CASE STUDY

XAMARIN APPLICATION

KEYHOLE SOFTWARE

OBJECTIVE

The main objective of this project was to modernize an existing web-based mobile application, converting it to Native iOS and Android. In addition, eliminating a costly Commercial-Off-the-Shelf (COTS) SaaS application, by integrating this with an in-house, custom developed solution as part of the new Native App.

TECHNOLOGIES

.Net, C#, XAML, Xamarin

Case Study: Xamarin Mobile Application

Client

A nationwide company providing business technology services across the United States. The client partners with its customers across all verticals to conceive, design, install, and maintain business technology solutions.

Project Objectives

The client had several goals for the successful completion of this project.

- Convert an existing web-based mobile application to a native mobile application
- Present a modern look and feel by improving the UI / UX experience
- Eliminate the use of a Commercial-Off-the-Shelf (COTS) application and integrate all required functionality into the new application
- Allow the new application to be fully operational on both iOS and Android platforms
- Integrate the client team into the development process to ensure its members possess the knowledge and skills to maintain the application moving forward

The client's existing application was a mobile-enabled web application that served its purpose but had several limitations and usability issues that continued to grow over time. The COTS application was full-featured and functional but contained many features that were no longer needed. The client wanted to combine two separate applications they were using for a single-source administration setup. They also requested that only one mobile application be used for completing work in the field.

Additionally, the project created a completely new UI experience that was custom to the business's needs and integrated the functionality needed from the COTS application into the new native mobile application.

Project Result

Keyhole Software designed and built a Xamarin enterprise mobile application to specification of the client. In this single, new application, Keyhole Consultants integrated all needed business functionality from multiple disparate applications including an existing mobile-enabled web application and expensive COTS application.

Technical Snapshot

Xamarin.Forms was the primary technology used in this project. Xamarin.Forms is a .NET technology used for building cross-platform mobile applications. The mobile application consumes RESTful APIs which were developed by the client.

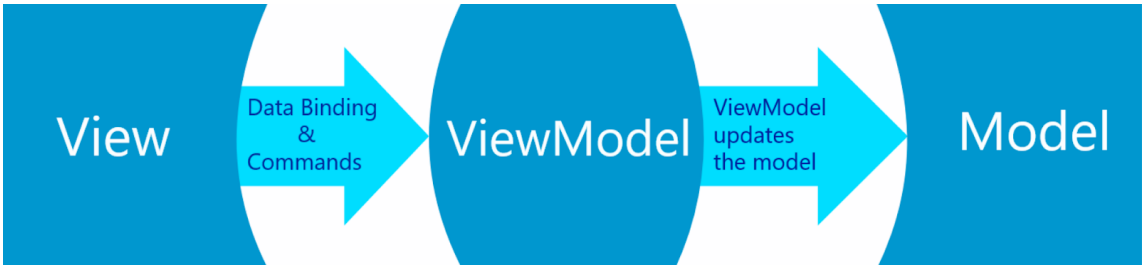
Project Architectural Patterns

While building this enterprise mobile application with Xamarin.Forms, Keyhole Consultants established an architecture that promoted code reuse, testability, and maintainability. The team used Model-View-View-Model (MVVM), loose-coupling through messaging patterns, and dependency injection.

MODEL-VIEW-VIEWMODEL (MVVM)

The Model-View-ViewModel (MVVM) pattern helps to cleanly separate the business and presentation logic of an application from its user interface (UI).

Maintaining a clean separation between application logic and the UI helps to address numerous development issues and can make an application easier to test, maintain, and evolve. It can also greatly improve code reuse opportunities that allow developers and UI designers to easily collaborate when developing their respective parts of an application.



On this project, one of the requirements was to list the job card with job-related information. In the image above, View represents the UI layout and appearance of what the user sees on the screen. View is defined in XAML, with a limited code-behind that does not contain business logic. ViewModel implements properties and commands so the View can data bind to, and notify the View of, any state changes through change notification events. ViewModel provides data from a Model in a form that the View can easily consume.

Model is a non-visual class that encapsulates the application data, and usually includes a data model along with business and validation logic. Model class is used in conjunction with services that encapsulate data access and caching.

DEPENDENCY INJECTION

The Autofac NuGet package was used to register and manage all dependency injection throughout the project. Autofac supports both constructor injection and method call injection. Dependency injection is a specialized version of the Inversion of Control (IoC) pattern, where the concern being inverted is the process of obtaining the required dependency. With dependency injection, another class is responsible for injecting dependencies into an object at runtime.

COMMUNICATION BETWEEN LOOSELY-COUPLED COMPONENTS: MESSAGINGCENTER

This project utilized the Xamarin.Forms MessagingCenter class that implements the publish-subscribe pattern. It allows message-based communication between components that are inconvenient to link by object and type references. This mechanism allows publishers and subscribers to communicate without having a reference to each other. This helps to reduce dependencies between components, while also allowing components to be independently developed and tested.

Use of NuGet

NuGet is a free and open-source package manager designed for the Microsoft development platform. Package managers, such as NuGet, make it simple for developers to integrate third-party, open-source code into applications. While Xamarin.Forms provided the basics, the application made extensive use of many available NuGet packages like Acr.UserDialogs, Akavache, Xamarin.Essentials, Xamarin.Plugin.FilePicker.

AUTOFAC

Autofac is an IOC container for .NET. It manages the dependencies between classes so that applications are easy to change as they grow in size and complexity.

AKAVACHE

Akavache provided caching. Caching strategies were used extensively throughout the application, allowing for fewer API calls and faster load times.

ACR.USERDIALOGS

Acr.UserDialogs is a cross-platform library that allows you to call for standard user dialogs from a shared or portable library. In this case, it provided the modal dialogs and alerts used throughout the application such as “Are you sure?” and “Save successful!”

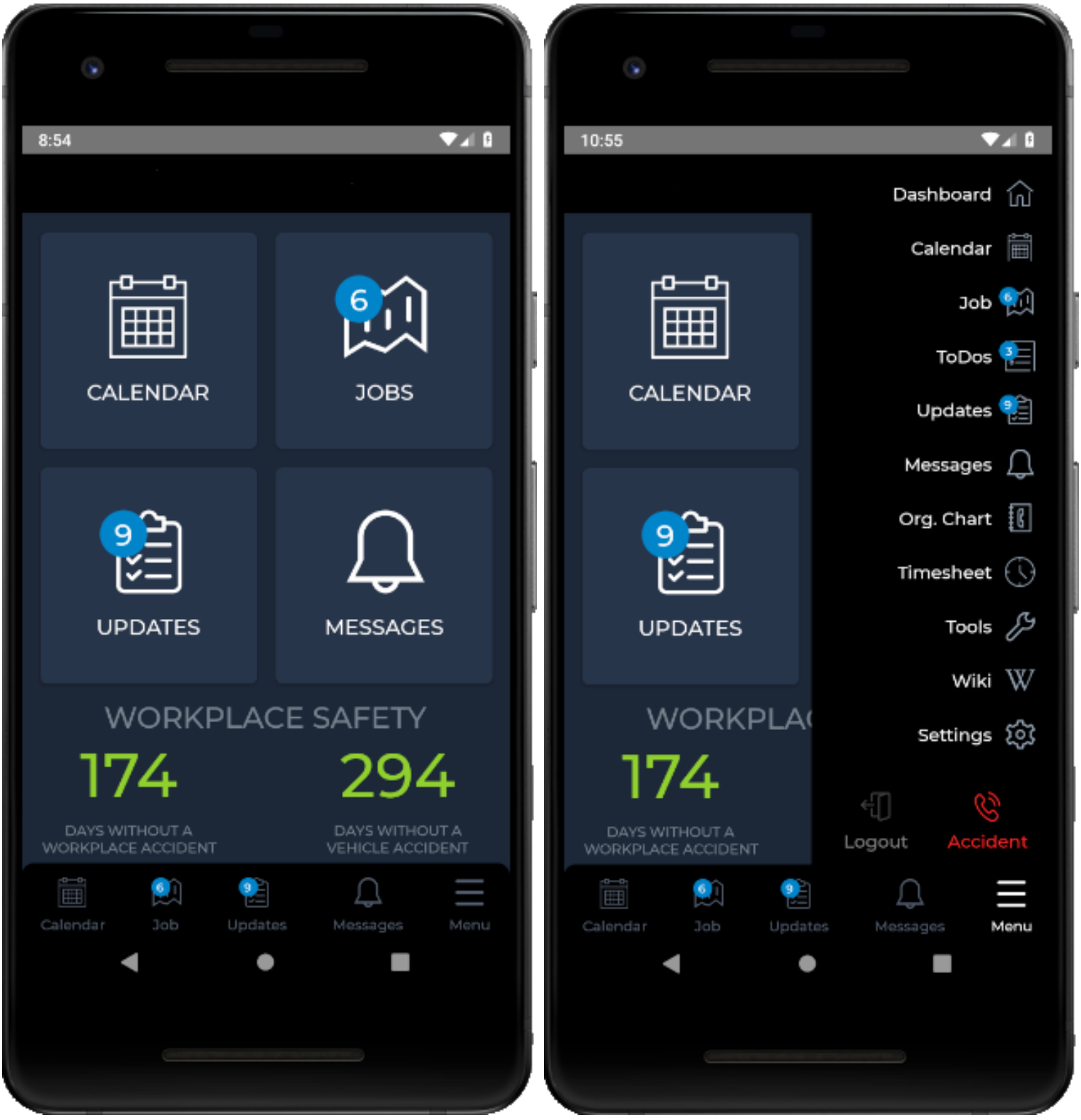
XAMARIN ESSENTIALS

Features of the Xamarin.Essentials add-on package were also used. Xamarin.Essentials provides developers with cross-platform APIs for their mobile applications. This application made use of Geolocation, Map, Phone Dialer, and many more.

XAMARIN.PLUGIN.FILEPICKER

One of the key project requirements was the ability to upload files to the server. Keyhole leveraged Xamarin.Plugin.FilePicker for this functionality. It provides the ability to browse one’s mobile device and select files or photos to upload.

Application Screenshots



Case Study: Xamarin Mobile Application

Project Scope

- Develop a mobile application to replace an existing mobile website with a native mobile application and integrating the required functionality previously contained in a disparate COTS application.
- Existing features included:
  - Labor entries (timesheet & expense reports)
  - Field technician check-in/check-out to a job site
  - Ability to take photos and upload files
- The new mobile application would have an offline mode feature to allow field technicians the ability to use the application without mobile or wi-fi connection. Offline mode data would be synced with the client's servers once a connection was re-established.
- The mobile application would also include a color-coded calendar view to indicate job types and statuses.
- A knowledge transfer between Keyhole Software and the client's development team to ensure the client's IT team could continue without Keyhole's assistance.

Unique Features

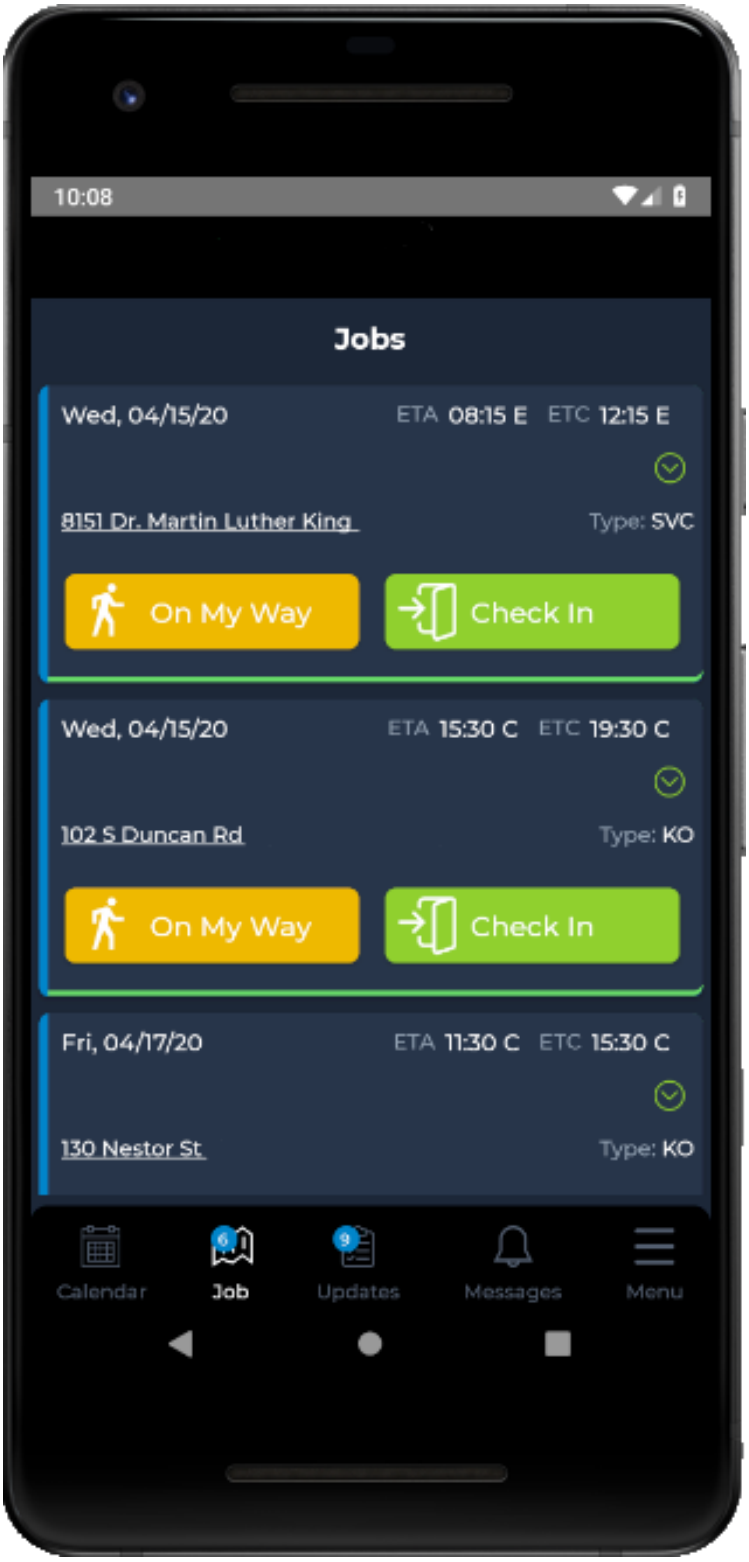
Early in the design process, the team decided on a standard look and feel across the application. A few features of the application included:

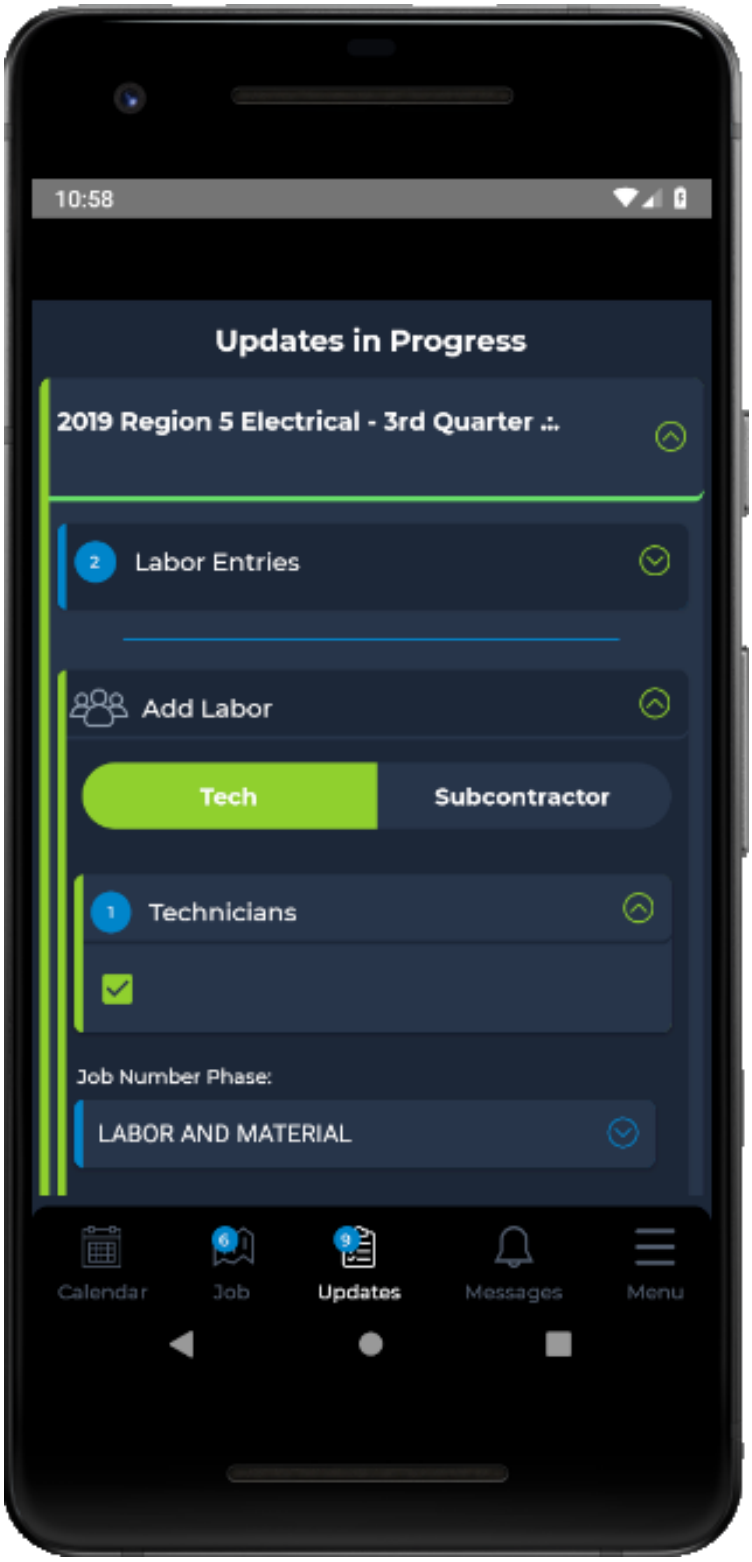
- A tile-based dashboard and "tile" based operation
- A fixed navigation bar across the bottom of the application in order to navigate to main functions at any time and from any location within the application
- An expanded hamburger menu (in this case, bottom right) which would allow the user to get anywhere they needed regardless of where they were in the application
- Color-coded "Expanders" and "Nested Expanders" were used to provide visual orientation within the application and rules were established to provide consistency and clarity.

  For example, if a list of expanders was presented, only one expander could be open at any time. Clicking on one expander would collapse any other expander that was open.

- All addresses were clickable to integrate with Google Maps™
- All phone numbers were click-to-dial with the mobile device
- Each display page had a quick-dial footer to call in case of an accident or emergency

Application Screenshots





Unique Features Continued

This mobile application was specifically made with custom created controls. One of these custom controls includes a nested expander frame that is the signature look and feel of the application itself and mostly implemented on populating the list of jobs, job updates, labor entry pages, etc.

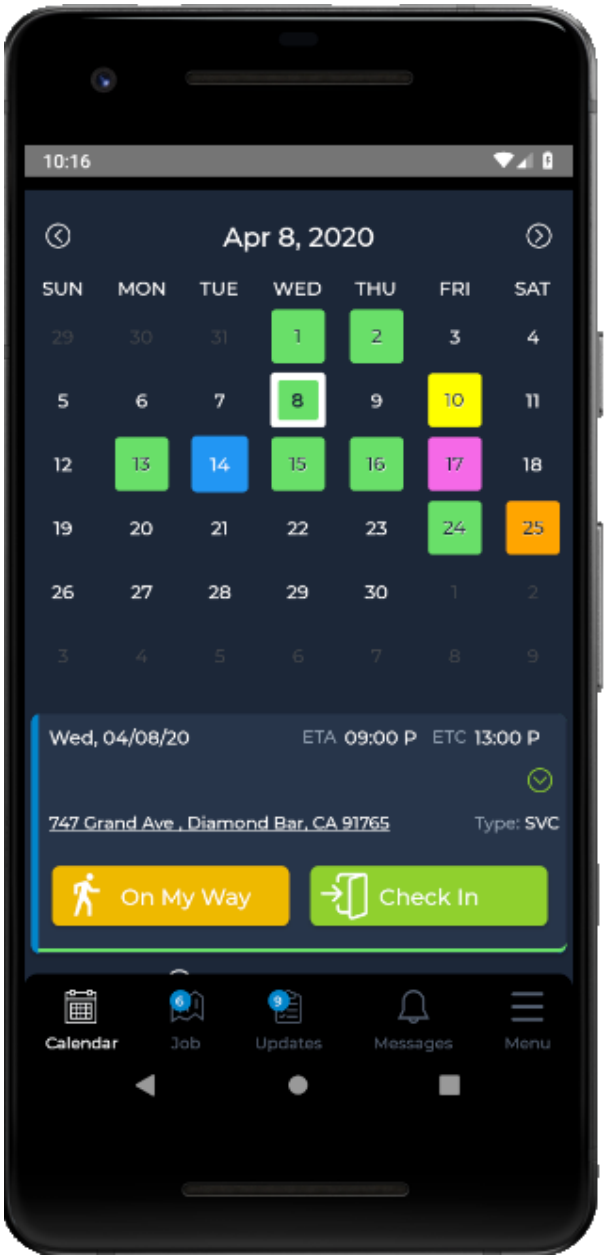
The depth of the nested expanders was limited to three levels. The left expander bar was color-coded to indicate its status as being expanded or collapsed. The calendar control was customized to allow for color coding according to event type, based on a prioritized list of types. Changing days would allow showing the events for that day in a prioritized list below the calendar.

The COTS application, a separate mobile application originally used by the client, was a full-featured standalone application. The client needed a subset of the functionality of that application and wanted to combine all required features into a single application. This would create a better user experience and remove the annual licensing fee associated with the COTS application.

This application allowed the client to create a custom set of data entries, called a “survey” specifically for each of their customers based on their needs.

Within a survey, a couple of key features needed to be created.

- First, the ability to do looping over a field, grouping of fields, or full screen of fields.
- And second, to allow conditional logic to present a field, group of fields, or full screen of fields, based on a value entered in another part of the application.



Application Screenshot

Conditionals and Loops

A conditional question is one whose visibility is determined by the answer to another question. For example:

- **Question 1:**
  - Should the next question be visible? Yes/No
- **Question 2:**
  - I am visible if the answer to Question 1 is "Yes"

Categories can be conditional as well. Looping is the dynamic addition of a category of questions within a survey. Thinking of the folder/file hierarchy in your computer's file system may help to visualize.

For example, let's say a particular job site has two rooms. The technician filling out the survey needs to answer the same set of questions about each of the rooms. Each room would then become its own category dynamically added to the screen as the technician completes the survey.

Room 1	Room 2
Question 1 Question 2 Question 3	Question 1 Question 2 Question 3

This looping structure can be nested another level to look something like this.

- **Room 1**
  - Question 1
    - Fax Machine
      - Question 1
      - Question 2
      - Question 3
  - Question 2
  - Question 3
- **Room 2**
  - Question 1
    - Fax Machine
      - Question 1
      - Question 2
      - Question 3
    - Computer
      - Question 1
      - Question 2
      - Question 3...
- **...Room 2 Continued**
    - Printer
      - Question 1
      - Question 2
      - Question 3
  - Question 2
  - Question 3

Looping and conditionals can also be combined, which presented unique programming challenges.

Field Entry Validation

Validation was handled by a Nuget package called MVVM Validation. Form entry validation was as simple as adding the package, creating a base class, then adding validation rules in the constructor of the derived classes. The base class was named ValidatableViewModelBase. View models requiring validation simply derived from this class.

A label was then added to each view requiring validation. Validation error labels were bound to the validation errors collection in ValidatableViewModelBase. Each validation error in the collection keyed off the property it validated so that the UI could display the correct error messages next to the fields being validated. Back-end validation was handled by the client APIs.

Security

The application authenticates via the client's Identity Server. Identity Server is based on OpenID Connect and OAuth 2.0 specification. OAuth 2.0 is a protocol that allows applications to request access tokens from a security token service and use them to communicate with APIs.

This delegation reduces complexity in both the client applications as well as the APIs since authentication and authorization can be centralized. This application is configured to authenticate and get authorization code after successfully authenticating, which can be further used to communicate with APIs.

App Store Deployment

One of the unique challenges of mobile application development is delivering the application to the end user. For this particular project, that involved pushing the application to both the Apple App Store and the Google Play Store.

This process involves obtaining keys, setting up the configuration, signing the application, creating a package, and finally, pushing the package to its respective store for the end-user to download.

While the steps are similar for both Android and iOS, the process is a bit different for both. Each store has its own process for preparing and pushing to a particular environment and the application itself needs to be configured correctly for each environment, whether it be testing, UAT, or production.

Conclusion

Keyhole Software team designed and built a Xamarin enterprise mobile application to specification. In this single, new application, Keyhole Consultants integrated all needed business functionality from multiple disparate applications including an existing mobile-enabled web application and expensive COTS application.

The development process utilized both standard Xamarin features and controls as well as several custom developed controls to provide the solution for this client. The new application provided an entirely new look, feel, and standardized operations using Xamarin technology.

The mobile application was deployed to both iOS and Android application stores. Additionally, Keyhole Software Consultants provided knowledge transfer to the client development staff in an effort to ensure they could maintain the application without Keyhole's assistance once the project concluded.