telliant
Intelligent Software. Delivered.

Microsoft Partner
Gold Application Development
Gold Collaboration and Content

SYSTEM ARCHITECTURE AND DESIGN – STRIVE MOBILE PALTFORM

Abstract

This document provides Architecture components, their details, abstract use cases at a High level

Author	Reviewed By	Approved By
Gokul Gopi	Amos	Seth
9/12/2019	9/12/2019	10/10/2019

Table of Contents

Introduction

Mammoth Detail Salons specialize in Hand car washing and detailing. At Mammoth hand car wash and detail there are many services available like Mini Wash, Mammoth Wash, Ultra – Mammoth Wash, Mega-Mammoth Wash and membership services. On the technical side, it has Web Applications and a

Mobile application to maintain the daily business seamlessly. Mammoth is building a point of sale software product called Strive with the following mobile applications.

Mobile Applications:

1. Employee App
2. Customer App
3. Owner App
4. Employee Time Management App (iPad App).
5. Greeter App (iPad mini App).

The purpose of this document is to provide High-level design details to create Strive mobile application from the scratch using latest technology.

Project Purpose

Multiple mobile applications are part of the Strive ecosystem. Applications include the Greeter/check in iPad mini app, iPad TIM app, Customer App, Employee App and Owner app.

The purpose of the project is to create multiple mobile solutions to ensure a full working mobile POS platform which is easy to carry, accessible and easily packaged as a white labelled mobile product.

Executive Summary

The existing applications of Mammoth Detail Salon which maintain the day to day operations of the car wash business are written in outdated technologies.

This document proposes a new software architecture and a new technology stack for the mobile application platform. This new platform will provide the following benefits:

1. improved maintainability
2. enhanced scalability
3. extensive security
4. robustness
5. extensibility

This application will use the latest Hybrid native approach of Xamarin and MVVM to provide an increased level of code reusability and ease of development.

Goals and Objectives

The goal is to design an easily scalable and maintainable mobile application platform for both Android and iOS, allowing seamless code sharing and business sharing.

Proposed Architecture

Mobile applications are more extensively used than web applications, therefore the architecture should be uncoupled and reusable for easy maintainenace and usability.

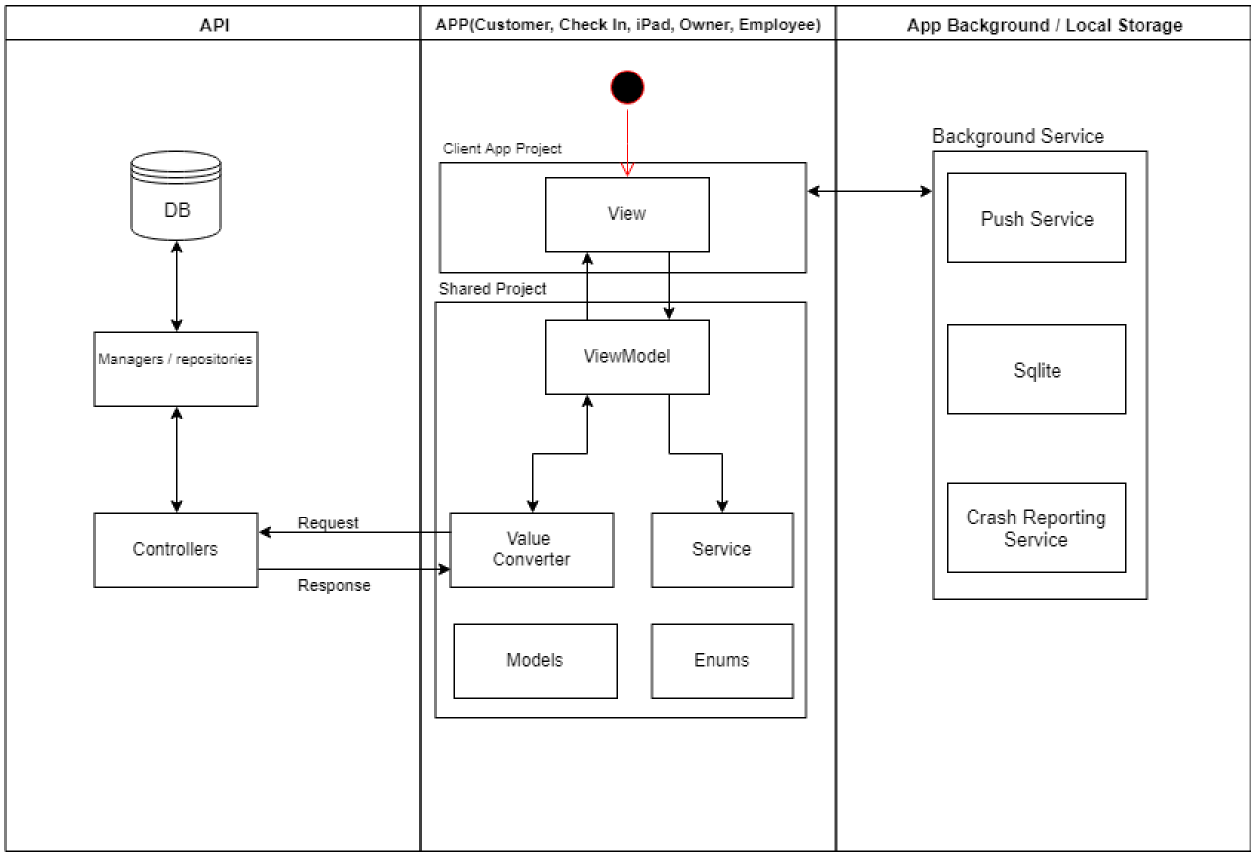
Application will be broken down into 3 major parts

1. Business Layer (Shared Project)
2. View Layer (Native device specific project for Android, iOS)
3. Service Layer (Implementation of Services)

This will be applicable to the following applications:

1. Check-In Station/Greeter App
2. Customer App
3. iPad TIM App
4. Owner App
5. Employee App

Proposed workflow



Assumptions

1. The proposed application architecture is high level, minor changes during the design phase are likely, though the overall concept will remain the same.

2. A technical design document will be derived from this document. The technical design document will include all the components required with specific details of the actual implementation.

3. The third-party commercial tools are listed in the section "3rd party Plugins". Additional third-party commercial tools may be included during the design phase, but the recommendation would be to utilize open source technologies.

4. All the plugins mentioned in the section “3rd Party Plugins” require validation by creating a Proof of Concept (POC) to ensure that the application can function as expected per the functional requirements specification document.

5. The application will be a “Hybrid Native” deployment and the code will be programmed using C# utilizing the .Net Framework.

Prerequisites

1. Apple developer license
2. Google Play console
3. Google cloud API for push notifications
4. Development server for API
5. Fastlane setup for sharing of keys

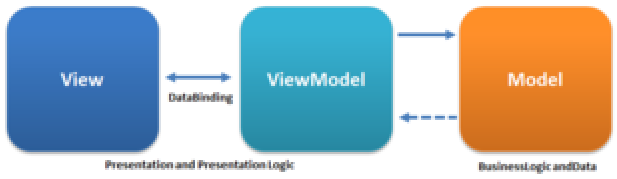
Environments

1. Dev
2. QA
3. UAT
4. Prod

Design Pattern

Model–view–viewmodel (MVVM) is a software architectural pattern.

The Model-View-ViewModel (MVVM) pattern helps to cleanly separate the business and presentation logic of an application from its user interface (UI). Maintaining a clean separation between application logic and the UI helps to address numerous development issues and can make an application easier to test, maintain, and evolve. It can also greatly improve code re-use opportunities and allows developers and UI designers to more easily collaborate when developing their respective parts of an app.



Source: MSDN

The view model implements properties and commands to which the view can data bind to, and notifies the view of any state changes through change notification events. The properties and commands that the view model provides define the functionality to be offered by the UI, but the view determines how that functionality is to be displayed.

The primary benefit of the ViewModel pattern to displace code from View's code-behind to ensure editing the view a more independent task. Controllers are still created as and when required to control the overall logic of our applications.

To support MVVM cross package there is an inbuilt two – way binding that will make the application reactive.

Technology Stack

Presentation Layer

Android	iOS
XML	Auto Layout views using constraints

Business Layer

Android	iOS
C# Shared Project	C# Shared Project

Data Layer

Android	iOS
C# Shared Project	C# Shared Project

IDE

Android	iOS
Visual Studio 2017	Visual Studio 2017

Configuration Management (CI/CD)

Release Pipeline

Azure DevOps

Source Control

Azure Repo

Deployment

App Store for Apple

Play Store for iOS

Appcenter for UAT and QA builds

Build Service

Azure

Security

Tools can be used to prevent security threats

Tool	Purpose

Visual Studio Code Analysis tool	Static code quality analysis tool based on MS coding best practices.

Possible threats

- Device Memory Constraints
- Clearing the Cache is mandatory to ensure customer satisfaction

Protected local storage

- Securing stored data by preventing unauthorized access
- Prevent accidental or intentional destruction, infection, or corruption
- Activity log monitoring
- Data encryption

Code obfuscation

Code obfuscation is the deliberate act of creating source or machine code that's difficult for humans (hackers) to understand. Developers can obfuscate code to conceal its purpose, logic, or implicit values embedded in it. A tool called an obfuscator can be used to automatically convert straightforward source code into a program that works the same way but is much harder to read and understand. Developers can also obfuscate code manually. Code obfuscation may include:

- encrypting some or all the code;
- stripping out potentially revealing metadata;
- renaming useful class and variable names to meaningless labels;
- adding unused or meaningless code to an application's binary.

No sensitive data stored on devices

When requesting information from different services, some data may require certain secret keys. Mostly, these secret keys are necessary for different services, for example for navigation with Google Maps or to use the Google search service. These services require a secret key that's generated on the service's website. A secret key gives a user access to a service. By entering this key, the system identifies that someone is an authorized user.

Secret keys are stored on the server side since it's the server that uses them. Keys have a better level of protection on the server side than when they're stored on the client side. Therefore, the server side requires more security.

If an app doesn't have a server side, then the secret keys need to be saved within the app. Not only is it necessary in this case to save them in code or adjust the open file settings — it's also necessary to encrypt them and limit access to them. The presence of this key in the app is a threat of its own.

3rd Party Plugins

Plugin Name	Purpose	Url
MvvmCross	Mvvm architecture implementation	https://github.com/MvvmCross/MvvmCross
Firebase	PushNotification	https://www.nuget.org/packages/Plugin.FirebasePushNotification
V7 Support Library	UI v7	https://www.nuget.org/packages/Xamarin.Android.Support.v7.AppCompat