RESOURCE FORECASTING SYSTEM (RFS)

Industrial training report submitted to the GITAM (Deemed to be University)

In partialfulfilment of the requirements for the award of Degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE ENGINEERING

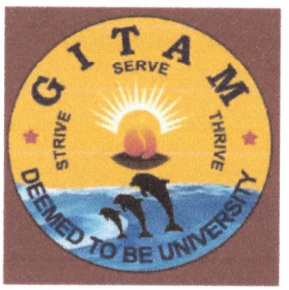
By

B V SAI CHARAN,2210315763

Under the esteemed guidance of

AMJAD KHAN MAHAMMED

PROJECT MANAGER, CYIENT



DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

GITAM SCHOOL OF TECHNOLOGY

GITAM (DEEMED TO BE UNIVERSITY)

HYDERABAD

JULY – 2018

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM INSTITUTE OF TECHNOLOGY

GITAM

(Deemed to be University)

CERTIFICATE

This is to certify that the internship report entitled "**RESOURCE FORECASTING SYSTEM**" is a bonafide record of work carried out by **B V SAI CHARAN (2210315763)** student submitted in partial fulfilment of requirement for the award of degree of Bachelors of Technology in Computer Science andEngineering.

Panel Reviewer 1:-

Panel Reviewer 2:-

Panel Reviewer 3:-

INTERNSHIPREVIEWER

Mr.S.Durga Prasad

GITAM (Deemed To Be University) Hyderabad 502329

CYIENT

28th May 2018

TO WHOMSOEVER IT MAY CONCERN

This is to certify that **Bandi Venkat Sai Charan**, a student of B Tech, Gandhi Institute of Technology and Management, Hyderabad has undergone Internship at Cyient Limited and has successfully completed the Project Work entitled **"Resource Forecasting system - RFS",** during the period from 23rd Apr 2018 to 25th May 2018.

We found **Venkat Sai Charan** to be sincere, hardworking and responsible during his tenure with Cyient. We wish him success in all future endeavors.

For Cyient Limited

M. Vij

Vijay Mavurappu
SENIOR MANAGER - HR,

ACKNOWLEDGEMENT

I express a profound sense of gratitude to my internship guide **Mr. Amjad Khan Mohammed**, Project Manager, CYIENT for the expert guidance and motivation given to me throughout my Internship.

I also express my thanks the Project Coordinator **Mr. B. Rajendra Prasad Babu**, for their valuable suggestions.

I also express my deep gratitude to our beloved **Dr. S. Phani Kumar**, Head of the Department, Department of Computer Science and Engineering for the encouragement and support given to me throughout Internship.

I take this opportunity to thank all the staff of the Department of Computer Science and Engineering and my family for their support during the internship.

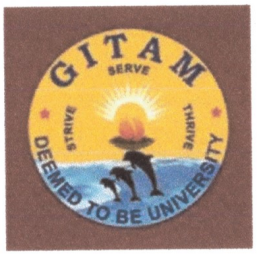
I take this opportunity to place on record my sincere thanks to Honorable Pro-Vice Chancellor **Prof. N. Siva Prasad** GITAM School of Technology, GITAM Deemed to be University for his help and for providing all the required resources for completing internship project till stage.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM INSTITUTE OF TECHNOLOGY

GITAM

(Deemed to be University)



DECLARATION

I, hereby declare that the internship review entitled "**RESOURCE FORECASTING SYSTEM**" is an original work done in the Department of Computer Science and Engineering, GITAM Institute of Technology, GITAM (Deemed to be University) submitted in partial fulfillment of the requirements for the award of the degree of B.Tech. in Computer Science andEngineering.

The work has not been submitted to any other college or University for the award of any degree or diploma.

Date: 3rd Aug 2018

RegistrationNo. 2210315763 Name BV Sai Charan Signature

Table of Contents

ABSTRACT

With the digitalization of almost every available piece of document, there has been an increase in the demand for software's that are personalized for each field according to their requirements. One such attempt to keep a detailed and easily retrievable data about the skill sets of the employees and to give a precise report of the available skills and expertise of the company to the management is RFS - Resource Forecasting System. It is developed to reduce the risk of excess hiring and shortcomings of resources of any required technology.

CHAPTER – 1

INTRODUCTION

The objective is to develop a cross platform application for CYIENT. Ltd which provides the details and appropriate information of the employees and their managerial resources. The purpose is to design an application which enables managers and higher positioned employees to view and select a required candidate from a pool of employee base.

CHAPTER – 2

BRIEF DESCRIPTION OF TECHNOLOGIES LEARNED DURING INTERNSHIP

2.1 C# Language

C# syntax is highly expressive, yet it is also simple and easy to learn. The curly-brace syntax of C# will be instantly recognizable to anyone familiar with C, C++ or Java. Developers who know any of these languages are typically able to begin to work productively in C# within a very short time. C# syntax simplifies many of the complexities of C++ and provides powerful features such as nullable value types, enumerations, delegates, lambda expressions and direct memory access, which are not found in Java. C# supports generic methods and types, which provide increased type safety and performance, and iterators, which enable implementers of collection classes to define custom iteration behaviours that are simple to use by client code. Language-Integrated Query (LINQ) expressions make the strongly-typed query a first-class language construct.

As an object-oriented language, C# supports the concepts of encapsulation, inheritance, and polymorphism. All variables and methods, including the Main method, the application's entry point, are encapsulated within class definitions. A class may inherit directly from one parent class, but it may implement any number of interfaces. Methods that override virtual methods in a parent class require the override keyword as a way to avoid accidental redefinition. In C#, a struct is like a lightweight class; it is a stack-allocated type that can implement interfaces but does not support inheritance.

In addition to these basic object-oriented principles, C# makes it easy to develop software components through several innovative language constructs, including the following:

•Encapsulated method signatures called delegates, which enable type-safe event notifications.

•Properties, which serve as accessors for private member variables.

•Attributes, which provide declarative metadata about types at run time.

•Inline XML documentation comments.

•Language-Integrated Query (LINQ) which provides built-in query capabilities across a variety of data sources.

If interaction with other Windows software such as COM objects or native Win32 DLLs, this in C# through a process called "Interop." Interop enables C# programs to do almost anything that a native C++ application can do. C# even supports pointers and the concept of "unsafe" code for those cases in which direct memory access is absolutely critical.

The C# build process is simple compared to C and C++ and more flexible than in Java. There are no separate header files, and no requirement that methods and types be declared in a particular order. A C# source file may define any number of classes, structs, interfaces, and events.

2.2 MICROSOFT VISUAL STUDIO

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs, as well as web sites, web apps, web services and mobiles apps. Visual Studio uses Microsoft software development platforms such as Windows API, Windows Forms, Windows Presentation Foundation, Windows Store and Microsoft Silverlight. It can produce both native code and managed code.

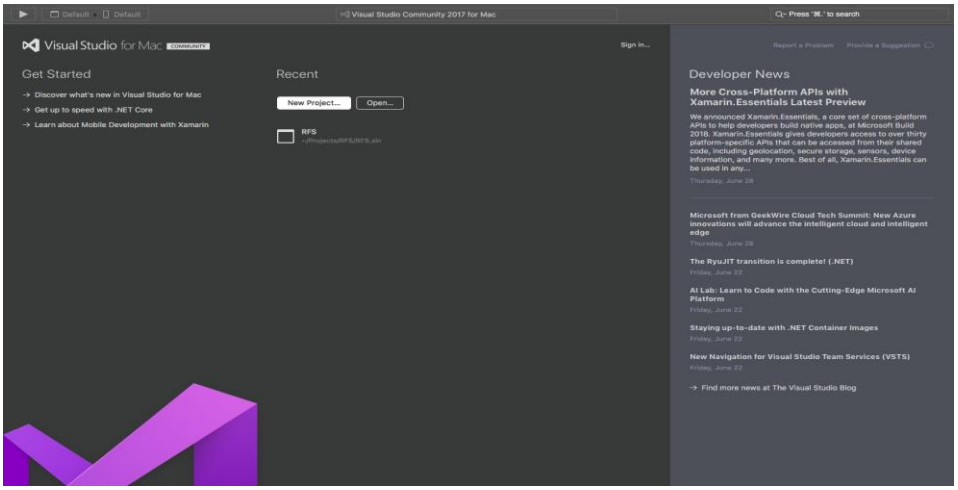


Figure 2.1 Visual Studio

Visual Studio supports 36 different programming language and allows th e code editor and debugger to support (to varying degrees) nearly any programming language, provided a language-specific service exists. Built-in languages include C, C++, C++/CLI, Visual Basic .NET, C#, F#, JavaScript, TypeScript, XML, XSLT, HTML and CSS. Support for other languages such as Python, Ruby, Node.js and M among others is available via plug-ins. Java (and J#) were supported in the past. The most basic edition of Visual Studio, the Community edition, is available free of charge.

2.2.1 Features

Code editor

Like any other IDE, it includes a code editor that supports syntax highlighting and code completion using IntelliSense for variables, functions, methods, loops and

LINQ queries. IntelliSense is supported for the included languages, as well as for XML and for Cascading Style Sheets and JavaScript when developing web sites and web applications. Autocomplete suggestions appear in a modeless list box over the code editor window, in proximity of the editing cursor. In Visual Studio 2008 onwards, it can be made temporarily semi-transparent to see the code obstructed by it. The code editor is used for all supported languages.

Debugger

Visual Studio includes a debugger that works both as a source-level debugger and as a machine-level debugger. It works with both managed code as well as native code and can be used for debugging applications written in any language supported by Visual Studio. In addition, it can also attach to running processes and monitor and debug those processes. If source code for the running process is available, it displays the code as it is being run. If source code is not available, it can show the disassembly. The Visual Studio debugger can also create memory dumps as well as load them later for debugging. Multi-threaded programs are also supported. The debugger can be

configured to be launched when an application running outside the Visual Studio environment crashes.

Designer

Visual Studio includes a host of visual designers to aid in the development of applications. These tools include:

Windows Forms Designer

The Windows Forms designer is used to build GUI applications using Window Forms. Layout can be controlled by housing the controls inside other containers or locking them to the side of the form. Controls that display data (like text box, list box and grid view) can be bound to data sources like database or queries. Data-bound controls can be created by dragging items from the Data Sources window onto a design surface. The UI is linked with code using an event-driven programming model. The designer generates either C# or VB.NET code for the application.

WPF Designer

The WPF designer, codenamed Cider, was introduced with Visual Studio 2008. Like the Windows Forms designer it supports the drag and drop metaphor. It is used to author user interface targeting Windows Presentation Foundation. It supports all WPF functionality including data binding and automatic layout management. It generates XAML code for the UI. The generated XAML file is compatible with Microsoft Expression Design, the designer-oriented product. The XAML code is linked with code using a code-behind model.

Web designer/development

Visual Studio also includes a web-site editor and designer that allows web pages to be authored by dragging and dropping widgets. It is used for developing ASP.NET applications and supports HTML, CSS and JavaScript. It uses a code-behind model to link with ASP.NET code. From Visual Studio 2008 onwards, the layout engine used by the web designer is shared with Microsoft Expression Web. There is also ASP.NET MVC support for MVC technology as a separate download and ASP.NET Dynamic Data project available from Microsoft.

Class designer

The Class Designer is used to author and edit the classes (including its members and their access) using UML modelling. The Class Designer can generate C# and VB.NET code outlines for the classes and methods. It can also generate class diagrams from handwritten classes.

Data designer

The data designer can be used to graphically edit database schemas, including typed tables, primary and foreign keys and constraints. It can also be used to design queries from the graphical view.

Mapping designer

From Visual Studio 2008 onwards, the mapping designer is used by LINQ to SQL to design the mapping between database schemas and the classes that encapsulate the data. The new solution from ORM approach, ADO.NET Entity Framework, replaces and improves the old technology.

2.2.2 Other Tools

Open Tabs Browser

The open tabs browser is used to list all open tabs and to switch between them. It is invoked using CTRL+TAB.

Properties Editor

The Properties Editor tool is used to edit properties in a GUI pane inside Visual Studio. It lists all available properties (both read-only and those which can be set) for all objects including classes, forms, web pages and other items.

Object Browser

The Object Browser is a namespace and class library browser for Microsoft .NET. It can be used to browse the namespaces (which are arranged hierarchically) in managed assemblies. The hierarchy may or may not reflect the organization in the file system.

Solution Explorer

In Visual Studio parlance, a solution is a set of code files and other resources that are used to build an application. The files in a solution are arranged hierarchically, which might or might not reflect the organization in the file system. The Solution Explorer is used to manage and browse the files in a solution.

Team Explorer

Team Explorer is used to integrate the capabilities of Team Foundation Server, the Revision Control System into the IDE (and the basis for Microsoft's CodePlex hosting environment for open source projects). In addition to source control it provides the ability to view and manage individual work items (including bugs, tasks and other documents) and to browse TFS statistics. It is included as part of a TFS install and is also available as a download for Visual Studio separately. Team Explorer is also available as a stand-alone environment solely to access TFS services.

Data Explorer

Data Explorer is used to manage databases on Microsoft SQL Server instances. It allows creation and alteration of database tables (either by issuing T-SQL commands or by using the Data designer). It can also be used to create queries and stored procedures, with the latter in either T-SQL or in managed code via SQL CLR. Debugging and IntelliSense support is available as well.

Server Explorer

The Server Explorer tool is used to manage database connections on an accessible computer. It is also used to browse running Window Services, performance counters, Window Event Log and message queues and use them as a data source.

Pre-emptive Protection-Dotfuscator Community Edition

Visual Studio includes a free 'light' version of Dotfuscator by PreEmptive Solution which obfuscates and hardens applications to help secure trade secrets (IP), reduce piracy/counterfeiting, protect against tampering and unauthorized debugging. Dotfuscator works with all flavours of .NET including ASP.NET, Xamarin, Unity and UWP.

Text Generation Framework

Visual Studio includes a full text generation framework called T4 which enables Visual Studio to generate text files from templates either in the IDE or via code.

ASP.NET Web Site Administration Tool

The ASP.NET Web Site Administration Tool allows for the configuration of ASP.NET websites.

Visual Studio Tools for Office

Visual Studio Tools for Office is a SDK and an add-in for Visual Studio that includes tools for developing for the Microsoft Office suite. Previously (for Visual Studio .NET 2003 and Visual Studio 2005) it was a separate SKU that supported only Visual C# and Visual Basic languages or was included in the Team Suite. With Visual Studio 2008, it is no longer a separate SKU but is included with Professional and higher editions. A separate runtime is required when deploying VSTO solutions.

2.2.3 Editions

Microsoft Visual Studio is available in the following editions or SKUs:

Community

The Community edition was announced on 12 November 2014, as a new free version similar in functionality to Visual Studio Professional. Prior to this date, the only free editions of Visual Studio were the feature-limited Express variants. Unlike the Express variants, Visual Studio Community supports multiple languages, and provides support for extensions. However, the license restricts use to individual developers or to teams developing open source projects. Visual Studio Community is oriented towards individual developers and small teams.

Professional

As of Visual Studio 2010, the Professional edition is the entry level commercial editor of Visual Studio. (Previously, a more feature restricted Standard edition was available.) It provides an IDE for all supported development languages. MSDN support is available as MSDN Essentials or the full MSDN library depending

on licensing. It supports XML and XSLT editing, and can create deployment packages that only use ClickOnce and MSI. It includes tools like Server Explorer and integration with Microsoft SQL Server also. Windows Mobile development support was included in Visual Studio 2005 Standard, however, with Visual Studio 2008, it is only available in Professional and higher editions. Windows Phone 7 development support was added to all editions in Visual Studio 2010.

Development for Windows Mobile is no longer supported in Visual Studio 2010; it is superseded by Windows Phone 7.

Enterprise

In addition to the features provided by the Professional edition, the Enterprise edition provides a new set of software development, database development, collaboration, metrics, architecture, testing and reporting tools.

Visual Studio Code

Visual Studio Code is a source code editor, along with other features, for Linux, macOS, and Windows. It also includes support for debugging and embedded Git Control. It is open-source, and on 14 April 2016 was released as version 1.0.

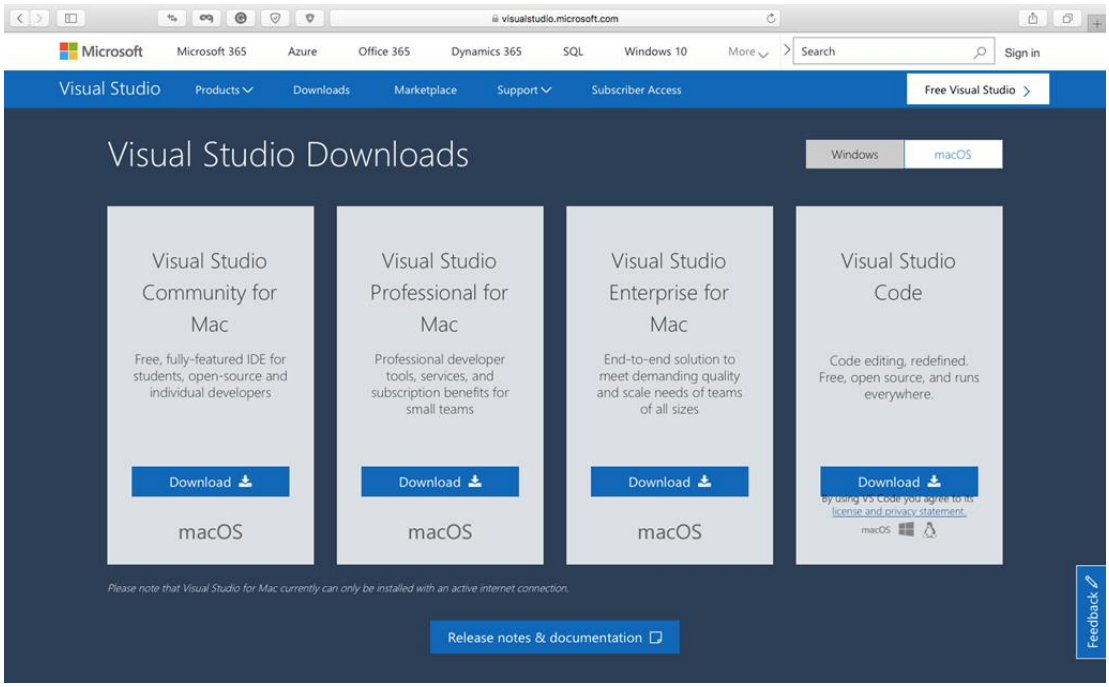


Figure 1.2 Visual Studio for Mac

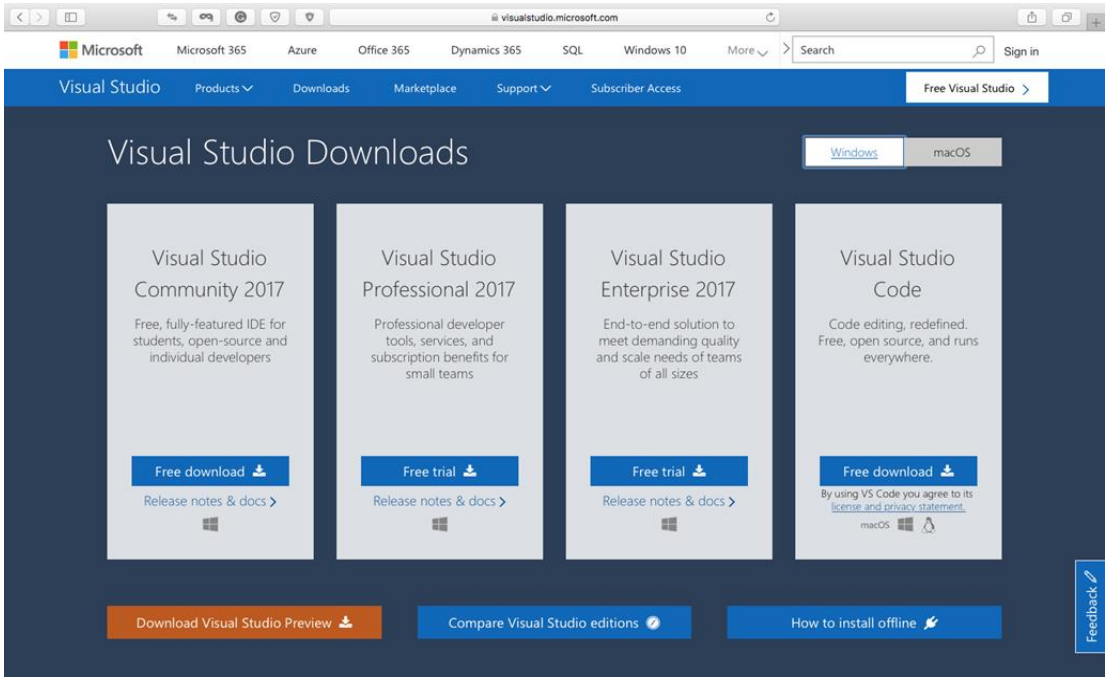


Figure 2.3 Visual Studio for Windows

2.3 XAMARIN

Xamarin is a Microsoft-owned San Francisco, California-based software company founded in May 2011 by the engineers that created Mono, Mono for Android and MonoTouch, which are cross-platform implementations of the Common Language Infrastructure (CLI) and Common Language Specifications (often called Microsoft .NET).

With a C#-shared codebase, developers can use Xamarin tools to write native Android, iOS, and Windows apps with native user interface and share code across multiple platforms, including Windows and macOS. According to Xamarin, over 1.4 million developers were using Xamarin's products in 120 countries around the world as of April 2017.

On February 24, 2016, Microsoft announced it had signed a definitive agreement to acquire Xamarin.

2.3.1 PLATFORM

2.3.1.1 Xamarin.Android

Xamarin.Android exposes the complete Android SDK for .NET developers. Build fully native Android apps using C# or F# in Visual Studio.

2.3.1.2 Application Fundamentals

- Accessibility

The Android Accessibility APIs to build apps according to the accessibility checklist.

- Android API Levels

Android uses API levels to manage app compatibility across different versions of Android, and it explains to configure Xamarin.Android project settings to deploy these API levels in the app. To write runtime code that deals with different API levels, and it provides a reference list of all Android API levels, version numbers (such as Android 8.0), Android code names (such as Oreo), and build version codes.

- Resources in Android

Android is the way to use resources in android application to support application localization, and multiple devices including varying screen sizes and densities.

- Activity Lifecycle

Activities are a fundamental building block of Android Applications and it can exist in several different states. The activity lifecycle begins with instantiation and ends with destruction and includes many states in between. When an activity changes state, the appropriate lifecycle event method is called notifying the activity of the impending state change and allowing it to execute code to adapt to that change. This will examine the lifecycle of activities and explains the responsibility that an activity has during each of these state changes to be part of a well-behaved, reliable application.

- Localization

It localizes a Xamarin.Android into other languages by translating strings and providing alternate images.

- Services

Android services are components that allow work to be done in the background. It explains the different scenarios that services are suited for and show to implement them both for performing long-running background tasks as well as to provide an interface for remote procedure calls.

- Broadcast Receivers

It is an Android component that responds to system-wide broadcasts, in Xamarin.Android.

- Permissions

The tooling support built into Visual Studio for Mac is to create and add permissions to the Android Manifest. This adds permissions in Visual Studio and Xamarin Studio.

- Graphics and Animations

Android provides a very rich and diverse framework for supporting 2D graphics and animations. These frameworks create custom graphics and animations and use them in a Xamarin.Android application.

- CPU Architectures

Xamarin.Android supports several CPU architectures, including 32-bit and 64-bit devices. This target an app to one or more Android-supported CPU architectures.

- Handling Rotation

Handle devices orientation changes in Xamarin.Android. work with the Android resource system to automatically load resources for a device orientation as well as to programmatically handle orientation changes and help in maintaining state when a device is rotated.

• Android Audio

The Android OS provides extensive support for multimedia, encompassing both audio and video. This Android covers playing and recording audio using the built-in audio player and recorder classes, as well as the low-level audio API. It also works with Audio events broadcast by other applications, so that developers can build well-behaved applications.

• Notifications

Implement local and remote notifications in Xamarin.Android and various UI elements of an Android notification. It also shows the API's involvement with creating and displaying a notification for remote notifications, both Google Cloud Messaging and Firebase Cloud Messaging are explained. Step-by-step walkthroughs and code samples are included.

• Touch

Concepts and details of implementing touch gestures on Android. Touch APIs are introduced by an exploration of gesture recognizers.

• HttpClient Stack and SSL/TLS

This are Implementation selectors for Android. These settings determine and implements that are used by Xamarin.Android apps.

• Writing Responsive Application

Threading to keep a Xamarin.Android application responsive by moving long-running tasks on to a background thread.

2.3.1.3 Platform Features

Features specific to Android will be known, topics such as using Fragments, working with maps and encapsulating data with Content Providers.

• Android Beam

Android Beam is a new Near Field Communication (NFC) technology in Android 4 that allows applications to share information over NFC when near.

• Fingerprint Authentication

Fingerprint authentication was first introduced in Android 6.0 to a Xamarin.Android application.

• Firebase Job Dispatcher

This discusses the Firebase Job Dispatcher and use it to simplify running background jobs in a Xamarin.Android app.

• Fragments

Android 3.0 introduced Fragments, showing support more flexible designs for the many different screen sizes found on phones and tablets. This will cover Fragments to

develop Xamarin.Android applications, and to support Fragments on pre-Android 3.0 (API Level 11) devices.

• App-Linking
A technique that allows mobile apps to respond to URLs on websites and implement app-linking in an Android 6.0 application and to configure a website to grant permissions to the mobile app to handle app-links for the domain.

• Android 8 Oreo
Provides an outline of the new features in Android Oreo and to prepare Xamarin.Android for Android Oreo development and provides links to sample applications that illustrate usage of Android Oreo features in Xamarin.Android apps.

• Android 7 Nougat
Provides overview of the new features introduced in Android 7.0 Nougat.

• Android 6 Marshmallow
Provides overview of the new features introduced in Android 6.0 Marshmallow.

• Android 5 Lollipop
Provides an overview of new Android 5.0 Lollipop features such as Material Theme, Card View, RecyclerView, and Heads Up Notifications, and it links to in-depth that helps the user to use new features in the app.

• Android 4.4 KitKat
Android 4.4 (KitKat) comes loaded with a cornucopia of features for users and developers both. Highlights several of these features and provides code examples and implementation details to help user to make the most out of KitKat.

• Android 4.1 Jelly Bean
Provide a high-level overview of the new features for developers that were introduced in Android 4.1. such as enhanced notifications, updates to Android Beam to share large files, updates to multimedia, peer-to-peer network discovery, animations, new permissions.

• Android 4.0 Ice Cream Sandwich
Several of the new features available to application developers with the Android 4 API - Ice Cream Sandwich, such as user interface technologies and then examines a variety of new capabilities that Android 4 offers for sharing data between applications and between devices.

• Working with the Android Manifest
Introduces the AndroidManifest.xml file and be used to control functionality and describe the requirements of a Mono for Android application.

• Introduction to Content Providers
A ContentProvider encapsulates a data repository and provides an API to access it. The provider exists as part of an Android application that usually also provides a UI

for displaying/managing the data. The key benefit of using a content provider is enabling other applications to easily access the encapsulated data using a provider client object (called a ContentResolver). Together a content provider and content resolver offer a consistent inter-application API for data access that is simple to build and consume. This access and build Content Providers with Xamarin.Android.

• Maps and Location
Use maps and location with Xamarin.Android. leveraging the built-in maps application to the Google Maps Android API v2 directly. Additionally, it explains use of a single API to work with location services, which use cellular triangulation to allow an application to obtain location fixes, Wi-Fi location, and GPS.

• Android Speech
Android Text to Speech and Speech to Text facilities. It also covers installing language packs and interpretation of the text spoken to the device.

• Binding a Java Library
Incorporate Java libraries into Xamarin.Android apps by creating a Bindings Library.

• Java Integration
Provides an overview of the ways that developers can reuse existing Java components in Xamarin.Android apps.

• Renderscript
This discusses Renderscript.

2.3.1.4 Xamarin.Essentials

Xamarin.Essentials provides developers with cross-platform APIs for their mobile applications.
Xamarin.Essentials provides a single cross-platform API that works with any Xamarin.Forms, Android, iOS, or UWP application that can be accessed from shared code no matter what the user interface is created.

- integrate these Xamarin.Essentials features into the application:

- Accelerometer – Retrieve acceleration data of the device in three-dimensional space.
- App Information – Find out information about the application.
- Battery – Easily detect battery level, source, and state
- Clipboard – Quickly and easily set or read text on the clipboard.
- Compass – Monitor compass for changes.
- Connectivity – Check connectivity state and detect changes.
- Data Transfer – Send text and website URLs to other apps.
- Device Display Information – Get the device's screen metrics and orientation.
- Device Information – Find out about the device with ease.
- Email – Easily send email messages.
- File System Helpers – Easily save files to app data.
- Flashlight – A simple way to turn the flashlight on/off.
- Geocoding – Geocode and reverse geocode addresses and coordinates.
- Geolocation – Retrieve the device's GPS location.
- Gyroscope – Track rotation around the device's three primary axes.

- Magnetometer – Detect device's orientation relative to Earth's magnetic field.
- Main Thread – Run code on the application's main thread.
- Open Browser – Quickly and easily open a browser to a specific website.
- Orientation Sensor – Retrieve the orientation of the device in three-dimensional space.
- Phone Dialer – Open the phone dialer.
- Power – Obtain the device's energy-saver status.
- Preferences – Quickly and easily add persistent preferences.
- Screen Lock – Keep the device screen awake.
- Secure Storage – Securely store data.
- SMS – Create an SMS message for sending.
- Text-to-Speech – Vocalize text on the device.
- Version Tracking – Track the applications version and build numbers.
- Vibrate – Make the device vibrate.

- Troubleshooting

Provides helps when running into issues.

- API Documentation

Browse the API documentation for every feature of Xamarin.Essentials.

2.3.1.5 Data and Cloud Services

Data and Cloud Services

Xamarin.Android applications often need access to data (from either a local database or from the cloud), and many of apps consume web services implemented using a wide variety of technologies. In this examine to access data and make use of cloud services is known.

- Data Access

Data access in Xamarin.Android is done using SQLite as the database engine.

- Google Messaging

Google provides both Firebase Cloud Messaging and legacy Google Cloud Messaging services for facilitating messaging between mobile apps and server applications. This provides overviews for each service by step-by-step explanation on the use of these services to implement remote notifications (also called push notifications) in Xamarin.Android applications.

2.3.1.6 Deployment and Testing

This section includes testing an application, optimize its performance, prepare it for release, sign it with a certificate, and publish it to an app store.

-Application Package Sizes

This examines the constituent parts of a Xamarin.Android application package and the associated strategies that can be used for efficient package deployment during debug and release stages of development.

- Building Apps

Describes the build process works and build ABI-specific APKs.

- Command Line Emulator

Starting the emulator via the command line.

- Debugging

Debugs app using Android emulators, real Android devices, and the debug log.

- Setting the Debuggable Attribute

This explains the debuggable attribute so that tools such as adb can communicate with the JVM.

- Environment

Describes the Xamarin.Android execution environment and the Android system properties that influence program execution.

- GDB

Used for debugging a Xamarin.Android application.

- Installing a System App

Installing a Xamarin.Android app as a System Application on an Android device or as part of a custom ROM is known.

- Linking on Android

This discusses the linking process used by Xamarin.Android to reduce the final size of an application. The various levels of linking that can be performed and provides some guidance and troubleshooting advice to mitigate errors result from using the linker.

- Xamarin.Android Performance

There are many techniques for increasing the performance of applications built with Xamarin.Android. Collectively these techniques can greatly reduce the amount of work being performed by a CPU and the amount of memory consumed by an application.

- Profiling Android Apps

Profiler tools to examine the performance and memory usage of an Android app.

- Preparing an Application for Release

After an application has been coded and tested, it is necessary to prepare a package for distribution. The first task in preparing this package is to build the application for release, which mainly entails setting some application attributes.

- Signing the Android Application Package

Learn to create an Android signing identity, create a new signing certificate for Android applications, and sign the application with the signing certificate. In addition, this topic explains exporting the app to disk for ad-hoc distribution. The resulting APK can be sideloaded into Android devices without going through an app store.

- Publishing an Application

Steps for public distribution of an application created with Xamarin.Android is known. Distribution can take place via channels such as e-mail, a private web server, Google Play, or the Amazon App Store for Android.

2.3.1.7 Advanced Concepts and Internals

This section contains topics that details the architecture, API design, and limitations of Xamarin.Android. In addition, it includes garbage collection implementation and the assemblies that are available in Xamarin.Android, because Xamarin.Android is open-source, it is also possible to understand the inner workings of Xamarin.Android by examining its source code.

- Architecture

The underlying architecture behind a Xamarin.Android application and Xamarin.Android applications run inside a Mono execution environment alongside with the Android runtime Virtual Machine. Explains key concepts such as Android Callable Wrappers and Managed Callable Wrappers.

- API Design

In addition to the core Base Class Libraries that are part of Mono, Xamarin.Android ships with bindings for various Android APIs to allow developers to create native Android applications with Mono. At the core of Xamarin.Android there is an interop engine that bridges the C# world with the Java world and provides developers with access to the Java APIs from C# or other .NET languages.

- Assemblies

Xamarin.Android ships with several assemblies. Just as Silverlight is an extended subset of the desktop .NET assemblies, Xamarin.Android is also an extended subset of several Silverlight and desktop .NET assemblies.

CHAPTER - 3

ANALYSIS OF APPLICATION WORKED/DEVELOPED

3.1 RESOURCE FORECASTING SYSTEM APPLICATION

3.1.1 Project Info/ Application Description

Using RFS Employees can register their skills experience and the expertise in all the skills through web and mobile applications. Respective Managers can approve the skills. This gives the management a bigger picture of all the available skills and expertise within the company to plan and forecast the new & existing projects which reduces the risk of excess hiring and shortcomings of resources for any given technology.

3.1.2 Project Application Code

3.1.2.1 Splash Screen

A splash screen is a graphical control element consisting of a window containing an image, a logo, and the current version of the software. A splash screen usually appears while a game or program is launching. A splash page is an introduction page. A splash screen may cover the entire screen or web page; or may simply be a rectangle near the centre of the screen or page. The splash screens of operating systems and some applications that expect to be run in full screen usually cover the entire screen.

Splash screens are typically used by particularly large applications to notify the user that the program is in the process of loading. They provide feedback that a lengthy process is underway. Occasionally, a progress bar within the splash screen indicates the loading progress. A splash screen disappears when the application's main window appears.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using Android.App;
using Android.Content;
using Android.OS;
using Android.Runtime;
using Android.Views;
using Android.Widget;

namespace RFS
{
   //Set MainLauncher = true makes this Activity Shown First on Running this Application
   //Theme property set the Custom Theme for this Activity
   //No History= true removes the Activity from BackStack when user navigates away from
the Activity
   [Activity(Label = "RFS", MainLauncher = true, Theme = "@style/Theme.Splash", NoHistory
= true, Icon = "@drawable/logo_3")]
   public class SplashScreen : Activity
   {
      protected override void OnCreate(Bundle bundle)
      {
         base.OnCreate(bundle);
         //Display Splash Screen for 0.5 Sec
         Thread.Sleep(500);
         //Start MAinActivity
         StartActivity(typeof(MainActivity));
      }}}
```

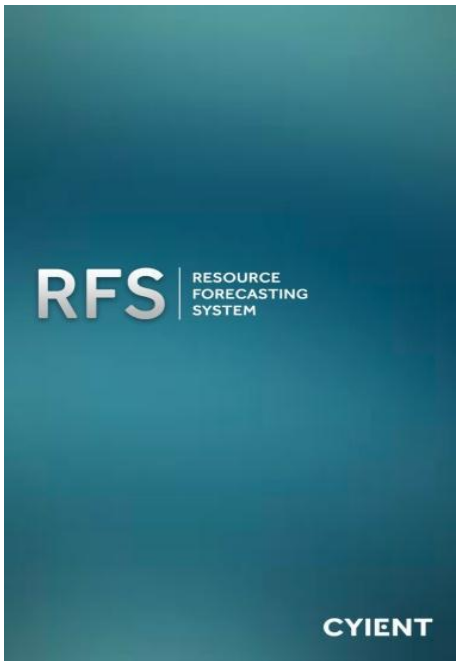


Figure 3.1 Splash Screen

3.1.2.2 Login Page

From here the use can login to the app to access RFS.

The user credentials are typically some form of "username" and a matching "password", and these credentials themselves are sometimes referred to as a login, (or a logon or a sign-in or a sign-on). In practice, modern secure systems also often require a second factor for extra security.

```
using Android.App;
using Android.Widget;
using Android.OS;
using Android.Content;
using Android.Content.PM;

namespace RFS
{
  [Activity(Label = "RFS", MainLauncher = true, Theme =
"@android:style/Theme.Holo.Light",ScreenOrientation = ScreenOrientation.Portrait)]
    public class MainActivity : Activity
    {
public static string corpID;
protected override void OnCreate(Bundle savedInstanceState)
{
base.OnCreate(savedInstanceState);
  // Set our view from the "main" layout resource
 SetContentView(Resource.Layout.Main);  ActionBar.Hide();
 EditText e1 = FindViewById<EditText>(Resource.Id.input_ID);
 EditText e2 = FindViewById<EditText>(Resource.Id.input_PWD);
 Button logBut = FindViewById<Button>(Resource.Id.btn_login);
 logBut.Click += B1_Click;
}

void B1_Click(object sender, System.EventArgs e)
{
string logVal = FindViewById<EditText>(Resource.Id.input_ID).Text;
string logPass = FindViewById<EditText>(Resource.Id.input_PWD).Text;
corpID = logVal;

//put below two lines in the if else block to authenticate
if (logPass == "xyz")
{
Toast.MakeText(this, logVal+" Login Successful", ToastLength.Short).Show();
Intent i = new Intent(this, typeof(menu_act));
StartActivity(i);
}
```

```
else
{
Toast.MakeText(this, "Incorrect Credentials", ToastLength.Short).Show();
}}}}
```

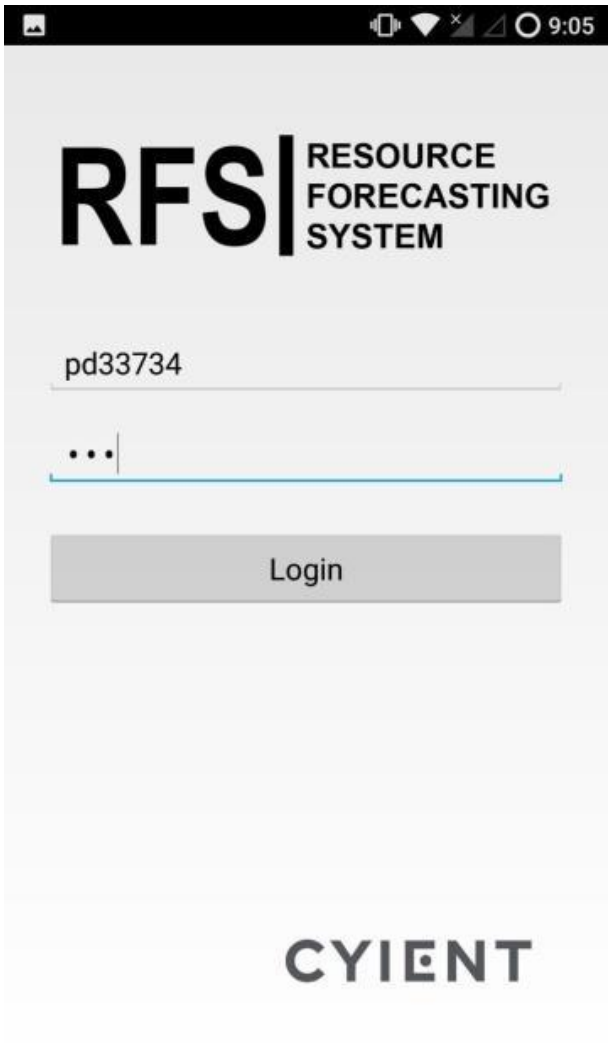


Figure 3.2 Login Page

3.1.2.3 Dashboard

Contains brief user details and access to intents of all pages in the app. From here the user can navigate to desired page.

A home page on a website giving access to different elements of the Intents functionality.

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using Android.App;
using Android.Content;
using Android.Content.PM;
using Android.Graphics;
using Android.Media;
using Android.OS;
using Android.Runtime;
using Android.Views;
using Android.Widget;
using Java.Lang;


namespace RFS
{
 [Activity(Label = "menu_act", ScreenOrientation = ScreenOrientation.Portrait)]
public class menu_act : Activity
{
int x = 0;
protected override void OnCreate(Bundle savedInstanceState)
{
base.OnCreate(savedInstanceState);
// Create your application here
SetContentView(Resource.Layout.menu);
ActionBar.Hide();
Button b1 = FindViewById<Button>(Resource.Id.button1);
b1.Click += B1_Click;
Button b2 = FindViewById<Button>(Resource.Id.button2);
b2.Click += B2_Click;
Button b3 = FindViewById<Button>(Resource.Id.button3);
b3.Click += B3_Click;
ImageView logiv = FindViewById<ImageView>(Resource.Id.iv2);
logiv.Click += Logiv_Click;
TextView name = FindViewById<TextView>(Resource.Id.nameID);
Typeface tf = Typeface.CreateFromAsset(Assets, "effra.ttf");
b1.SetTypeface(tf, TypefaceStyle.Normal);
b2.SetTypeface(tf, TypefaceStyle.Normal);
b3.SetTypeface(tf, TypefaceStyle.Normal);
name.SetTypeface(tf, TypefaceStyle.Normal);

Button test1 = FindViewById<Button>(Resource.Id.test1);
test1.Click += Test1_Click;
x = 1;
if(x==1)
{
test1.PerformClick();
}}
```

```
void B1_Click(object sender, System.EventArgs e)
{
Intent a = new Intent(this, typeof(res_act));
a.PutExtra("corpid", "");
StartActivity(a);
}

void B2_Click(object sender, System.EventArgs e)
{
Intent b = new Intent(this, typeof(prof_act));
StartActivity(b);
}

void B3_Click(object sender, System.EventArgs e)
{
Intent c = new Intent(this, typeof(search_act));
StartActivity(c);
}


void Test1_Click(object sender, EventArgs e)
{
try
{
TextView txtview = FindViewById<TextView>(Resource.Id.nameID);
rfs.Addnew obj = new rfs.Addnew();
string s = obj.GetUsers(MainActivity.corpID);
DataTable dt = ConvertJSONToDataTable(s);
txtview.Text =Convert.ToString(dt.Rows[0][1]);
Toast.MakeText(this, "Welcome,  " +txtview.Text, ToastLength.Short).Show();
}
catch (System.Exception ex)
{
Toast.MakeText(this, ex.Message, ToastLength.Long).Show();
}}

private DataTable ConvertJSONToDataTable(string jsonString)
{
// ErrorHandling.CreateLogfile(AppDomain.CurrentDomain.BaseDirectory.ToString());
//
ErrorHandling.CreateLogfile(ConfigurationManager.AppSettings["LogFolderPath"].ToString()
);
DataTable dt = new DataTable();
//strip out bad characters
string[] jsonParts = Regex.Split(jsonString.Replace("[", "").Replace("]", ""), "},{");
//hold column names
List<string> dtColumns = new List<string>();
//get columns
foreach (string jp in jsonParts)
{
//only loop thru once to get column names
```

```
string[] propData = Regex.Split(jp.Replace("{", "").Replace("}", ""), ",");
foreach (string rowData in propData)
{
try
{
int idx = rowData.IndexOf(":");
string n = rowData.Substring(0, idx - 1);
string v = rowData.Substring(idx + 1);
if (!dtColumns.Contains(n))
{
dtColumns.Add(n.Replace("\"", ""));
}}
catch (System.Exception ex)
{
// ErrorHandling.WriteLine("Error: Manage User" + ex.Message);
throw new System.Exception(string.Format("Error Parsing Column Name : {0}", rowData));
}}
break; // TODO: might not be correct. Was : Exit For
}
//build dt
foreach (string c in dtColumns)
{
dt.Columns.Add(c);
}
//get table data
foreach (string jp in jsonParts)
{
string[] propData = Regex.Split(jp.Replace("{", "").Replace("}", ""), ",");
DataRow nr = dt.NewRow();

foreach (string rowData in propData)
{
try
{
int idx = rowData.IndexOf(":");
string n = rowData.Substring(0, idx - 1).Replace("\"", "");
string v = rowData.Substring(idx + 1).Replace("\"", "");
nr[n] = v;
}
catch (System.Exception ex)
{
//ErrorHandling.WriteLine("Error: Manage User" + ex.Message);
continue;
}}
dt.Rows.Add(nr);
}
return dt;
}
void Logiv_Click(object sender, EventArgs e)
{
Android.App.AlertDialog.Builder dialog = new AlertDialog.Builder(this);
```

```
AlertDialog alert = dialog.Create();
alert.SetTitle("Log Out");
alert.SetMessage("Are You Sure");
alert.SetButton("OK", (c, ev) =>
{
Intent lo = new Intent(this, typeof(MainActivity));
StartActivity(lo);
// Ok button click task
});
alert.SetButton2("CANCEL", (c, ev) => { });
alert.Show();
}}}
```

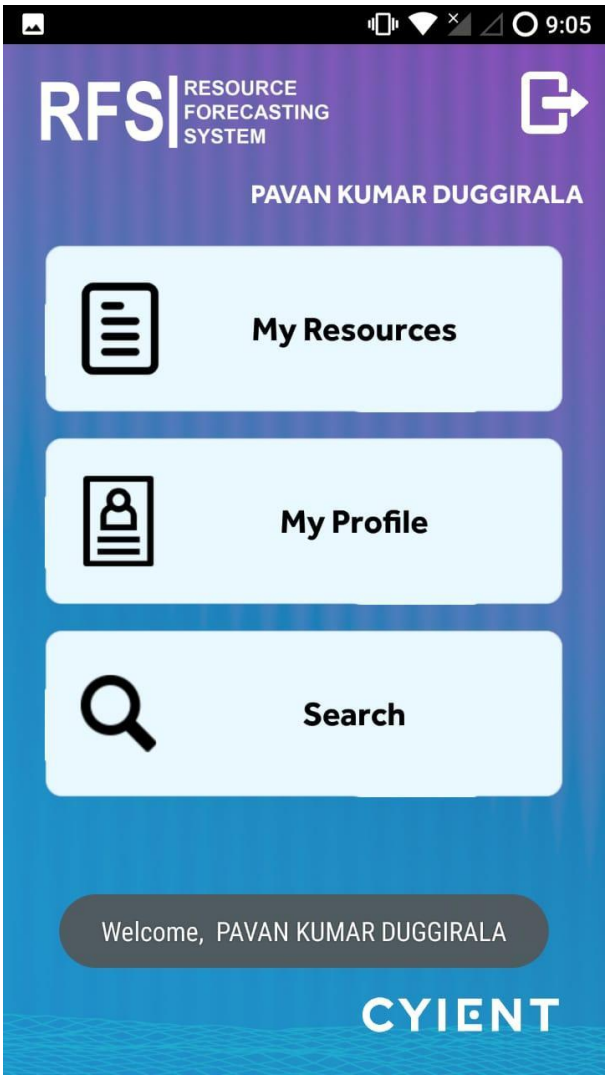


Figure 3.3 Dashboard

3.1.2.4 My Resources

Visual representation of direct and indirect reportees working under the manager. Clicking on the profile will generate a new page which creates a new list of further branches.

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using Android.App;
using Android.Content;
using Android.Content.PM;
using Android.OS;
using Android.Runtime;
using Android.Views;
using Android.Widget;


namespace RFS
{
 [Activity(Label = "RFS", Theme = "@android:style/Theme.Holo.Light", ScreenOrientation =
ScreenOrientation.Portrait)]
public class res_act : Activity
{
protected override void OnCreate(Bundle savedInstanceState)
{
base.OnCreate(savedInstanceState);
SetContentView(Resource.Layout.res);
Button test3 = FindViewById<Button>(Resource.Id.test3);
test3.Click += Test3_Click;
Button home = FindViewById<Button>(Resource.Id.homebut);
home.Click+= Home_Click;
x3 = 1;
if (x3 == 1)
{
test3.PerformClick();
}
ActionBar.Hide();
}


void Home_Click(object sender, EventArgs e)
{
Intent menu = new Intent(this, typeof(menu_act));
```

```
StartActivity(menu);
}


void Test3_Click(object sender, EventArgs e)
{
//GetOrgChart(MainActivity.corpID, 1);
var selecteditem = Intent.Extras.GetString("corpid");
if (selecteditem.ToString() != "")
{
//Toast.MakeText(this, selecteditem.ToString(), ToastLength.Long).Show();
GetOrgChart(selecteditem.ToString(), 1);
}
  else
{
GetOrgChart(MainActivity.corpID, 1);
}}
int x3 = 0;
ListView list;
string[] listName = { };
//int[] imageId = { };
string[] listDept = {  };
int[] listDr = {  };
int[] listIr = {  };


string[] corpid = { };//corp id
List<string> lstres = new List<string>();
List<string> lstres1 = new List<string>();
List<int> lstres2 = new List<int>();
List<int> lstres3 = new List<int>();
List<string> lstres4 = new List<string>();//corp id
public void GetOrgChart(string username, int level)
{
try
{
TextView homename = FindViewById<TextView>(Resource.Id.homebar);
//list.ItemSource = myobservablecollection;
//myobservablecollection.Clear();
//Toast.MakeText(this, "This toast", ToastLength.Short).Show();
rfsres.ResourcesApprovals objres = new rfsres.ResourcesApprovals();
string name = objres.GetMyOrgChart(username, level);
string PresentDesignation = objres.GetMyOrgChart(username, level);
string DirectReportees = objres.GetMyOrgChart(username, level);
string IndirectReportees = objres.GetMyOrgChart(username, level);
string CorpId = objres.GetMyOrgChart(username, level);//corp id
string homenamehigh = objres.GetMyOrgChart(username, 2);//home name


DataTable dtres = ConvertJSONToDataTable(name);
DataTable dtres1 = ConvertJSONToDataTable(PresentDesignation);
DataTable dtres2 = ConvertJSONToDataTable(DirectReportees);
```

```
DataTable dtres3 = ConvertJSONToDataTable(IndirectReportees);
DataTable dtres4 = ConvertJSONToDataTable(CorpId);//corp id
DataTable dtres5 = ConvertJSONToDataTable(homenamehigh);
homename.Text = Convert.ToString(dtres5.Rows[0][2]);//home name

int len = dtres.Rows.Count;
Console.WriteLine("Entered ---------------------------------------->"+len);
if (dtres != null && dtres.Rows.Count > 0)
{
for (int i = 1; i < len; i++)
{
/*Toast.MakeText(this, "This is first", ToastLength.Short).Show();
imageId[i] = Resource.Drawable.ic_launcher;
Toast.MakeText(this, "This is second", ToastLength.Short).Show();*/
Console.WriteLine("Entered looop---------------------------------------->");
string RMName = Convert.ToString(dtres.Rows[i][2]);
string presentDesignation = Convert.ToString(dtres1.Rows[i][5]);
string directReportees = Convert.ToString(dtres2.Rows[i][9]);
string indirectReportees = Convert.ToString(dtres3.Rows[i][10]);
string corpId = Convert.ToString(dtres4.Rows[i][1]);//corp id
int directReportees1= Convert.ToInt32(directReportees);
int indirectReportees1 = Convert.ToInt32(indirectReportees);



lstres.Add(RMName);
lstres1.Add(presentDesignation);
lstres2.Add(directReportees1);
lstres3.Add(indirectReportees1);
lstres4.Add(corpId);//corp id
listName = lstres.ToArray();
listDept = lstres1.ToArray();
listDr = lstres2.ToArray();
listIr = lstres3.ToArray();
corpid= lstres4.ToArray();//corp id
  }}


CustomAdapterList adapter = new CustomAdapterList(this, listName, listDept, listDr,
listIr,corpid);
list = (ListView)FindViewById(Resource.Id.listView);
list.Adapter = adapter;
list.ItemClick += List_ItemClick;
}
catch (Exception ex)
{
Toast.MakeText(this, ex.Message, ToastLength.Long).Show();
}}
private DataTable ConvertJSONToDataTable(string jsonString)
{
DataTable dt = new DataTable();
string[] jsonParts = Regex.Split(jsonString.Replace("[", "").Replace("]", ""), "},{");
```

```
List<string> dtColumns = new List<string>();
foreach (string jp in jsonParts)
{
string[] propData = Regex.Split(jp.Replace("{", "").Replace("}", ""), ",");
foreach (string rowData in propData)
{
try
{
int idx = rowData.IndexOf(":");
string n = rowData.Substring(0, idx - 1);
string v = rowData.Substring(idx + 1);
if (!dtColumns.Contains(n))
{
dtColumns.Add(n.Replace("\"", ""));
}}
catch (System.Exception ex)
{
throw new System.Exception(string.Format("Error Parsing Column Name : {0}", rowData));
}}


break;
}
foreach (string c in dtColumns)
{
dt.Columns.Add(c);
}
foreach (string jp in jsonParts)
{
string[] propData = Regex.Split(jp.Replace("{", "").Replace("}", ""), ",");
DataRow nr = dt.NewRow();
foreach (string rowData in propData)
{
try
{
int idx = rowData.IndexOf(":");
string n = rowData.Substring(0, idx - 1).Replace("\"", "");
string v = rowData.Substring(idx + 1).Replace("\"", "");
nr[n] = v;
}
catch (System.Exception ex)
{
//ErrorHandling.WriteLine("Error: Manage User" + ex.Message);
continue;
}}
dt.Rows.Add(nr);
}
return dt;
}
private void List_ItemClick(object sender, AdapterView.ItemClickEventArgs e)
{
Toast.MakeText(this, "You Clicked at " + listName[e.Position], ToastLength.Long).Show();
```

```
//((list)sender).SelectedItem = null;
//GetOrgChart("GA42141", 1);
}}}
```

3.1.2.5 CustomAdapterList

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Android.App;
using Android.Content;
using Android.OS;
using Android.Runtime;
using Android.Views;
using Android.Widget;


namespace RFS
{
class CustomAdapterList : BaseAdapter
{
private Activity context;
private string[] listName;
//private int[] imageId;
private string[] listDept;
private int[] listDr;
private int[] listIr;
private string[] corpid;
//List<int> arrayList;
public override int Count
{
get
{
return listName.Length;
} }


public CustomAdapterList(Activity context, string[] listName, string[] listDept, int[] listDr,
int[] listIr,string[] corpid)
 {
this.context = context;
this.listName = listName;
// this.imageId = imageId;
this.listDept = listDept;
this.listDr = listDr;
this.listIr = listIr;
this.corpid = corpid;
}
```

```
public override Java.Lang.Object GetItem(int position)
{
return null;
}


public override long GetItemId(int position)
{
return listName.Length;  }
public override View GetView(int position, View convertView, ViewGroup parent)
{
var view = context.LayoutInflater.Inflate(Resource.Layout.list, parent, false);
TextView txtname = (TextView)view.FindViewById(Resource.Id.emp_name);
//ImageView imageView = (ImageView)view.FindViewById(Resource.Id.profile_img);
TextView txtdept = (TextView)view.FindViewById(Resource.Id.emp_dept);
Button txtdr = (Button)view.FindViewById(Resource.Id.dr);
Button txtIr = (Button)view.FindViewById(Resource.Id.ir);


txtdr.Click+=Txtdr_Click;
txtname.Text = (listName[position]).ToString();
//imageView.SetImageResource(imageId[position]);
txtdept.Text = (listDept[position]).ToString();
txtdr.Text = (listDr[position]).ToString();
txtIr.Text = (listIr[position]).ToString();


//int b = listDr.GetValue();
//Console.WriteLine("this--> " + b);


void Txtdr_Click(object sender, EventArgs e)
{
//Intent directreport;
switch(listDr[position])
{
case 0: break;
default:
//((res_act)parent.Context).
//((res_act)parent.Context).GetOrgChart((corpid[position]),2);
Intent a = new Intent(parent.Context, typeof(res_act));
a.PutExtra("corpid", corpid[position]);
context.StartActivity(a);
//GetOrgChart myobj=new GetOrgChart();
//directreport = new Intent(parent.Context, typeof(direct_report));
// context.StartActivity(directreport);
break;
}}
return view;
}}}
```

Figure 3.4 My Resources

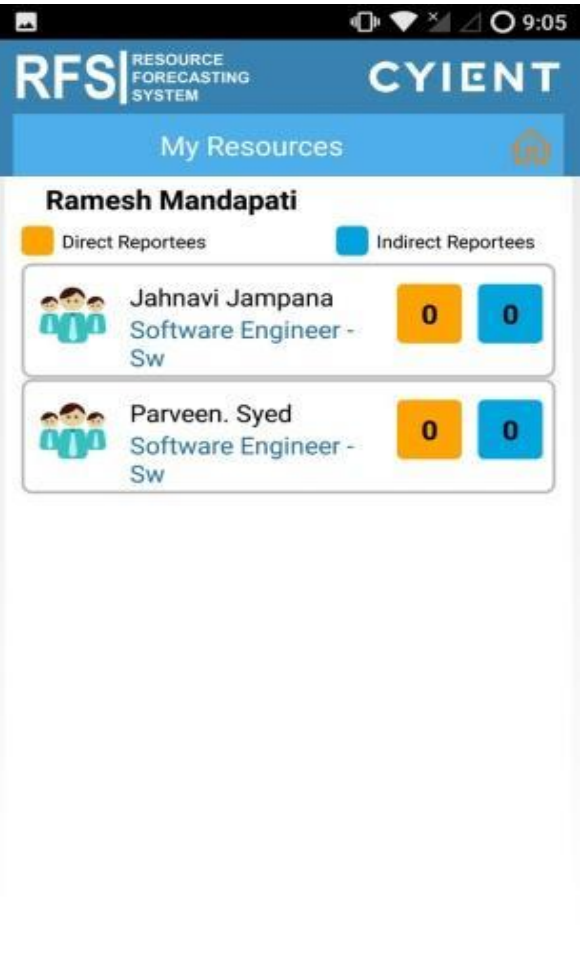


Figure 3.5 My Resources with Adapter List

3.1.2.6 Profile

A user profile is a visual display of personal data associated with a specific user, with associated skills and years of expertise in the skill. It is described by a progress bar and skill level indicator. Beside each skill lies a edit button and a separate add new skill button for additional skills.

```
using System;
using System.Collections.Generic;
using System.Data;
using System.IO;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using Android.App;
using Android.Content;
using Android.OS;
using Android.Runtime;
using Android.Views;
```

```
using Android.Widget;
using SQLite;

namespace RFS
{
[Activity(Label = "RFS", Theme = "@android:style/Theme.Holo.Light")]
public class prof_act : Activity
{
public static string skillID;
public static string skillEXP;
public static string skillLEVEL;
public static string skillLAST;

Button skillbut;
protected override void OnCreate(Bundle savedInstanceState)
{
base.OnCreate(savedInstanceState);
// Create your application here
SetContentView(Resource.Layout.prof);
skillbut = FindViewById<Button>(Resource.Id.skillbut);
skillbut.Click+=delegate {
StartActivity(typeof(edit_act));
};
CreateDB();

Button test2 = FindViewById<Button>(Resource.Id.test2);
test2.Click += Test2_Click;
x2 = 1;



if (x2 == 1)
{
test2.PerformClick();
}

ActionBar.Hide();
RatingBar r0 = FindViewById<RatingBar>(Resource.Id.ratingBar);
RatingBar r1 = FindViewById<RatingBar>(Resource.Id.ratingBar1);
RatingBar r2 = FindViewById<RatingBar>(Resource.Id.ratingBar2);
float x = 4.0F, y = 2.5F, z = 3.5F;
r0.Rating = x; r1.Rating = y; r2.Rating = z;
}

public string CreateDB()
{
var output = "";
output += "Creating Databse if it doesnt exists";
string dpPath =
Path.Combine(System.Environment.GetFolderPath(System.Environment.SpecialFolder.Perso
nal), "Person.db3"); //Create New Database
```

```
var db = new SQLiteConnection(dpPath);
output += "\n Database Created....";
return output;
}

ListView list;
int x2 = 0;
List<string> lst = new List<string>();
List<string> lst1 = new List<string>();
List<int> lst2 = new List<int>();
List<int> lst3 = new List<int>();
List<string> lst4 = new List<string>();//skill level
List<string> lst5 = new List<string>();//last used

void Test2_Click(object sender, EventArgs e)
{
getUser(MainActivity.corpID);
}

string[] imageProgress = { };//skill level
string[] list1 = { };
string[] list2 = { };
int[] prog = { };
string[] lastUsed = { };//last used

public void getUser(string username)
{
try
{
TextView emp_name = FindViewById<TextView>(Resource.Id.emp_name);
TextView emp_dept = FindViewById<TextView>(Resource.Id.emp_dept);
TextView skill = FindViewById<TextView>(Resource.Id.skill);
TextView communication = FindViewById<TextView>(Resource.Id.textView8);
rfs.Addnew obj = new rfs.Addnew();
string s = obj.GetUsers(username);
string OBS = obj.GetUsersOBS(username);
string skils = obj.GetUserSkillsDetails(username, 2);
string comm = obj.GetUserRating(username);
string listof = obj.GetUserSkillsDetails(username,2);//skill level
string lastused = obj.GetUserSkillsDetails(username, 2);//last used


DataTable dt = ConvertJSONToDataTable(s);
DataTable dt1 = ConvertJSONToDataTable(OBS);
DataTable dt2 = ConvertJSONToDataTable(skils);
DataTable dt3 = ConvertJSONToDataTable(comm);
DataTable dt4 = ConvertJSONToDataTable(listof);//skill level
DataTable dt5 = ConvertJSONToDataTable(lastused);//last used


int len = dt2.Rows.Count;
//Console.WriteLine("viewA---------------------------------------->" + len);
```

```
emp_name.Text = Convert.ToString(dt.Rows[0][1]);
communication.Text = Convert.ToString(dt3.Rows[0][1]);
string f = Convert.ToString(dt1.Rows[0][0]);
string[] obs = f.Split(',', '{', '[', ']', '}', ':');
emp_dept.Text = obs[1];
for (int i = 0; i <= len - 1; i++)
{
//  Android.App.AlertDialog.Builder dialog1 = new AlertDialog.Builder(this);
//  AlertDialog alert1 = dialog1.Create();
//  alert1.SetTitle("Log Out");
string a = Convert.ToString(dt2.Rows[i][2]);//skii id
string b = Convert.ToString(dt2.Rows[i][23]);//yrs
string c = Convert.ToString(dt2.Rows[i][23]);//progress barar
string d = Convert.ToString(dt4.Rows[i][3]);//skill level
string e = Convert.ToString(dt5.Rows[i][5]);//last used
double val = Convert.ToDouble(c);
int valint = Convert.ToInt32(val);
lst.Add(a);
lst1.Add(b + " yrs");
lst2.Add(valint);
lst4.Add(d);//skill level
lst5.Add(e);
list1 = lst.ToArray();
list2 = lst1.ToArray();
prog = lst2.ToArray();
imageProgress = lst4.ToArray();//skill level
lastUsed = lst5.ToArray();//last used
Console.WriteLine("viewA---------------------------------------->" + list1.Length);
Console.WriteLine("viewA---------------------------------------->" + list1[i]);
Console.WriteLine("viewA---------------------------------------->" + skill.Text);
}
CustomList adapter = new CustomList(this, imageProgress, list1, list2, prog,lastUsed);
list = (ListView)FindViewById(Resource.Id.listView1);
list.Adapter = adapter;
list.ItemClick += List_ItemClick;
}

catch (System.Exception ex)
{
Toast.MakeText(this, ex.Message, ToastLength.Long).Show();
}}

private DataTable ConvertJSONToDataTable(string jsonString)
{
DataTable dt = new DataTable();
string[] jsonParts = Regex.Split(jsonString.Replace("[", "").Replace("]", ""), "},{");
List<string> dtColumns = new List<string>();
foreach (string jp in jsonParts)
{
string[] propData = Regex.Split(jp.Replace("{", "").Replace("}", ""), ",");
foreach (string rowData in propData)
```

```
{
try
{
int idx = rowData.IndexOf(":");
string n = rowData.Substring(0, idx - 1);
string v = rowData.Substring(idx + 1);
if (!dtColumns.Contains(n))
{
dtColumns.Add(n.Replace("\"", ""));
}}
catch (System.Exception ex)
{
throw new System.Exception(string.Format("Error Parsing Column Name : {0}", rowData));
}}
break;
}
foreach (string c in dtColumns)
{
dt.Columns.Add(c);
}
foreach (string jp in jsonParts)
{
string[] propData = Regex.Split(jp.Replace("{", "").Replace("}", ""), ",");
DataRow nr = dt.NewRow();
foreach (string rowData in propData)
{
try
{
int idx = rowData.IndexOf(":");
string n = rowData.Substring(0, idx - 1).Replace("\"", "");
string v1 = rowData.Substring(idx + 1).Replace("\"", "");
nr[n] = v1;
}
catch (System.Exception ex)
{
//ErrorHandling.WriteLine("Error: Manage User" + ex.Message);
continue;
}}
dt.Rows.Add(nr);
}
return dt;
}
private void List_ItemClick(object sender, AdapterView.ItemClickEventArgs e)
{
//Toast.MakeText(this, "You Clicked at " + list1[e.Position], ToastLength.Long).Show();
Toast.MakeText(this, "welcome to edit" +
list1[e.Position]+list2[e.Position]+lastUsed[e.Position], ToastLength.Long).Show();
skillID = list1[e.Position];
skillEXP = list2[e.Position];
skillLEVEL = imageProgress[e.Position];
skillLAST = lastUsed[e.Position];
```

```
for (int z = 0; z <1; z++)
{
Intent description = new Intent(this, typeof(discription));
StartActivity(description);
}}}}
```

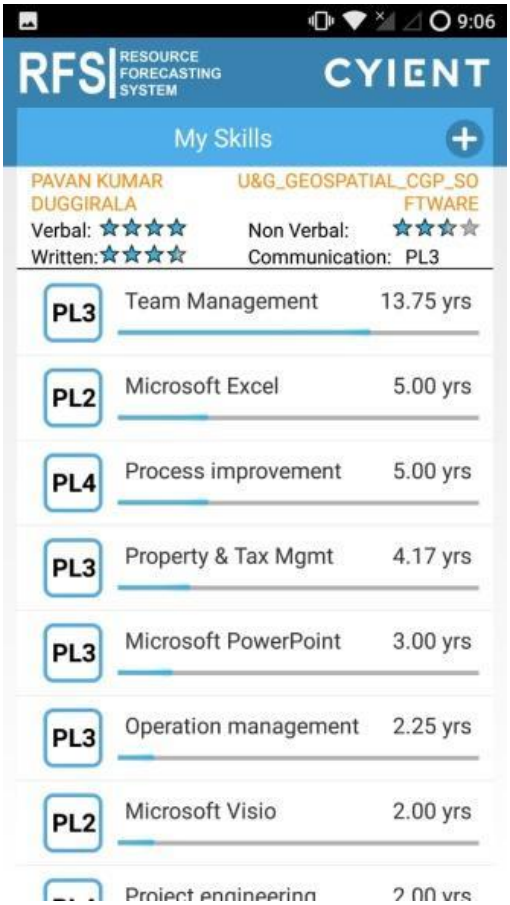


Figure 3.6 My Profile

3.1.2.7 Edit Profile

Can edit previously existing skills or add new skills.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using Android.App;
using Android.Content;
using Android.Content.PM;
using Android.OS;
using Android.Runtime;
using Android.Views;
using Android.Widget;
```

```
using SQLite;

namespace RFS
{
[Activity(Label = "edit_act", ScreenOrientation = ScreenOrientation.Portrait, Theme =
"@android:style/Theme.Holo.Light")]
public class edit_act : Activity
{
Button b1;
EditText ed1;
EditText ed3;
EditText ed4;
EditText ed5;
EditText ed6;
Spinner spinner1;

protected override void OnCreate(Bundle savedInstanceState)
{
base.OnCreate(savedInstanceState);
ActionBar.Hide();
SetContentView(Resource.Layout.edit_layout);

ed1 = FindViewById<EditText>(Resource.Id.ed1);
spinner1 = FindViewById<Spinner>(Resource.Id.spinner1);
ed3 = FindViewById<EditText>(Resource.Id.ed3);
ed4 = FindViewById<EditText>(Resource.Id.ed4);
ed5 = FindViewById<EditText>(Resource.Id.ed5);
ed6 = FindViewById<EditText>(Resource.Id.ed6);
b1 = FindViewById<Button>(Resource.Id.b1);
b1.Click += Btncreate_Click;

var levels = new List<String>() { " ", "PL1", "PL2", "PL3", "PL4" };
var adapter = new ArrayAdapter<String>(this, Android.Resource.Layout.SimpleSpinnerItem,
levels);
adapter.SetDropDownViewResource(Android.Resource.Layout.SimpleSpinnerDropDownIte
m);

spinner1.Adapter = adapter;
spinner1.ItemSelected += (sender, e) =>
{
var s = sender as Spinner;
// Toast.MakeText(this, "My favorite is " + s.GetItemAtPosition(e.Position),
ToastLength.Short).Show();
};
}

private void Btncreate_Click(object sender, EventArgs e)
{
try
{
```

```
string dpPath =
Path.Combine(System.Environment.GetFolderPath(System.Environment.SpecialFolder.Perso
nal), "Person.db3");
var db = new SQLiteConnection(dpPath);
db.CreateTable<Persontable>();
Persontable tbl = new Persontable();
tbl.skill = ed1.Text;
tbl.Proficience = Convert.ToString(spinner1.Adapter);
//tbl.Proficience = spinner1;
//tbl.proficence = ed2.Text;
tbl.Months = ed3.Text;
tbl.date = ed4.Text;
tbl.Description = ed5.Text;
tbl.Documenent = ed6.Text;
db.Insert(tbl);
clear();
Toast.MakeText(this, "Data Store Successfully...,", ToastLength.Short).Show();
}
catch (Exception ex)
{
Toast.MakeText(this, ex.ToString(), ToastLength.Short).Show();
}
}
void clear()
{
ed1.Text = "";
ed3.Text = "";
ed4.Text = "";
ed5.Text = "";
ed6.Text = "";
}}}
```

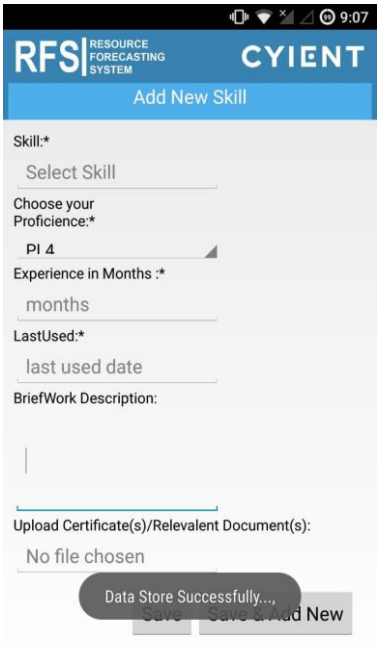


Figure 3.7 Add Skill

CHAPTER - 4

DESIGN/IMPLEMENTATION

4.1 UML DIAGRAMS

4.1.1 FLOW CHART

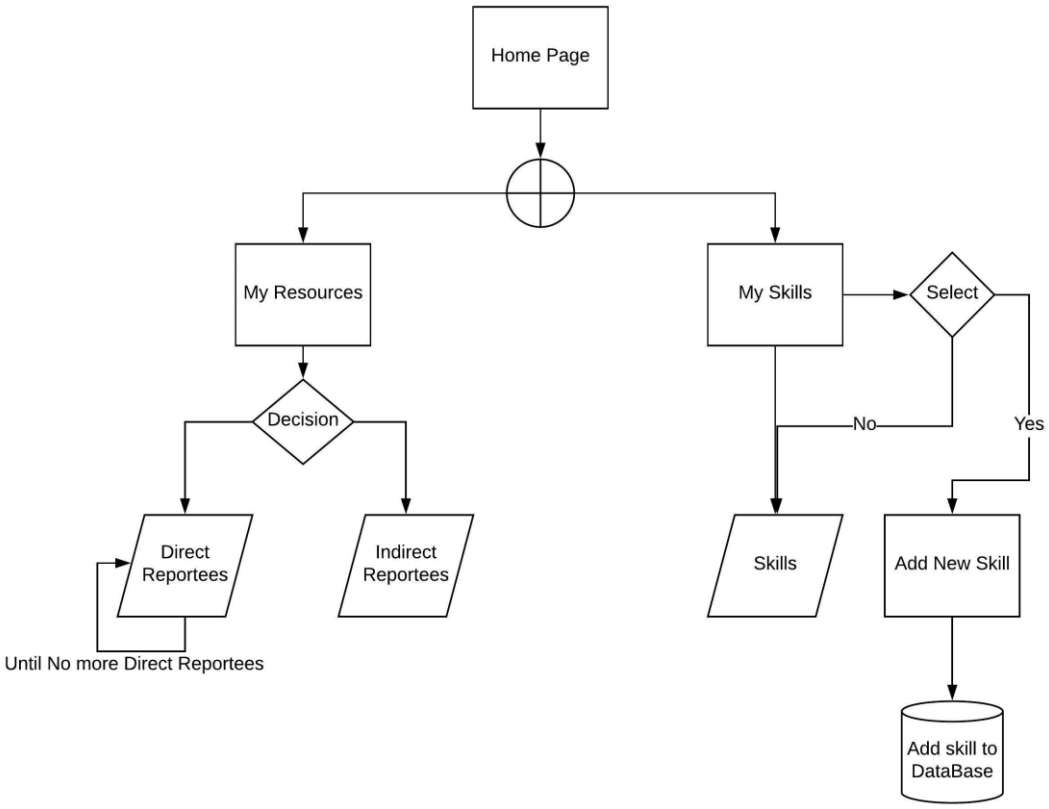


Figure 4.1 RFS Flow Chart

4.1.2 USE CASE

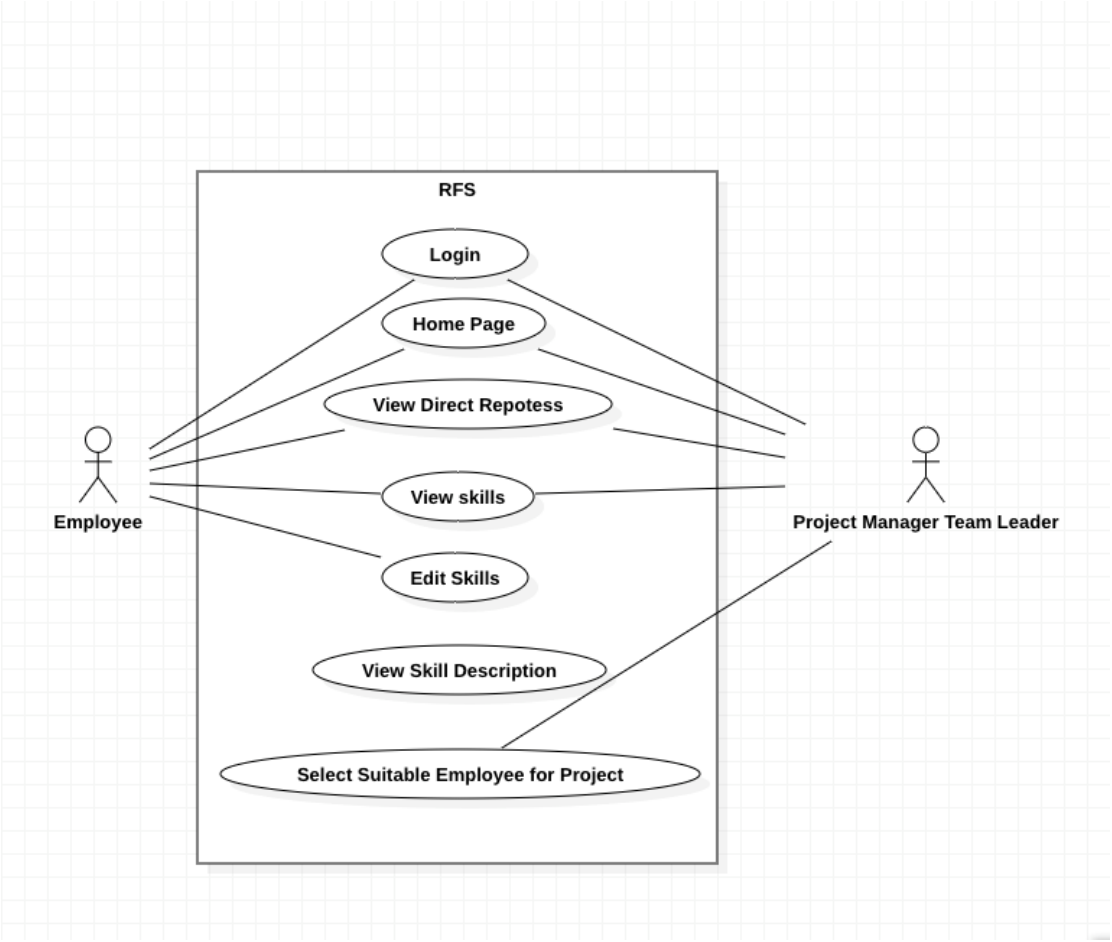


Figure 42 RFS Use Case

4.1.3 STATE CHART

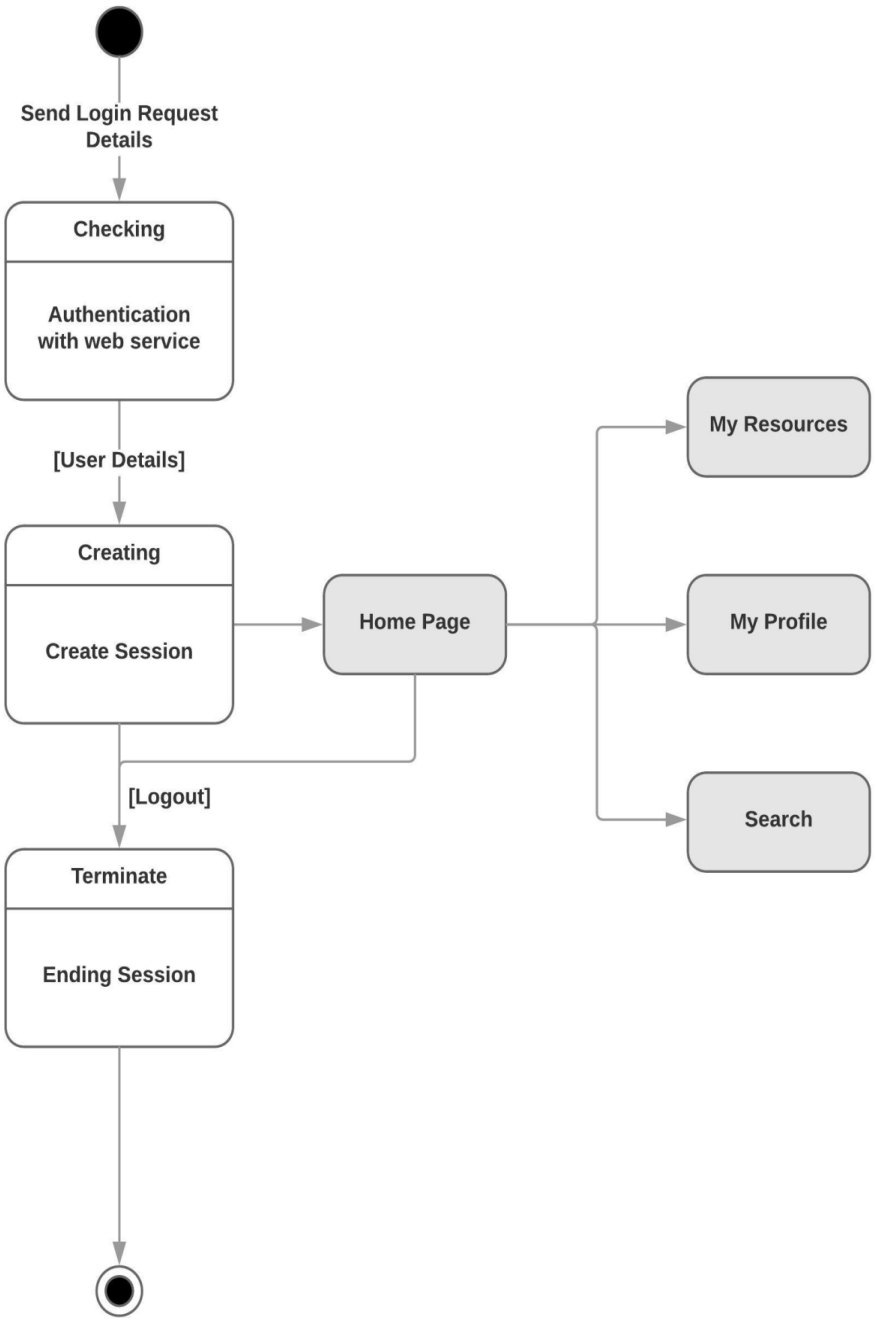


Figure 4.3 RFS State Chart

4.1.4 SEQUENCE DIAGRAM

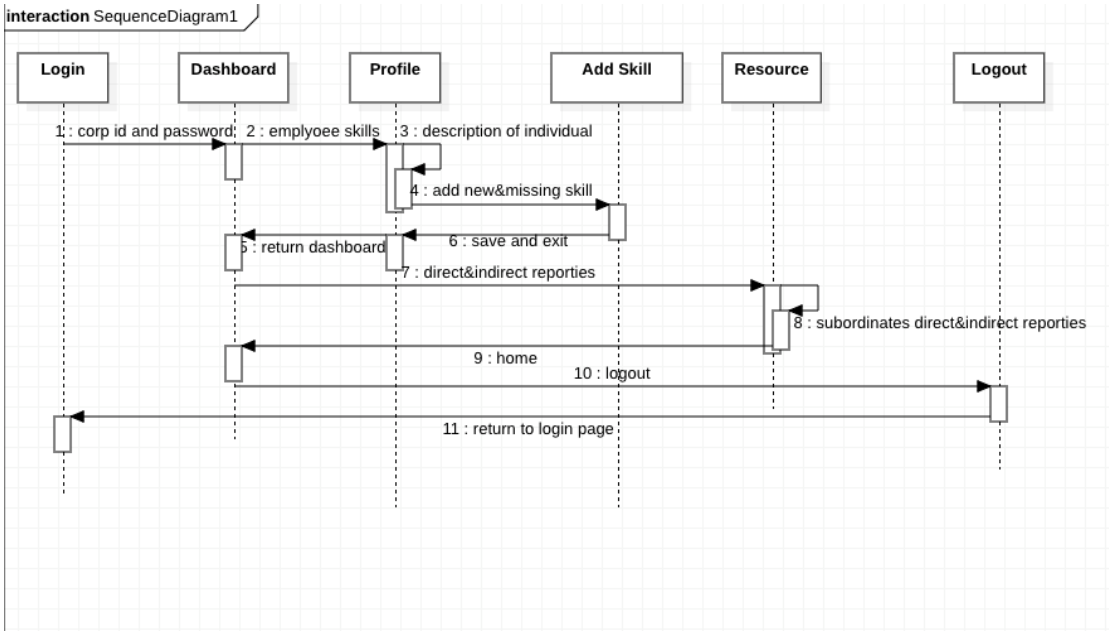


Figure 4.4 RFS Sequence Diagram

4.2 USER INTERFACE

The various tools and building blocks that are used to compose user interfaces in Xamarin.Android apps.

4.2.1 ANDROID DESIGNER

The Android Designer to layout controls visually and edit properties. The Designer to work with user interfaces and resources across various configurations, such as themes, languages, and device configurations, Design for alternative views like landscape and portrait.

Xamarin.Android supports both a declarative style of user interface design based in XML files, as well as programmatic user interface creation in code. When using the declarative approach, XML files can be either hand-edited or modified visually by using the Xamarin.Android Designer. Use of a designer allows immediate feedback during UI creation, speeds up development, and makes the process of UI creation less laborious.

The Designer is launched automatically when a layout is created, or it can be launched by double-clicking an existing .axml file. For example, double-clicking **Main.axml** in the **Resources > Layout** folder will load the Designer as shown below:

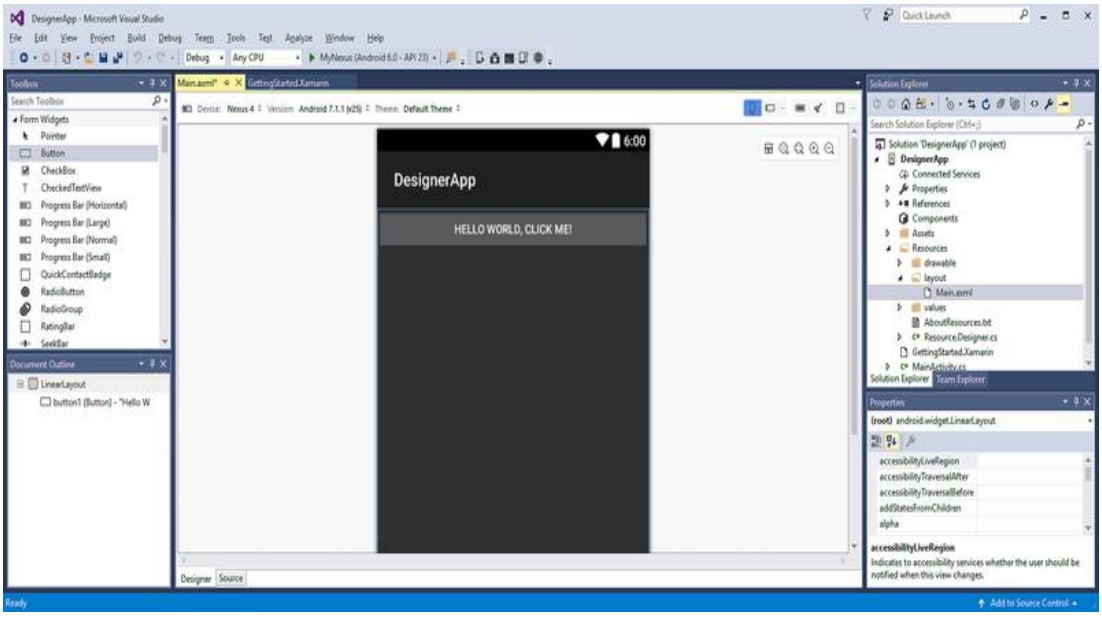


Figure 4.5 Android Designer

4.2.2 DESIGNER FEATURES

The Designer is composed of several sections that support its various features, as shown in the following screenshot:

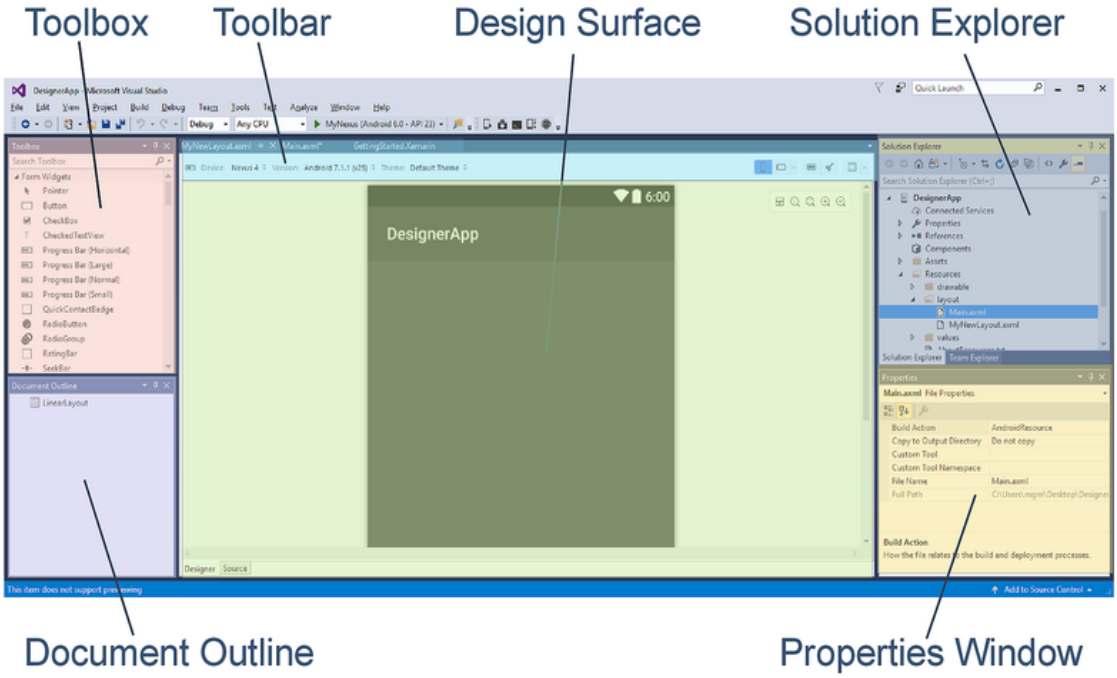


Figure 4.6 Designer Features

When editing a layout in the Designer, usage of the following features to create and shape design:

- **Design Surface** – Facilitates the visual construction of the user interface by giving an editable representation of how the layout will appear on the device.
- **Toolbar** – Displays a list of selectors: **Device**, **Version**, **Theme**, layout configuration, and Action Bar settings. The Toolbar also includes icons for launching the Theme Editor and for enabling the Material Design Grid.
- **Toolbox** – Provides a list of widgets and layouts that can drag and drop onto the Design Surface.
- **Properties Window** – Lists the properties of the selected widget for viewing and editing.
- **Document Outline** – Displays the tree of widgets that compose the layout. Click an item in the tree to cause it to be selected in the Designer. Also, clicking an item in the tree loads the item's properties into the Properties window.

4.2.3 Toolbar

The Toolbar (positioned above the Design Surface) presents configuration selectors and tool menus:

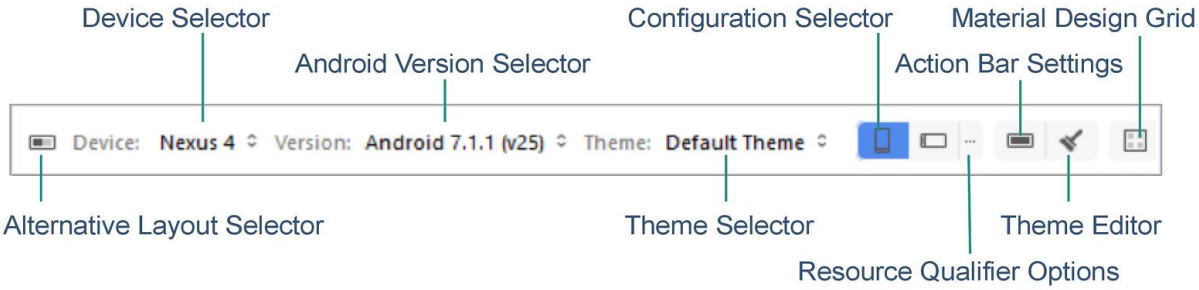


Figure 4.7 Toolbar

The Toolbar provides access to the following features:

- **Alternative Layout Selector** – Allows to select from different layout versions.
- **Device Selector** – Defines a set of qualifiers associated with a particular device such as screen size, resolution, and keyboard availability. Can also add and delete new devices.
- **Android Version Selector** – The Android version that the layout is targeting. The Designer will render the layout according to the selected Android version.
- **Theme Selector** – Selects the UI theme for the layout.
- **Configuration Selector** – Selects the device configuration, such as portrait or landscape.
- **Resource Qualifier Options** – Opens a dialog that presents drop-down menus for selecting Language, UI Mode, Night Mode, and Round Screen options.
- **Action Bar Settings** – Configures the Action Bar settings for the layout.
- **Theme Editor** – Opens the Theme Editor, which makes it possible customize elements of the selected theme.

- **Material Design Grid** – Enables or disables the Material Design Grid. The drop-down menu item adjacent to the Material Design Grid opens a dialog that enables to customize the grid.

4.2.4 MATERIAL THEME

Material Theme is a user interface style that determines the look and feel of views and activities starting with Android 5.0 (Lollipop). Material Theme is built into Android 5.0, so it is used by the system UI as well as by applications. Material Theme is not a "theme" in the sense of a system-wide appearance option that a user can dynamically choose from a settings menu. Rather, Material Theme can be thought of as a set of related built-in base styles that can be use to customize the look and feel of an app.

Android provides three Material Theme flavours:

Theme.Material – Dark version of Material Theme; this is the default flavour in Android 5.0.

Theme.Material.Light – Light version of Material Theme.

Theme.Material.Light.DarkActionBar – Light version of Material Theme, but with a dark action bar.

Examples of these Material Theme flavours are displayed here:

Example screenshots of the Dark theme, Light theme, and Dark Action Bar theme

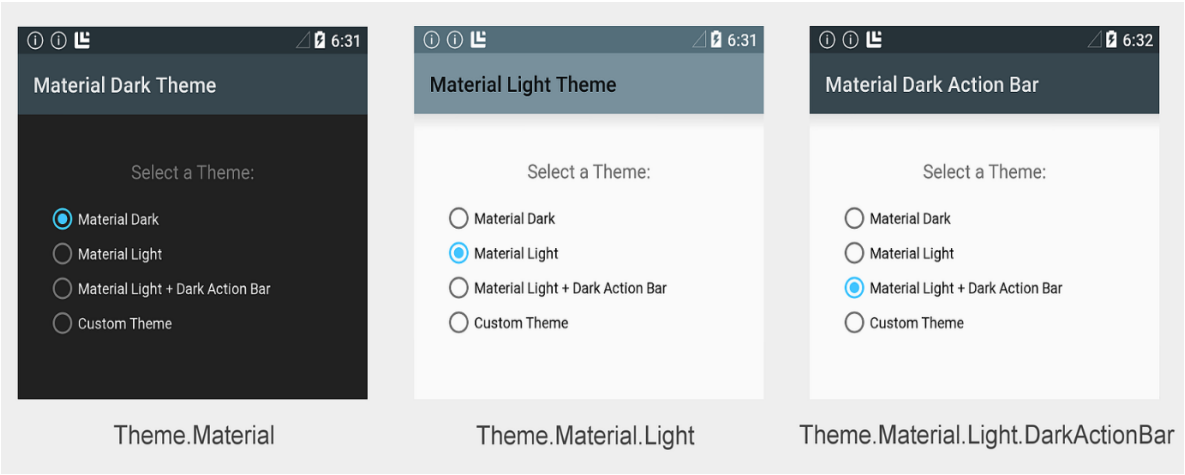


Figure 4.8 Material Theme

Deriving from Material Theme to create own theme, overriding some or all color attributes. For example, create a theme that derives from Theme.Material.Light, but overrides the app bar color to match the color of a brand. Also style individual views. For example, creating style for CardView that has more rounded corners and uses a darker background color.

A single theme can be used for an entire app, or a different theme for different screens (activities) in an app. In the above screenshots, for example, a single app uses a different theme for each activity to demonstrate the built-in color schemes. Radio buttons switch the app to different activities, and, as a result, display different themes.

Because Material Theme is supported only on Android 5.0 and later, it cannot be used (or a custom theme derived from Material Theme) to theme an app for running on earlier versions of Android. However, app can be use Material Theme on Android 5.0 devices and gracefully fall back to an earlier theme when it runs on older versions of Android (see the Compatibility section of this article for details).

Requirements

The following is required to use the new Android 5.0 Material Theme features in Xamarin-based apps:

Xamarin.Android – Xamarin.Android 4.20 or later must be installed and configured with either Visual Studio or Visual Studio for Mac.

Android SDK – Android 5.0 (API 21) or later must be installed via the Android SDK Manager.

Java JDK 1.8 – JDK 1.7 can be used for specifically targeting API level 23 and earlier. JDK 1.8 is available from Oracle.

4.2.5 LAYOUTS

Layouts are used to define the visual structure for a user interface. Layouts such as ListView and RecyclerView are the most fundamental building blocks of Android applications. Typically, a layout will use an Adapter to act as a bridge from the layout to the underlying data that is used to populate data items in the layout. This section explains how to use layouts such as LinearLayout, RelativeLayout, TableLayout, RecyclerView, and GridView.

Layouts are used to arrange the elements that make up the UI interface of a screen (such as an Activity). The following sections explain how to use the most commonly-used layouts in Xamarin.Android apps.

- **LinearLayout** is a view group that displays child view elements in a linear direction, either vertically or horizontally.

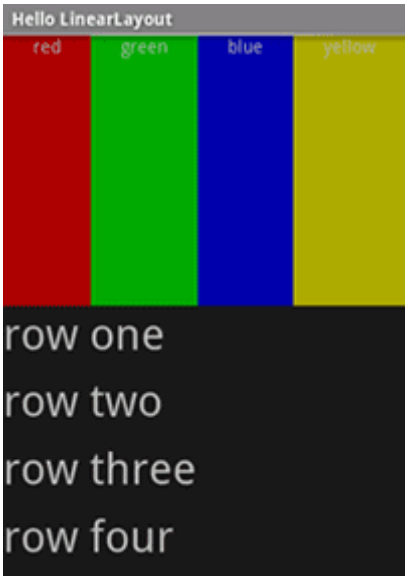


Figure 4.9 Linear Layout

- **RelativeLayout** is view group that displays child view elements in a relative position. The position of a view can be specified as relative to sibling elements.

Figure 4.10 Relative Layout

- **TableLayout** is a view group that displays child view elements in rows and columns.

Figure 4.11 Table Layout

- **RecyclerView** is a UI element that displays a collection of items in a list or a grid, enabling the user to scroll through the collection.

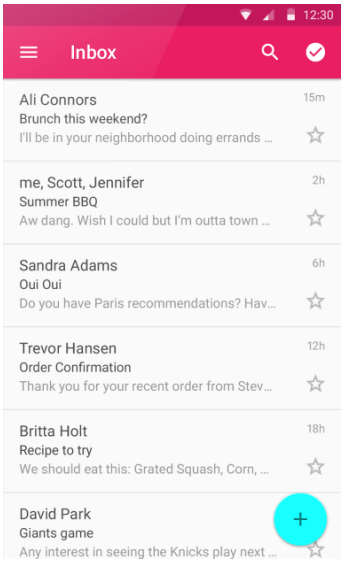


Figure 4.12 Recycler View

- **ListView** is a view group that creates a list of scrollable items. The list items are automatically inserted into the list using a list adapter. The ListView is an important UI component of Android applications because it is used everywhere from short lists of menu options to long lists of contacts or internet favourites. It provides a simple way to present a scrolling list of rows that can either be formatted with a built-in style or customized extensively. A ListView instance requires an Adapter to feed it with data contained in row views.

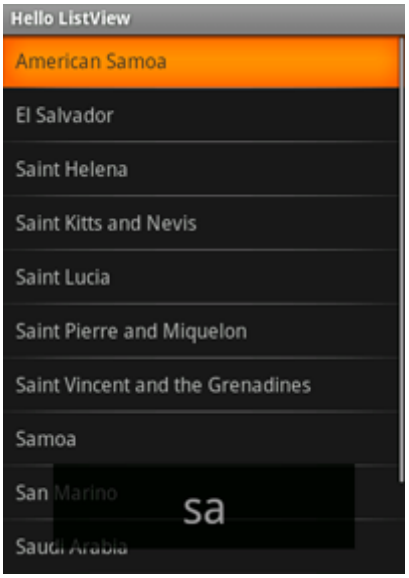


Figure 4.13 List View

- **GridView** is a UI element that displays items in a two-dimensional grid that can be scrolled.

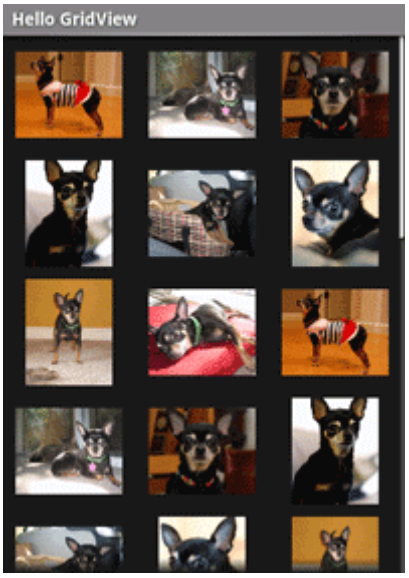


Figure 4.14 Grid View

- **GridLayout** is a view group that supports laying out views in a 2D grid, similar to an HTML table.

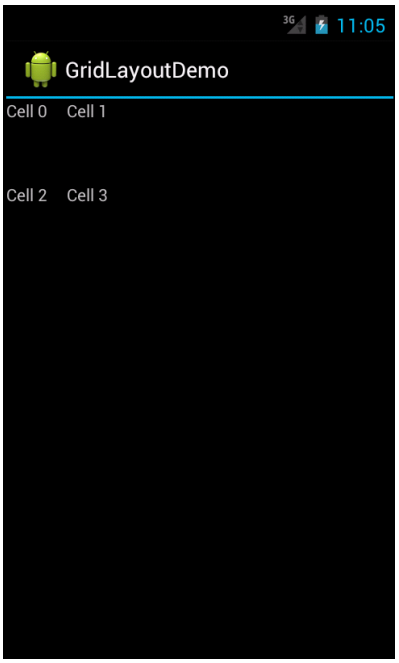


Figure 4.15 Grid Layout

- **Tabbed Layouts** are a popular user interface pattern in mobile applications because of their simplicity and usability. They provide a consistent, easy way to navigate between various screens in an application.

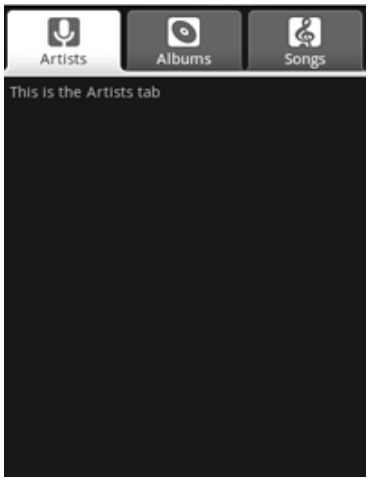


Figure 4.16 Tabbed Layout

4.2.6 CONTROLS

Android controls (also called widgets) are the UI elements that can be uses to build a user interface. This section explains how to use controls such as buttons, toolbars, date/time pickers, calendars, spinners, switches, pop-up menus, view pagers, and web views.

Xamarin.Android exposes all of the native user interface controls (widgets) provided by Android. These controls can be easily added to Xamarin.Android apps using the Android Designer or programmatically via XML layout files. Regardless of which method chosen, Xamarin.Android exposes all of the user interface object properties and methods in C#. The following sections introduce the most common Android user interface controls and explain how to incorporate them into Xamarin.Android apps.

Action Bar

`ActionBar` is a toolbar that displays the activity title, navigation interfaces, and other interactive items. Typically, the action bar appears at the top of an activity's window.

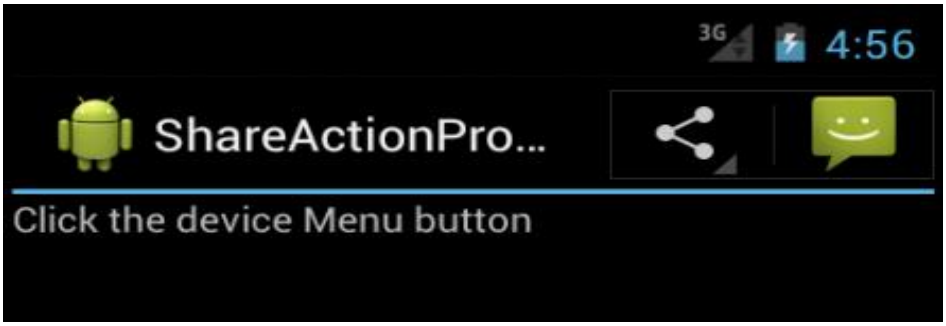


Figure 4.17 Action Bar

Auto Complete

AutoCompleteTextView is an editable text view element that shows completion suggestions automatically while the user is typing. The list of suggestions is displayed in a drop down menu from which the user can choose an item to replace the content of the edit box with.

Figure 4.18 Auto Complete

Buttons

Buttons are UI elements that the user taps to perform an action.

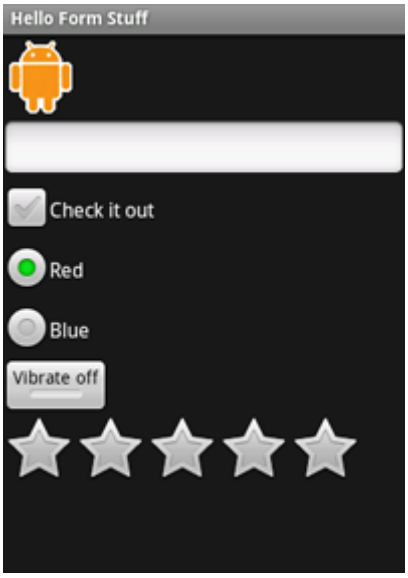


Figure 4.19 Buttons

Calendar

The `Calendar` class is used for converting a specific instance in time (a millisecond value that is offset from the epoch) to values such as year, month, hour, day of the month, and the date of the next week. `Calendar` supports a wealth of interaction options with calendar data, including the ability to read and write events, attendees, and reminders. By using the calendar provider in a application, data can be added through the API will appear in the built-in calendar app that comes with Android.

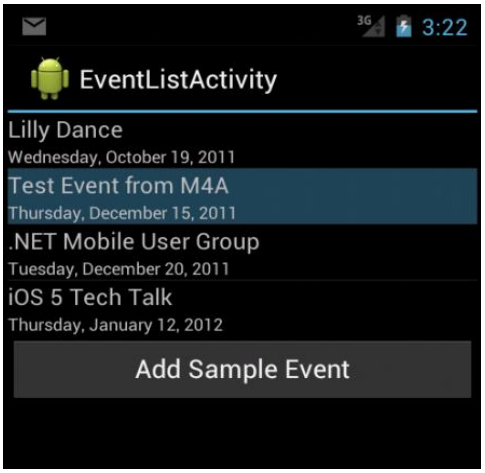


Figure 4.20 Calendar

CardView

CardView is a UI component that presents text and image content in views that resemble cards. CardView is implemented as a FrameLayout widget with rounded corners and a shadow. Typically, a CardView is used to present a single row item in a gListView or GridView view group.

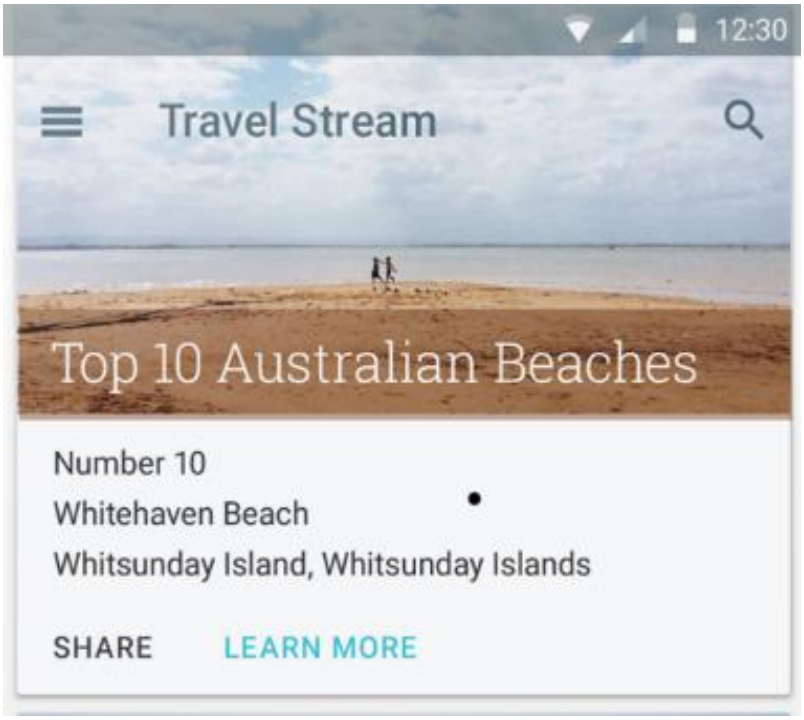


Figure 4.21 Card View

Edit Text

EditText is a UI element that is used for entering and modifying text.

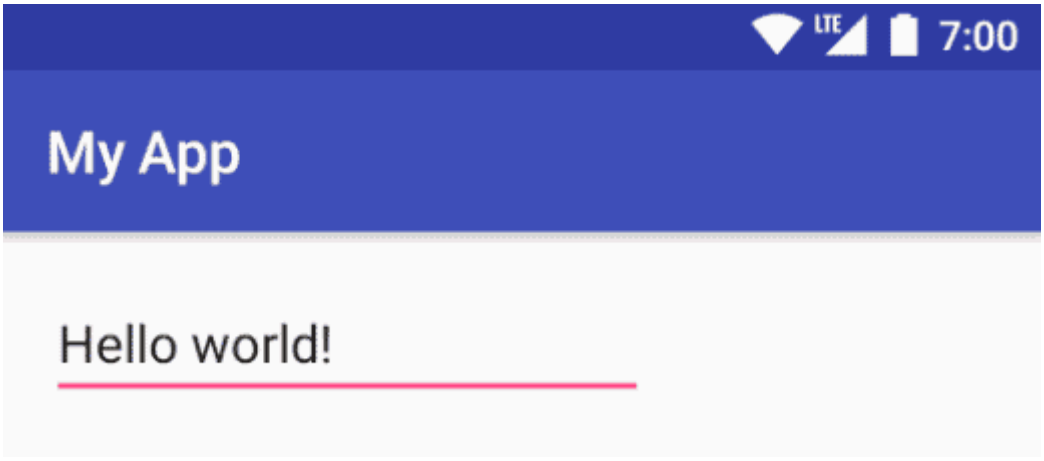


Figure 4.22 Edit Text

Gallery

Gallery is a layout widget that is used to display items in a horizontally scrolling list; it positions the current selection at the centre of the view.

Figure 4.23 Gallery

Navigation Bar

The *Navigation Bar* provides navigation controls on devices that do not include hardware buttons for **Home**, **Back**, and **Menu**.

Figure 4.24 Navigation Bar

Pickers

Pickers are UI elements that allow the user to pick a date or a time by using dialogs that are provided by Android.

Figure 4.25 Pickers

Popup Menu

PopupMenu is used for displaying popup menus that are attached to a particular view.

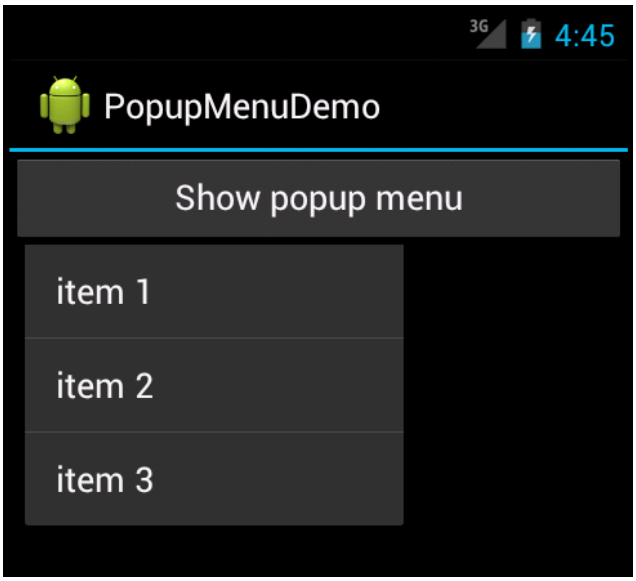


Figure 4.26 Pop Up

Spinner

Spinner is a UI element that provides a quick way to select one value from a set. It is similar to a drop-down list.

Figure 4.27 Spinner

Switch

Switch is a UI element that allows a user to toggle between two states, such as ON or OFF. The Switch default value is OFF.

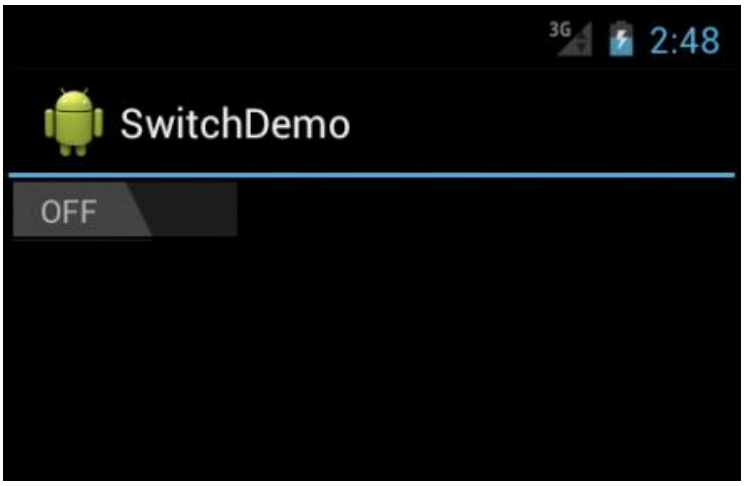


Figure 4.28 Switch

Texture View

TextureView is a view that uses hardware-accelerated 2D rendering to enable a video or OpenGL content stream to be displayed.

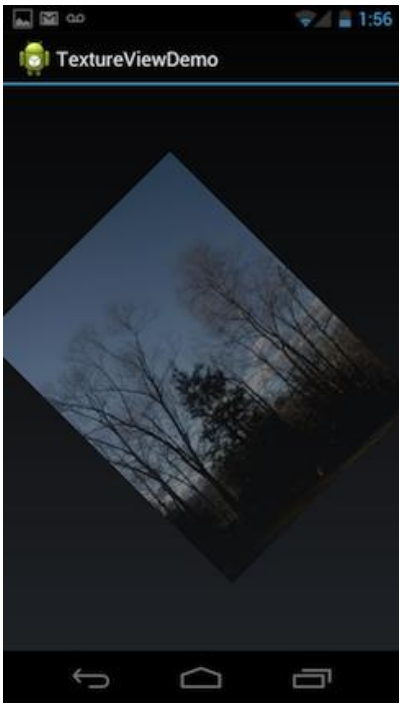


Figure 4.29 Texture View

Toolbar

The Toolbar widget (introduced in Android 5.0 Lollipop) can be thought of as a generalization of the action bar interface – it is intended to replace the action bar. The Toolbar can be used anywhere in an app layout, and it is much more customizable than an action bar.

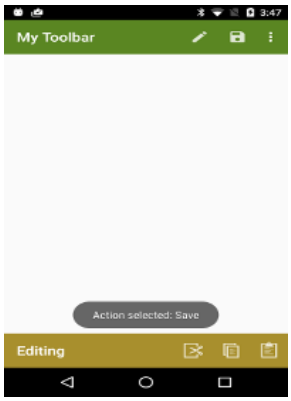


Figure 4.30 Toolbar Widget

ViewPager

The ViewPager is a layout manager that allows the user to flip left and right through pages of data.

Figure 4.31 View Pager

WebView

WebView is a UI element that allows creating an own window for viewing web pages (or even developing a complete browser).

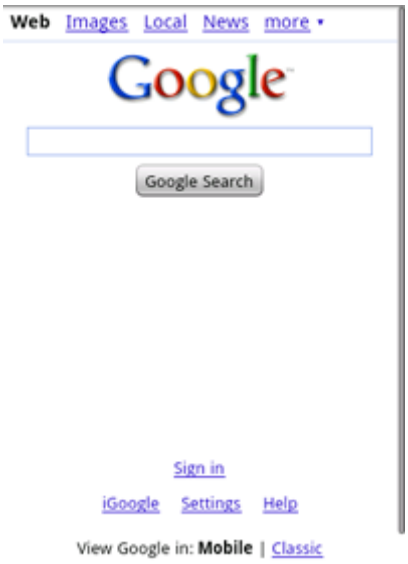


Figure 4.32 Web View

4.3 INSERTING DATA IN SQLITE DATA BASE

Introduction

Xamarin is a platform to develop cross-platform and multi-platform apps (for example, Windows phone, Android, iOS). In Xamarin platform, the code sharing concept is used. In Xamarin Studio, Visual Studio is also available.

Android, iOS and Windows support SQLite3, the SQL-based relational database management system (RDMS), for local data storage. There are a few things to note when working with SQLite, that may influence the way developed with it:

- SQLite stores data in a simple text file. There is no granular security or user privileges for data therefore anyone with file system access to it may read its contents.
- There are only five underlying data types; NULL, INTEGER, REAL, TEXT and BLOB. For more detail see Data types In SQLite Version 3.
- Binary objects (BLOBs) are stored as text representations, thus access to BLOBs is not optimal. It is recommend storing BLOBs on the filesystem and storing the filesystem path in the database.
- SQLite supports concurrent read access, but enforces sequential write access. This is because a filesystem lock is placed on the file during write operations. This is an important point to bear in mind with multi-threaded applications.
- Referential integrity is not enabled by default. See SQLite Foreign Key Support for more information.
- RIGHT and FULL OUTER JOINs are not supported.
- There is limited ALTER TABLE support; columns may not be modified or deleted.

4.3.1 Prerequisites

Visual Studio 2015 Update 3.

In order to insert data in SQLite Data Base in the Xamarin Android app, using Visual Studio 2015.

4.3.2 Usage of a Database

While the storage and processing capabilities of mobile devices are increasing, phones and tablets still lag behind their desktop and laptop counterparts. For this reason it is worth taking some time to plan the data storage architecture for the app rather than just assuming a database is the right answer all the time. There are a number of different options that suit different requirements, such as:

- **Preferences** – Android offers a built-in mechanism for storing simple key-value pairs of data. When are storing simple user settings or small pieces of data (such as personalization information) then use the platform's native features for storing this type of information.
- **Text Files** – User input or caches of downloaded content (eg. HTML) can be stored directly on the file-system. Use an appropriate file-naming convention to help organize the files and find data.
- **Serialized Data Files** – Objects can be persisted as XML or JSON on the file-system. The .NET framework includes libraries that make serializing and de-serializing objects easy. Use appropriate names to organize data files.
- **Database** – The SQLite database engine is available on the Android platform, and is useful for storing structured data that needs to query, sort or otherwise manipulate. Database storage is suited to lists of data with many properties.
- **Image files** – Although it's possible to store binary data in the database on a mobile device, it is recommended that it is stored directly in the file-system. If necessary it can store the filenames in a database to associate the image with other data. When dealing with large images, or lots of images, it is good practice to plan a caching strategy that deletes files no longer needed to avoid consuming all the user's storage space.

If a database is the right storage mechanism for a app, the remainder of this document discusses how to use SQLite on the Xamarin platform.

4.3.3 Advantages of using a Database

There are a number of advantages to using an SQL database in a mobile app:

- SQL databases allow efficient storage of structured data.
- Specific data can be extracted with complex queries.
- Query results can be sorted.
- Query results can be aggregated.
- Developers with existing database skills can utilize their knowledge to design the database and data access code.
- The data model from the server component of a connected application may be re-used (in whole or in part) in the mobile application.

4.3.4 SQLite Database Engine

SQLite is an open-source database engine that has been adopted by Google for their mobile platform. The SQLite database engine is built-in to both operating systems so there is no additional work for developers to take advantage of it. SQLite is well suited to cross-platform mobile development because:

- The database engine is small, fast and easily portable.
- A database is stored in a single file, which is easy to manage on mobile devices.
- The file format is easy to use across platforms: whether 32- or 64-bit, and big- or little-endian systems.
- It implements most of the SQL92 standard.

Because SQLite is designed to be small and fast, there are some caveats on its use:

- Some OUTER join syntax is not supported.
- Only table RENAME and ADDCOLUMN are supported. It cannot perform other modifications to the schema.
- Views are read-only.

CHAPTER - 5

CONCLUSION

At the end of this application it is a sophisticated approach for users to have a best selection and gives better performance for Administrator so that he can easily add, update and view employee details. This initiative of making managers task of selection and display of attributes easier and convenient to view the details.

REFERENCES

1.Jesse Liberty, “Learning C#”, O’Reilley & Associates, September 2002: First Edition

2.Grady Brooch, James Rambaugh.1998, “Unified Modelling Language User Guide”, Addison Wesley Publishing, chapter 8-31.

3.Lars Powers, Mike Shell, “Microsoft Visual Studio 2015 Unleashed”, Pearson Education

WEBSITES:	**REFERRED URLS:**
www.android.com	http://developer.android.com/index.html
www.google.com	http://en.wikipedia.org/wiki/SQLite
www.microsoft.com	https://visualstudio.microsoft.com/xamarin/
www.tutorialspoint.com	https://www.tutorialspoint.com/csharp
www.cyient.com	http://www.cyient.com/about-us/who-we-are/