VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Yonas Azmera Gindo

WORKFORCE-
A XAMARIN MOBILE APPLICATION FOR HOMECARE SERVICE

Information and Technology
2017

ACKNOWLEDGMENT

I would like to begin by expressing my sincere gratitude to my thesis supervisor Timo Kankaanpää for providing me with an opportunity to work on an interesting project. I would also like to thank him for his guidance and motivation throughout the process.

I would also like to take this opportunity to express my appreciation to the entire VAMK staff for having me acquired with all the necessary knowledge and to their continuous support during my studies.

A special thanks to my wife and best friend, Milka Gindo, the completion of this thesis and my degree would have not been possible without her continuous support.

VAASAN AMMATTIKORKEAKOULU
(UNIVERSITY OF APPLIED SCIENCES)
Degree Program in Information Technology

ABSTRACT

Author	Yonas Gindo
Title	Workforce - A Xamarin Mobile Application for Homecare Service
Year	2017
Languages	English
Page	48
Supervisor Name	Timo Kankaanpää

In the past decade, the introduction of smart phones has revolutionized the way we live very much. Mobile applications have become an integral part of our daily lives by making numerous services available at our finger tips.

In this thesis, an Android application was designed and implemented to improve homecare service sector. Although popular in Finland and in other developed countries, the homecare service sector is yet to benefit from today's software solutions. The efficiency and effectiveness of this service sector heavily relies on establishing a smooth communication network between task allocators and field workers (healthcare professionals). The application is intended to be used by field workers; it displays job offers, provides necessary features, sends and receives data to and from the back-end solution, thereby creating a centralized and well-organized homecare service network. The mobile application together with Azure Mobile Apps back-end cut out the unnecessary time, effort and finance wasted on traditional manner of carrying out the service.

Although the project was limited to Android mobile application, it keeps Xamarin cross-platform project structure to make it ready for future updates. The project integrates Google's Material design guidelines to deliver user-friendly and responsive UI. Android Support Library was used to ensure backward compatibility. Azure Mobile Apps was implemented as a server-less back-end for data storage and Google authentication.

This document provides a detailed documentation of the application's use cases, design and implementations carried out throughout the project.

Keywords Xamarin, Material design, Android Support Library, Microsoft Azure

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBRIVATIONS

AOT	Ahead of Time
APP	Application
API	Application Programming Interface
BCL	Base Class Library
CSS	Cascading Stylesheet
IaaS	Infrastructure as a Service
IDP	Identity Provider
IDE	Integrated Development Environment
HTML	Hypertext Markup Language
JIT	Just in Time
PaaS	Platform as a Service
PCL	Portable Class Library
REST	Representational State Transfer
SaaS	Software as a Service
SDK	Software Development Kit
UI	User Interface
VS2015	Visual Studio 2015

1 INTRODUCTION

The studies conducted in this project and the experience derived from it were used to tackle and bring about a software solution to one of healthcare service sectors, namely homecare service. Workforce is an Android application built using Xamarin Platform and it consumes a server-less Azure Mobile Apps back-end. The application targets professionals in the healthcare sector (doctors, nurses etc) and is used to handle all the communication between allocators and these professionals in homecare service.

Homecare service has been around for a long time, but recently this service sector is becoming more and more popular in the developed countries like Finland due to the elderly population increase, which will eventually result in high homecare service demand.

It is a necessity that the traditional home to home healthcare service should be improved by innovating technological solutions to address these problems and to provide an optimized service. Workforce addresses this issue and aims to revolutionize homecare services, it emphasises on the need to create a seamless communication system between allocators and field workers (doctors, nurses), which is usually easily overlooked but vital in guaranteeing an effective and efficient service.

1.1 Constraints

The initial agreement with the client company was to build a cross-platform mobile application solution using Xamarin platform. While the need to use Xamarin platform was to build a cross-platform mobile application, building a cross platform application would require more time and, obviously, a team of developers.

The fact that only a single developer was available has made it difficult to carry on the initial agreement. Due to these constraints, the project’s scope has been reduced to just an Android Xamarin project, which still leaves room for future cross-platform updates.

2 TECHNOLOGIES

This chapter will cover the technologies, frameworks and tools used to design and implement the application. As briefly mentioned in the introduction of this document, Workforce is developed using Xamarin platform and it consumes a server-less Azure back-end.

Xamarin platform enables developers to build both native and cross-platform mobile applications using C#. This chapter will introduce Xamarin platform. Moreover, it will also discuss how it stands out from other similar technologies.

Microsoft Azure offers countless cloud services, among those Azure Mobile Apps. Azure Mobile Apps is a Platform as a Service (PaaS) and brings rich capabilities to mobile applications. Workforce integrates key mobile features like cloud data storage and OAuth 2.0 provided by Azure Mobile Apps.

2.1 Xamarin Platform

Xamarin was first founded in 2011 by a Mexican software engineer, Miguel de Lcaza, and an American programmer, Nat Friedman, and was later owned by Microsoft in 2016 /1/. Xamarin platform is among the products offered by Xamarin; it is a framework for building cross-platform as well as platform specific mobile apps (Android, iOS).

Visual Studio 2015 (VS2015) comes already bundled with Xamarin platform making it easily available for C# developers. It can also be downloaded from the company's official site, in addition to Xamarin platform; the universal installer from Xamarin website bundles the necessary tools and the official Xamarin IDE; Xamarin Studio.

When it comes to cross-platform mobile development, Xamarin platform overshadows alternative technologies in the following core features and functionalities.

- **Native User Interfaces**

  One of the challenges in developing cross-platform mobile applications is to keep native UI look and behaviour across each platform, which it vital to delivering better user experience. Cross-platform mobile applications built with most alternative technologies fail to address this issue, therefore look, feel and behave the same across

all platforms. Xamarin is unique as it provides a native UI feel and behaviour across each platform.

- **Native API Access**

  Xamarin applications (both cross-platform and platform specific) have 100% access to both Android and iOS native SDKs. This enables developers to take advantage of platform specific APIs and functionalities.

- **Native Performance**

  Most alternative technologies (Phone Gap, Accelerator etc) employ runtime interpreters to compile code to native assemblies, which significantly affects performance. Xamarin compiler produces native iOS and Android assembly languages ensuring Xamarin apps a native performance; the Xamarin AOT (Ahead-of-Time) compiler compiles Xamarin.iOS apps directly to native ARM assembly code whereas Xamarin.Android apps are compiled down to native Android APKs by JIT (Just-in-Time) compiler /2/.

- **Single Language; C#**

  Another unique feature of Xamarin is that it does all the things discussed above by using a single language; C#. This eliminates the need to learn or use many languages (Objective C, Java, HTML, CSS, JavaScript etc) that are usually required by alternative technologies for achieving the same results that Xamarin can with a single language.

Xamarin Platform mainly provides two frameworks: Xamarin.iOS and Xamarin.Android; both are built on top of Mono (open source version of .Net framework) which runs almost on every device and OS. Xamarin applications are built against a subset of .NET BCL known as Xamarin Mobile Profile which is responsible for mobile applications. Xamarin Mobile Profile ships Mono.Android.dll (for Android) and MonoTouch.dll (for iOS) providing developers have the possibility to develop both native or cross-platform mobile applications using C# /2/.

2.1.1 Xamarin Cross-platform

Xamarin cross-platform development offers an environment for developing iOS, Android and Win phone mobile applications that share a single logic code base. Cross-platform applications in Xamarin can be developed by using Xamarin.Forms and Native cross-platform approach. Both approaches can be implemented as shared projects or PCL projects.

The VS2015 snap shot picture below (Figure 1) shows the different types of project choices for cross-platform Xamarin development.

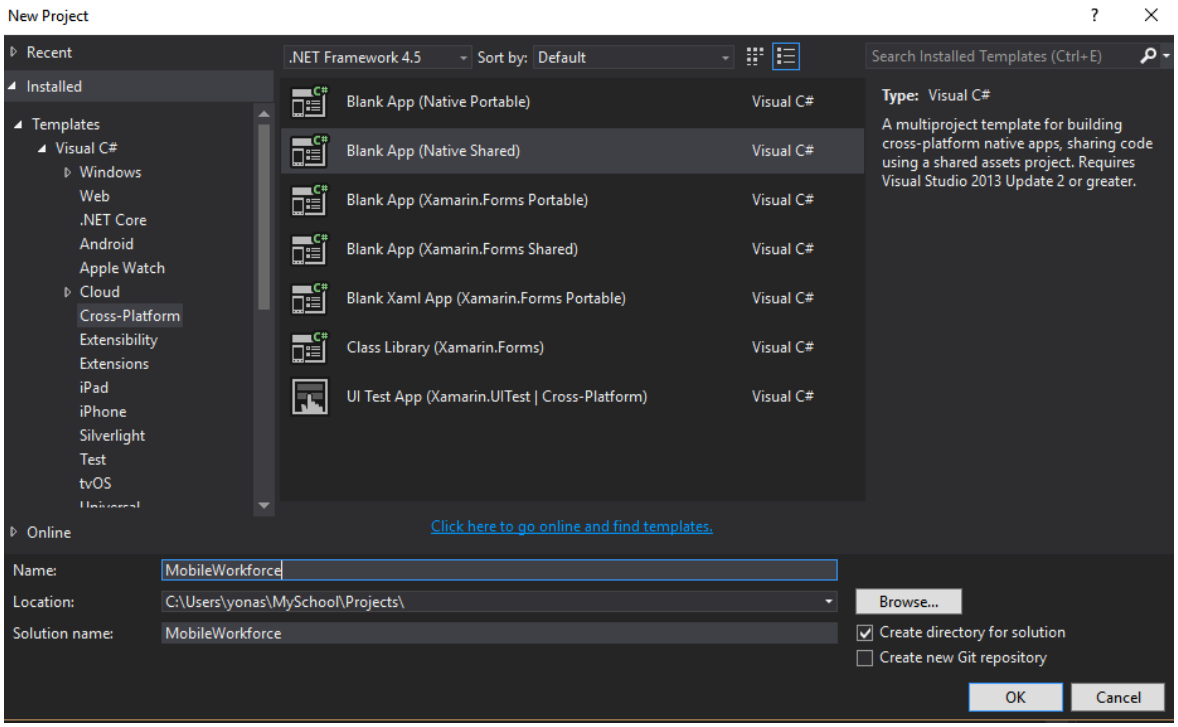


Figure 1. Xamarin Cross-Platform development choices (Visual Studio 2015)

Xamarin.Forms is a magnificent cross-platform UI toolkit. What makes Xamarin.Forms fascinating is that it allows developers to create native UIs that can be shared across different platforms in addition to the usual logic code sharing technique.

Native cross-platform approach also employs sharing of a single logic codebase, but unlike Xamarin.Forms writes UI code inside the platform specific projects. As of this document writing, applications built with Xamarin.Forms can share more than 96% of code whereas the native approach guarantees up to 75% of code sharing across platforms /3/.

Even though it is obvious that Xamarin.Forms is a better option it is, however, still in beta version and has a lot of bugs. In addition, it is not recommended for projects that require platform specific API implementations. As it stands now, Xamarin.Forms is best suited for applications that require little access to platform specific APIs and in cases where code sharing is an important criterion.

Workforce, even with the limited scope to Android platform only, it is already mentioned that an update to cross-platform development is a must. Taking that into consideration and the fact that the project requires access to platform specific features like maps, calls, messaging and more, it is developed in a way to make it ready for future updates using Native Xamarin cross-platform approach (PCL project).

2.1.2 Xamarin Native (Platform specific)

In addition to cross-platform applications, developers also have an option to develop platform-specific apps. As discussed in the previous sections, Xamarin platform ships Mono.Android.dll (for Android development) and MonoTouch.dll (for iOS development) using only C#.

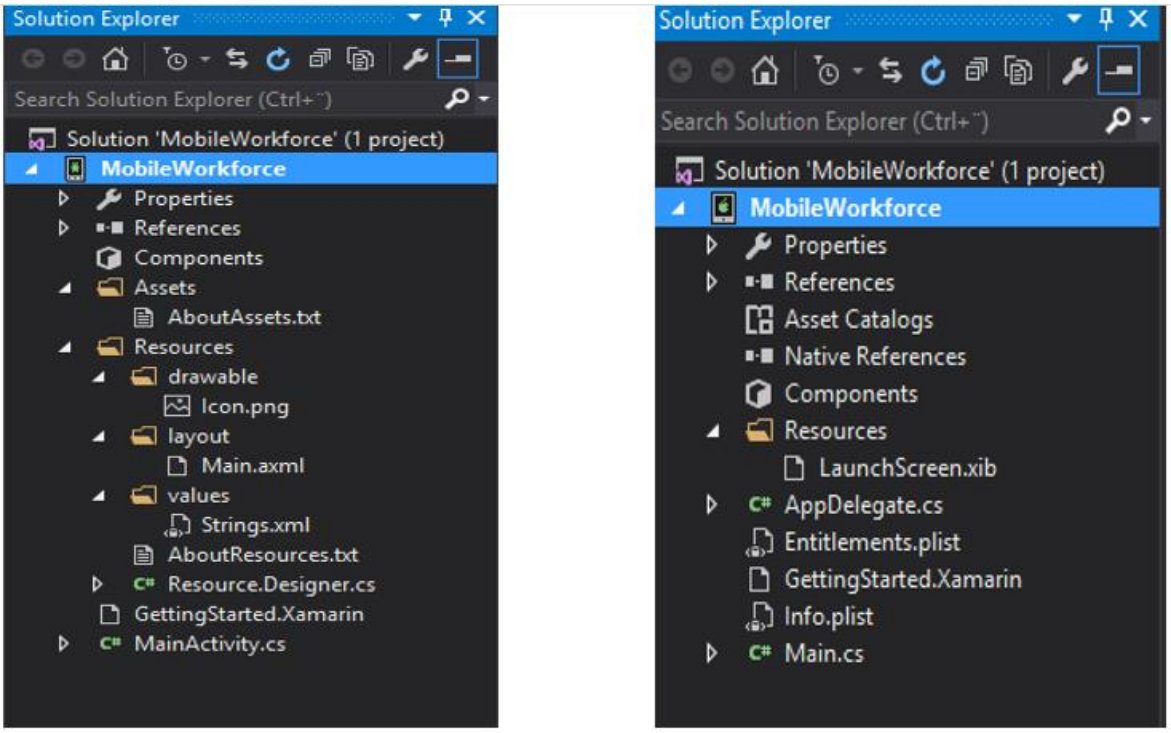


Figure 2. Xamarin Platform Specific

Yet another cleverness of Xamarin platform is that all the native platform-specific development styles are kept as they would appear in the native Java (left) and Objective C (right) development styles. Xamarin, in addition to keeping the native development style and native SDK access, offers developers the entire .NET BCL plus the flexibility and elegance of C#.

2.2 Google Material Design

One of the criteria that make up for great mobile apps is a great UI design. Mobile apps with great UI designs not only stand out from others but also provide a better user experience. In 2014 Google introduced Material Design, a comprehensive guide for visual, motion and interaction design across platforms and devices which helped Android developers deliver the best mobile apps in the business.

Material design was introduced in Android 5.0 (Lollipop, API level 21), Android 5.0 and subsequent versions come with new UI components and features that enable integrating material design patterns specified by Google. Even though Material design is only available in Android 5.0 and above, Android Support Library (discussed more below) has made it possible to apply the beauty of material design on earlier versions of Android too.

2.2.1 Material Theme

Material Theme allows android developers to set colour palette and animations to widgets that support material design. Android Material theme offers three different base themes; each defined in the style folder of Android SDK. It is also possible to extended from these themes to create a customized look and give a distinctive brand to our application /4/.

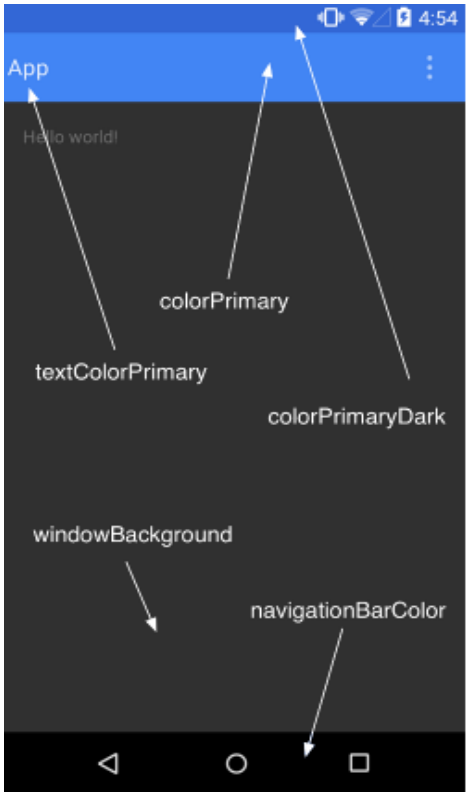


Figure 3. Material Theme colour palettes /5/

Figure 3 shows customizable colour attributes which can help brand our application; by extending one of Android's base themes, developers can set colour palettes, customize UI controls, animate and more according to material design specifications.

2.2.2 Widgets

Material Widgets with rich and complex views are one of the reasons behind the beauty of the recent Android apps. Android provides widgets like *RecyclerView*, which can be used instead of *ListView*, a materially designed widget with rich features. Developers also have the option of adding more material widgets from Android Support Library.

2.2.3 Animations

Android has made an API available to create custom animations for touch gestures in UI controls and widgets to meet with Google Material Design specifications.

2.3 Android Support Library

Android Support Library was first introduced in 2011. The main purpose of the library is to provide backward compatibility so that new widgets and features can be added to earlier

versions of Android. In addition, Support Library provides material design UI controls/elements that are not included in the native Android SDK /6/.

The features and specifications provided by Material Design together with Android Support Library's compatibility support and new UI controls are the key to today's best looking and efficient mobile applications.

2.4 Microsoft Azure

Microsoft Azure is a complete cloud computing platform which provides an overwhelming number of cloud services needed to develop, test, deploy and manage applications both in the cloud and on-premises.

Microsoft Azure has been around since early 2010. It is the only cloud provider to have the leading cloud computing services in SaaS (Software as a service), PaaS (Platform as a service) as well as IaaS (Infrastructure as a service). Microsoft Azure supports almost all popular operating systems, programming languages, databases and third-party software and APIs /7/. Figure 4 below shows cloud solutions provided by Microsoft Azure.

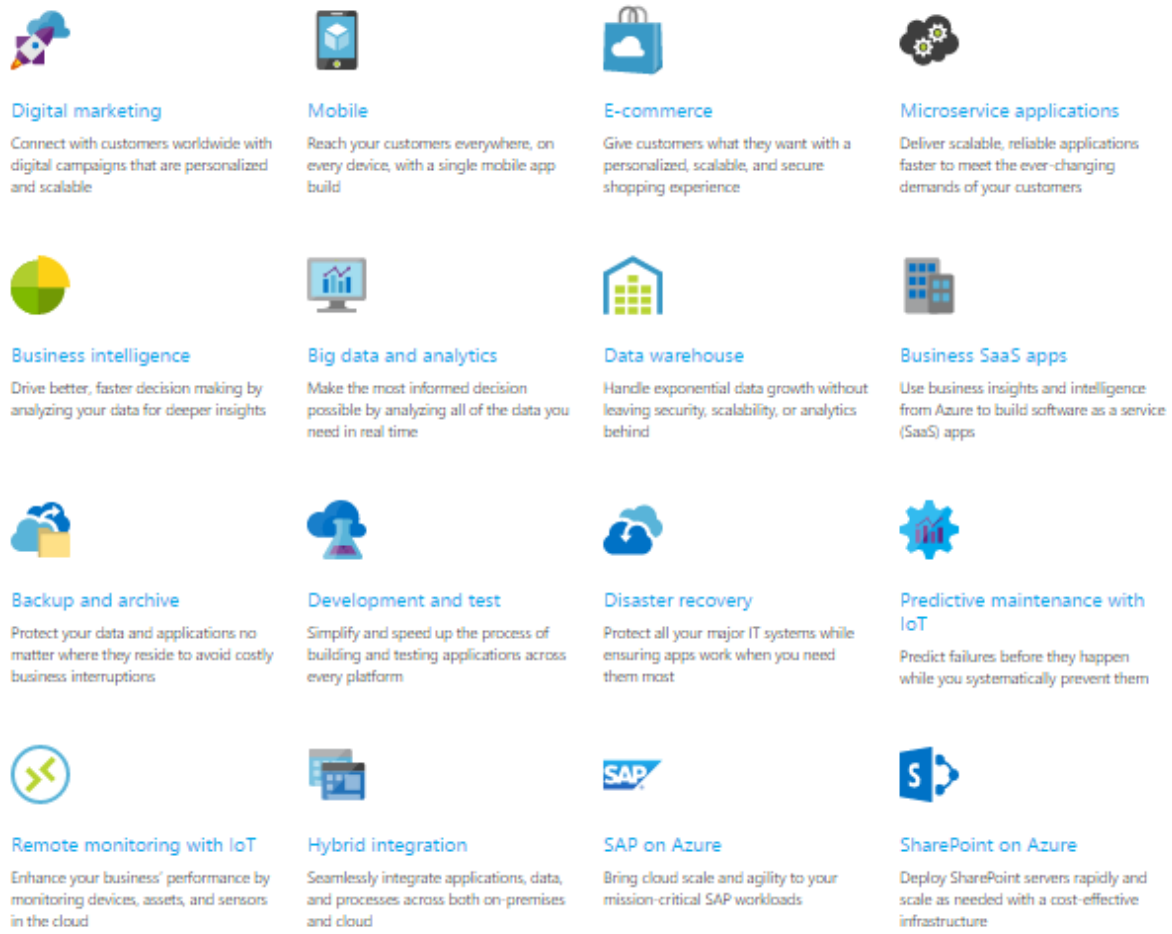


Figure 4. Microsoft Azure cloud solutions /9/

Microsoft Azure covers more geographical regions than any other cloud provider (more than AWS and Google Cloud combined), with 38 Azure regions, Azure is keen to provide and ensure high performance applications and preferred cloud location across the globe /10/.

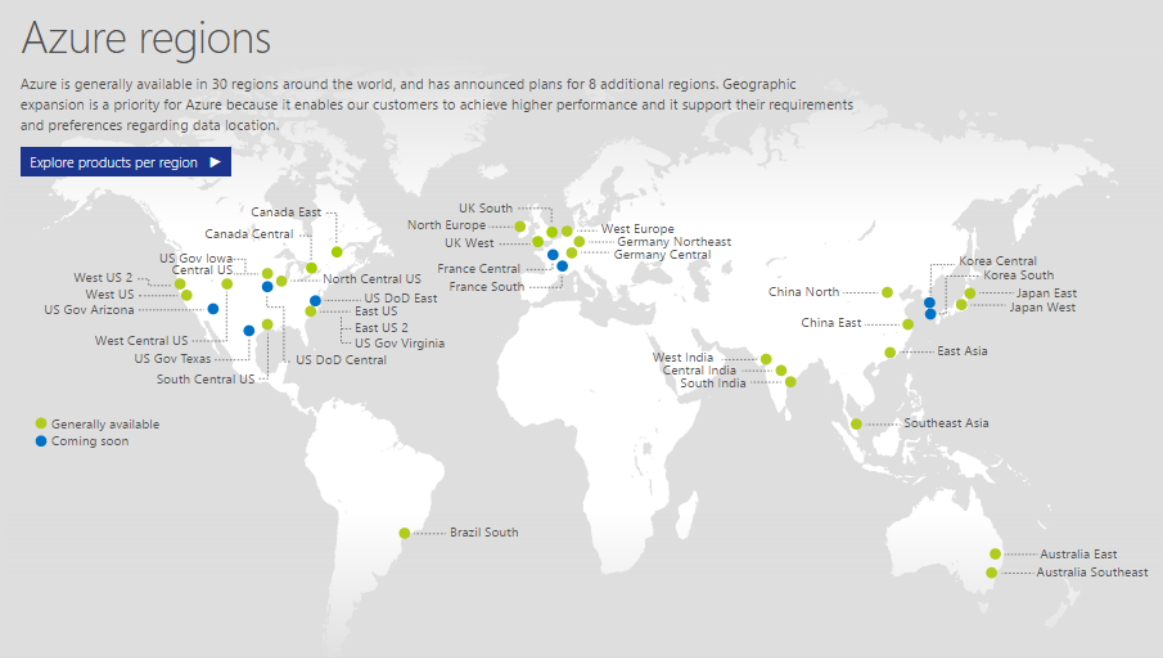


Figure 5. Azure Regions /11/

Workforce makes use of Azure Mobile Apps service as a back-end to store data, authenticate users using Google as an external IdP.

2.5 Visual Studio 2015

Visual Studio 2015 is one of the best, if not the best IDE's available for mobile, desktop and web application development and, of course, the primary choice for developing C# applications. VS2015 comes bundled with Xamarin framework, making it easy to create mobile applications out of the box. VS2015 offers a variety of Xamarin development environments both for cross-platform as well as native projects. In both approaches VS2015 provides a straightforward way to integrate third party libraries through Nuget Package Manager and Component store. For instance, Android Support Library can easily be added and installed into android projects via both methods.

3 SYSTEM DESCRIPTION

3.1 General Requirements

Bases on their priority level, the requirements for Workforce can be classified under three main specification categories (Table 1).

Requirements	Description	Priority level
Normal Requirements	basic requirements of the project as specified by the client, these specifications must be met, therefore given a high priority	1
Expected Requirements	expected features and functionalities of Mobile Workforce as a typical mobile application and therefore given a priority as well	2
Exciting requirements	requirements that would enhance functionality and user experience if met, exciting requirements are given a low priority as they are not obligatory	3

Table 1. Project requirement classification

A successful implementation of these requirements in accordance with their priority level plays a key role in delivering the desired product. Table 2 shows the requirements for Workforce.

Requirements	Priority Level
User should be able to set his/her availability (online/offline)	1
User can accept/decline allocated task	1
User can view accepted tasks/task	1
User can navigate to client address using the application	1
User can contact allocator (message, call)	1
Workforce should be able to track and record distance during navigation	2
Workforce should update server/allocator about task progress	1
Workforce should send the time taken to complete a task to server/allocator	2
Users can synchronize tasks with native android 'event'	3
Users have an option to add contacts	2
The application should be a cross-platform mobile application using Xamarin Technology	2

Table 2. Mobile Workforce Requirements

3.2 Workforce Use Case

Figure 6 shows the interaction between Workforce application and the user (field worker); who can be a doctor, a nurse or a registered healthcare worker. Each use case is also explained in the sections that follow.

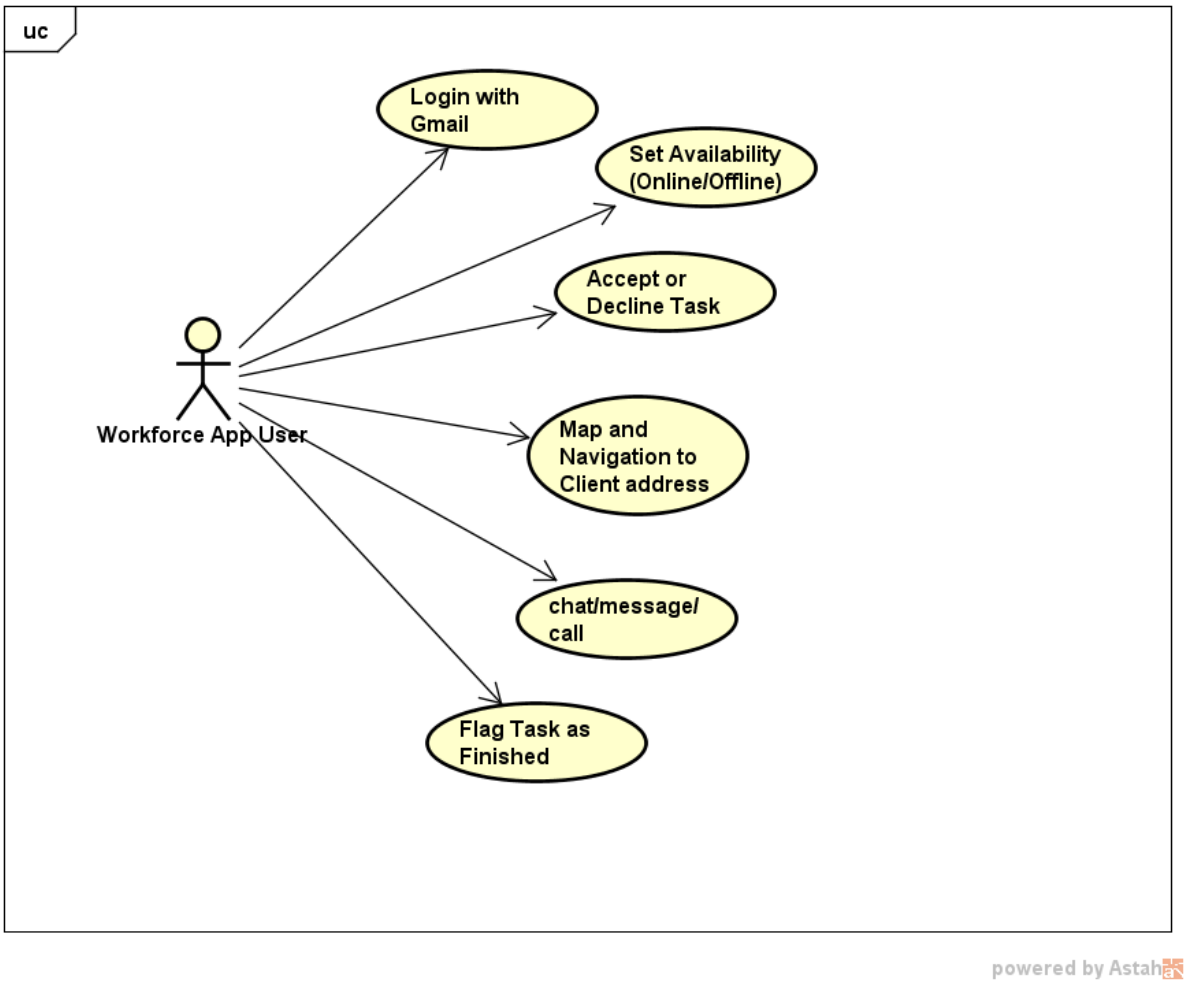


Figure 6. Workforce Use Case Diagram

3.2.1 Login with Gmail

Users of this mobile application should login with a valid Google account to access features in the application.

3.2.2 Set Availability

A Workforce user must be able to set availability status as ‘online’ or ‘offline’ through the application so that allocators on the server side would know which users are available. Allocators on the back-end depend heavily on user’s availability status to offer/assign tasks.

A User only gets job offers if availability is set to online and gets notified through the application when offered.

3.2.3 Accept or Decline New Task

As mentioned in the above section, allocators send job offers to available users only. When notified about a new task offer, a user has an option to view the task offered and information associated with it such as job address, time and duration.

The offered tasks must be responded to as quickly as possible by users, and users have the option to accept or decline the offer.

3.2.4 View Tasks/Task

Each time a user accepts an offer, the task is added to a list of accepted tasks, which is available to the user on the home screen of the application. The user can view task lists and/or select any task from the list to view further details, features and functionalities.

Table 3 shows properties associated with each task available for Workforce user to view. This data provide all the necessary information about the task to be carried out.

Task Properties/Information	**Description**
Task name	The type of healthcare service needed; for example, ‘elderly care’, ‘physiotherapy’…
Client name	The name of the person or organization in need of healthcare service
Client address	Map view of the client’s address
Date	Task date
Time	Task start time
Duration	Amount of time that the task should take
To do	Description of the task to be carried out

Table 3. Task Data Properties

3.2.5 Location and Navigation

In addition to displaying the location of client's address in a map view, Workforce application should also provide direction and navigation to the address as well as track and record the distance travelled during navigation.

3.2.6 Contacts

A Workforce user may need guidance or other assistance regarding his/her tasks and therefore users should be able to chat, message or call allocators or colleagues from within the application. Users also have the option to save new contact as well as modify saved ones.

3.2.7 Finished Tasks

Upon completion of a task, a Workforce user should flag the task as 'finished task' to let the application move the task from the accepted task list to finished tasks.

4 APPLICATION DESIGN

This chapter provides an overview of application designs and architectures implemented from the development to deployment of the application.

4.1 Three-tier Architecture

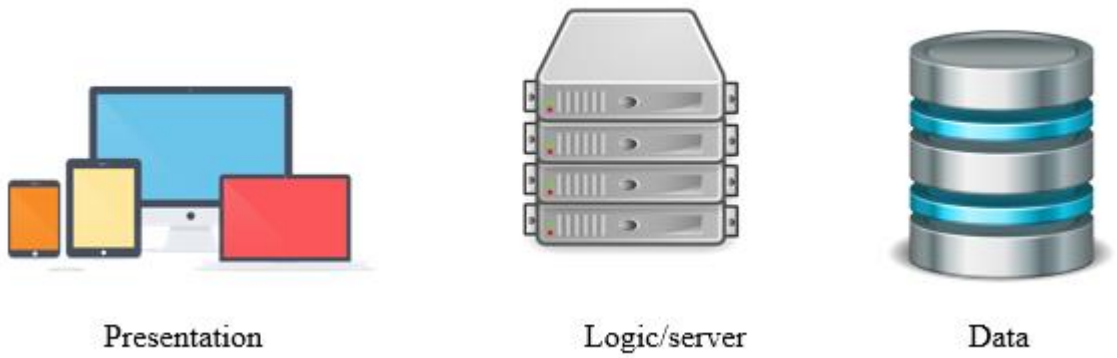


Figure 7. Three-tier Architecture

The Three-tier Architecture, as can be seen from Figure 7, consists of three separate tiers; Presentation, Logic and Data. At the top sits the Presentation tier, it refers to any UIs in the application where users can directly interact with the application. The Logic tier sits between the Presentation and Data tier and it is responsible for the application's business rules as well as moving data in between the two tiers. The final tier, Data tier, is where data is persisted.

4.2 Layered Architecture

Not to be confused with tier architecture, Layered architecture refers to the logical representation of how the application code is organized. Layered architecture groups related functionalities and responsibilities into layers. It essentially includes data, business as well as presentation layer but it can also include more layers depending on the application. Each layer is introduced in the sections that follow.

Layers in layered architecture are loosely coupled to ensure separation of concerns, code reusability and maintainability. As mentioned in the previous sections, Workforce uses Portable Class Libraries to handle data and service manipulations. Workforce Android consumes this PCL library to handle its data and service manipulations.

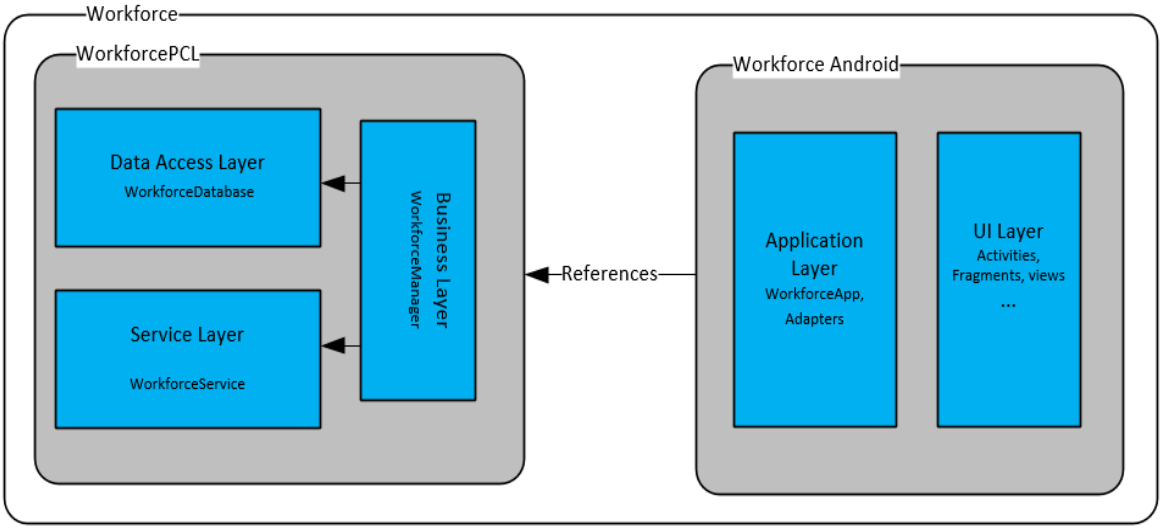


Figure 8. Workforce Application Architecture

The figure above shows the architectural design of how code is organized in Workforce. *WorkforcePCL* project includes data access, service and business logic layers and therefore *WorkforcePCL* is where all the shared logic code is written and it can be consumed by any of the platforms, in this project, by Workforce Android.

4.2.1 Data Access Layer

Data Access Layer includes data access classes that contain methods for querying and persisting data to local SQLite database. *WorkforceDatabase* is the data access layer and consists of two classes *WorkforceDatabase* and *WorkforceRepository*. Table 3 gives the list of classes that reside in data access layer along with their functions.

Layer	Classes	Function
WorkforceDatabase	*WorkforceDatabase*	Performs database related tasks (CRUD operations)
	WorkforceRepository	Makes database connection and provides methods for *WorkforceDatabase* operations

Table 3. Data Access layer

4.2.2 Service Layer

Nuget manager makes Azure Mobile Apps Client sdk available, this sdk provides features to connect to Azure Mobile Apps and thereby consume services like data storage, authentication, push notification and offline sync. Workforce consumes Azure Mobile App services through a RESTful API to carry out http call operations to manipulate and persist data.

Layer	Classes/interfaces	Function
WorkforceService	*IWorkforceService*	Contains method signatures for common service scenarios
	WorkforceService	Implements *IWorkforceService*, Handles Azure Mobile Apps client and consumes Azure Mobile App services

Table 4. Service Layer

4.2.3 Business Layer

Business Layer provides two manager classes to manipulate data throughout the application. *WorkforceDatabaseManager* class holds business logic for local SQLite data manipulations whereas the second class, *WorkforceServiceManager*, class holds business rules for cloud based operations.

Layer	Classes/interfaces	Function
WorkforceManager	*WorkforceDatabaseManager*	Application business rules for local database manipulations, encapsulates the data access layer
	WorkforceServiceManager	Application business rules for Azure Mobile App Services, encapsulates the service access layer

Table 5. Business Layer

4.2.4 Application Layer

Application Layer is what glues the PCL to the Android project and it is also the place where platform specific features are included during cross-platform development.

Layer	Classes/interfaces	Function
Application Layer	*WorkforceApp*	Glues the portable class library to the android project, provides database connection properties and exposes business layer
	ContactsRecyclerAdapter	Used to bind data to UI
	NewTasksRecyclerAdapter	
	TasksRecyclerAdapter	

Table 6. Application Layer

4.2.5 UI Layer

Also known as presentation layer, this layer includes activities, fragments and all contents under *Resource/Layout* folder. UI layer handles the appearance of application's screen and its behaviour during user interaction.

Workforce makes use of support libraries to add material design UI components with rich features like recycler view, navigation view, floating action button and more.

4.3 Sequence diagrams

Sequence diagrams are useful ways to visualize and demonstrate how different parts of the application infrastructure work in a sequential manner and as per the application's architectural design. In this section, the key events involved in the application are illustrated through sequence diagrams.

4.3.1 New Task Notification

Workforce users receive new task offers and in addition users can also pull notifications by pressing the notification action menu.

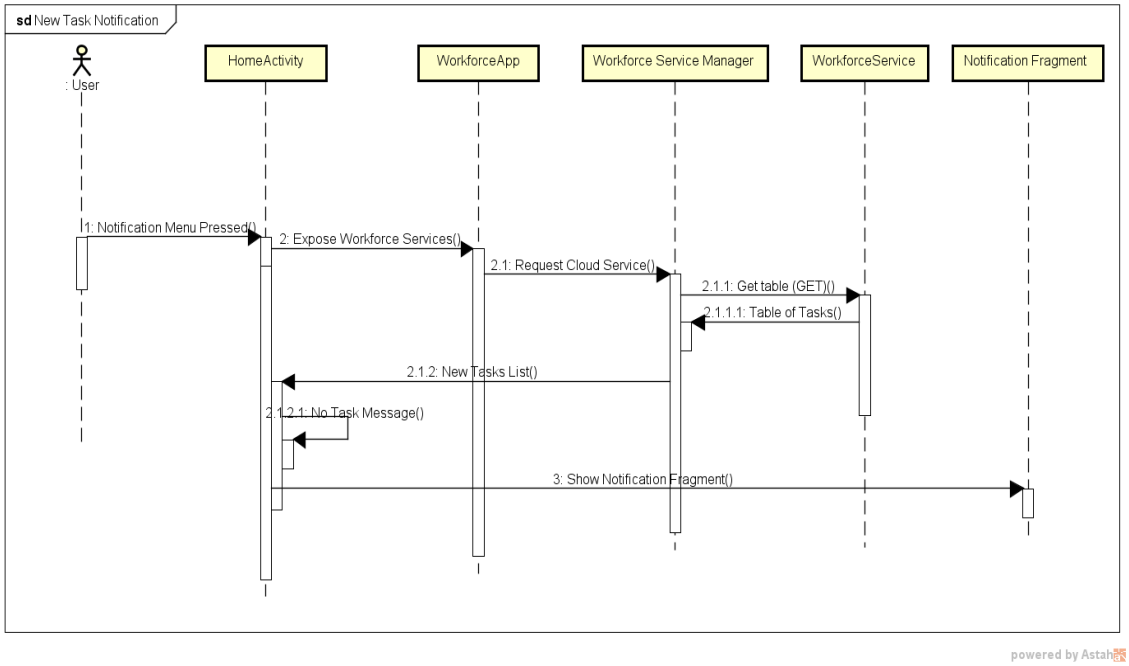


Figure 9. New task notification

4.3.2 Accept/Decline Task

One of the main scenarios includes the user's action when new tasks are pushed from allocators. The user either accepts or declines the task offered.

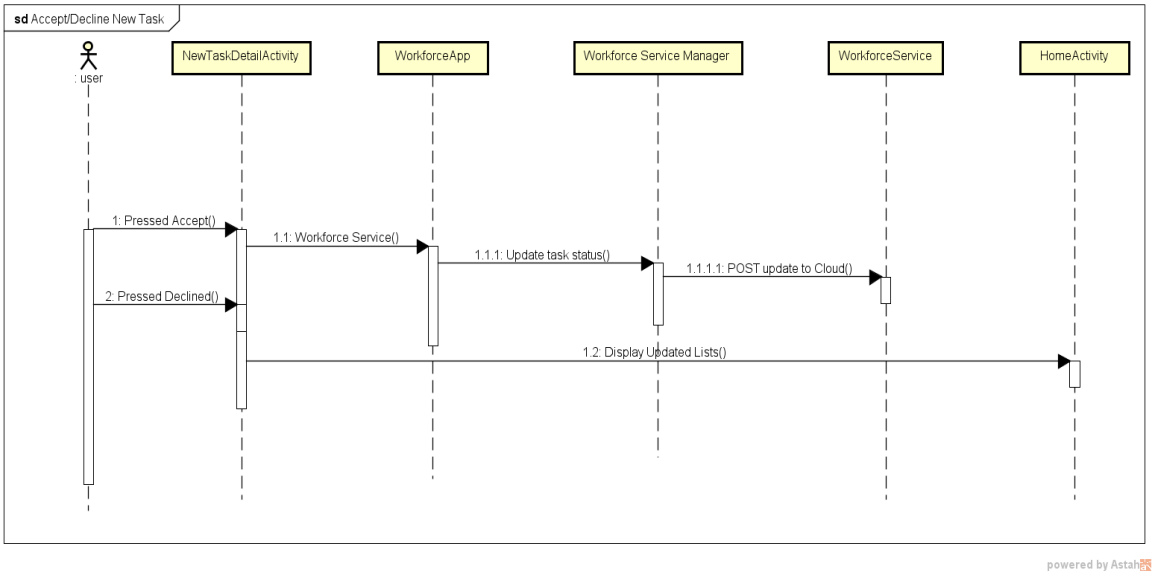


Figure 10. Sequence diagram 'Accept' or 'Decline' task offer

When a user presses the 'Accept' or the 'Decline' button, two events happen depending on which action is taken. If a user decides to decline the offer and presses 'Decline', a response

is sent to a server notifying of the decision taken. The same scenario occurs when the user accepts, but in addition to updating the server, Workforce also saves the accepted task into the local SQLite for offline access. Figure 11 illustrates the sequential flow of events when a new task is added to SQLite database.

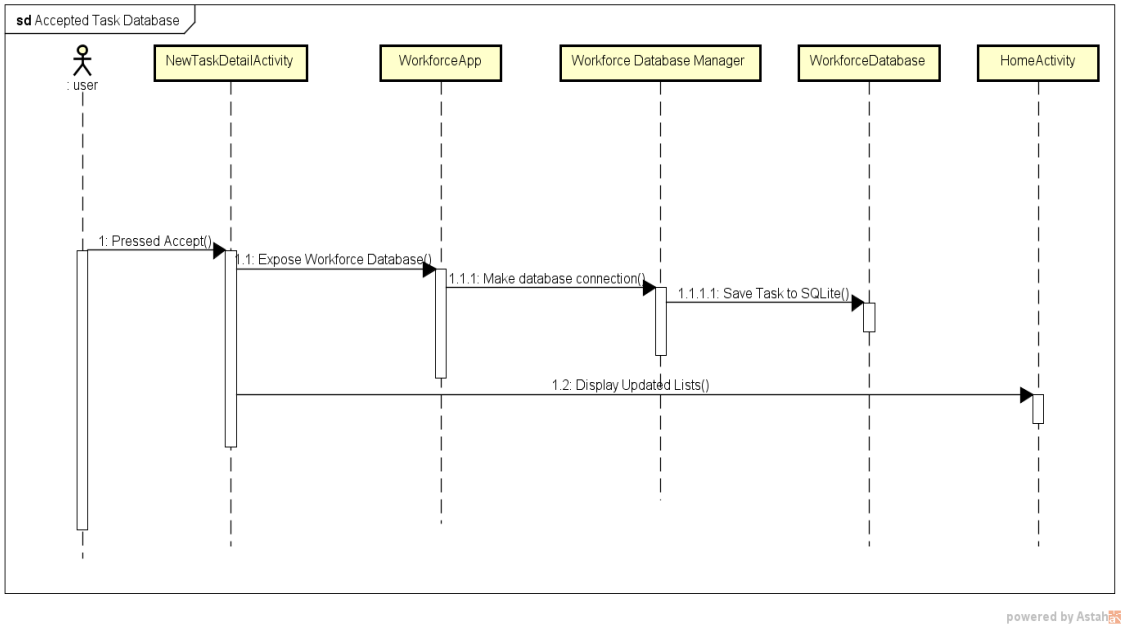


Figure 11. Adding 'accepted task' to local SQLite

4.3.3 Displaying Tasks

The homepage screen shows the list of accepted tasks. As mentioned above tasks are saved to the local SQLite database when accepted to make the application functional offline. Figure 12 shows the sequential flow of how accepted tasks are retrieved from the local SQLite and displayed.

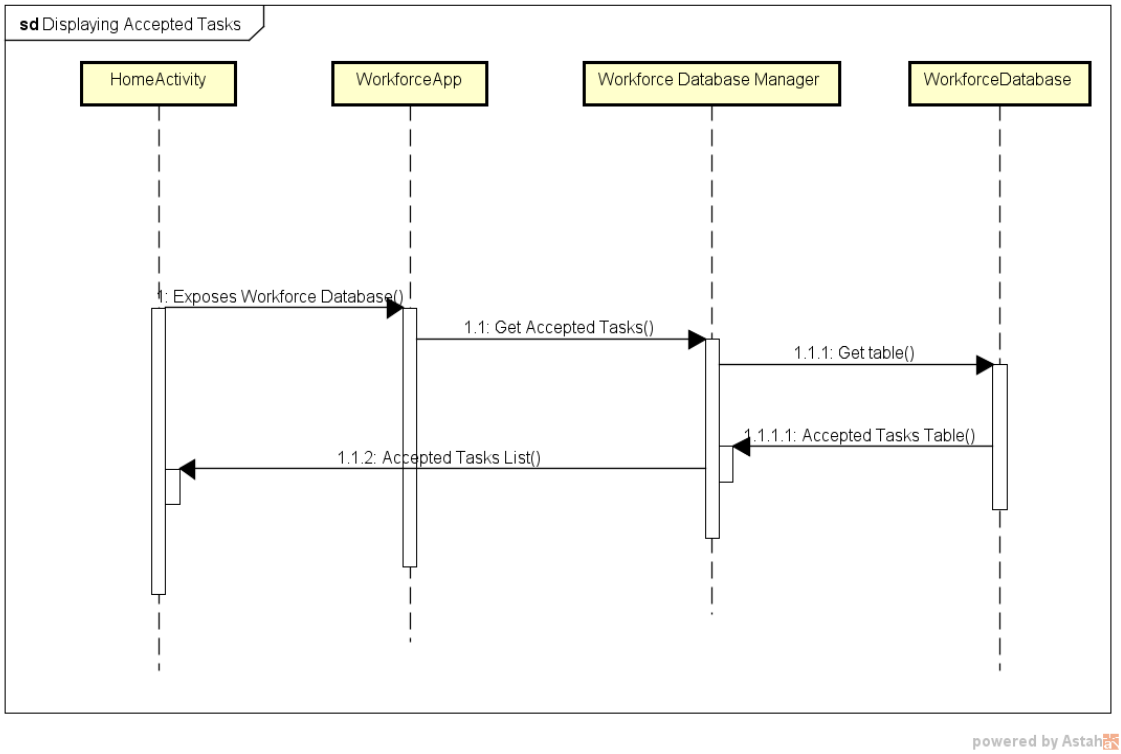


Figure 12. Displaying 'accepted tasks'

4.3.4 Finished Task

Workforce users must mark tasks as ‘Finished’ upon completing a task. Both Azure back-end and the local SQLite are updated to persist the changes made. Figure 13 shows Azure back-end scenario where as Figure 14 shows the local SQLite data update.

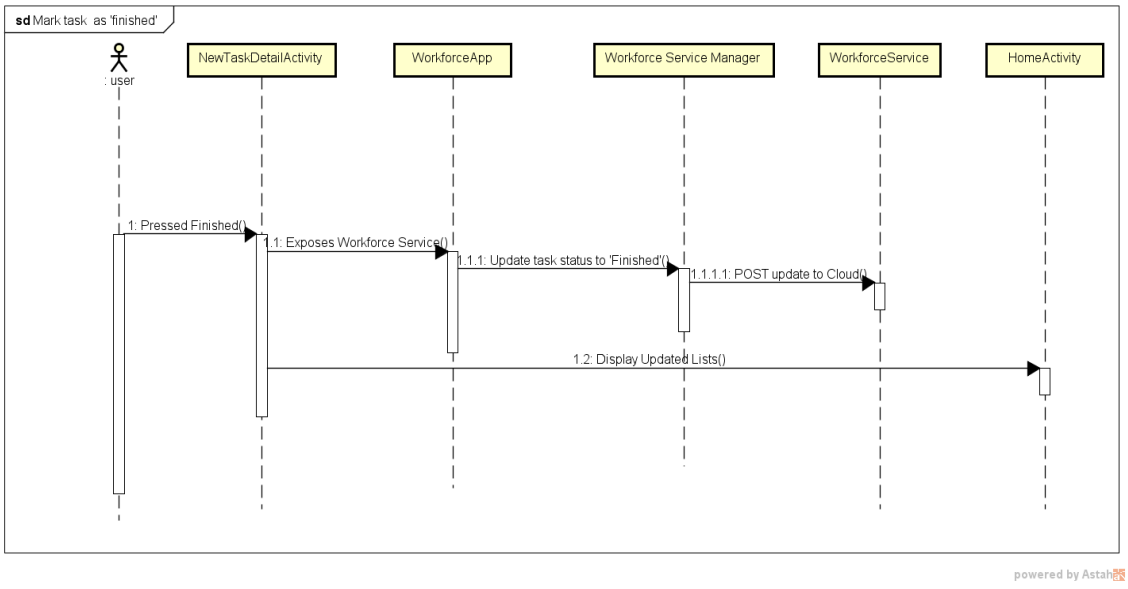


Figure 13. Mark task as 'Finished' (Azure back-end scenario)

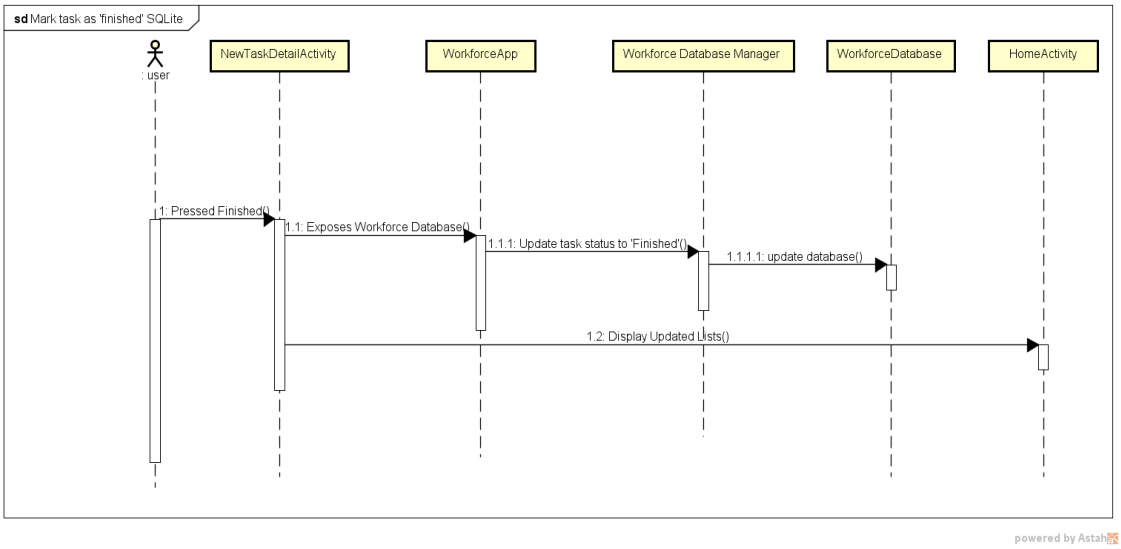


Figure 14. Mark task as 'Finished' (local SQLite scenario)

4.4 **Server-less backend (Azure Mobile Apps)**

Azure server-less backend makes it possible to host mobile applications without the need to implement server side implementations and with the flexibility of ‘pay-as-you-go’ service. Azure Mobile App is amongst the most popular PaaS (Platform as a Service) cloud services offered by Microsoft Azure and it provides a platform to develop easily scalable, high performance and globally available mobile back-ends.

Azure Mobile App brings rich capabilities to client mobile applications like cloud data storage, offline data Synching, authentication and push notifications amongst many others /8/. Mobile Workforce implements some of these services and features to deliver Enterprise level product.

Figure 15 shows the most popular services available in Azure Mobile apps and it also shows the supported mobile platforms (note that it supports almost all the popular mobile development platforms).

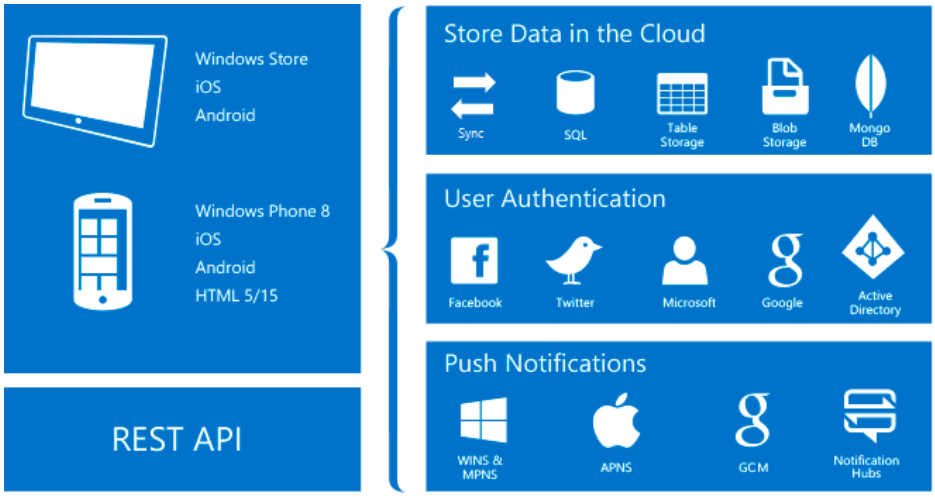


Figure 15. Azure Mobile Apps /8/

Microsoft Azure offers a free trail for any developer with a Microsoft account. Once registered, Azure cloud services can easily be configured through Azure Portal. Azure Portal is an official web app for Microsoft Azure, its rich features offer an environment for enterprise developers to create, manage and remove Azure services. Azure Portal also lets developers manage their Azure account, subscription as well as billing information settings /12/.

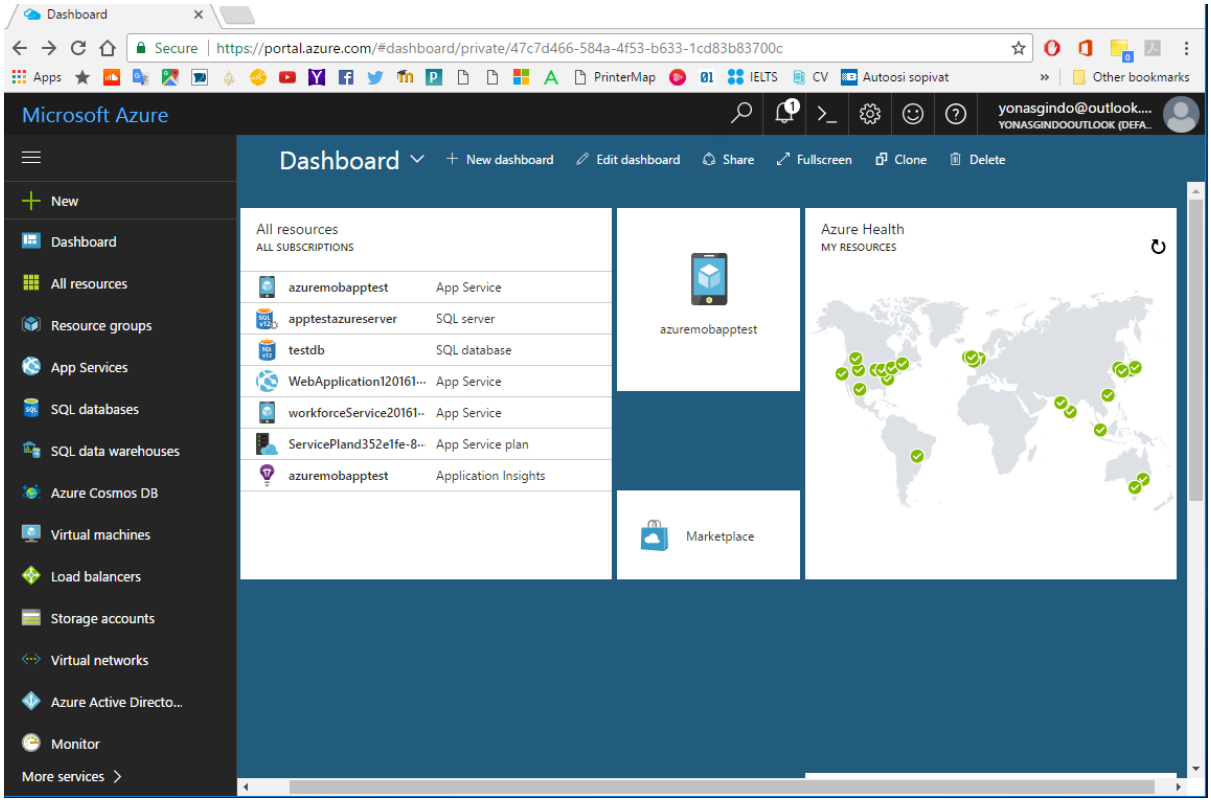


Figure 16. Azure Portal Dashboard

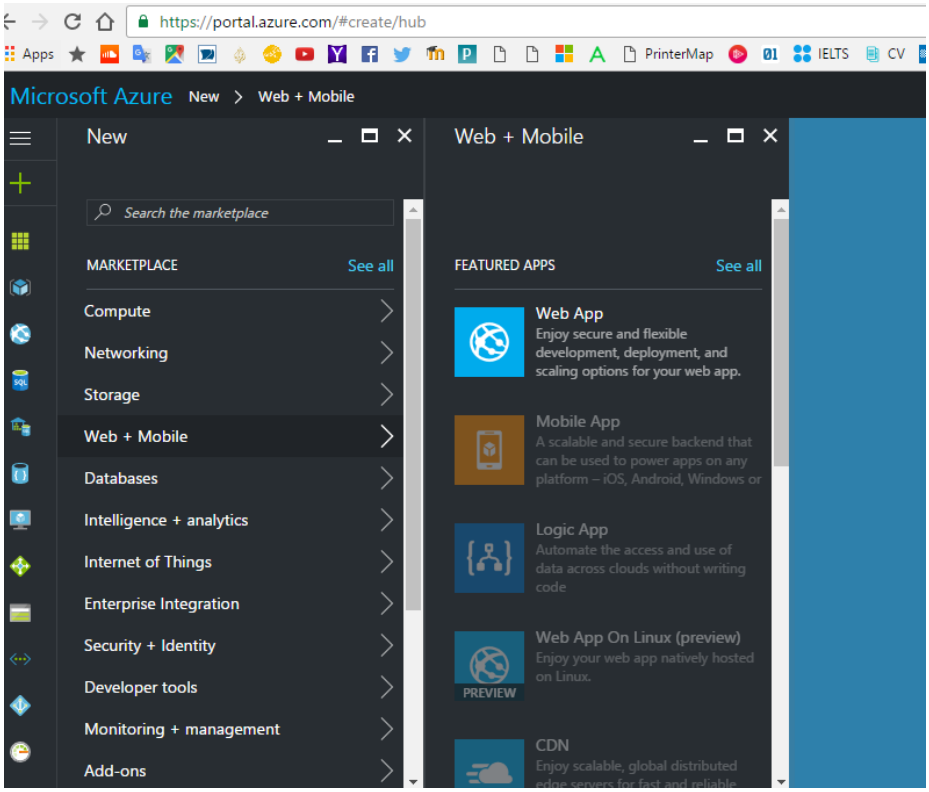


Figure 17. Creating Azure Mobile App

As mentioned above, Azure Mobile App service can easily be created, configured and maintained through Azure Portal (alternatively Azure Mobile Apps can also be created and deployed to Microsoft Azure from VS2015).

The following sections discuss Azure Mobile App services consumed by Workforce.

4.4.1 Easy Tables

Data is at the heart of any software application. Almost all applications must store data and Azure Mobile Apps provide a variety of data storage services /Figure 15/. One of the interesting data storage services provided by Azure Mobile Apps is Easy Tables. Easy tables allow developers to easily create and manage tables for mobile applications with only a few steps and it is backed by Azure SQL and supports all mobile development platforms; both native and cross-platform.

4.4.2 OAuth 2.0

Azure Mobile Apps offer authenticating and authorization of client mobile users from external identity providers including Twitter, Facebook, Google, Microsoft and Azure Active Directory. Developers have a variety of providers to choose from to satisfy user needs. Workforces app implements Google provider for authenticating and authorizing users.

To use Google authentication in Azure Mobile Apps, the mobile application must first be registered to use Google's OAuth service, and the Google API service must be enabled trough Google developer console page.

After successfully registering the application to use OAuth, Google provides a client ID and a client secrete and by using these credentials Azure Mobile Apps can be configured to support third party authentication and authorization services by Google /13/.

5 IMPLEMENTATION

The features and purpose of the technologies used for the application have already been discussed in chapter 2. This section of the document illustrates how these technologies were implemented in accordance with the software design specification discussed in chapter 4.

5.1 User Interface

Workforce is material from top to bottom, it implements all the features that material design has to offer. The application has integrated material widgets, material theming and animations, providing optimized user experience, efficiency as well as native UI look.

5.1.1 Material Theme

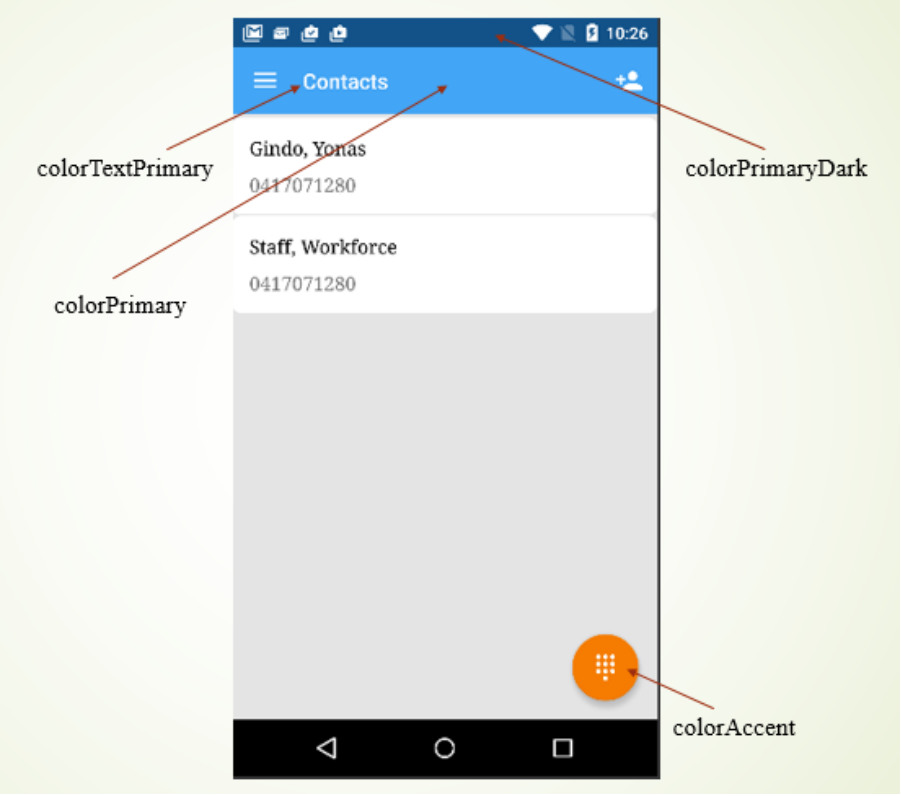


Figure 18. Workforce theme

As discussed in chapter 2, Material theming is used to brand applications by applying identity colour palettes to UI controls as well as creating characteristic touch gestures. Android SDK provides three base material themes out of the box to inherit from /5/.

Theme	
@android:style/Theme.Materail	dark version theme
@android:style/Theme.Materail.Light	light version theme
@android:style/Theme.Materail.Light.DarkActionBar	light version theme with dark action bar

Table 7. Android Themes

This application, however, makes use of *AppCompat* theme provided by support library instead of android SDK themes. The *v4* and *v7* support libraries provide themes with more features and support to implement material design patterns and best practices. These libraries were added to the project via NuGet Package Manager.

```
<style name="AppTheme" parent="WorkforceTheme.Base">
  <item name="android:windowContentTransitions">true</item>
</style>

<style name="WorkforceTheme.Base" parent="Theme.AppCompat.Light.NoActionBar">
  <item name="android:colorPrimary">@color/ColorPrimary</item>
  <item name="android:colorAccent">@color/ColorAccent</item>
  <item name="android:colorPrimaryDark">@color/ColorPrimaryDark</item>
  <item name="android:statusBarColor">@color/ColorPrimaryDark</item>
</style>
```

Android theme is declared in the *Resource/Styles* folder of the project, the code snippet above shows that Workforce theme inherits from *Theme.AppCompat.Light.NoActionBar* which is provided by *v7 support Library*. It is the light versioned theme without action bar, themes without action bar prevent the native action bar from appearing so that the application can use custom toolbar instead of an app bar. The following code snippet shows how the customized theme is applied to an activity.

```
[Activity(Label = "Workforce", Theme = "@style/AppTheme")]
8 references
public class HomeActivity : AppCompatActivity
{
```

5.1.2 Material Widgets

This application uses several material widgets like recycler view, snack bar, Navigation view and floating action button to provide beautiful UI and best user experience. Android support design library was added via NuGet package manager to add these rich and complex material widgets to Workforce.

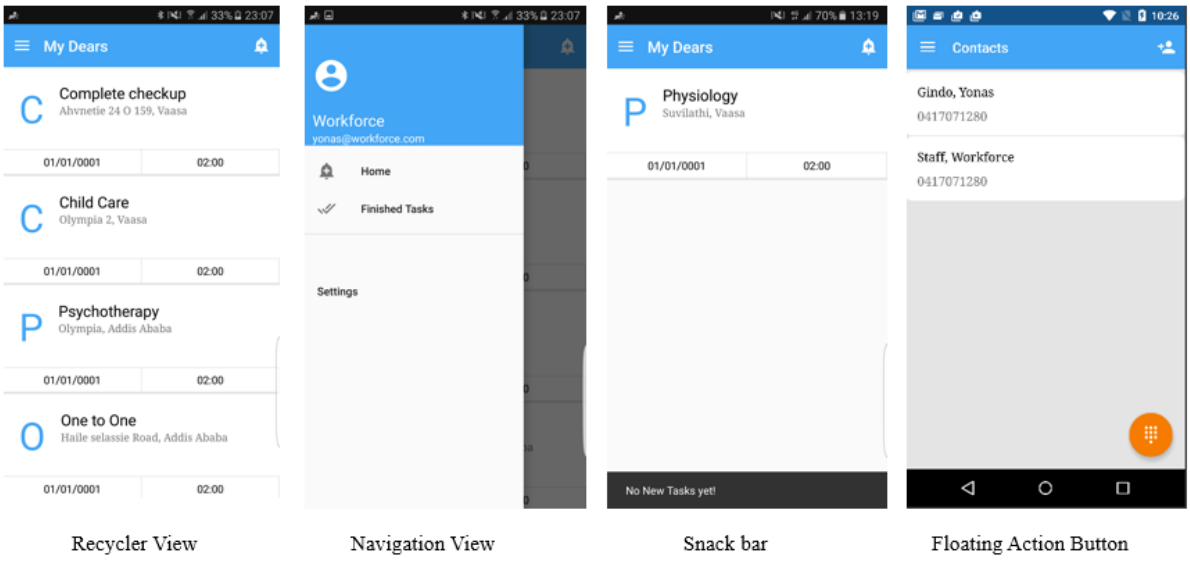


Figure 19. Material Widgets

The application has mainly used recycler view to display a list of data. Figure 19 above shows that it was used to display the list of accepted tasks. Recycler view brings rich capabilities such as animating views, scrolling effects, styling items and much more; which are difficult and in some cases impossible to achieve using list view.

5.2 Calls and Messaging

Workforce integrates calls and messaging to let users contact allocators or clients when needed. Android SDK provides Intents to launch calls and SMS messaging for a specified phone number.

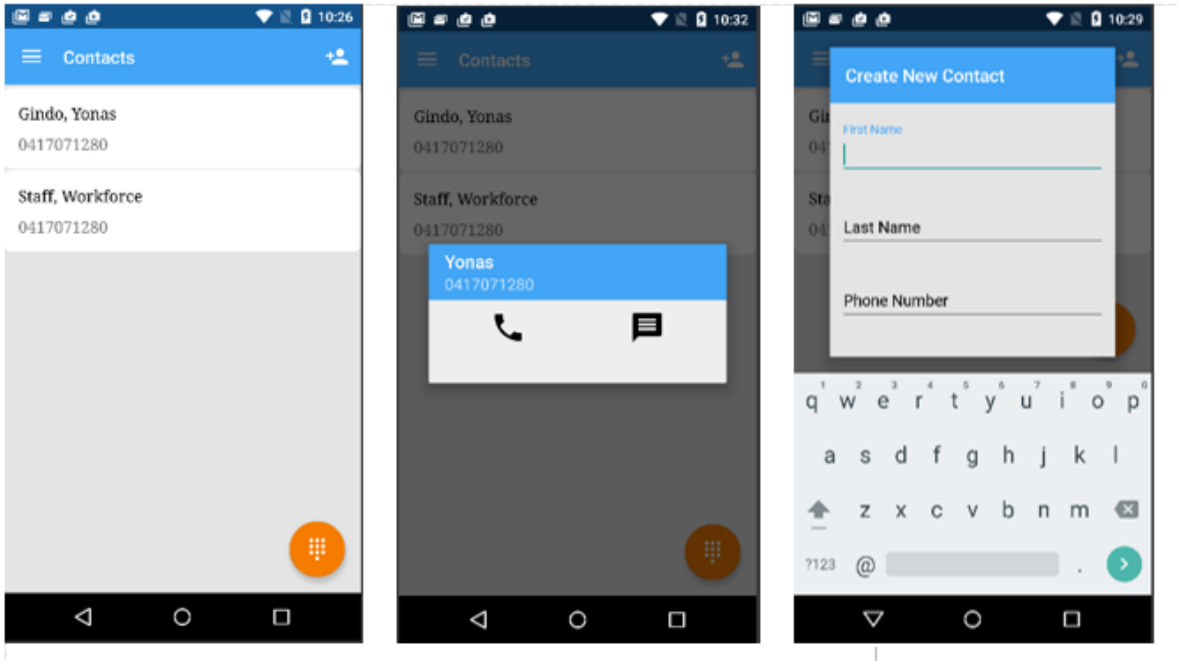


Figure 20. Contacts UI

As can be seen from Figure 20, in addition to making calls and sending SMS messages Workforce also provides an option for adding new contacts, editing contacts as well as a floating action button for navigating to the dial pad.

To use Android's phone call and SMS services, the *AndroidManifest.xml* file must be updated with the following permission. The SMS permission is not needed unless the application sends a message programmatically, which is not the case here.

```
<uses-permission android:name="android.permission.CALL_PHONE" />
```

The *ContactsActivity* and the *ContactActionFragment* classes handle operations related to Workforce contacts. *ContactsActivity* class makes use of the recycler view to display a list of saved contacts. In addition, this class is responsible to load the *ContactActionFragment* when a user clicks on a specific contact.

ContactActionFragment class displays the selected contact and provides the possibility to make a call, message or edit contact. The code snippets below show how call and SMS actions are invoked for a specified phone number.

```
private void MBtnCall_Click(object sender, EventArgs e)
{
    Dismiss();
    var callIntent = new Intent(Intent.ActionCall);
    callIntent.SetData(Android.Net.Uri.Parse("tel:" + mCurrentContact.PhoneNumber));
    StartActivity(callIntent);
}


private void MBtnMessage_Click(object sender, EventArgs e)
{
    Dismiss();
    var messageIntent = new Intent(Intent.ActionSendto);
    messageIntent.SetData(Android.Net.Uri.Parse("smsto:" + mCurrentContact.PhoneNumber));
    StartActivity(messageIntent);
}
```

5.3 Maps and Navigation

One of the key implementations of Workforce app is Maps and Navigation. Allocators rely on Workforce user locations to assign tasks to the right candidate. In addition, field workers should be able to check the task's location as well as navigate to the client's address. During navigation to the client address, Workforce sends updates on the location of the field worker to the Azure back-end.

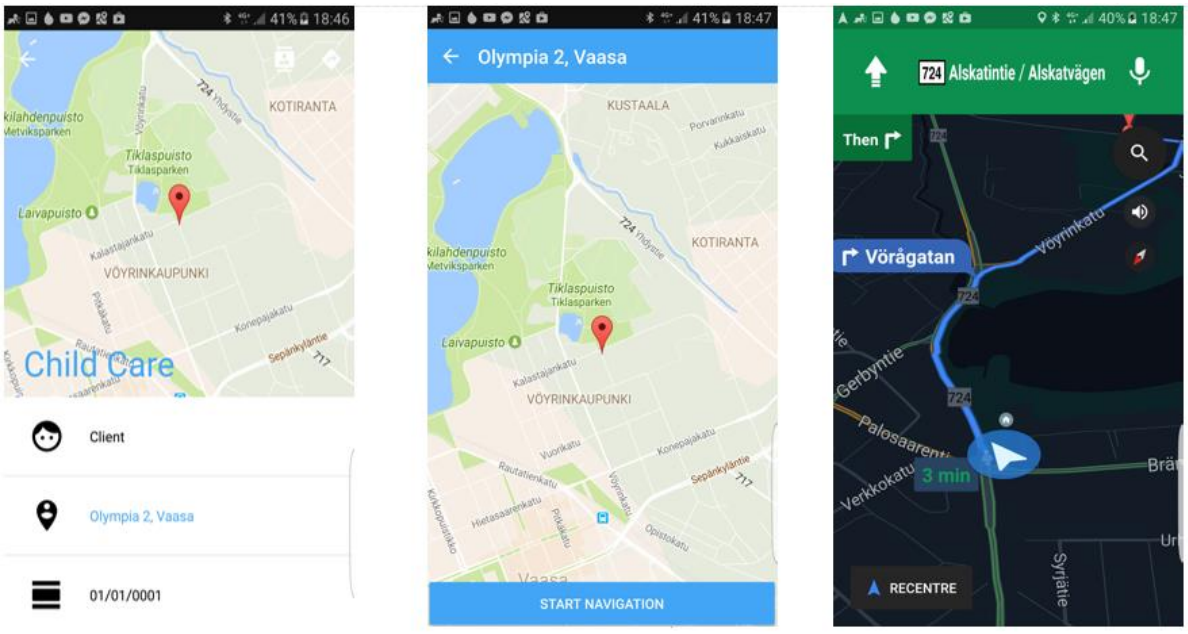


Figure 21. Workforce Location and Navigation

To use Maps in Android applications, Google Play Services client library must first be added to the project via Nuget Package Manager or through component store.

```
PM> Install-Package Xamarin.GooglePlayServices
```

Before Google Play Services can be consumed, Workforce was first registered to Google and the Maps API Key was obtained. The *AndroidManifest.xml* file should also be updated to grant permissions related to maps and location. The Maps API Key obtained from Google and its version should also be specified in the 'application' node of the *AndroidManifest.xml* file.

Once Google play services has been added and *AndroidManifest.xml* file updated, Google Maps view was added to the host view by using *MapFragment* as shown below.

```
<fragment
    android:id="@+id/mapPreview"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:name="com.google.android.gms.maps.MapFragment" />
```

5.4 Calendar sync

The application offers field workers the option to synchronize accepted tasks to Android's built-in event. Since the scenario involves writing to and reading from calendar, the *AndroidManifest.xml* file was updated with the necessary permissions requirements.

The Calendar API provides *ContentValues* and *InterfaceConsts* classes to populate accepted task properties to Android's event. Once the *ContentValues* is populated and the time zone specified, events are inserted to Calendar using the *ContentResolver* class.

5.5 SQLite

Workforce app saves accepted tasks and contact details to Android's local SQLite database. SQLite.NET Library was added to the project via Nuget Manager. It is a simple ORM tool for storing and retrieving objects to the local SQLite database. The SQLite.NET library provides the *SQLiteConnection* object, it consists of various methods which help execute various database operations.

5.6 Azure Mobile Apps

Chapter 4 has covered how Azure Mobile App can be created and configured to provide services like authentication and data storage. This section will discuss how these services

were consumed in the client application, Workforce. Microsoft Azure provides *Azure Mobile Client SDK* library; this library provides classes which help carry out table operations (read, insert, update and delete) in addition to the services mentioned above.

5.6.1 CRUD Operations

Once Azure Mobile Apps has been created successfully, it is possible to check if it is up and running by navigating into the application URL provided. It is also possible to view and test different operations available for the resources created.

Workforce mobile application mainly performs two operations as shown in Table 8.

Operation	**Description**
GET /tables/TaskModel	Gets all new tasks for a given user
PATCH /tables/TaskModel	Updates task progress (accepted, declined, finished)

Table 8. Workforce data operations

On the client side, before Azure Mobile Apps back-end can be accessed for service, the *MobileServiceClient* class should be instantiated and the application URL to the Azure service must be provided. As already mentioned, *Azure Mobile Client SDK* provides infrastructure to access Azure Mobile Apps instance and there by carry out CRUD operations.

```
private static MobileServiceClient client;

public WorkforceService()
{
    client = new MobileServiceClient(applicationURL);

}
```

```
public Task<List<TaskModel>> GetNewTasks(string userId)
{
    tasksTable = client.GetTable<TaskModel>();
    return tasksTable.Where(task =>
                                (task.Status == (int)TaskModel.TaskStatus.NewTask)
                                && (task.userId == userId))
                                .ToListAsync();
}


public async Task UpdateTaskAzure(TaskModel task)
{
    tasksTable = client.GetTable<TaskModel>();
    if (client == null) { return; }
    await tasksTable.UpdateAsync(task);
}
```

5.6.2 Adding Authentication

Chapter 4 of this document has demonstrated how Google Authentication can be configured and enabled in Google's console page as well as in Azure Mobile Apps. To add Google Authentication on the client application; once again, the *Azure Mobile Client SDK* provides the *MobileServiceClient* class which lifts a heavy load off by simplifying the login process using a single line of code.

```
private async Task<bool> Authenticate()
{

    var success = false;
    try
    {
        user = await WorkforceApp
                    .Current
                    .WorkforceServiceManager
                    .CurrentClient
                    .LoginAsync(this, MobileServiceAuthenticationProvider.Google);

        success = true;
    }
    catch (Exception)
    {
        Snackbar.Make(mMainLayout, "Login Failed!", Snackbar.LengthLong)
              .Show();
    }

    return success;
}
```

5.7 Testing and Analysis

This section briefly discusses the tests carried out to check the core functionalities of both the mobile application and Azure Mobile Apps back-end. It also analyses the process in terms of learning outcome.

5.7.1 UI Test

It was discussed that Android introduced Material design UI as of API level 21, however, the minimum targeted version for Workforce is API level 17, and therefore the project involved the implementation of support libraries which help apply these Material design features to versions earlier to API level 21. The application has been tested in all the Android API levels that it targets and has proved to be working as it was intended to.

5.7.2 Azure Mobile App Tests

Azure Mobile Apps provides an interface to test operations for resources/tables created; Once Azure Mobile App is deployed and the application URL obtained, navigating into the application URL should display the page below (Figure 22). It confirms that the Azure Mobile Apps back-end is successfully created and that it is running.

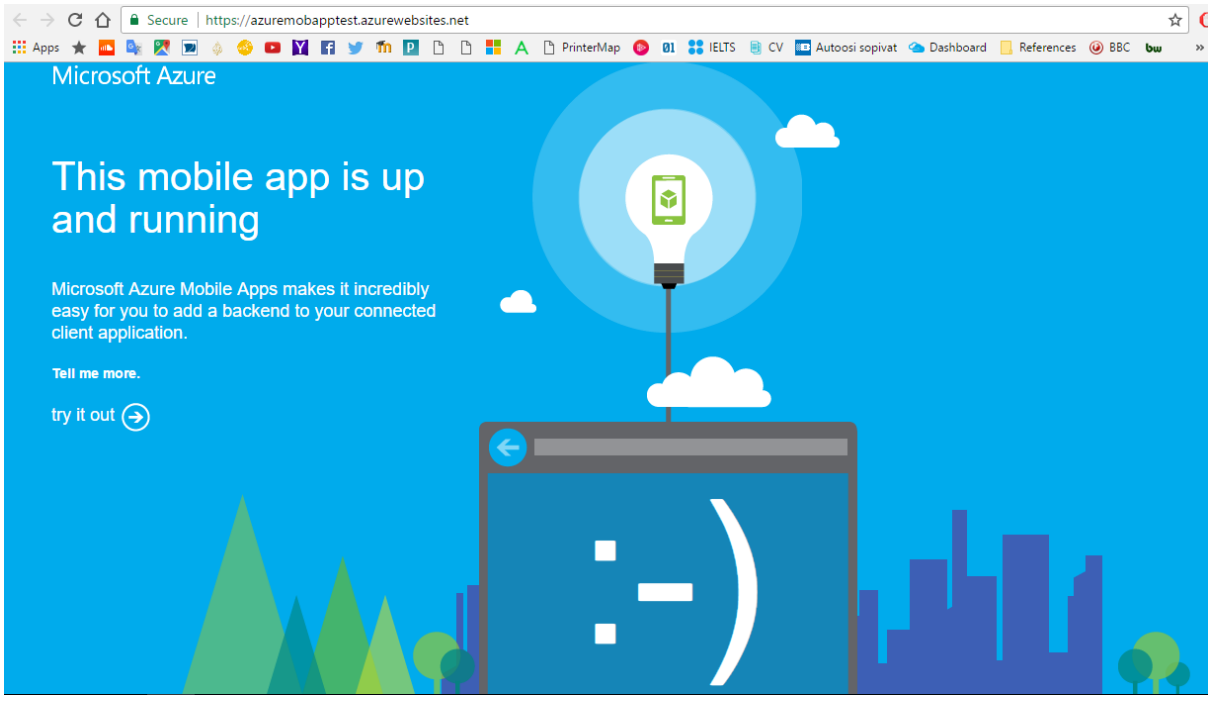


Figure 22. Azure Mobile Apps

By clicking the ‘try it out’ option, it is possible to view all the available operations for each resource created under the Azure Mobile App. In this project two resources (tasks and users) were created by making use of Easy Tables storage. Figure 23 shows the operations available for these resources.

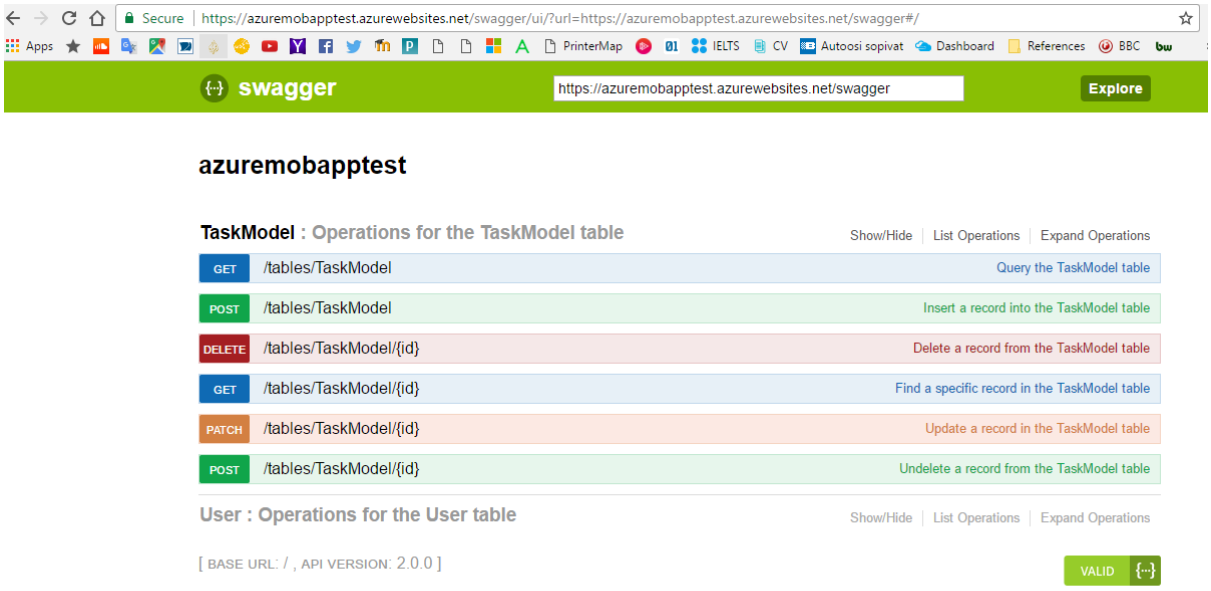
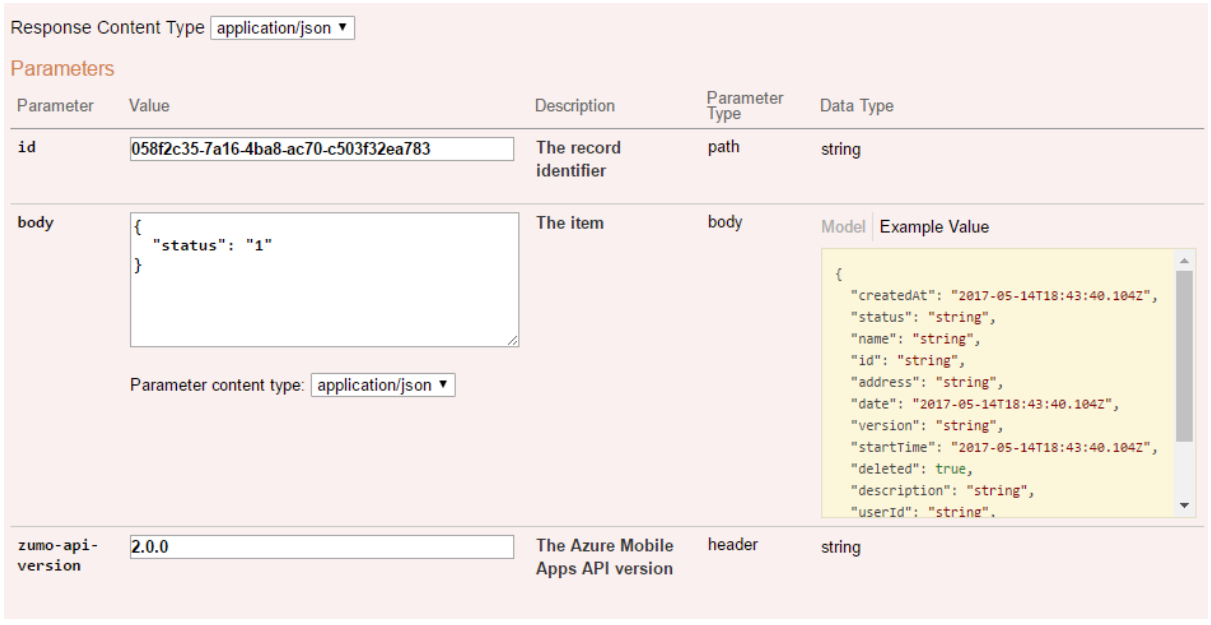


Figure 23. Azure REST

It is possible to test any of these operations and the screen shot taken below shows the test made for updating a task status.



Request URL

```
https://azuremobapptest.azurewebsites.net/tables/TaskModel/058f2c35-7a16-4ba8-ac70-c503f32ea783
```

Response Body


```
{
  "createdAt": "2016-11-30T15:10:15.509Z",
  "status": "1",
  "name": "Complete checkup",
  "id": "058f2c35-7a16-4ba8-ac70-c503f32ea783",
  "address": "Ahvnetie 24 O 159, Vaasa",
  "date": "0001-01-01T00:00:00.000Z",
  "version": "MQ==",
  "startTime": "2017-04-12T13:00:00.000Z",
  "deleted": false,
  "description": "blood pressure checkup, suger level, diet followup...",
  "userId": "sid:22aaf9789ca43f4ed32f283805ba85cf",
  "updatedAt": "2016-11-30T15:10:15.509Z",
  "endTime": "2017-04-12T16:00:00.000Z"
}
```


Response Code

```
200
```

Response Headers


```
{
  "pragma": "no-cache",
  "date": "Sun, 14 May 2017 18:53:17 GMT",
  "content-encoding": "gzip",
  "server": "Microsoft-IIS/8.0",
  "x-powered-by": "Express, ASP.NET",
  "etag": "\"MQ==\"",
  "vary": "Accept-Encoding",
  "content-type": "application/json; charset=utf-8",
  "access-control-expose-headers": "Link,Etag",
  "cache-control": "no-cache",
  "content-length": "436",
  "expires": "0"
}
```


5.7.3 Analysis

The specification for the project required that the Workforce to be a cross-platform mobile application built with Xamarin Platform. At the time, Xamarin.Forms was believed to be the best approach; however, it was later dropped to adopt Native cross-platform approach instead.

At first, Xamarin.Forms proved to be a better approach; since it achieves 96% code sharing (compared to 75% by Native cross-platform approach). However, it was rather recommended for simple applications, and not for applications that require access to platform specific features. This, along with other limitations has forced the application design to shift to the

Native cross-platform development approach which is suitable for more complex cross-platform mobile applications that involve platform specific implementations.

6 CONCLUSION

This project was designed and implemented to improve the traditionally conducted homecare service by providing a better communication network between the two main acting parties, i.e. the field workers and allocators. Workforce solved the unnecessary effort, resources and time invested during the traditional way of communicating between the field workers and the allocators (messaging, phone calls, emails etc). Workforce brings all the necessary entities in making this service sector more effective and efficient and it creates a common platform where field workers get all the necessary information about tasks to be carried out whereas the allocators can allocate and monitor tasks and the assigned field worker.

The technologies chosen and the implementations carried out take the maintainability and scalability of the project into account, Workforce is built using Xamarin, one of the best technologies for cross-platform mobile development, and Microsoft Azure, which is the leading cloud computing provider. It keeps the guidelines of Material design to guarantee a beautiful UI, responsive touch gestures and animations and it also makes use of Android support Library to ensure backward compatibly so that new features and widgets can be applied to earlier Android versions.

The project achieves the objectives set forth in the beginning, however, it can further be improved and the existing code architecture makes it possible to add new features or update the application to a cross-platform app with minimum code effort. Adding a server side implementation of Azure Mobile Apps back-end will also contribute to adding more functionalities and features to Workforce.

As the developer of the project and the author of this document, the most important take away was the opportunity to learn how to design and implement scalable and maintainable software solutions by keeping to today’s software standards.

7 REFERENCES

/1/ About Xamarin

https://www.xamarin.com/about

/2/ Xamarin Platform overview

https://developer.xamarin.com/guides/cross-platform/getting_started/introduction_to_mobile_development/

/3/ Xamarin code sharing (Forms vs Native approach)

https://www.xamarin.com/platform

/4/ Material theming

https://developer.xamarin.com/guides/android/user_interface/material_theme/

/5/ Customizing material theme

https://developer.android.com/training/material/theme.html

/6/ Android Support Library

https://developer.android.com/topic/libraries/support-library/index.html

/7/ About Microsoft Azure

https://docsmsftpdfs.blob.core.windows.net/guides/azure/azure-developer-guide.pdf

/8/ Azure Mobile Apps

https://docs.microsoft.com/en-us/azure/app-service-mobile/app-service-mobile-value-prop

/9/ Microsoft Azure cloud solutions

https://azure.microsoft.com/en-us/solutions/?v=17.23h

/10/ Azure Overview

https://azure.microsoft.com/en-us/overview/what-is-azure/

/11/ Azure Regions

https://azure.microsoft.com/en-us/regions/?v=17.23h

/12/ Azure Portal

https://azure.microsoft.com/en-in/features/azure-portal/

/13/ Azure-Google Authentication

https://docs.microsoft.com/en-us/azure/app-service-mobile/app-service-mobile-how-to-configure-google-authentication