is an advantage because it makes it easier and faster to develop and maintain them. However, hybrid development can occasionally compromise quality and performance.

Open Source Code –
A big advantage of software development is knowledge sharing. Many developers share the source code of the new product or app that they develop. This accelerates learning and strengthens the whole industry, which is why Crowdbotics leverages open source technology in its app builder platform. If your company is hawking a technical product, you can use open-source code for recruiting or marketing.

Programming Languages –
Writing code can be compared to preparing a cake recipe. The computer will follow each instruction provided by you. But a single mistake can mess things up. Coding languages delineate the rules about how you create your recipe. Similar to spoken languages, each programming language has its syntax. Further, each coding language is built around a different philosophy and offers different benefits. Some apps or devices only support specific programming languages.

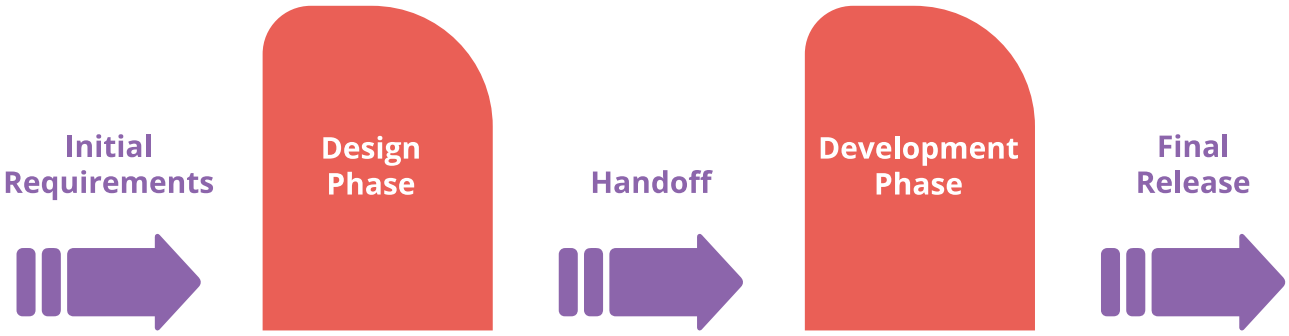
Most coding languages have the same fundamentals, though the syntax may be different. Competent developers are well-versed in a few languages and can easily learn new ones. This means you should recruit for overall ability and not a specific language.

Ch. 5

How Teams Can Get to Market 4x Faster with Automated Design-to-Development

Transforming brilliant ideas and sketches into fully functional products can cause a lot of trouble, confusion, and hurdles down the road, especially when working on digital products that require both design and development. Converting design files to code is an early phase in the development lifecycle, and difficulties at this stage can make it hard for teams to get their products and features to market on time.

The point at which a designer has finalized (not finished–more on this distinction below) an app's UI design and is ready to hand it over to developers for further engineering and implementation is known as the "design-to-development handoff." This concept can be illustrated as follows:



This handoff is crucial because there is no such thing as a "finished" design. There is always more to add before the final release, with ongoing iterations based on requirements from stakeholders and clients. To avoid wasting time while exchanging specs between your design and development teams, it is vital to get all of your product teams on the same page.

Who Is Involved in the Design-to-Development Handoff?

A good designer-developer collaboration is important to make this handoff successful and bring the product to life. To ensure accuracy in production, the following teams must be kept in the loop to discuss design specifics, and communication must be established to make the handoff go smoothly.

Designers

During the designing process, it is effective for the designer's team to involve the engineering team in the design process at a high level to avoid extra feedback loops later. Designers should seek insights into the implementation process to pinpoint likely points of confusion that may emerge when translating design elements to code.

Developers

To avoid an unwelcome clash in the future, developers must get involved early with the designer team. This can be done by running initial design prototypes to identify bugs in code, which can be removed by making suitable adjustments in design sets.

Product managers

This team should encourage collaboration from both sides (designers and developers). A proper handoff meeting makes it possible to hash out questions from both teams regarding implementation, QA, and launch period. PMs should act as a source of truth that both designers and developers can consult about the current status of an application's design process.

Other concerned persons, such as stakeholders or clients, can be included in this communication loop to ensure that they have an insight to how their requirements are going to transform into the final product.

Causes of Friction in the Design-to-Development Handoff

The design-to-development handoff sounds easier than it actually is. There are predictable logjams that can emerge if the two teams do not synchronize their efforts from time to time. “Design is not just what it looks like and feels like. Design is how it works.” - Steve Jobs

Some potential causes of friction that can complicate the handoff process include:

- A lack of designer-developer communication
- Differing skill sets between designers and developers
- A lack of clarity from clients or PMs
- Inconsistent naming conventions on either the design or development side
- Vague holes or missing details in the designer’s handoff checklist

Each of these holdups will affect the overall development cycle of a product and can be the most nerve-wracking part for some teams. If mishandled at an early stage, the design-to-development handoff can come back to haunt you at the execution and launch phase.

Some specific risks of a failed design-to-development handoff include:

- Slower overall development due to different interpretations of the product
- Needless iterative feedback loops between designer and developer
- Adverse user experience due to missing or overlooked details
- Time-consuming bugs arising in QA, at an advanced stage of development when deadlines are approaching

Automating the Design-to-Development Handoff

Companies need to adopt some carefully chosen ways to alleviate the above-mentioned challenges during handoff.

The optimal way to structure this handoff is to incorporate tech tools and processes that can automate key steps, like providing interactive prototypes, gathering teams and assets on one board, and maintaining documentation to avoid misunderstandings between designer and developer. Let's take a deeper look at some ways to automate your design-to-development handoff:

Task management and organization

Getting all teams on one project board from the very beginning is essential because it automates the overall workflow for design and development. Asana and Jira are popular task management tools for grouping all employees in a single communication loop.These tools enable everyone to see who is working on which task, and it provides total visibility into the content and objectives of each task. They automatically create a project board with timelines, task prioritization, and team-level tracking. They also act as a home for all necessary assets of the project.

Cloud tools to streamline communication

It is sometimes physically impossible to get all team members to work in the same place and be aware of each other's progress. One best practice here is to use cloud-based tools to streamline product management.

Mockplus provides the functionality to edit and share PRDs online, offering faster design and better collaboration features. It displays design files and documents from a bird's-eye view for each product and allows one-click downloads. It also has the ability to accelerate a design-to-development handoff by generating historical versions of source files, which eliminates traditional ways of archiving files and data

Google Drive and **Dropbox** can also be adopted to make assets accessible to each team on the cloud, although they also present a risk of over-documentation if not properly organized.

Providing interactive prototypes

Rendering a design prototype is important for clients, as it visualizes the basic workflow, interactions, and insights of the final product for them. It is equally beneficial to provide developers with action-oriented prototypes of design assets.

For example, while building a website, tools like Marvel and InVision provide the ability for designers to convert mockups into functional prototypes that define the internal links between pages of the website. These clearly defined interactions eliminate complexity for developers while transitioning the design files into code. The automatically generated prototypes can then be stored directly in code databases.

Design system

Providing developers with a style guide in the form of a design system helps to improve the designer-developer collaboration. Although the ideation of a design system cannot be automated, the creation of design templates can enable developers to quickly grab assets from a central repository. This practice promotes both procedural and design consistency, which are the keys to successful collaboration.

Advantages and Aftereffects of Automation

As competition increases in the software development industry, companies will need to speed up development to get new features into production faster. Intelligent automation can simplify development tasks that typically take hours so that they take seconds.

Cost savings via automation

Development work is expensive, so any reduction in development costs can be a significant cost-saver for an organization. Design-to-development automation is a high-value (yet underutilized) way to dramatically cut the total dev hours required.

Imagine, for example, that your product team pays for seats on a design platform such as Figma, which typically costs something like $50-$100 per seat. Without design-to-development automation, the productivity gained by using Figma is defined exclusively by how much faster your team works due to Figma's features.

However, if you can automate the conversion of design files into code, you have now amplified the value of each Figma seat, since the productivity gained now includes both how much faster your team works plus the development hours eliminated plus the value added by the fully functional applications they've created directly from designs.

These productivity gains have massive effects on development costs. Hypothetically, automated design-to-development could cut down development time by up to 80% for a given build. Most full-time developers bill their time at around $100-150+ per hour. So, at $125/hour, on a 500-hour build, you could cut costs from $62,500 to $12,500.

Now, extrapolate that for a team of 10 (or 100) devs working full time. The savings are compelling.

Operational advantages of automation

You can also realize benefits related to procedural improvements. The highest-impact forms of operational automation for the design-to-development stage are outlined below:

Handoff meeting –
Task organization using task management software eliminates the time required to compile information from multiple disparate sources. Preliminary automation is beneficial in creating, sequencing, and assigning all product-related tasks from beginning to final release.

Subtasks –
Subtasks allow teams to tick off every essential design and development step directly within task management software, rather than by preparing a traditional checklist.

Releases –
Faster development is a breeze if you use an agile management approach to automatically push releases and updates into production.

Workflow and AI automation –
Automatic code checks and script diffing in a single board can save a team a great deal of time. It makes it easy to track bug origins, which in turn frees up more time to test different deployment techniques.

Communication –
Streamlined communication through a dedicated channel allows teams to detect errors earlier in the development process. Early detection and feedback empower designers to restructure their designs for easier implementation by developers.

CI/CD pipeline –
Continuous integration and continuous delivery tools allow each developer to merge code changes into a central repository. This technique enables faster iteration, which is a must when implementing design change requests.

Power Digital Marketing Case Study

A case study from **Power Digital Marketing** shows that they have automated their design-to-development process since the start of 2018 and found it to be fruitful.

They used Asana for managing tasks and organized their workflow through it. Its features like subtasks and project boards, which they did not use beforehand, have streamlined their development process.

They also conducted a search for a familiar design platform or a single design system where both designers and developers could access current data. They tried using Sketch, Photoshop, and InVision for integrating designs, but this approach did not work out for them to get things properly labeled and in one place.

Eventually, they switched to **Adobe XD**, which became their go-to design platform. Adobe XD provides the option to publish design specs from a design file, so that developers can consult those specs when implementing designs.

They have also used **Zeplin** for their “handoffs” and for design specifications. Zeplin generates the required CSS for the design specs of Adobe XD. Adobe supports a direct integration between Zeplin and XD, which dramatically reduces the necessary coding time for all websites. Interactive prototypes have also done wonders for their team, as prototypes show the overall navigation for each product and make it easier for developers as they begin implementing and mapping the product. **Marvel** and **InVision** have done wonders in converting mockups to live prototypes.

All of these improvements made their release cycles more efficient than before. Although specific requirements vary from project to project, technological alignment and streamlined management are all you need to get a product to market faster. This is how intelligent design-to-development automation can get teams into the market 4x faster.

Wrapping Up

There is no magical way to avoid all handoff errors, but adopting the above approaches can help your team grow faster and build better products. Too few product teams are focusing on automated design-to-development, but the gains that you can realize in this stage are significant.

At Crowdbotics, we are tackling this inefficiency head-on with our upcoming Figma integration for the **Crowdbotics App Builder**. Soon, users will be able to automatically import Figma designs into the Crowdbotics platform and see those designs rendered as fully functional apps inside the Crowdbotics App Builder.

From there, users can configure interactions, backend models, and third-party APIs with code-free tooling or directly edit the source code for highly custom builds. Deployment is fully automated, meaning that a user can go from design to delivery with zero coding required in a fraction of the time required by a traditional development cycle.

For users with designs who would like us to turn those designs into working apps quickly, we also offer managed app development services. This is an excellent option for teams looking to get to market as soon as possible. If you have designs but don't know how to code, we recommend **getting in touch** with one of our expert PMs today.

Ch. 6

Build Your App with Crowdbotics

It can be intimidating to develop software without the experience or know-how it takes to actually build a product. At Crowdbotics, we're ready to help you close that gap and turn your idea into a market-ready software application.

Managed app development with Crowdbotics isn't a typical outsourced development experience. It's a hands-on collaboration between you and our dedicated team of experts to plan, design, build, and release your app at the highest possible quality and on your preferred timeline.

Here's what to expect:

- Our sales team will reach out to schedule a brief scoping call to understand exactly what you're building.
- Once we clearly understand your build plan, we'll send you a detailed proposal explaining the full scope of work to be delivered.
- Once you approve our proposal, we will assign a product manager and team of developers who will work with you every step of the way.
- You'll enjoy daily and weekly check-ins until your app is live, and ongoing support and iteration from there.

What's Different About Building With Crowdbotics?

Our process leverages a simple truth about software development: most new applications look like applications that have been built before.

No matter how unique your idea is, there are fundamental best practices and code packages that are used by developers across the globe to build leading applications. Crowdbotics rapidly reduces your time-to-market and ensures that your app adheres to modern code standards by providing access to a vast library of common code modules.

This enables us to assemble the basic structure of your application instantaneously, leaving more time for customization and fine-tuning.

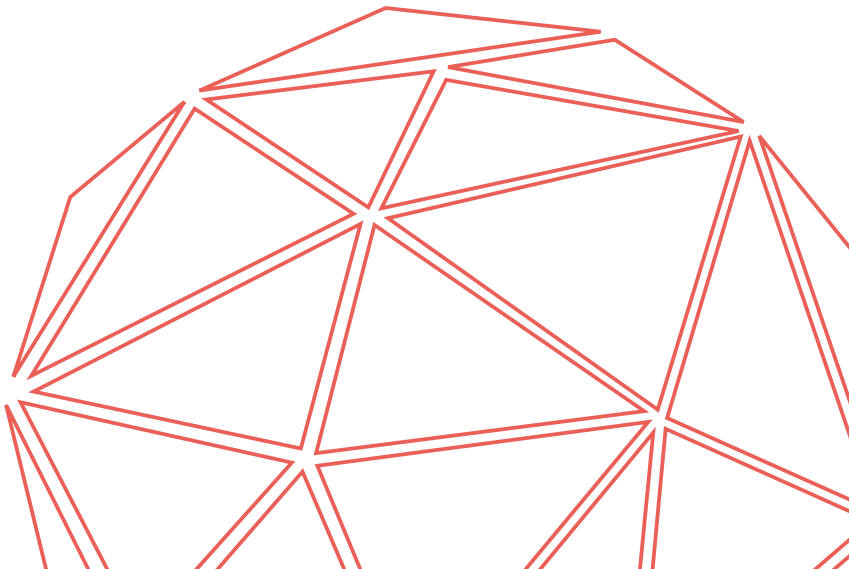
What Do People Build With Crowdbotics?

Professional teams choose Crowdbotics for simple SaaS integrations, enterprise-scale trading platforms, and everything in between.

Our core stack is React Native + Django, but our expertise extends to all modern (and some not-so-modern) frameworks and languages.

Whatever your software development needs are, our experts are standing by to deliver your build on time and as specified. Here are a few industries that we're especially experienced in:

- Health & Wellness
- Finance & Investment
- Analytics & Data Visualization
- Startups & Technology
- Non-profit & B2G



What are the Steps in A Typical Application Build?

Whether you work with Crowdbotics or a different third-party development partner, the overall best practices for building an app will be consistent for any team:

Outline requirements –
Create a detailed build plan for each feature of your app in collaboration with the development team.

Wireframing and basic UX –
The team prepares basic “placeholder” screens for each unique screen in your app and walks through the intended user flows with you.

Technical assessment and architecture –
The team determines how to structure your app’s underlying components and designs the most optimal system for data storage and retrieval.

Solidify design –
The team replaces the basic wireframes with final design elements and requests your approval.

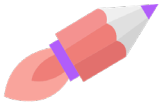
Front end development –
The team builds the user-facing components of your application.

Back end development –
The team builds the underlying components that power the user’s interactions with the front end.

Testing –
The team works with you to run through the intended user flows of your app and fix any bugs or undesirable behaviors.

Deploy –
The team launches your app to your chosen platform(s).

Iterate and scale –
As your app attracts new users, you work with the team to add features and improve performance.



crowdbotics

Ready to Get Started?

Get business-ready applications and features into production faster. Contact us today to tell us what you need.

Contact Us Today!