Digital Transformation Roadmap for

Flagship Mobile Application

On behalf of:

The U.S. Department of Veterans Affairs

Submitted by:

580 STRATEGIES
GOVERNMENT. TECHNOLOGY. SOLUTIONS.

September 19, 2018

TABLE OF CONTENTS

580 Strategies LLC
1100 7th Street NE
Unit #3
Washington, DC 20002
(415) 915-4679

September 19, 2018

U.S. Department of Veterans Affairs
810 Vermont Ave NW
Washington, DC 20571

Dear Messrs. Quinones and Melton:

I am pleased to submit the attached deliverables as a starting point to putting modern mobile technology in the hands of veterans who utilize VA services. 580 Strategies has brought our unique combination of familiarity with government constraints and extensive experience in establishing and delivering digital products, including mobile applications, to bear on a technology framework and product roadmap that will help the VA get the job done faster and with greater veteran satisfaction.

Over the three-week duration of this project, we have analyzed the marketplace of mobile application development options and put forth a recommendation of *React Native* as a cross-platform technology stack that will balance technical overhead and serve veterans well through a consumer-oriented app for years to come. We have also interviewed veterans and logged their direct feedback on existing features of *Vets.gov* and what they would like to see included in the mobile app. We used those data points to assemble an initial product backlog ranked by notional risk versus reward factors. Finally, we assembled this work into a 12-month product roadmap of sprints and releases with a clear delineation for what we feel is an achievable minimum viable product (MVP) that would delight veterans.

These deliverables will give VA a rapid-startup approach if your stakeholders choose to fund and move forward with development of the app. I am confident we have put our best offering forward in crafting for VA a roadmap which will meet the demands of agency stakeholders and ultimately deliver a robust mobile application that will increase access to Department services and improve outcomes for veterans.

Thank you for the opportunity to execute the project, and I look forward to continuing to support the agency's mission as its technology portfolios evolve!

Very Respectfully,

Rusty D. Pickens
Founder and Principal

EXECUTIVE SUMMARY

The 580 Strategies team was tasked under a micropurchase agreement by the U.S. Department of Veterans Affairs (VA) with envisioning and recommending a starting point for a best-practice implementation approach and product roadmap to support the delivery of a robust mobile application (app) for popular services on Vets.gov using both a technology and development framework that can be extended across the agency to other online properties such as VA.gov.

As part of the three-week engagement, we assembled a technical team steeped in mobile application development using agile frameworks, a skillful design team that has supported world-class user experience products, and placed the voice of the veteran front and center in our planning.

The output of this effort has produced three major deliverables:

1. A recommended technology stack, *React Native,* upon which to build the app and an agile framework methodology pioneered by 580 Strategies that will empower engineers to work collaboratively with the VA to ship working software quickly
2. A starting product backlog of user stories taken directly from collaboration with veterans who utilize services from Vets.gov with a notional sizing, ranking, and priority order to kickstart the project
3. A product roadmap that charts major features across a sample calendar and agile iterations to denote major milestones, including the Minimum Viable Product (MVP) for the first year

We utilized a rapid startup approach; 580 Strategies partnered with the VA U.S. Digital Service team and conducted veteran interviews to define and jumpstart the requirements, people, and processes needed to support the development of this new product offering. We then analyzed Vets.gov and the technical marketplace to derive product requirements, technology stack options, development framework approaches, and desired features. Finally, we bound all of this together in a product roadmap that should inform stakeholders and delight veterans as the VA implements new technology tools to reach their customers wherever they are online.

Utilizing the *React Native* cross-platform technology stack, VA will be able to focus efforts with a single development team working on a single application that can then be translated to many platforms such as iOS and Android. *React Native* supports reuse of code while still retaining the ability to use native functions of the mobile devices such as the GPS, camera, Touch/Face ID, and document storage capacity. This will allow the developers to put extended features in the hands of veterans that complement and enhance the Vets.gov experience.

Here are just a few highlight use-cases which will be possible with the mobile app:

- Veterans can directly call, text, or chat with the Crisis Line from their device
- Veterans can locate VA facilities near them using the GPS features of the device and contact or navigate to them with ease
- Veterans can create and modify appointments, claims, prescriptions, and benefits requests while on or offline, without data loss or unnecessary app refreshes

- Veterans can sync their appointment details to the calendar on their device
- Veterans can use the camera and documents features of the device to submit evidence for claims justification
- Veterans can message and respond to their provider team from within the app
- Veterans can use Siri or similar voice-assistant features to navigate the app hands-free

The project deliverables provide an ample starting point for the Department that can be refined and developed into greater detail. Our intent is to provide, first and foremost, an actionable framework and development approach VA can use to execute this work but also as much detailed information we could uncover for features and services specific to the Department. We are confident this micropurchase project has set VA up for success with a scalable framework and enough starting information should you choose to move forward with the project and begin development of the mobile app.

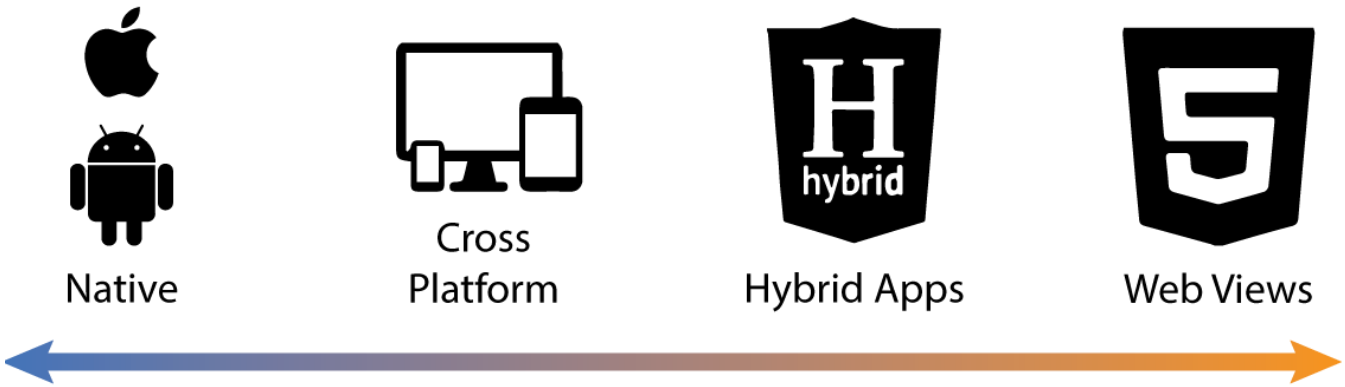
TECHNOLOGY AND DEVELOPMENT FRAMEWORK

The 580 Strategies team recommends VA implement the following technology stack and proven agile framework as the underlying technology for application development but also to quickly deliver working software within the constraints and challenges associated with a government software initiative.

Mobile Technology Application Stack

In the recent past, the mobile app development environment has been largely narrow and barren in terms of the enabling technology that goes into designing, developing and deploying mobile applications. Apps were largely built in Objective C and Java. But, fast forward to the present state of the industry, and you'll see there are a variety of approaches VA can adopt for developing a mobile application. The ecosystem has grown by leaps and bounds, and with an ever growing community, the tools have also increased as well.

Today, mobile app development approaches vary widely on the following scale: fully native apps written in native programming languages that are platform or device-specific, yet only function on the chosen platform reside on one end of the spectrum while the wholly generic "Web View" approach that simply calls web pages from within an app wrapper yet offer no device-specific features resides on the other. In the middle, varying degrees of cross-platform and hybrid approaches offer combinations of strengths and weaknesses from both extremes.



Technology Approach	Pros	Cons
Native	• Fastest performance • Natural look-and-feel • Easy, direct access to all device features	• Apps are specific to a single platform/device • No realistic code reuse • Each update/patch/release must be maintained separately • Programming languages are highly specialized

Cross Platform	• Close-to-native look and feel • Can maintain good performance with the right toolkit • Uses standard libraries to access native features • Simplifies operations by managing a single app • Maximizes code reuse and minimizes platform-specific code • Utilizes a variety of mainstream coding languages • Supports robust continuous integration / continuous development and high-quality control processes	• Heavily specific to the chosen toolkit • Can be enhanced or hindered by third-party plugins • Can encounter performance issues given certain toolkits • May present compatibility issues with the wrong toolkit
Hybrid	• Utilizes standard web coding approaches • Faster and cheaper development • Commodity programming languages enable non-mobile development teams to produce apps	• Abstraction away from the device layer limits access to native features • No offline access options (everything is web based) • No guarantees of rigid performance • Could be rejected from the App Store
Web Views	• Requires minimal app startup efforts • Simple web pages wrapped in a mobile app	• No access to native features or functions • Little control over how information is rendered; no native look and feel • No offline capability (everything is web based) • Likely to encounter frustrating page refreshes and data loss if connections or devices aren't stable

We will examine the tradeoffs VA should consider for each type of approach below.

Native App Development

"Native" apps are platform-specific (to the manufacturer and hardware devices) apps which are developed in a platform-specific programming language and development environment. Native apps require complete, direct access to all the hardware and features/functionality of a specific device but won't work on anything else.

Each native platform provides app developers with a standardized "Software Development Kit" (SDK); i.e. iOS SDK and Android SDK, containing a set of tools, code samples, libraries, documentation and guides that allow developers to create apps on a particular platform but work only on devices supported for that platform. For example, an iOS app developed using the iOS SDK will only function on

devices that support iOS. This is a major decision point when considering whether or not to develop in exclusively native SDKs.

Pros

- Given native apps are native to the platform, they work faster and offer utmost performance
- Native apps offer offline capabilities and work even without internet connectivity
- Native look and feel allow the user to get acquainted with the app quickly and understand the natural flow of the subsequent features
- Access to all functionalities of a smartphone like GPS, cameras, multiple sensors, offline storage are provided by native libraries for easy use
- Maintaining screen resolution and aspect ratios across native devices is simple. Native apps have the best control over the orientation, size, and resolution of the app.

Cons

- Apps are only supported by one platform. Developers have to spend time on developing applications for each platform entirely separately with no shared code
- Updates, patches, and compliance processes must be completed separately for each app
- Project management overhead and technical debt are expanded dramatically if multiple-platform support is desired
- Development is expensive compared to other alternatives due to the overhead associated with maintaining multiple apps
- Applications are relatively bloated compared to other alternatives as static content must be coded individually, adding to app size and development overhead

Sample Use Cases and Considerations with Native Apps

For VA use cases, native capabilities will allow use of the Camera and Gallery to capture and upload documents. Veterans could fill out forms offline and save progress as well. They can later submit form data when back online. Moreover, the native look and feel will acquaint Veterans with form filling behavior quickly. The forms will not disperse, nor will they pixelate on devices with higher resolutions. However, for every screen showing static content, a separate piece of code must be written. This will increase application size as well as the level of effort for development.

Cross Platform App Development

A cross-platform app development environment allows a developer to create one source code for an app, which the platform then converts into separate codebases for the native environment. Developers can reuse code that's already been written while letting the platform handle the compiling, rather than having to write separate source codes as with a from-scratch native app.

There are many cross-platform SDKs and frameworks that can be used to develop an app, but we've focused our options to ones that are widely accepted by the developer communities.

Xamarin

Xamarin is a tool used for cross-platform app development that allows engineers to share around ~90 percent of code across major platforms. Being a comparatively new tool, it is based on the Microsoft technology stack and has an active community of over 1.4 million developers. Xamarin uses a single language, C#, to create apps for all mobile platforms.

Xamarin shares 90% of app code while splitting the remainder across platform-specific stacks for integration across multiple devices

Pros

- Xamarin does not require switching between development environments: developers can build Xamarin apps in Visual Studio for ease of coding
- Xamarin uses C#, which is a mature language with strong safety-typing that prevents code from unexpected behavior
- As C# is one of the .NET framework languages, it can be used with a number of useful .NET features such as Lambdas, LINQ, and Asynchronous programming, which makes Xamarin an adroit solution option

Cons

- With Xamarin, developers only share the logic; User Interface (UI) code will be mostly platform-specific. This makes building complex form structures, rich custom UI, or exquisite animations in Xamarin more difficult without access to the freedom and fluidity Javascript offers
- Xamarin is geared for maximizing shared functionalities across platforms and a wide range of APIs but which lends it to suboptimal performance
- Xamarin requires an arduous “binding” process when developers need access to third-party libraries that don’t already exist

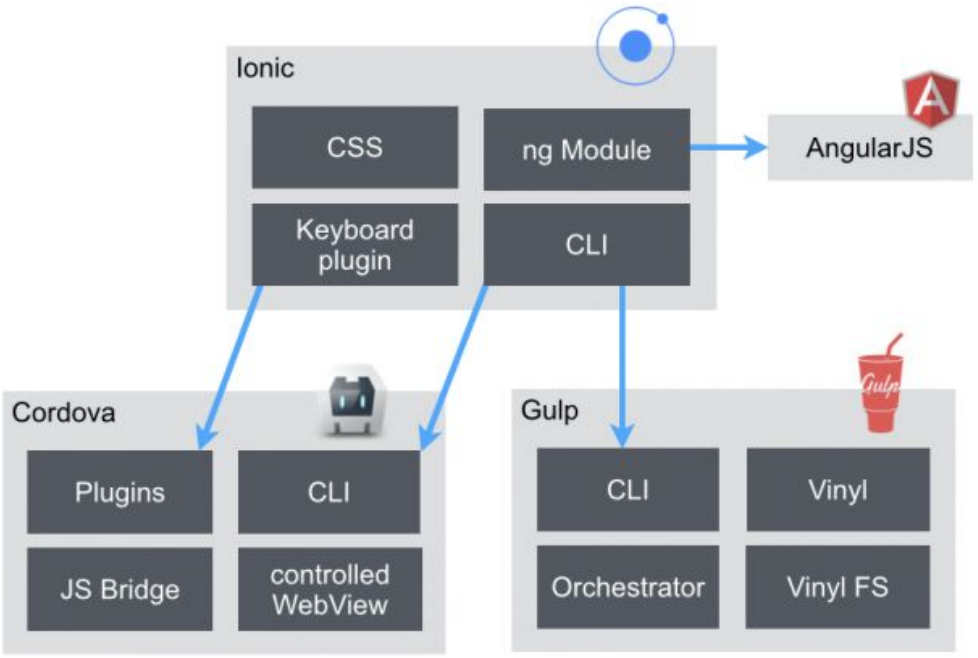
Sample Use Cases and Considerations with Xamarin Apps

Xamarin is helpful for providing a closer-to-native look and feel to the app that will allow for easier adoption and familiarization with the app UI/UX. But it is likely that VA will rely upon multiple APIs for service delivery and it is critical that the app doesn’t lag in carrying and relaying information between APIs.

While a Xamarin app may look good, imagine a veteran wanting to schedule an appointment scheduled at a health facility, but instead must wait for a discouraging amount of time while appointment call is made through the API. With Xamarin, chances are that relaying and transmission might not be smooth and at times become latent, which could lead to a frustrating experience for the veteran.

Ionic

Ionic is an open source, cross-platform app framework based on the HTML5 programming language provided via Apache Cordova/PhoneGap. Ionic provides tools and services for developing cross-platform mobile apps using standard web technologies like CSS, HTML5 and JS instead of native programming languages. But this approach relies heavily on many third-party plugins to achieve more robust functionality.



Ionic allows developers to write in web-languages and leaves the mobile device heavy-lifting to Cordova and standardized APIs.

Pros

- Ionic uses a powerful HTML5 SDK that lets developers write in widely accepted and easily accessible programming languages
- Ionic has a command line interface which uses Cordova on the back-end and allows developers to build apps for iOS and Android with minimal native integration
- Ionic can easily access the full native functionality of the devices using Cordova

Cons

- Ionic's app reliance on third-party plugins lengthens the app startup/boot time
- Ionic apps have relatively sub-optimal UI/UX that are ill-equipped to handle heavy graphics transitions and animations
- Ionic has very few best practices to optimize mobile battery consumption
- Ionic does not support older version mobiles OSes (Android 4.1 or iOS 9 and older)

- Ionic is missing many code quality options and workflows needed by enterprises when building an app in a Continuous Integration/Continuous Development environment. Thus, Ionic apps do not lend themselves well to producing enterprise grade applications

Sample Use Cases and Considerations with an Ionic App

Due to relative lack of backward compatibility, a VA app built on Ionic may not run on older devices and OSes. Also, even with older, support versions, the app could be unstable relative to devices with newer OSes.

Forms built within the application built on Ionic can pixelate or even fail if run on a suboptimal device with an encompassed resolution and aspect ratio.

Finally, longer app start times can be frustrating to veterans and they likely would be discouraged from leaving the application running due to battery consumption. Both these considerations would likely impact adoption and usage.

Hybrid App Development

In a hybrid app development environment, the code is written in traditional HTML, CSS, or JavaScript and later wrapped in an "invisible" native WebView browser. Hybrid applications are web applications executed in the native browser on a device, such as UIWebView in iOS and WebView in Android (not Safari or Chrome). This approach seeks maximum translation across devices with minimal integration consideration but sacrifices use of native features in lieu of more generic "web view" approaches.

Examples of Hybrid apps include Sworkit, which allowed a single developer to create the app for both iOS and Android, and the National Museum of African American History and Culture that loads content from the website and exhibits.

Pros

- Hybrid app development is the fastest and cheapest approach due to easy commodity access of programming languages and little translation to device-specific stacks
- Hybrid development use of HTML, CSS and JS make finding competent developers easier
- Hybrid development allows coding assets written in HTML, CSS, JavaScript to be packaged through Apache Cordova and translated to the target platform. Once built, the app can run like any other kind of application across many devices.

Cons

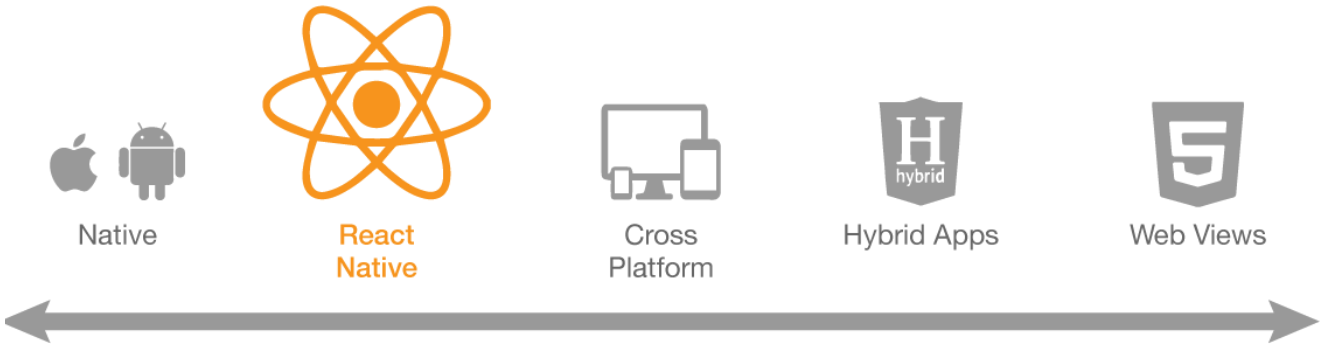
- Hybrid apps are separated from the device hardware abstraction and lack access to many features and functionalities
- Hybrid apps have no option to function in offline mode given that they are wrapped in the native browser
- Hybrid development does not guarantee excellent and rigid app performance
- Hybrid apps that are completely built on webview have higher chances of getting rejected from the Apple Store; a considerable project risk

Sample Use Cases and Considerations with a Hybrid App

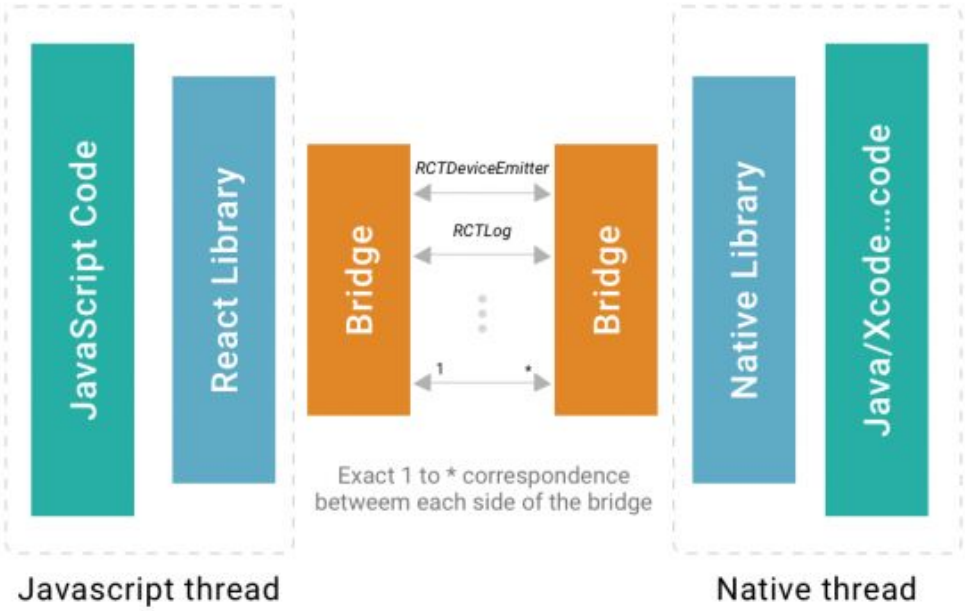
Developers can create hybrid apps for multiple platforms with the same shared application codebase and quickly translate the app to additional devices with an additional line of code per device. The application will be extremely lite and can access static content on the Vets.gov website easily. But veterans will not be able to access the application offline at all.

Also, the app won't allow the Veteran to access the device-specific features from within the app at all. Every time a veteran fills a form, they cannot turn off the application, come back, and expect the information to still be there. These considerations will lead to a very generic, less than useful experience for the veteran.

Our Recommendation: *React Native*



The 580 Strategies team recommends Veterans Affairs pursue a mobile app development approach using the *React Native* cross-platform framework. React Native is a great option for creating performant apps for both iOS and Android that feel at home on their respective platforms without the need to manage entirely separate applications. React Native is a relatively new framework and the very first that allows developers to create a mobile app which will work both on iOS and Android without compromising speed, efficiency, and the overall look and feel while continuing to take advantage of native features of the platforms and devices.



React Native offers all the ease of development JavaScript supplies but manageable translation to native devices and features as well

There are three parts of the React Native framework: the Javascript side, the Bridge, and the Native side. Each vertical acts in an encapsulated yet relatively performant manner.

Developers write React Native apps using the mainstream programming language Javascript, which is then connected to the native app platforms such as iOS and Android via a bridging layer. This bridge allows React Native apps to work as if they were built natively but allows developers to focus their efforts on shared code assets but access native features simply by invoking libraries. React Native uses libraries and native APIs to render platform-specific components: it invokes Objective-C/Swift or Java APIs to render to iOS or Android components, respectively.

This distinctive framework sets React Native apart from other cross-platform mobile application development technologies. Hence, the apps built in React Native not only look consistent and leverage native device capabilities, they are also performant.

Leveraging React Libraries for Native Capabilities

Integrate Touch ID and Face ID

React Native with the help of react-native-touch-id library allows developers to easily make use of Touch ID and Face ID features for iOS devices if desired.

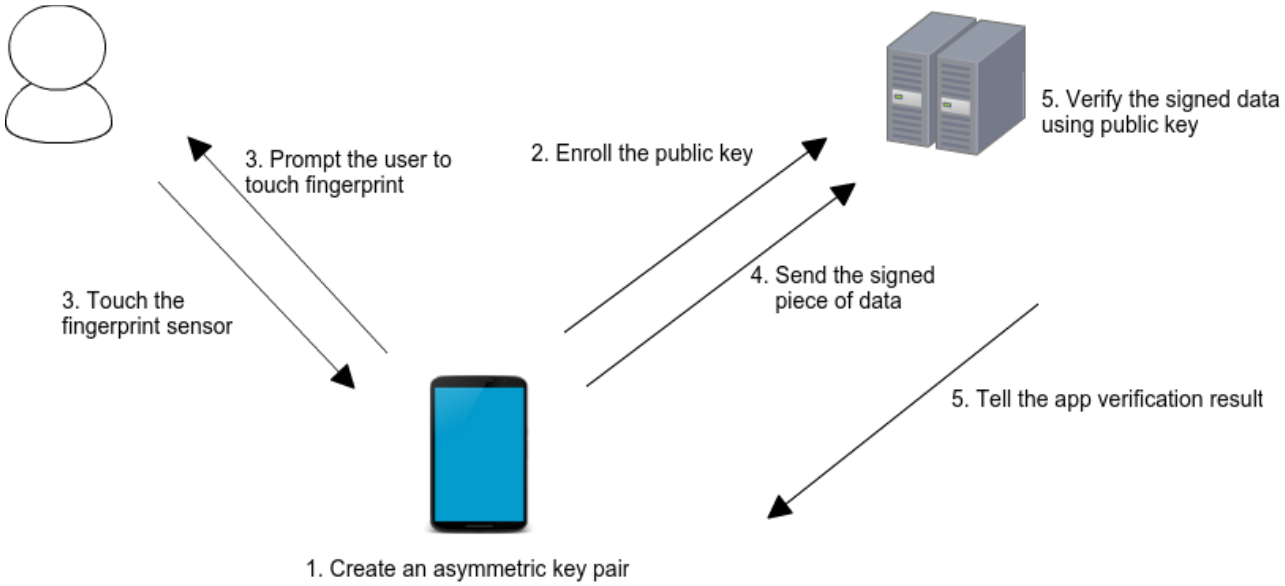
Adding a Touch ID and Face ID makes the most sense if:

- Veterans should sign in to use the app (and there is something worth protecting)
- Veterans need to remain signed in (otherwise there is no real security gain)

Store Credentials

The library react-native-keychain allows developers to store credentials in a device's secure storage (the "keychain" in iOS and the "keystore" on Android). This can be particularly helpful if veterans need to store credentials for multiple systems such as ID.me or My HealtheVet.

Managing Keychains and Stores



After storing credentials, the key task is to manage them efficiently and secure them effectively. React

Native can leverage the *react-native-biometrics* library to create public/private key pairs that are stored in native keystores/keychains and protected under biometric authentication on the device.

When a user enrolls in biometrics, a key pair is generated. The private key is stored securely on the device and the public key is sent to a server for registration. When the user wishes to authenticate, the user is prompted for biometrics, which unlocks the securely stored private key. Then a cryptographic signature is generated and sent to the server for verification. The server then verifies the signature. If the verification was successful, the server returns an appropriate response and authorizes the user.

All this means that VA will be providing securely managed authentication options on the devices that take advantage of the biometric features of the platform while keeping veteran credential information secured by reasonable means from theft or tampering.

Extending Identity Providers and Leveraging Multi-factor Authentication with *React Native*

React Native supports a variety of authentication mechanisms and identity providers to accommodate multi-factor authentication and enterprise identity management systems that may be in place.

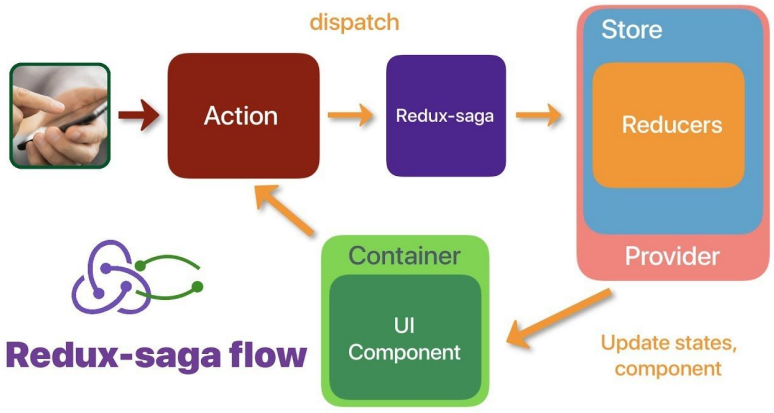
React Native supports an *App Auth* library to connect OAuth and OpenID Connect providers. The following providers are compliant with OpenID:

- Identityserver4
- Identityserver3
- Google
- Okta
- KeyCloak

React Native also supports Auth0 as a native SDK as well as an additional option.

For VA specifically, getting started with Simple Auth may be enough for MVP, with plans to expand into fully mature identity management systems such as Okta that are already in place at the agency.

Storing Application State for Offline Use



One of the best use case for this is that whenever a new data stream has to be shown or updated, redux-saga will enable the UI to update without having to refresh the page/screen.

Redux-saga is a library that allows the app to communicate and stay in sync with external APIs and keep track of application state even while offline, which can be a major advantage to the VA. *Redux-saga* is middleware, which enables it to respond to app actions, access the application state data, and dispatch actions to APIs as well.

What this will mean for the veteran is that they can fill out forms within the app and not need to refresh the pages or submit the form to the server before seeing errors or omissions on form data and that data will not be lost if the app is offline or an error occurs.

Local Database Solutions using SQLite

Building a successful app relies not only on programming languages and client-side APIs for accessing data across networks, but the actual client-side database on the device also plays a crucial role. The 580 Strategies team recommends using *SQLite* for this function to compliment application state data using *redux-saga*.

SQLite is a relational database management system designed specifically to meet the storage demands of mobile applications. The word “lite” in *SQLite* describes it as being a lightweight database which requires minimal setup. *SQLite* can be integrated with the mobile application to directly access the database transparently to the user.

SQLite is an ACID-compliant (atomicity, consistency, isolation and durability) database that implements most mainstream SQL-based standards. To easily enable offline persistence with *SQLite*, developers can use a *react-native-SQLite-storage* plugin to manage the data within the app.

The addition of *SQLite* will play well with redux-saga for autosaving/autofilling forms and keeping user data maintained for proper submission. The end result of the combination of *redux-saga* and *SQLite* is data that is cached and managed locally on the app, which gives the user a better experience and peace of mind that their form data will be sent where it’s supposed to go properly without having to re-enter it multiple times.

Additionally, *SQLite* also supports data encryption with an extension named *SQLite Encryption Extension (SEE).* This is a public domain addon that allows the app to read and write encrypted database files to protect both VA data and metadata stored on the mobile device. This will allow VA to protect veteran PII and health information that may be stored on the device for various features.

A good example of this is the Adobe Suite apps, they store local data, which allow (is essential for) the user to run the application offline. Moreover, they also use local storage to store cache in SQLite in order to initiate faster initialization of the app. For files created on web app, they use server side storage, whereas when user uses the desktop/ mobile app, they sync the data with SQLite. A healthy combination of both server-side and locally stored data provides optimal user experience.

React.JS and *React Native* Compatibility

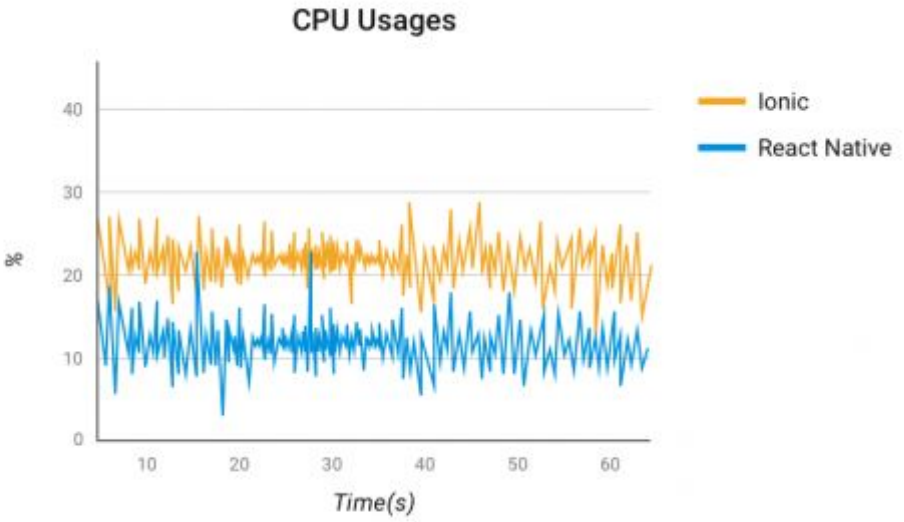
React.JS apps are similar to React Native apps but are written with more of a Hybrid approaching using HTML and CSS. They can be complementary of one another in that *React.JS* apps or endpoints produce *React Native* readable views and inputs. This can save substantial amounts of recoding if VA is already be using *React.JS* apps for critical services. For example, the forms associated with the Prescription Refill service could be recreated with minimal ease directly in the mobile app.

Sample Use Cases and Considerations with the React Native App Recommendation

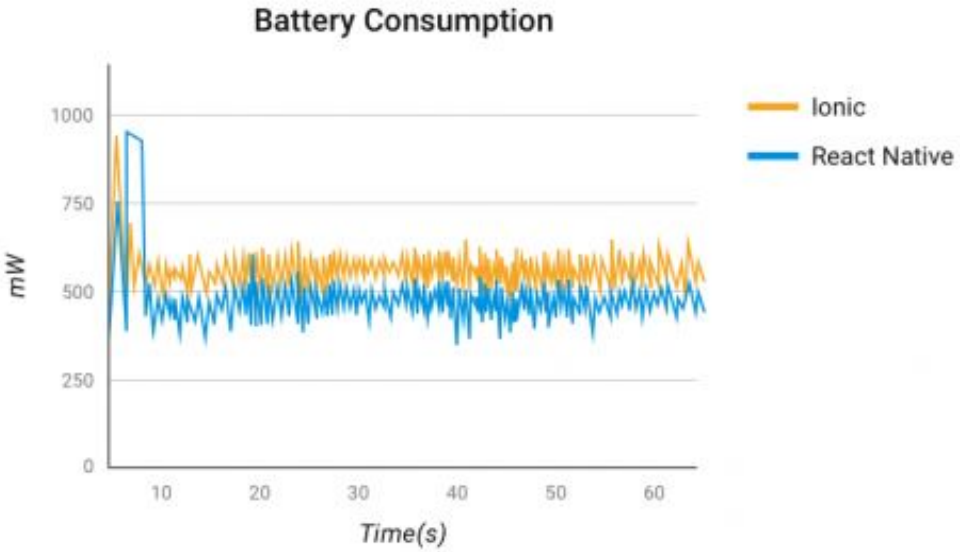
- The close-to-native look and feel of the app will help veterans get easily acquainted with the app
- Veterans can fill forms easily, on the go, and using offline storage, return to forms any time without losing data
- When a veteran wants to sign in using external identity providers, they'll be able to do so from within the application itself using secure storage and key management
- Veterans will be able to use Touch ID and Face ID as authentication options instead of typing their credentials to Sign-in to the app each time
- When a veteran wants find information and access to features within the app without having to navigate manually, they can do so via Siri(iOS) or Google Assistant (android), as the app won't restrict access to native functionalities like voice assistants
- When a veteran locates their records within systems connected to the mobile app, they can save those records as .pdf files for later access
- Veterans can use their smartphone cameras or gallery to submit photos of documentation or ID
- Using GPS and mapping features veterans can locate and filter VA facilities near them easily from within the app
- After veterans have located a facility, they can also schedule, modify, or cancel appointments from within the app as well
- Veterans can receive push notifications on their mobile from the app whenever their application status changes, they get an appointment status update, or they leave a form midway, etc...
- Veterans can not only submit claims/forms from within the app itself, but also track their progress
- A veteran can use the GI-Bill Comparison Tool natively to search for school details and programs that they offer including tuition, housing and books fees associated with each

Performance: Comparing React Native with Ionic

Beyond features and use-cases, it is also important to consider app performance. 580 Strategies partner, Simform, built a small expense management app and benchmarked performance for the app in a side-by-side comparison of React Native versus Ionic which produced the following results.



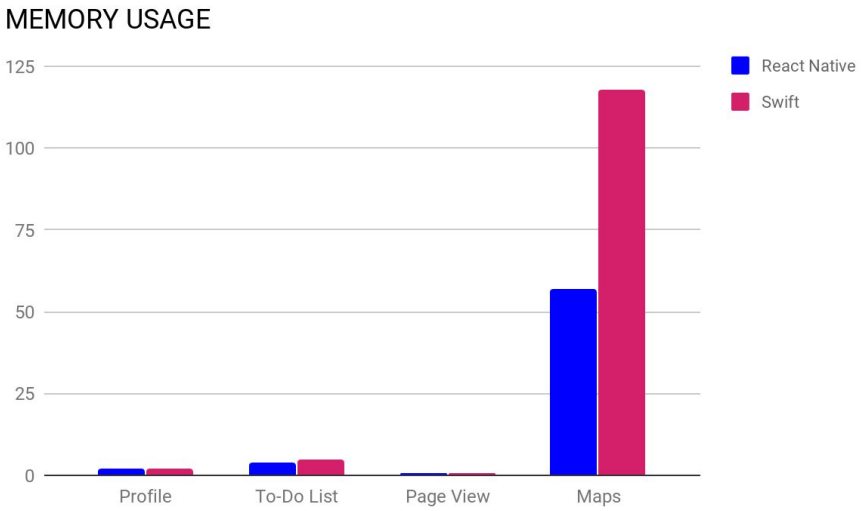
CPU Usage: The graph above represents the CPU usage comparison when both the application were kept on startup and idle states. The average CPU usage of the Ionic application was recorded at 21.3% whereas the React Native application CPU usage was recorded as 12.8%.



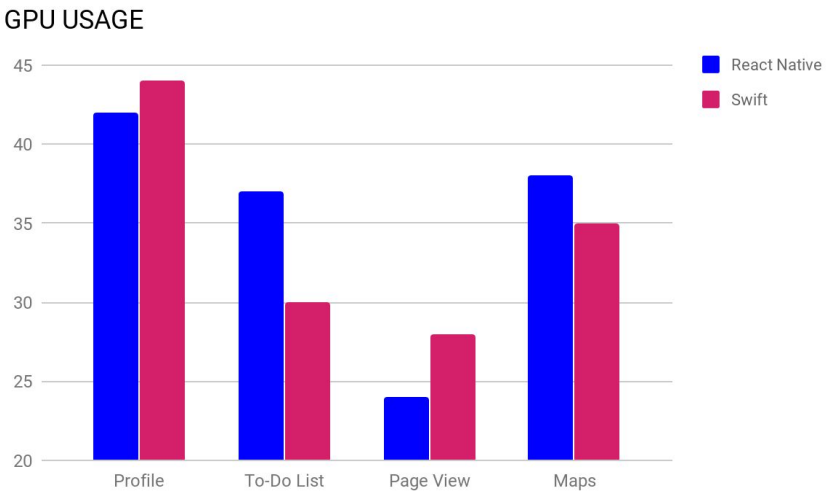
Power Consumption: The graph above illustrates battery consumption of both the applications. Battery usage was significantly higher for Ionic application at the time of application startup with 520 mW average battery usage compared to React Native with an average of 473.2 mW.

Comparing React Native with true Native (Swift)

To compare React Native versus Swift, we use [the following case study] that produced these results.



Memory Usage: The chart represents the memory usage comparison for different features within a sample application. Memory usage of the React Native app is a bit lower if not equal, than the native application, except for when the apps were running maps, where React Native greatly outperformed.



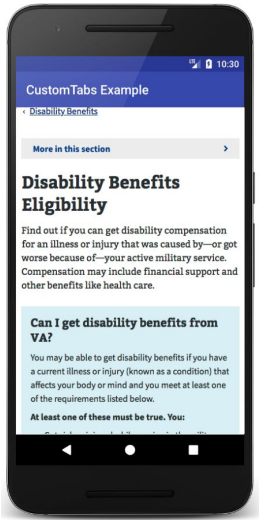
GPU Usage: Both applications were tested to monitor their GPU usage along sample screens. React Native edges out Swift in terms of aggregate score.

Static Content and Web View Considerations

Rather than coding static content or relying on native browsers for the occasional web view, the 580 Strategies team recommends “Chrome Custom Tabs” to elegantly work with static content. Chrome Custom Tabs give apps more control over their web experience and make transitions between native and web content more seamless without having to resort to a standard web view through a default browser.

For auxiliary static content, we can have a Chrome Custom Tab assist with:

- Synchronized AutoComplete data across devices for faster and better form completion
- Quickly return to app from Chrome with a single tap
- Chrome can preload many of the frequently accessed pages used in the app to speed up load times



Chrome Custom Tabs also offer additional benefits:

- Security: the browser uses Google's Safe Browsing to protect the user and the device from dangerous sites such as known Phishing offenders
- Performance optimization:
    - Pre-warming of the Browser in the background, while avoiding stealing resources from the application.
    - Providing a likely URL in advance to the browser, which may perform speculative work, speeding up page load time.

Utilizing Chrome Custom Tabs will require the user to have Chrome installed on the device, but also will function with failback measure to work with whatever default browser is registered with the device if Chrome is missing. This will allow veterans to take full advantage of the additional features Chrome provides but still support basic web view functionality if it is missing.

In summary, the 580 Strategies team recommends VA utilize the React Native mobile development framework with associated plugins listed above. This framework will give VA multiple top-level benefits including:

- Leveraging a cross-platform development approach for a single app that maximizes shared code reuse and minimizes overhead associated with maintaining multiple apps
- Utilizing a technology stack that is consumer-oriented and will grow with the app, supporting our placement of the veteran at the heart of the technology endeavor
- Leveraging a commodity industry programming language, Javascript, to lower overall cost of development
- Balancing access to native features and functions of various mobile platforms and devices while still focusing on a single application with a large amount of shared code
- Utilizing a modern mobile framework which has demonstrated performance gains for CPU, GPU, and memory considerations when tested in real-world scenarios

Agile Software Development Framework

580 Strategies has over six years of experience in both delivering product using agile development methods within government and training civil servants how to oversee their own agile development programs. We recommend the following framework as a guideline that VA can use either as requirements for a new Request for Proposal or as a starting point for an experienced agile development team. This framework is intended to be lean enough to get started but is tried and tested enough to shortcut many of the failings of embarking upon a new agile development journey within the constraints of government:

Agile Software Development provides the following benefits:

- Faster, more lightly-scoped acquisition processes that encourage innovation
- User-centered design efforts that focus on designing with users, not for them
- Gathering requirements just-in-time, when needed, instead of too far ahead so that they're out of date when coding actually begins
- Building in flexibility to adjust capacity and priority to account for change
- Shipping software in smaller parts with more immediate feedback and more time for adjustment if expectations aren't met
- The ability to scale output, grow, and perform better over time
- A better software development experience for both staff and veterans

The Agile Manifesto

These changes are driven by some very simple and straightforward guiding principles that software development teams call "The Agile Manifesto." When confusion or decisions arise in the course of operating a product team, referring to the Agile Manifesto can usually help guide a team to a decision that allows it to move forward in a productive manner.

We value:

Individuals and interactions *over* processes and tools

Working software *over* comprehensive documentation

Customer collaboration *over* contract negotiation

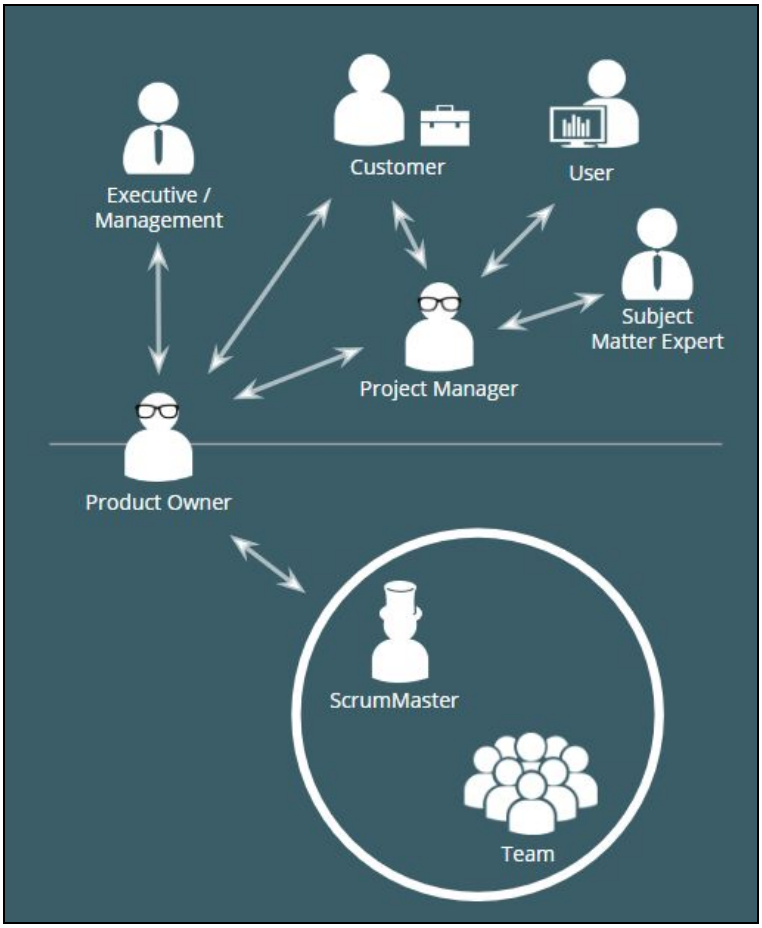
Responding to change *over* following a plan

By following the framework recommended here, we can ensure that agile principles from the Manifesto are applied in a way that works within the constraints of government to ensure a better product for veterans. And in cases where the Manifesto doesn't give enough guidance, an easy North Star can be summed up as: *Above all else: Ship working software!*

The Team

We recommend keeping the team size small and smart, but large enough to cover the capacity needed. When constructing the agile team, government will often contract for the technical and soft skills needed to do advanced software development work. Many times the agile team will come prepackaged as a part of the contract with the project or ScrumMaster, business analysts, engineers, developers, testers, trainers, and other subject-matter experts as needed.

But what's missing from this equation are the government stakeholders and the users. This is where we recommend teams place emphasis on bringing to the table engaged government staff to serve as Product Owners and enthusiastic end-users -- in VA's case, veterans -- to work alongside the team.



A Typical Agile Team in Government

The people above the line are generally VA stakeholders including everyone from the Secretary down to the veteran using the services the team is building. This can be any mix of people interested in what the team is shipping. It also allows for additional project managers to interface with the agile Product Owner and Scrum team as part of larger initiatives or in cases where there are multiple teams working in tandem toward a large goal.

The people below the line are a turnkey Scrum Team often provided by contract. For this project, we're assuming they are expert mobile application developers and associated team roles to deliver high

quality software for VA. These teams will vary in size, skill, and capacity as determined by the specific project for which they've been contracted.

You'll notice that in the middle is the Product Owner, who is key to making the entire project work! Ideally the Product Owner (or Owners) is a government stakeholder who is well-versed in the program and mission needs of what the team is building, but with the associated agile fluency and technology skills necessary to work closely with the Scrum Team on a daily basis.

A skilled Product Owner can mean the difference between a failed project or a successful one. This person needs the right demeanour to work on both sides of the team, both keeping stakeholders engaged and briefed, as well as digging in to the details of the product on a daily basis with the team. They should also be empowered to make all necessary decisions when it comes to the product. They are the primary decision maker the Scrum Team comes to whenever there are questions to resolve.

Beyond these core roles, it is important to connect the end user, the veterans directly into a working relationship with both the Product Owner and the Scrum Team. We value working with and designing *with* the users and not for them. This relationship will take many different forms given the specific group makeup and project, but it is important to emphasize this relationship and give it top-billing whenever any question of product feature/functionality is concerned. The closer the actual end user is to the process and team, the higher the quality and satisfaction levels will be in the end. Which brings us to our work: User Stories.

User Stories

User Stories are the basic unit of work for the Scrum Team. These are well-crafted, one-line sentences written in plain English about what a desired function or feature of the product will do.

It has three parts:

As a [role], I want to [action] so that [business value].

The role is who the user is and is key to considering how the feature or function will be built. An analytics dashboard for the Secretary will be vastly different than the veteran's accessing the Crisis Line which will be different than the VA staffer's accessing the backend of the system. Starting with the proper role and mindset is key for every story.

The action is the core of the feature or function. What does the user want the product to do? Submit a case? Call a phone number? Send an email? This is chief point of design and engineering consideration in any user story.

The business value is the reason the team is spending time and effort to build the feature. Each user story should be justified in and of itself as a reason to be built. If the team finds that a story is lacking business value, it's likely a candidate for being removed.

Sample user stories for the VA mobile application could be as follows:

As a *veteran*, I want to *call the Crisis Line directly from the app* so that I can reach help instantly.

or

As a *veteran*, I want to *search for facilities* so that *I can access the services I'm entitled to in my area*.

As a part of this effort, the 580 Strategies team has compiled an initial list of User Stories as a Product Backlog to get the VA team started quickly when the time comes.

Well-written User Stories meet the following criteria:

- Are written *with* users not for them
- Have only one "ask" per story
- Demonstrate clear business value

It is important to note that user stories should often be just that: a story developed by the user. Again, putting users -- veterans -- at the heart of our work, the vast majority of these stories should be written by veterans and managed by the Product Owner as the team builds out the app.

Once the stories are written as one-liners, the Scrum Team will take additional action on them and add information to get them ready for use:

- They are assigned a relative priority and/or risk v. reward score by the Product Owner
- "Acceptance Criteria" are added by business analysts that define pass/fail criteria for testing to determine if the feature or function as-built delivers what the user is asking for
- They are assigned estimated "size" by the team during story grooming or sprint planning sessions

At this point, the user stories are candidates for work items that the Scrum Team can accept and build. And over the course of a project, there can be hundreds of user stories, which brings us to the Product Backlog.

Product Backlog

The Product Backlog is a mainstay for any agile development project. It's just a fancy way of saying "a to-do list of all work." And in the case of the VA mobile application, it's a list of all of the user stories for the features and functions we think we can build for the application itself. The starting/notional Product Backlog is attached to this deliverable as a starting place for VA.

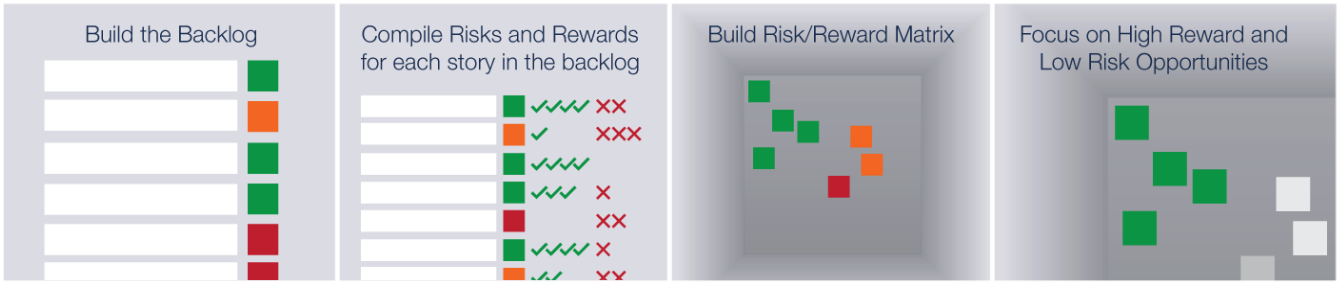
Initial Product Backlogs can be very simple and usually begin in spreadsheet form like this:

ID	Story	Rank	Notes:
Feature #1	Sample user story	Must Have	This is a feature the system must have in order to be considered a success
Feature #2	Sample user story	Should Have	This is a feature the system should have if schedule/capacity allows

Feature #3	Sample user story	Nice to Have	This is a feature that is desirable but not necessary and to be built only if additional capacity becomes available

This simple Product Backlog is arranged by ID number, the story itself, a notional rank/priority, and associated notes. Using this format is enough to get a team started transcribing user stories from 3x5 cards or by adding them directly to the spreadsheet.

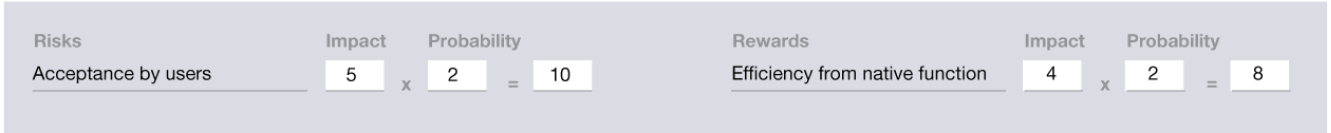
For this project specifically, we've gone beyond the simple starting point and have added two key areas: Epics and Risk/Reward scores. Epics are simply ways of gathering groups of user stories together into similar categories of functions that make logical sense. They help agile teams break down sequencing of development and are sometimes helpful in status reporting, but they aren't always necessary.



Risk and Reward scores help surface feature opportunities quickly.

The Risk/Reward scores have been introduced by the 580 Strategies design team and sourced directly from veterans when possible. These are combination scores on a sliding-point scale that consider items such as technical complexity, relative importance to the veteran, likelihood of adoption, etc. These key factors are then assigned a point value which is ultimately computed by formula to give the overall user story a numeric ranking value that can be used to determine priority relative to the others. It's intended to allow the team to consider multiple unrelated factors - including intangibles - in the feature selection process! This balances strategic and user-centered goals with implementation and operating risks and rewards.

Risks can range from implementation complexity, user acceptance, bleeding-edge technology dependencies, or lack of supporting infrastructure. Rewards can be similarly varied. For example: user satisfaction or need fulfillment, greater efficiency, or faster task completion. Rewards can also be drawn from supporting user research interviews as well as user validation sessions. Risks and rewards are weighed by potential impact (on a scale of 1 - 5) multiplied by the probability of occurring (on a scale of 1 - 3 or low, medium, and high).

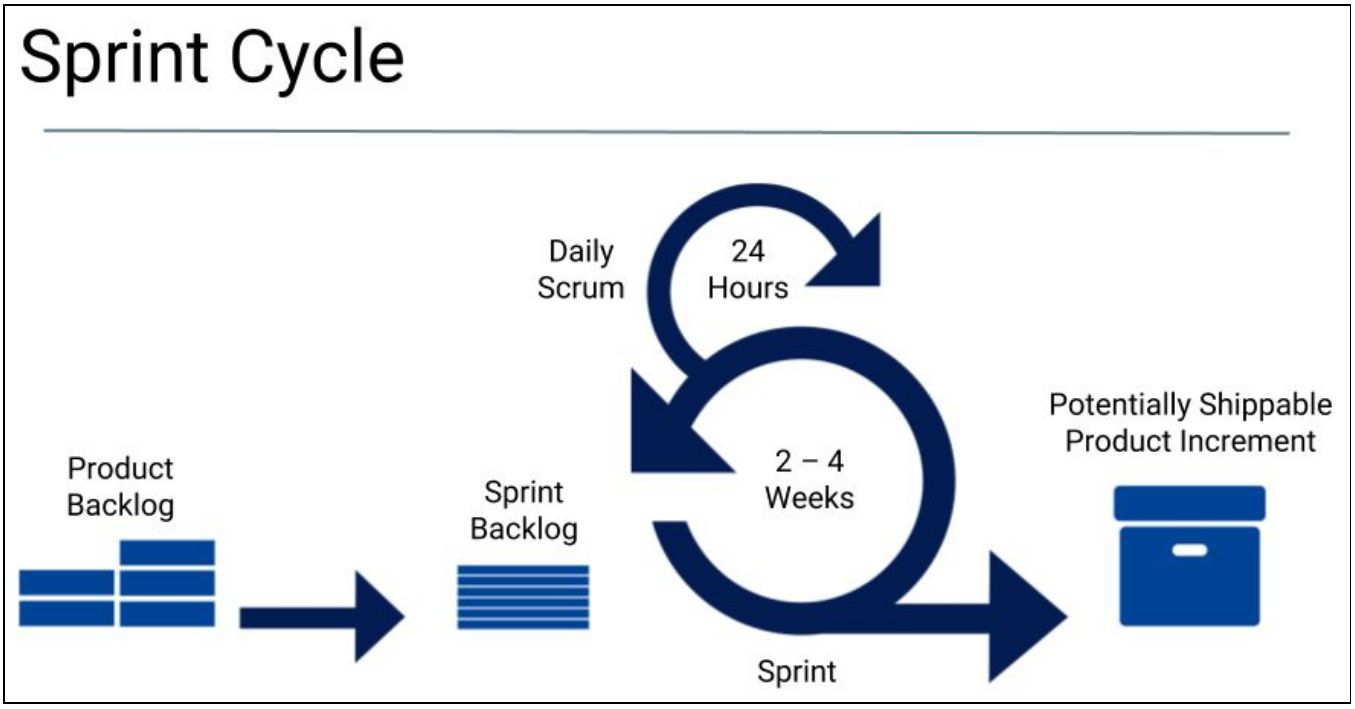


Risks and Rewards are scored as Impact x Probability.

In addition to working with users, stakeholders, and the Scrum Team, the Product Owner's primary job is maintaining and managing the Product Backlog. Given the challenges of building large software projects in an ever-changing environment, the Product Owner's most important job is to keep the Product Backlog "groomed" in terms of priority. This becomes helpful for the team where if they have questions or doubt about what to build next, it's very clear that the Product Owner has expressed their opinion through prioritization.

The Sprint Cycle

Typical sprints range from two to four weeks in industry best practice, and those have proven to be a good fit for most government agencies as well. Two weeks can be aggressive in some instances when there is a lot of process or governance overhead (it may take more time to get sign-off than the two weeks allows), and four weeks sometimes can be too slow in some cases where the agile team is very practiced and work gets a little stale by the fourth week of the sprint. The timeframes will need to be experimented with and adjusted to what works for VA.



A sprint is a single agile development cycle that delivers an amount of working product.

For each sprint there are several minimum "ceremonies" or meetings that the team will participate in:

- Stand Ups / Scrums
- Sprint Planning
- Sprint Retrospective

Stand ups or "Scrums" are the basic check-in for the team and consist of an open and democratic process whereby each member of the team is empowered to speak-up and reports out on the following three items:

1. What did you accomplish since the last stand up?
2. What are you working on next?
3. Do you have any blockers keeping you from being productive?

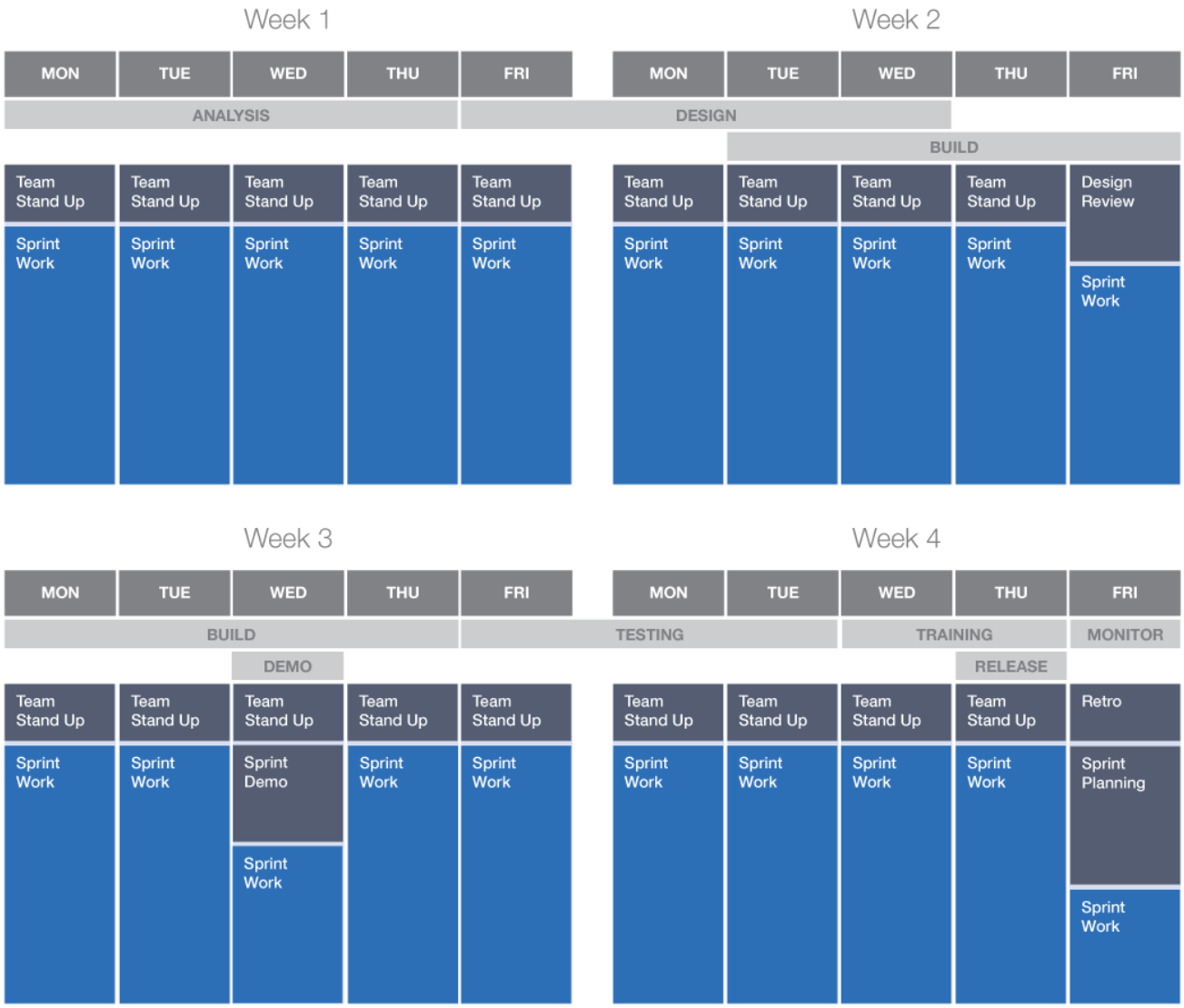
The Sprint Planning meeting is a special meeting that occurs at or near the beginning of a sprint when the scrum team works together with the Product Owner to agree upon the work that will be accepted and delivered by the end of the sprint. Sprint planning is a critical function that should take into account available team capacity, up-to-date business priorities, any specific project considerations, and the overall balance of what the team is working on. Ideal outcomes from a good sprint planning session are a team that feels good about the amount of work they're taking on and are confident they can deliver on-time with high quality and a Product Owner satisfied with the amount of product that will be shipped at the end of the sprint.

The Sprint Retrospective meeting is another special meeting that takes place at or near the end of the sprint and ideally before sprint planning for the next sprint. The retrospective is a review of the previous sprint when the entire team is empowered to speak up and answer the following two questions: What went well that we should continue doing, and what went poorly that we could improve upon? The retrospective is critical to the team continually improving upon and ultimately shipping more software sprint-over-sprint.

Further, the 580 Strategies team has found that it is beneficial to define various phases of the sprint cycle when the team is focused on specific types of activities. These phases can also include various checkpoints that satisfy government processes and keep stakeholders engaged.

The recommended phases for a sprint:

- **Analysis:** Confirming user stories for clarity and answering any questions developers might have about them
- **Design:** Crafting notional solutions to the problems presented by the user stories
- **Build:** Where the actual work of coding, creating, and innovating is conducted.
- **Demo:** A live walkthrough of product built during the sprint to elicit feedback from stakeholders
- **Testing:** Structured user, quality assurance, and technical compliance testing of product built during the sprint
- **Training:** Formal and informal training for product created during the sprint; creation or update of any required documentation
- **Release:** Technical release of product into production systems
- **Monitoring:** Supporting users, eliciting feedback, and resolving any problems that may arise from the newly released product



A typical four-week sprint cycle with defined agile phases

Most of these phases and ceremonies are necessary for the basic functioning of the Scrum Team. But other process milestones specific to government can be added to augment the sprint and help it function better within government. For example:

- Design reviews for technical acceptance
- Sprint Demo for stakeholder engagement/feedback
- Release management checkpoints
- Security approvals or Center of Excellence participation

Special Sprints

Kickoff: Sprint 0

For any new agile development effort, we recommend beginning with a “Sprint 0” that is a short period of time to complete all the initial logistical tasks a new team needs to complete before formally starting

software work. For the VA mobile application, we are recommending a full four-week Sprint 0 for this purpose as well.

During Sprint 0, the team will complete a few critical task including:

- Identifying all team members and starting onboarding
- Solidifying and defining team member roles
- Agreeing to a sprint-cycle duration
- Establishing and identifying necessary process steps specific to the project and agency
- Acquiring and establishing basic logistical tools such as:
    - A project management tool for the product backlog
    - A repository for document storage
    - Telephony and computing equipment for conference calls and work items
- Establishing a stand-up meeting cadence
- Configuring any technical tools needed such as cloud instances, Git repositories, databases, and Continuous Integration tools
- Importing or building an initial Product Backlog
- Establishing a Risk Management Plan and Quality Assurance Surveillance Plan (QASP)
- Setting performance metrics for tracking progress
- Initializing the project calendar
- Conducting user research
- Laying the groundwork for the design system by:
    - Identifying the core interaction model for the app
    - Setting the initial visual direction

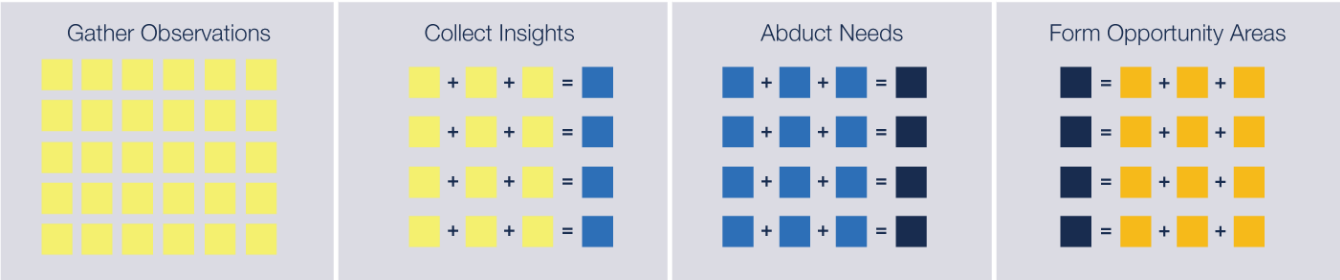
At the conclusion of Sprint 0, the team should have the basic working norms established as well as the tools in place to get started with actual build work. From there, the first Sprint Planning ceremony for Sprint 1 can be held and the team will get started iterating and shipping product.

Design or Analysis Sprints

Often, scrum teams may need to do preliminary work that involves focused design or engineering work that produces documentation or planning artifacts instead of writing actual code or shipping product. For development of a mobile app particularly, utilizing design sprints will be key to understanding the needs of veterans while laying the foundation for the app's interaction model, visual direction, and major experiences before the detailed development work begins.

User Research

User Research focuses on understanding users' needs, expectations, and motivations through observation techniques, task analysis, and other feedback methodologies. Captured insights and observations are processed (synthesized) to identify opportunities for improvement as well as themes or patterns. The findings are intended to be used as potential risks and/or rewards in the backlog prioritization process as well as serve as a guide to future design activities.



Design synthesis is crucial to putting the user at the heart of development efforts.

During synthesis, multiple rounds of affinity sorting distill raw notes and observations into insights. By repeating the sorting process, we further distil the collective insights into core needs and opportunities or ideas that address those needs.

Interaction Model

An interaction model helps define how the various features and concepts in an application are assembled and interrelate. Defining the right interaction model before sprints begin is critical to the cohesion, intuitiveness, and navigability of the overall app experience.

Initial Visual Direction

Initial visual explorations yield a range of compelling directions that provoke or inspire thought**.** The resulting concepts form the foundation of the app's visual design language including branding, color, typography, imagery and iconography. These explorations serve as a starting point for subsequent design sprints.

User Validation

Ideally, user validation sessions are conducted at regular intervals throughout the project to gather direct input on the performance of the app based on real interactions with veterans. The result is a clear understanding of what veterans face while using the app and a backlog of refactoring stories to iteratively address any unexpected challenges that may arise.

When planning design sprints, we recommend either running standalone design sprints to complete this work before actual build begins *or* staggering the design sprints one or two iterations ahead of the build to allow the design team to execute and complete its work so that UI kits, sliced assets, and documentation will be available and approved by the time the development team begins the corresponding build sprint.

Translation Sprints

Similar to a design or analysis sprint, there are times when a core software product will need to be refactored or extended to another platform or system integration. In this case, after the bulk of the core work is done, the team may execute a translation sprint that focuses on taking the core product and "translating" it to the new target platform. These sprints will largely be an exercise in testing and quality

assurance instead of heavy lifting with coding, but they are equally important ensuring the app functions on the target platform properly.

Performance Management

Beyond typical agile implementations, there are additional measures that are helpful for government projects that both give a good sense of contract performance and allow Contracting Officers to gauge how well an agile team is doing.

These performance metrics break down into three general areas:

> **Schedule.** Is the team staying within the agreed-upon sprint cycle phases and hitting the required milestones? Are releases being deployed according to the release management plan?
>
> **Velocity.** How much product is being shipped sprint-over-sprint? Is unfinished work slipping between sprints? Are any team members idle at any point during the sprint? Can we adjust team capacity to meet the work need and vice-versa?
>
> **Quality.** How many defects are we discovering? How quickly are defects being addressed? Are any defects being released into production? Are defects disrupting capacity to take on new feature work? Are users happy with what the team is delivering? Are users proficient with and actually using the features? Are users satisfied with the amount of training and documentation?

Contractually, handling performance management as a set of "Acceptable Quality Levels" (AQLs) with defined metrics is very helpful. Below is a sample table of to consider.

Sample *Acceptable Quality Levels*		
Measure	**Satisfactory**	**Excellent**
Iteration schedule is met no less than 85% of the time during the period of performance	Met 80% of the time	Met 85% of the time
User story completion within sprint	Completed within 80% of commitment	Completed within 90% of commitment
Code coverage and test classes	Meet or exceeds 80%	Exceeds 85%

Velocity in Story Points	Throughput is no less than 10% below 30 story points per week	Throughput is no less than 5% below 30 story points per week
Pre-release defects	All defects identified and resolved prior to production release	All defects identified and resolved prior to production release
Post-release defects (after current sprint release but identified within two sprints)	<= 2 defects identified after release	No post-release defects identified
Defect resolution	Within two sprints of identification	Within one sprint of identification

A sample table of Acceptable Quality Levels (AQLs).

If teams are new to agile or embarking upon a new project or technology stack, 580 Strategies recommends crafting notional Acceptable Quality Levels during Sprint 0 in collaboration with the COR and then using 2-3 sprints to baseline the average metrics for them before enacting them as contractually binding. AQLs can then further be enhanced or refined over time and contracts modified either bi-laterally or when they are renewed to achieve greater levels of performance.

Building in metrics and the answers to these questions in the contract will allow both the Product Owner and COR/Contracting Officer to keep a close eye on team performance and make adjustments as the project matures.

PRODUCT BACKLOG

The 580 Strategies team worked directly with veterans and analyzed the Vets.gov website to produce a backlog of user stories that VA can use to kickstart the development of a mobile application. This deliverable is provided as an attachment in spreadsheet format that can easily be imported into a project management tool of VA's choosing.

Each row represents a potential feature the team can build for the application. But beyond simply notionally ranking each story by "Must Have," "Should Have," and "Nice-to-Have" priorities, we applied our risk versus reward framework to these stories for a more nuanced ranking system that easily identifies candidate stories for Minimum Viable Product.

Scoring Framework

Size: Each story has a notional size that represents the overall level of effort associated with the story. This gives us a quick way to align the number of stories and their overall size with the capacity of the team that will be doing the development. Our scale ranges from "Small" stories that could be completed in roughly a half-day of coding to "Extra-Large" (XL) stories that are likely too big for a single sprint and will need to be distilled into more granular stories before build.

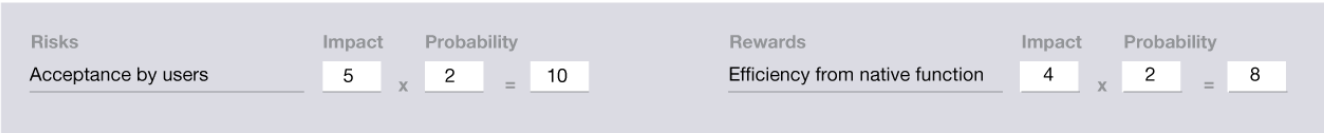
Estimated/notional sizing by points and t-shirt size used for this exercise are:

- **Small**: 1-2 story points (easily completed in a half-day of coding)
- **Medium**: 3-6 story points (1-2 days of coding)
- **Large**: 7-12 story points (multiple days of coding)
- **Extra Large**: 13+ story points (likely too big for a single sprint)

Risk: Risks can include technical implementation complexity, user acceptance or adoption likelihood, bleeding-edge technology dependencies, or lack of supporting infrastructure. They are specific to each user story and represent challenges or barriers to successful delivery of the feature.

Reward: Rewards can be similarly varied. For example: user satisfaction or need fulfillment, greater efficiency, or faster task completion. Rewards can also be drawn from supporting user research interviews as well as user validation sessions. They represent the value to or satisfaction of the user when delivered.

Priority Score: Risks and rewards are weighed by *potential impact* (on a scale of 1 - 5) multiplied by the *probability of occurring* (on a scale of 1 - 3 or low, medium, and high).



Epic: Epics are simply ways of gathering groups of user stories together into similar categories of functions that make logical sense. They help agile teams break down sequencing of development and are sometimes helpful in status reporting, but they aren't always necessary.

Backlog Creation

Using this scoring framework, the 580 Strategies team groomed the Product Backlog accordingly using the following process:

1. Build the Backlog with each user story as its own row
2. Compile Risks and Rewards for each story individually
3. Build a Risk/Reward Matrix according to their Priority Scores
4. Focus on the High Reward and Low Risk stories as candidates for Minimum Viable Product

Backlog with Priority Scores

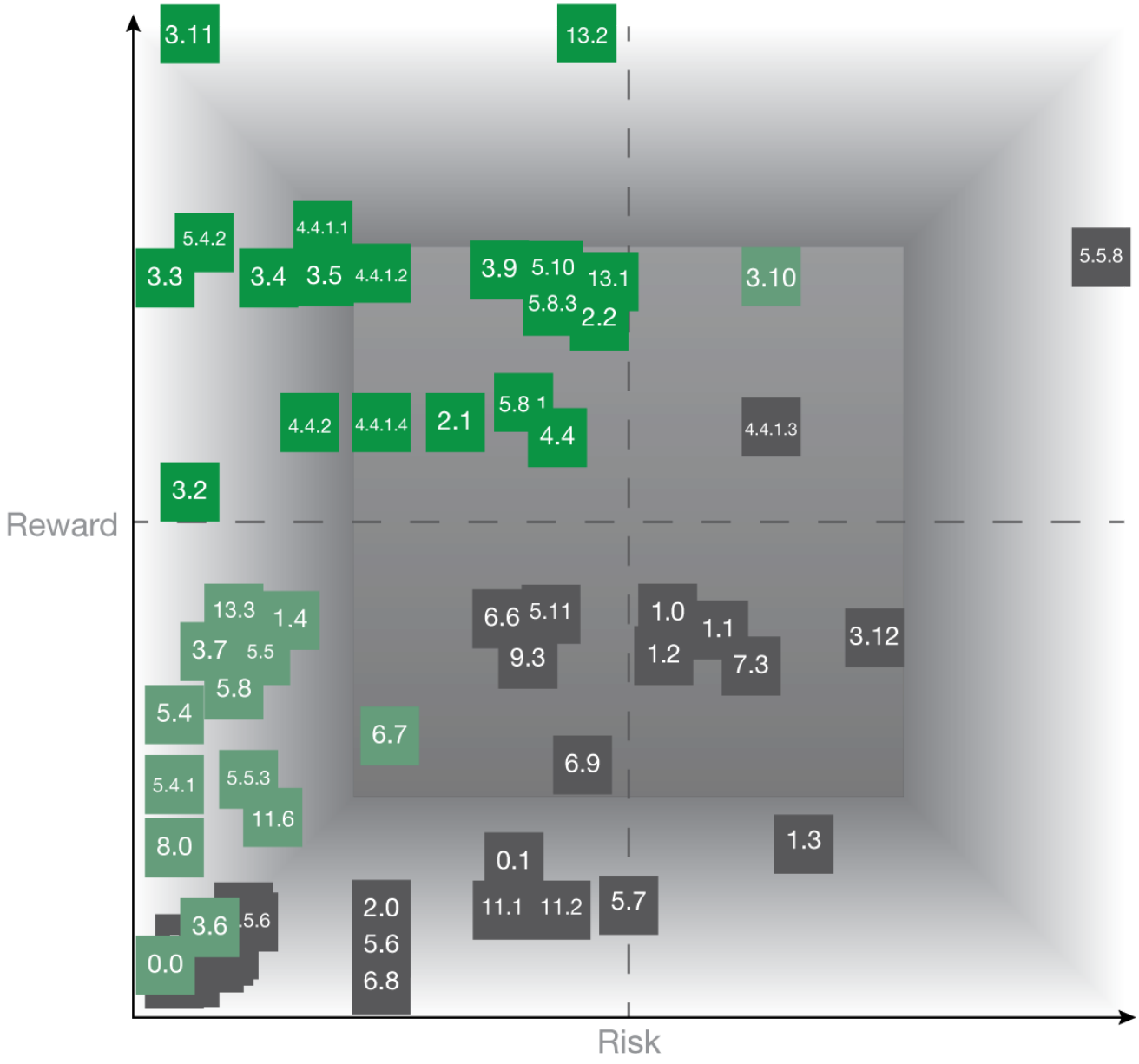
Below is a sample from the VA Mobile App Product Backlog. It illustrates each user story as a row with the Risk and Reward scoring and sizing framework as columns. The Priority Scores are used to plot the Risk and Reward Matrix as well as sort the stories by relatively priority:

ID #	Epic	Story	Risk				Reward				Risk Score	Reward Score	Priority Score	Size	Notes:
			R1	R2	R3	R4	R1	R2	R3	R4					
0.0	Launcher	As a veteran I want to see an app launch experience so I know the app is loading	0				1				0	0.25	0.00	S	Reward 1: Users know the app is alive when loading - 1 imp x 1 prob
3.11	Help	As a veteran, I want to get help from Veterans Crisis Line (Facetime)	1				15	15	10		0.25	10	0.03	S	Risk 1: Technical LOE of integrating with telephony - 1 imp x 1 prob Reward 1: Users can directly connect with Vets.gov - 5 imp x 3 prob Reward 2: Users can immediately speak with Crisis Line agents to receive help - 5 imp x 3 prob Reward 3: Face-to-face conversations provide a visually rich experience - 5 imp x 2 prob

3.3	Help	As a veteran, I want to get help from Veterans Crisis Line (Call)	1				15	15			0.25	7.5	0.03	S	Risk 1: Technical LOE of integrating with telephony - 1 imp x 1 prob Reward 1: Users can directly connect with Vets.gov - 5 imp x 3 prob Reward 2: Users can immediately speak with Crisis Line agents to receive help - 5 imp x 3 prob
3.2	Help	As a veteran, I want to call the Vets.gov Help Desk	1				15	6			0.25	5.25	0.05	S	Risk 1: Technical LOE of integrating with telephony - 1 imp x 1 prob Reward 1: Users can directly connect with Vets.gov - 5 imp x 3 prob Reward 2: Users can immediately get general help for VA services - 3 imp x 2 prob
5.4.2	Healthcare	As a veteran, I want to refill a prescription	2				15	15			0.5	7.5	0.07	M	Risk 1: Technical LOE of integrating content (to start requests) - 2 imp x 1 prob Reward 1: Veterans wanted access to their services (research observation) - 5 imp x 3 prob Reward 2: Being able to intiate/request services directly through the app - 5 imp x 3 prob
5.4	Healthcare	As a veteran, I want to view my prescription refills	1				12				0.25	3	0.08	M	Risk 1: Technical LOE of integrating content - 1 imp x 1 prob Reward 1: Veterans wanted access to their services (research observation) - 4 imp x 3 prob
5.4.1	Healthcare	As a veteran, I want to view my past prescriptions refills	1				9				0.25	2.25	0.11	M	Risk 1: Technical LOE of integrating content - 1 imp x 1 prob Reward 1: Veterans wanted access to their services (research observation) - 3 imp x 3 prob
3.4	Help	As a veteran, I want to get help from Veterans Crisis Line (Text)	4				15	15			1	7.5	0.13	M	Risk 1: Technical LOE of integrating with sms messaging functions - 2 imp x 2 prob Reward 1: Users can directly connect with Vets.gov - 5 imp x 3 prob Reward 2: Users can immediately text with Crisis Line agents for "asychronous" help - 5 imp x 3 prob

Risk and Reward Matrix

Utilizing the combined Risk and Reward scores to arrive at a Priority Score value, we then plot each story ID on a matrix with the relative risk on the X-axis and the relative reward on the Y-axis. This results in a rich visual representation of which user stories are likely the ones to focus on prioritizing from the upper-left quadrant (High Reward/Low Risk) for earlier build and/or inclusion in Minimum Viable Product.



The Risk and Reward Matrix plotted from the VA Mobile App Product Backlog

Feature and Use-Case Highlights

As the 580 Strategies team constructed the Product Backlog, we found many interesting use-cases and potential features from Vets.gov that can use native mobile device capabilities worth highlighting.

Get Help epic (3.11, 3.2, 3.3, 3.4, and 3.5):

Right away, the feature epic of getting help directly from the app stands out. With very little technical difficulty and a very high reward, these features can quickly take advantage of the app and mobile device capabilities to directly call, text, or chat with a Crisis Line agent. Unlike the web version where a veteran would be required to find a phone or utilize the web-based chat app, a mobile device version of these features will empower veterans to use these services and receive potentially life-saving help in just a click or two on their device.

Of special note is story 3.11 that would utilize the Facetime features to conduct a video chat quickly and directly. We feel the reward value on this story tops the list by adding valuable body language and additional context to conversations beyond what calls, texting, or chatting provide.

Search and Navigation epic (2.1 and 2.2):

Additional interesting stories are the searching and navigation stories. These features are necessary in order to streamline use and give the user a good sense of how to find the services within it. But for VA specifically, we feel using voice assistant technology such as *Siri* and *Google Assistant* for spoken word searching and navigation could be quite valuable. Not only will these technologies extend and enhance 508-accessibility compliance but they could also greatly assist disabled veterans who may have trouble manipulating mobile devices.

Then, beyond mere searching and navigating, we think exploring what translating the services and features themselves to a natural language user interface that would allow veterans to access and utilize app services entirely by voice-command could be a very powerful and worthy effort all on its own.

Facilities, Navigation, and Appointments (5.8, 11.6, 13.1, 13.2, and 13.3):

Finally, a major set of features for the app would simply be providing information about VA facilities (and even partner organizations). Immediately, this can manifest in two simple ways: compiling lists of facilities that are searchable, sortable, and able to be filtered by the user so they can quickly find the facility they're looking for and providing those facilities on an interactive map that utilizes the GPS functions of the mobile device to show the veteran the ones nearest to them.

Beyond just using the GPS for showing facilities near the user, we could also offer directions and navigation to those facilities using native device features.

Additionally, powerful design features could be provided here that take advantage of many data points across many systems. Veterans could easily look up the hours of operation for a facility, contact

information for a facility that could then use native features to call, or even initiate or change appointment details at a given facility.

Veteran ID Card (3.8):

Another interesting story leverages the feature of requesting a hard-copy Veteran ID card. But an interesting use-case here could be generating a soft-copy card as well and securely storing it within the app as a temporary value add that would provide immediate benefit to the veteran while they wait for the hard copy card to arrive.

My Files (3.9, 3.10, 4.4.1.2, and 4.4.1.4):

So much of what veterans do when interacting with the VA involves paperwork and forms. For these stories we feel that adding the option to securely store copies of frequently used artifacts could be an essential tool as well. This would give the veteran not only the ability to complete forms and access the end results of those workflows from the app but could also allow them to store ones they'd like to keep and access again in the future as an encrypted local copy.

Claim Status Tracking and Document Submission (4.4, 4.4.1.1, 4.4.1.2, 4.4.1.3 and 5.11):

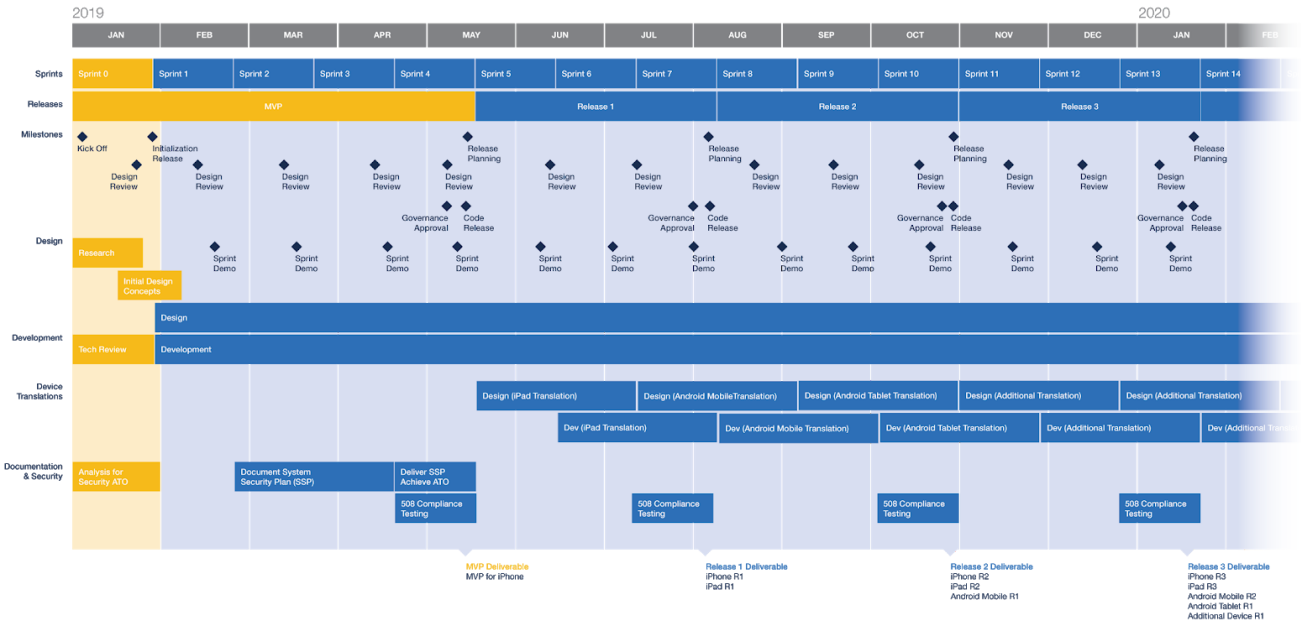
Feedback in both our analysis of Vets.gov and veteran interviews revealed that a set of helpful features would include the ability to initiate and track claims from within the app. Getting claim status is likely the simplest technological inclusion here, but initiating the claims from built-in forms would be an easy enough step. Then, another very valuable addition to the app would be integrating the camera and gallery features to allow veterans to snap photos of documentation or upload other files as evidence and supporting documentation for their claims. Not only would these features be value-add for the app itself, they could drastically reduce service times by allowing for soft-copy uploads of these required documents.

These are just a few of the interesting use cases and areas where mobile device features could be very powerful for enhancing the experience a veteran has when working with the VA. As the project grows and matures, we are confident the product owners and engineers will find even more ways to take advantage of these technologies beyond the initial ones noted for this engagement.

PRODUCT ROADMAP

The Product Roadmap builds upon all the preceding knowledge gained for this engagement and combines it into an actionable plan VA can use to begin development. The 580 Strategies team coupled the general capabilities and functions of the *React Native* technology stack with the desired features and services specific to Vets.gov into one contiguous timeline that conforms to agile development best practices within government.

We've taken the stories from the backlog and organized them roughly by priority into four major software releases over thirteen sprints and a 12-month project calendar. For ease of use, we began in January 2019 and assumed a four-week sprint duration. For capacity estimation purposes, we assumed a team of five that includes two developers, a ScrumMaster, a designer, and a business analyst/tester. The team in this scenario can product approximately 30-35 story points each sprint.



The Product Roadmap organizes work into four releases across thirteen sprints over a calendar year

Roadmap Highlights

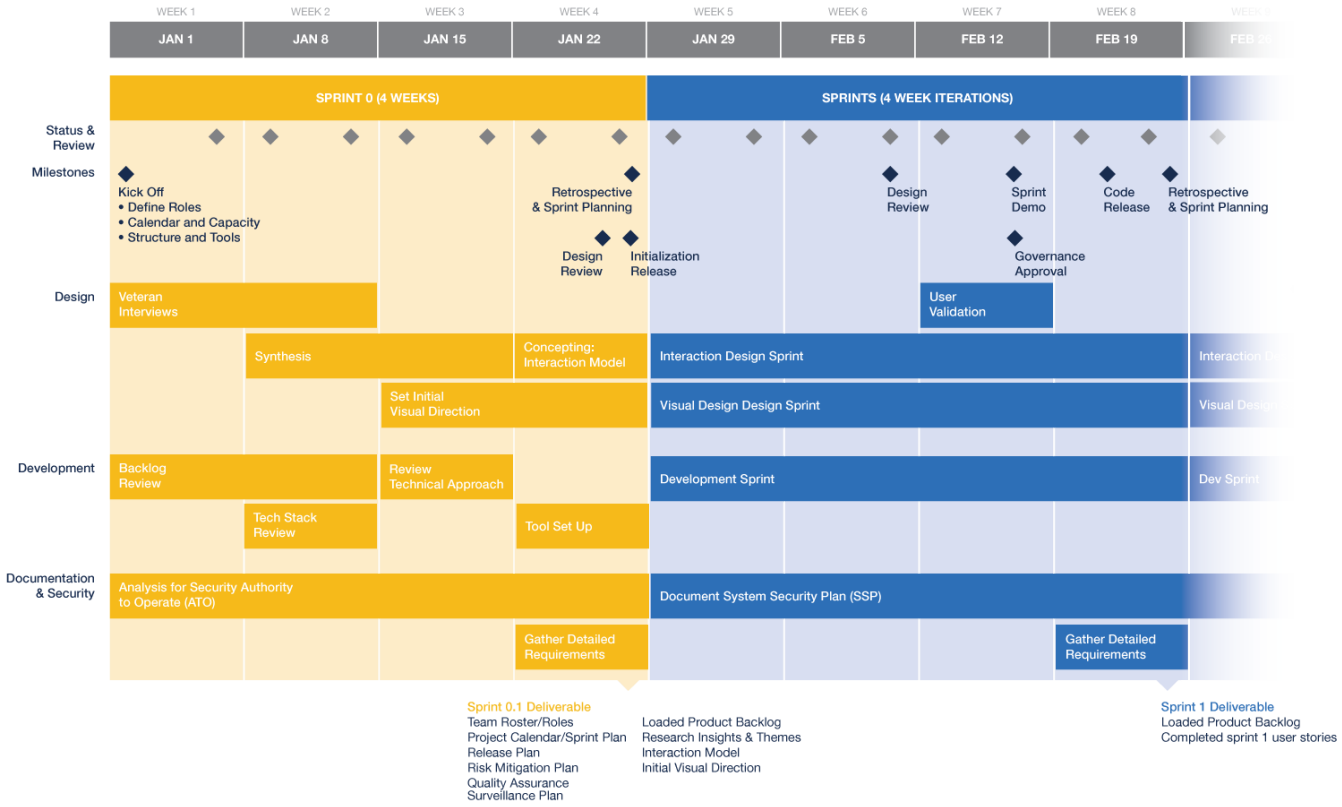
The roadmap also takes several other items into consideration, including:

- Getting an immediate start on design work and user feedback in Sprint 0
- Identifying security and 508 compliance requirements in Sprint 0
- Achieving Minimum Viable Product (MVP) for iPhone in four sprints of development
- Developing the system security plan and achieving Authority to Operate in parallel with MVP
- Three additional full releases within the first year
- Staggered design sprints preceding each translation sprint
- Translations sprints iPad, Android Mobile, Android Tablet, and one additional translation to be determined later
- 508-compliance testing factored in before each major release

- Approximately one week of schedule cushion in December for transition to Operations and Maintenance mode or additional work in preparation for a second year of development

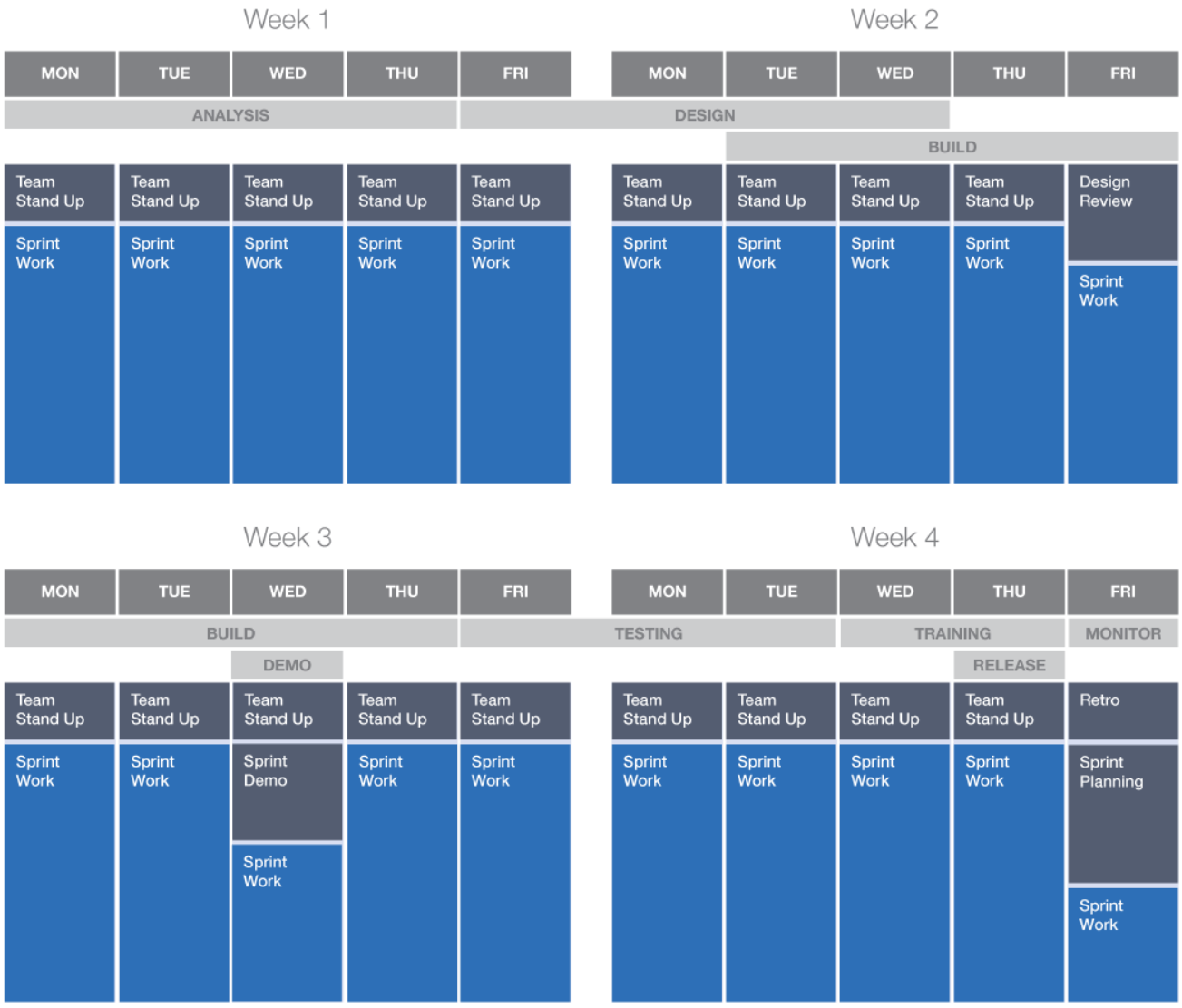
Sprint 0 into Sprint 1

We begin with a project kickoff and Sprint 0 to set the team up for successful iterations throughout the remainder of the year. Sprint 0 also includes immediately starting on design elements and soliciting user feedback as well as analyzing the requirements for security and 508 compliance to launch the application.



Sprint 0 organizes the team, configures tools, and produces initial project deliverables

After the conclusion of Sprint 0, the team will move into Sprints 1 through 12 in a structured manner that conforms to roughly the same process each time as seen below.



A typical 4-week sprint cycle with associated, repeatable milestones and phases.

During Sprint 0, the roadmap can be adjusted to account for VA specific holiday, business priorities, or major milestones that may not have been considered during this exercise. Then, each subsequent sprint planning meeting can account for inevitable change in terms of business priorities and as estimated levels of effort change across both engineering and design team analysis.

Minimum Viable Product

Utilizing the Risk and Reward Matrix, we selected the low-risk and high-reward stories that we thought made sense to produce a functional, useful, and desirable Minimum Viable Product (MVP). These 19 user stories totaled 131 points of work and will take an estimated four sprints to complete.

ID #	Story	Points
Sprint 1		34

2.1	As a veteran, I want to use Siri to find information and features within the app	8
2.2	As a veteran, I want to navigate with a natural language UI (chat-like)	13
3.2	As a veteran, I want to call the Vets.gov Help Desk	1
3.3	As a veteran, I want to get help from Veterans Crisis Line (Call)	5
3.4	As a veteran, I want to get help from Veterans Crisis Line (Text)	9
3.5	As a veteran, I want to get help from Veterans Crisis Line (Chat)	1
Sprint 2		33
3.9	As a veteran, I want to download my data locally and have easy access to "My Files"	9
3.11	As a veteran, I want to get help from Veterans Crisis Line (Facetime)	1
4.4	As a veteran, I want to track claims and appeals	5
4.4.1.1	As a veteran, I want to view files associated with a claim/appeal	9
4.4.1.2	As a veteran, I want to upload new files/evidence from my device	9
Sprint 3		32
4.4.1.4	As a veteran, I want to save claim files to My Files	9
4.4.2	As a veteran, I want to see the details of my claim/appeal	5
5.4.2	As a veteran, I want to refill a prescription	5
5.8.1	As a veteran, I want to automatically sync my appointments with my device calendar	13
Sprint 4		32
5.8.3	As a veteran, I want to schedule a VA Appointment	9
5.10	As a veteran, I want to download my VA Letters to My Files	5
13.1	As a veteran, I want to locate facilities and services	9
13.2	As a veteran, I want to locate facilities and Services on a map	9
	Total Story Points	**131**

This delineation of Minimum Viable Product pulls to the front of active development some key stories that were identified as priorities through analysis and veteran interviews. Our target MVP will produce an app that allows veterans to access all aspects of the Crisis Line, navigate the app using voice-assistant technology, locate VA facilities on a map, track appeals status and upload documents

to claims, download important files to My Files, schedule an appointment, and track a prescription - all with a ship-date of mid-May.

While this is our best estimate of Minimum Viable Product given our feedback from veterans and reward scores from analysis, it is obviously up to the formal product owners to accept and act upon. There may be more important infrastructure and technology groundwork to be completed upfront such as two-factor authentication and integration with all VA identity providers or one of the major lines of business such as healthcare prioritized over another. True MVP will be a combination of our best collective guess at what to put forward in an app offering first, constrained by the realities of integrating with large enterprise systems like the VA's. This MVP is recommended merely as a starting point to begin development with were no other factors a consideration.

After MVP is achieved, the team can continue to iterate on features and functions both large and small across the remainder of the calendar year. The roadmap produced assumes the team will focus on three more major releases with associated design, translation, and testing required. But other minor releases could be introduced as well that might allow for value-added features to be released sooner than expected.

Summary

These project deliverables provide an ample starting point for the Department that can be refined and developed into greater detail. Our intent is to provide, first and foremost, an actionable framework and development approach VA can use to execute this work but also as much detailed information we could uncover for features and services specific to the Departement. We are confident this micropurchase project has set VA up for success with a scalable framework and enough starting information should you choose to move forward with the project and begin development of the mobile app.

PROJECT TEAM

Rusty D. Pickens -Agile for Government Expert

OVERVIEW / Rusty is the Founder and Principal of 580 Strategies. He is the former Senior Advisor for Digital Platforms at the U.S. Department of State, and former Acting Director for New Media Technologies at the White House, where he led teams who operated cloud platforms for the Obama Administration to increase public engagement, improve user experience, enhance staff productivity, and heighten security posture. During this time, Rusty created new systems for and built new teams to lead Whitehouse.gov, the White House email outreach services, the Presidential correspondence system, the We The People petitions system, the White House Appointment Center, and the U.S. Embassy contact management systems.

Rusty's two decades of leadership experience aligning organizational vision with technology strategy across top federal agencies and start-up environments included the Federal Salesforce Community of Excellence, the U.S. Small Business Administration, the 2009 Presidential Inaugural Committee, Obama for America 2008, and the Chickasaw Nation of Oklahoma. He currently advises clients on unlocking the potential of cloud computing and agile software delivery to vastly improve their digital presence and citizen experience.

Carl Seiber -Human Centered Designer

OVERVIEW / Carl is a process-driven, user-centered, multidisciplinary designer with experience creating design systems, products, services, apps, mobile experiences, websites (intranet and extranet), rich applications, touchscreen kiosks, and all sorts of deliverables.

He is adept in all phases of client engagements including: sales, planning, ethnographic research, collaborative workshops, usability testing, and stewardship (fully integrated agile or other methodologies).

EXPERTISE / Illustrator | Photoshop | Sketch | InDesign | HTML/CSS | JavaScript | Zeplin/UXPin | Axure

Hiren Dhaduk -Simform Chief Technology Officer

OVERVIEW / Hiren is a versatile analytics leader with extensive experience in helping enterprises streamline their business performance through data-driven innovation and help them turn their data into the asset it should be. Proficient at manipulation and analysis of complex, high-volume data from multiple sources. Skilled at qualitative problem solving. Adept at communicating with executives, partners, and end-users.

Enabled Analytics Transformation for multiple organizations over last 8+ years. Diverse background including software engineering, business intelligence, data warehousing, analytics, SaaS platforms and IOT.

Specialties: Business Intelligence, Predictive Analytics, Data modeling, Data visualization, Data warehouse architecture, Big Data, Financial forecasting, User behavior analysis.

EXPERTISE / App Development | Web Development | BI | Predictive Analytics | HTML/CSS | JavaScript | Zeplin | Axure | Azure | Enterprise Mobility