

Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of Information Engineering



Master's Thesis

iOS Application in the XCode environment

Saravana Kumar Obula Meganath

© 2022 CZU Prague

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

DIPLOMA THESIS ASSIGNMENT

Saravana Kumar Obula Meganath

Systems Engineering and Informatics
Informatics

Thesis title

iOS application in the XCode environment

Objectives of thesis

The main goal of this thesis is to demonstrate how to create a user-friendly and fully functional mobile application – a *Multitasker* app for the iPhone, and increase awareness of mobile development for the efficient development of iOS projects, the Xcode interface and the SwiG programming language.

Methodology

The Diploma thesis will be divided into two sections: theoretical and practical. The theoretical part will be based on an analysis of secondary data sources, programming language documentations, including professional literature, online videos, Internet publications about developing for the iOS mobile platform and documents about mobile technology. The second part will be practical on the example of how to use these tools and techniques for developing a mobile application named *Multitasker*.

The proposed extent of the thesis

80 – 120 pages

Keywords

iOS; Mobile Application; SwiG; iPhone; XCode; Programming

Recommended information sources

Arthur M. Langer. 2012. Guide to SoGware Development. New York: Springer.
Christian Keur and Aaron Hillegass. 2015. iOS Programming: The Big Nerd Ranch Guide. Atlanda: Pearson Technology Group.
Stefan Kaczmarek, Brad Lees, and Gary Bennett. 2019. SwiG 5 for Absolute Beginners: Learn to Develop Apps for iOS. New York: Apress Media.

Expected date of thesis defence

2021/22 SS – FEM

The Diploma Thesis Supervisor

doc. Ing. Vojtěch Merunka, Ph.D.

Supervising department

Department of Information Engineering

Electronic approval: 1. 11. 2021

Ing. Martin Pelikán, Ph.D.

Head of department

Electronic approval: 23. 11. 2021

Ing. Martin Pelikán, Ph.D.

Dean

Prague on 28. 03. 2022

Declaration

I declare that I have worked on my master's thesis titled "iOS application in the XCode environment" by myself and I have used only the sources mentioned at the end of the thesis. As the author of the master's thesis, I declare that the thesis does not break any copyrights.

In Prague on 30.03.2022

Acknowledgement

I would like to thank doc. Ing. Vojtech Merunka, Ph.D, friends and my family for their advice and support during my work on this thesis.

iOS Application in the XCode environment

Abstract

The thesis's goal is to raise knowledge of efficient iOS mobile app development, the XCode interface, and the Swift programming language. The primary purpose of this thesis is to illustrate how to construct a user-friendly and fully working iOS native application — an iPhone Multitasker app. A novice user will have a fundamental understanding of how a native iOS application is created from the ground up. The thesis also explains how the Multitasker mobile app was created using the Swift programming language. The subjects covered in the literature study demonstrate that it is surely feasible to construct an iOS application by following the user manuals and tutorials on iOS programming.

Keywords: iOS, Mobile Application, Swift, iPhone, XCode, Programming.

Aplikace pro iOS v prostředí XCode

Abstrakt

Cílem práce je rozšířit znalosti o efektivním vývoji mobilních aplikací pro iOS, rozhraní XCode a programovacím jazyku Swift. Primárním účelem této práce je ukázat, jak vytvořit uživatelsky přívětivou a plně funkční nativní aplikaci pro iOS — aplikaci iPhone Multitasker. Začínající uživatel bude mít základní znalosti o tom, jak se od základu vytváří nativní aplikace pro iOS. Práce také vysvětluje, jak byla vytvořena mobilní aplikace Multitasker pomocí programovacího jazyka Swift. Předměty zahrnuté v literární studii ukazují, že je jistě možné vytvořit aplikaci pro iOS podle uživatelských příruček a návodů na programování iOS.

Klíčová slova: iOS, mobilní aplikace, Swift, iPhone, XCode, programování.

Contents

1.	Introduction.....	10
2.	Objectives and Methodology of the thesis	11
3.	Part 1- Literature Review	12
3.1	Why Develop Mobile Apps	12
3.1.1	Increase business revenue	12
3.1.2	Reduce marketing cost	12
3.1.3	Improved customer experience	12
3.1.4	Get a competitive edge.....	13
3.1.5	Broaden your user base	13
3.1.6	AI and latest phone sensors.....	13
3.2	History of iPhones	13
3.3	Mobile OS Market Share	16
3.4	iOS Application Lifecycle	20
3.4.1	Life Cycle based on State Management.....	20
3.4.2	Life Cycle based on Framework methods.....	22
3.5	Drawbacks of Mobile Apps	24
3.5.1	Mobile apps cannot replace a website.....	24
3.5.2	Developing for Both Android and iOS	24
3.5.3	Listing for Both Apple and Google Stores.....	25
3.5.4	Other Drawbacks.....	25
3.6	Types of Mobile Apps	27
3.6.1	Native Apps	28
3.6.2	Web Apps.....	29
3.6.3	Hybrid Apps.....	31
3.7	iOS Native Development Vs Flutter Vs React Native	32
3.7.1	Installation & Architecture.....	32
3.7.2	User Interface.....	33
3.7.3	App Reloading	33
3.7.4	CI/CD Integration	34
3.7.5	Size of Application Build.....	34
4.	Part 2 – Development of Multitasker app	35
4.1	Introduction to XCode	35
4.2	Introduction to Swift.....	38

4.2.1	Closures.....	38
4.2.2	Tuples and multiple return values	39
4.2.3	Generics	39
4.2.4	Class and Structs	40
4.2.5	Extensions	41
4.3	Built in Error Handling in Swift	43
4.3.1	Throws and Throw	44
4.3.2	Try, do-catch	45
4.3.3	Try? and try!	46
4.4	Common mistakes made by swift developers.....	46
4.4.1	Unwrapping Optionals	47
4.4.2	Too much use of <i>self</i> keyword	48
4.4.3	Not using new features like Generics, Protocol Oriented Programming, Enums etc.	48
4.4.4	Not using functional programming features in swift	48
4.5	TestFlight by Apple	49
4.5.1	Benefits of Conducting TestFlight.....	49
4.5.2	The Pre-Requisites of TestFlight Beta Testing	49
4.6	Design Patterns Used in iOS App.....	50
4.6.1	MVVM (Model - View – View Model).....	50
4.6.2	MVC (Model – View – Controller)	50
4.7	Technical Details of Multitasker Modules.....	51
4.7.1	Details of plist.info.....	53
4.7.2	Source Code Management	54
5.	How Did I Develop Multitasker iOS App.....	55
5.1	Introduction to Multitasker application.....	55
5.2	Application Screenshots.....	57
5.3	Storyboard & Technical Components.....	68
5.4	UI Components used in App.....	81
5.5	Errors faced during development.....	82
5.6	Used Hardware & Software	88
6.	Future Vision	90
7.	Conclusion	91
8.	List of Figures Used in this Document	92
9.	Abbreviations/Acronyms Used	94
10.	References.....	95

1. Introduction

The recent technology advancements have a great impact on the mobile phones. With the advent of latest inexpensive chipsets in phones and modern phone sensors have changed the way people use their phones. With every passing day, the number of mobile apps is increasing as well as the number of consumers who use those applications. The arrival of AI and machine learning into the mobile phones has also increased the role of mobile phone apps in our everyday life.

Technology companies are also progressing in terms of ease and simplicity in software development of mobile apps. Cross platform applications are slowly making their way into the mobile market and giving tough competition to native iOS and android. The frameworks and tools to develop mobile apps are also becoming simpler day by day.

The thesis serves a basic information frame for developing a native iPhone application using Swift language. It should serve as a quick basic guide for new iOS developers. Although this thesis has covered all the major topics needed to develop an iOS application, however, due to the practical nature of iOS mobile app development, this document should not be considered a full-fledged step-by-step guide for developing an iOS application.

2. Objectives and Methodology of the thesis

The main goal of this thesis is to demonstrate how to create a user-friendly and fully functional iOS native application – a Multitasker app for the iPhone. This thesis will increase awareness about efficient mobile app development in iOS, the XCode interface and the Swift programming language. After reading this thesis, a beginner iOS developer should be in a position to create small but fully functional iOS application using Swift and XCode.

The diploma thesis is divided into two main parts. The first one is literature review, which is based on an analysis of secondary data sources, programming language documentations, including a professional literature, online blog articles, Internet publications about developing iOS mobile applications and documents about mobile technology.

The second section demonstrates the development of Multitasker application. Thesis will go through all the practical aspects of how the application was developed. It will also explain various sections of the application, technical details, what configuration was used etc. After going through this section, a beginner user will get the basic idea of how a native iOS application is developed from scratch.

3. Part 1- Literature Review

3.1 Why Develop Mobile Apps

Mobile applications deliver lot of value and business advantage as compared to web applications. Yes, there are certain use cases where mobile app may not be a perfect replacement of a web application. However, in most cases, mobile app complement web application to attract more customers and add revenue. As discussed at (matchboard, n.d.), mobile app delivers following advantages:

3.1.1 Increase business revenue

Mobile apps bring additional revenue in following ways:

- With mobile apps, there are more chances of repeat order. This is because customer is almost always carrying the phone and it is very easy to place order on the go.
- Company can take advantage from a new advertising revenue channel through mobile application
- Features like in-app purchase, premium features, ads etc. also contribute to increased revenue

3.1.2 Reduce marketing cost

Take the example of push notification to a mobile app user when he is near to his favorite restaurant. Traditional marketing would have costed more as compared to push notifications for example. This reduces the overall marketing cost.

3.1.3 Improved customer experience

Mobile apps are purposely built for handheld devices, so they offer much better UI/UX experience as compared to a traditional web application which struggles on small devices. Also, the customized experience based on the phone features or latest high-tech sensors creates a bond between mobile app and the consumer. Take the example of Uber mobile app. The latest location of car moving towards you is an example of an excellent customer experience.

3.1.4 Get a competitive edge

If your business belongs to a particular niche and your competitors do not have a mobile app, then developing a mobile app will offer great advantage over your competitors. Even if your competitors have mobile apps, then you can create a mobile app containing the various different features which are present in other mobile apps but not in one place.

3.1.5 Broaden your user base

Young consumers prefer to use mobile app instead of a web application. Not having a mobile application for your business will be losing the revenue from these consumers. Mobile application will never decrease your user base; it will always increase it. The millennials are major target user group of the mobile apps.

3.1.6 AI and latest phone sensors

The introduction of Machine learning and latest phone sensors have changed the way people use mobile phones. Take the example of Fall detect feature from (Apple, n.d.), the apple watch detects if person has fallen, and it generates an emergency SOS message to the contacts in the watch. Similarly, the trend in latest fitness trackers and mobile wearable has improved the quality of health of many consumers. The latest wearables offer highly complex medical features like Oxygen saturation level, ECG, blood pressure etc.

3.2 History of iPhones

Apple iPhones has evolved over the years. (Jones, 2022) and (Verizon, n.d.) has discussed the history of iPhones in detail. Find below a quick overview of history of iPhones.

June 2007: The first-generation iPhone is launched

The first iPhone was announced in January 2007, the original iPhone was launched as a combination of iPod, a revolutionary mobile phone and a groundbreaking Internet communicator. It featured a 3.5-inch screen, a multi-touch touchscreen displays, a microphone and headset controls.

July 2008: The first phone to beat the iPhone

Around a year after the first iPhone, iPhone 3G hit the market as its successor. It included various new hardware features like 3G data and GPS, but perhaps most notable introduction was launch of Apple Store. Apple store allowed users to browse and download millions of third-party applications

This iPhone 3G addressed the two main issues of the previous iPhone: cost and inability to access the high-speed cell phone networks.

June 2010: iPhone 4

iPhone 4 introduced a high-resolution retina display, multi-tasking feature and FaceTime. It was the first phone in which front facing camera was introduced by Apple. iPhone 4 was an excellent combo of software, hardware, performance, app selection etc.

October 2011: iPhone 5

iPhone 5 was launched in stores in September 2012. The “s” in iPhone 4S stands for Siri, Apple’s first intelligent personal assistant introduced at the time of the 4s.

It also introduced iOS 5, brought along iMessage, iCloud and Notification Center along with other notable features. iPhone 4s also housed Apple’s first 8-megapixel camera with 1080p video recording.

September 2014: Introduction of Plus size models

Two years after iOS 5 was released, Apple added 2 new models to the series. iPhones 6 and 6 Plus brought along faster processors, better cameras, and improved cell data connectivity. The 7 and 7 Plus added new color options and added water and dust resistance. The 3.5mm headphone jack was also removed with these models.

September 2017: iPhone 8 and 8 plus

The iPhone 8 and 8 Plus introduced wireless charging with the glass cover on the back of the iPhone. It brought in a much-improved camera with better tools for editing and filtering images. The true tone displays greatly improved the viewing experience by automatically reducing the blue light exposure.

November 2017: iPhone X

iPhone X was a revolutionary introduction by Apple. iPhone X brought in dual front-facing cameras through which consumers could take amazing selfies in Portrait mode. It was the first time, an Apple phone included Portrait mode for the front-facing camera. Some of the other super features included OLED screen technology, wireless charging, FaceID, Digital image stabilization, optical image stabilization etc.

September 2018: iPhone XS and XS Max

Apple introduced three new models: iPhone XS, iPhone XS Max and iPhone XR. Some of the cool features XS and XS Max brought, were faster Face ID, super retina in two sizes, largest display ever on iPhone, and a revolutionary dual-camera system.

iPhone XR was housed with a Liquid Retina display, which facilitated users to view true-to-life color from one edge of your screen to the other and that too on the largest LCD ever for an iPhone.

October 2020: iPhone 12

iPhone 12 had exactly the same features as the iPhone 12 mini, except that it claimed to have a 16-hour video playback compared to the iPhone 12 mini's 14 hours. This model also featured the dual-lens camera, 5G support, ceramic shield, A14 bionic chip, 16 core neural engine and exceptional battery life.

In the imaging department, iPhone 12 introduced an Ultra-Wide camera that captured Night mode images and a Wide camera that now captured 27% more light.

September 14, 2021: iPhone 13

The iPhone 13 introduced a 20% smaller notch, accompanied by a new camera layout. It runs on new A15 chip, which is much improved version over the previous generation of iPhones. Its camera brought some significant improvements including cinematic mode and photographic Styles. Excellent battery life and cost were some of the other salient features.

September 24, 2021: iPhone 13 Pro Max

The iPhone 13 Pro Max is one step ahead of iPhone 13 Pro. It added an additional GPU bringing it to total 5. Apple claims that the battery is able to support up to 28 hours of non-stop video playback.



Figure 1 - History of iPhones (Source (Proulx, n.d.))

3.3 Mobile OS Market Share

The market share of mobile operating systems has varied in last 20 years. There was a time when Apple was the sole leader, however, Android caught up with iPhone pretty fast. Find below a brief report on mobile OS market share based on the statistics from (Wise, 2022) and (7t.co, n.d.)

Find below an overview of Mobile OS market share from Jan 2012 to May 2021.

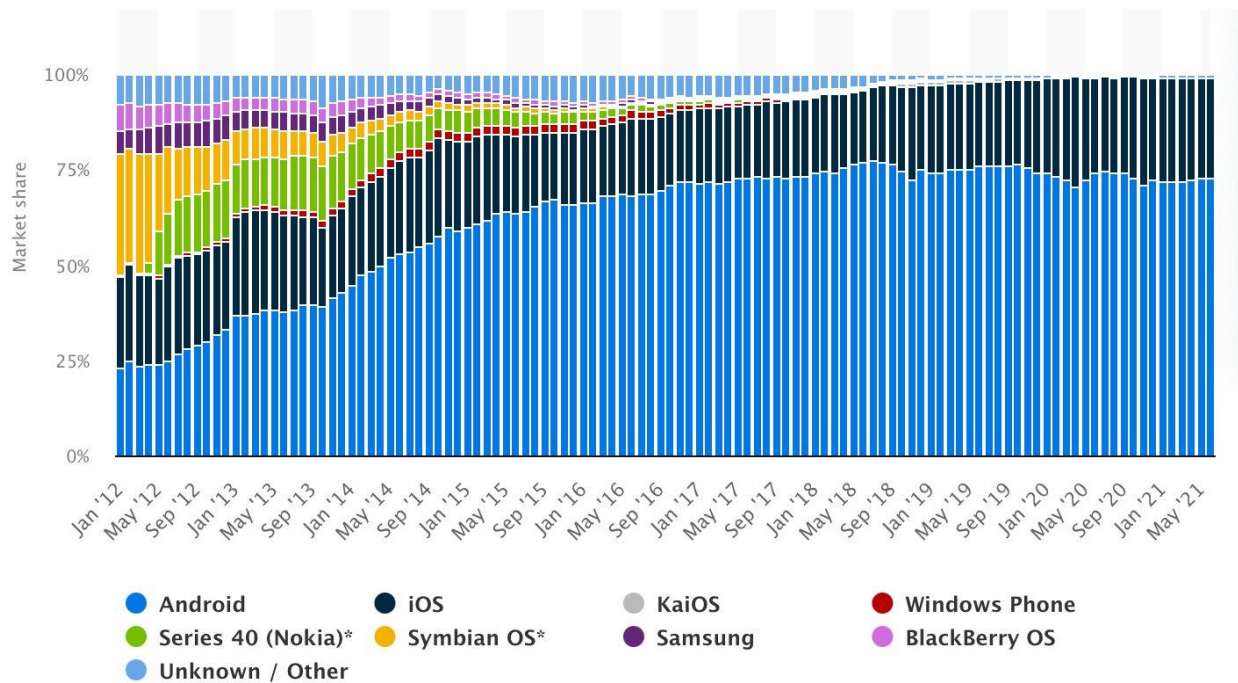


Figure 2 - Mobile OS Market Share

Apple phones have a large user base in china and world has seen a huge increase in Apple phone usage in China recently. Apple was able to capture more than 72% of China’s market share in the Q1 of 2021. One of the factors which played the role in this increase was the decline of the top end segment of Huawei phones in the mobile market. Globally, all Chinese brands are gaining popularity, with Xiaomi making new record volume this quarter with 86.6%.

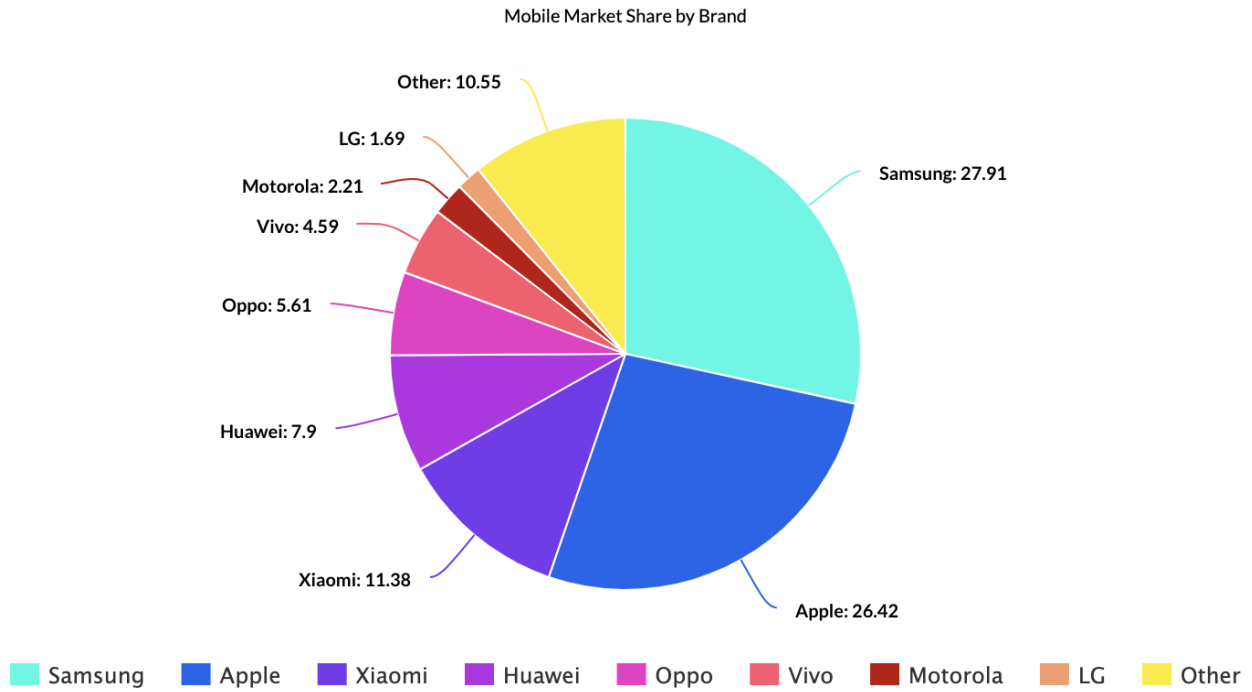


Figure 3 - Mobile Vendor Market Share

US & UK has more Android users as compared to iOS

iOS gained popularity in UK during the Q1 of 2021 with a 53% market share, leaving Android behind.

At the same time, Android is trailing in USA with just 38% market share during the Q1 of 2021. User of US and UK like Android more than iOS clearly.

Find below a graph of UK smartphone users, both android and iOS.

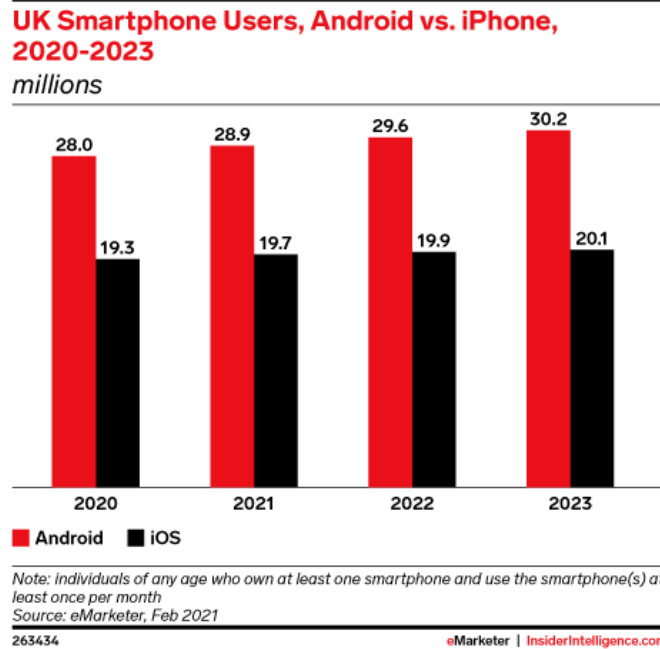


Figure 4 - Android vs iOS Market Share UK

iOS has 16.76% Market Share During Q1 of 2021

Although the devices produced by Apple are high quality they offer lot of diversity like iPhone, iPad, Apple TV, Apple watch etc. However, the growth of Apple products is not as fast as Google Android devices; this is because of high price of apple devices as compared to android and also Apple is considered slightly difficult to be handled by common people.

According to the statistics, iOS showed a decline of 14% from 2015 to 2020. It is believed that this decline is the aftereffect of the competition between Chinese rivals Xiaomi and Huawei.

iOS Devices Gained a 50.4% Year-over-Year Increase during Q1 of 2021

In 2020, Apple shipped 206.1 million iPhone units, which is 7.9 percent more compared to the previous year. Not only that, but they also managed to distribute 55.2 million devices in the market during the first quarter of 2021, gaining a 50.4 percent year-over-year increase.

19.3% Market Share in China

With a 19.3% market share, iPhone got more than twice the market share compared to the previous year, which is huge growth. The main rivals for iPhone in the Chinese market are Xiaomi and Huawei. iPhone sales have dropped from 71 million to 34 million during 2015-2020. Similarly, Q1 of 2021 saw a 13% drop in iPhone market share.

3.4 iOS Application Lifecycle

Understanding of iOS application life cycle is basis for developing good iOS applications. Let's divide the application life cycle in two parts. The first one will focus on life cycle based on the state management. While the second part will focus on, which methods will be called on various events in the application. To grasp the concept of life cycle in iOS, inspiration was taken from (Ramnath, 2014, p. 138)

3.4.1 Life Cycle based on State Management

Every iOS application goes through the following states whether it is developed in Swift or in Objective-C:

1. Not Running
2. In-active
3. Active
4. Background
5. Suspended

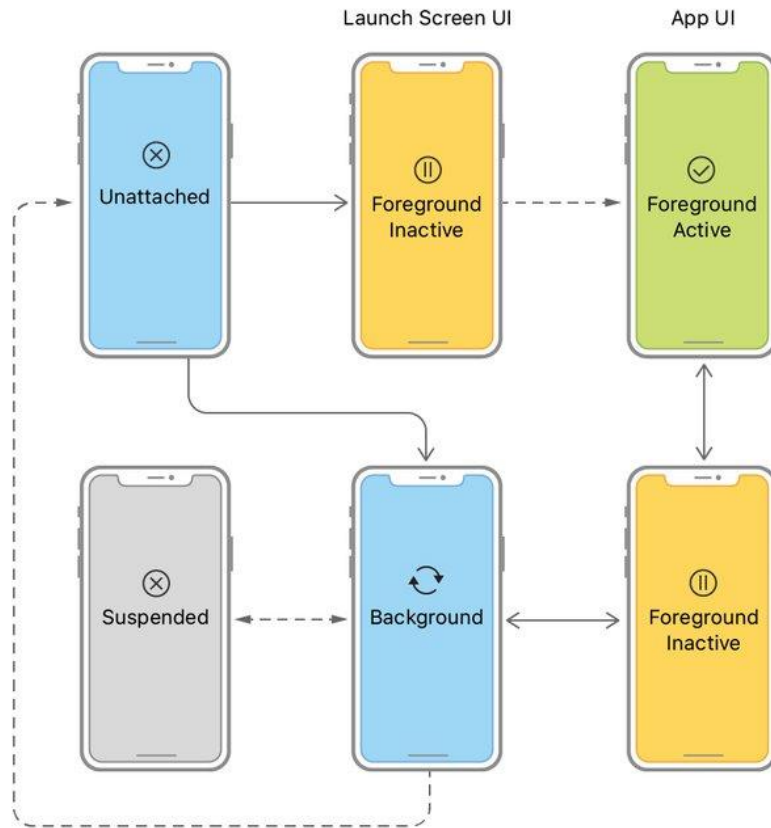


Figure 5 - Life Cycle Management

Not Running (Unattached): Application has not started yet or has been terminated by user/system.

In-active: The app is just entering the foreground state but cannot process events yet. It remains in this state for very brief amount of time.

Active: The app enters the foreground state and can process user events. This is the normal state when application is actively being used by the user.

Background: In this state, the application is in background, but it is executing the code.

Following are some of the scenarios when application will go into background states:

- When user clicks on home screen while using the application
- When application is doing some complex processing and it needs some extra execution time.
- Just before application goes into suspended state, it also transitions into background state for a small amount of time.

Suspended: In this state, the application goes into the background, and it does not execute any code. It is sitting idle in memory. It cannot execute any code and is considered frozen. System will give priority to other apps on foreground. System can terminate an application in suspended state anytime based on needs.

3.4.2 Life Cycle based on Framework methods

During the life cycle of an application, many iOS methods are called in a particular sequence and on particular events. (Prasad, 2018) has discussed these methods in detail. Find below a brief summary of the sequence of these methods along with a visual description.

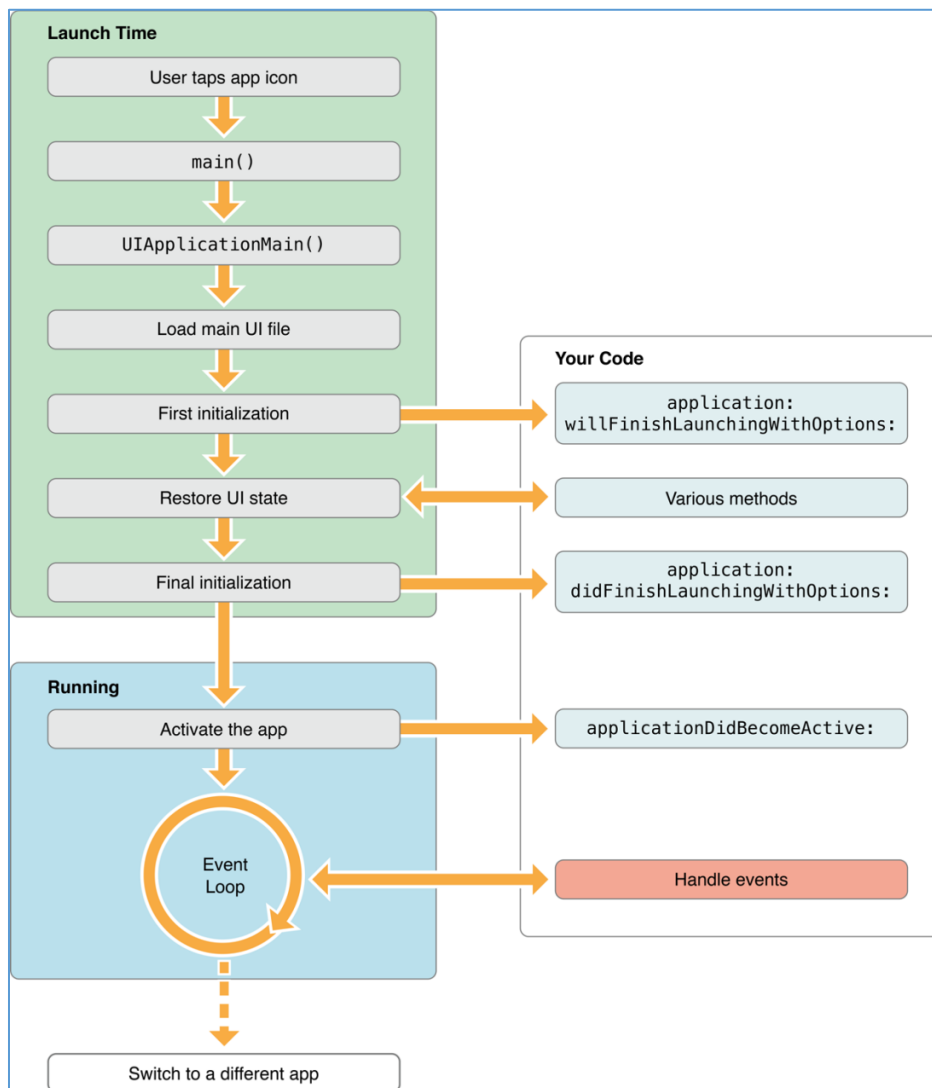


Figure 6 - Life Cycle Methods

1. *application: willFinishLaunchingWithOptions* - This is the first method from app delegate which is called after the application is launched successfully. Your code will be executed after the successfully launch.
2. *application: didFinishLaunchingWithOptions* - This method is called just before the app's screen is displayed. You can finalize your interface and can provide the root View Controller to the window.
3. *applicationDidBecomeActive* - There are two scenarios in which this method is called:
 - a. It informs your application that it moved from the inactive to active state
 - b. User ignores an interrupt e.g., incoming phone call, that sent the application to inactive state temporarily.
 - c. The best use case of this method is to resume any tasks that were paused (or not yet started) because the app had gone to inactive state.
4. *applicationWillResignActive* - This method is almost opposite to above method. This method informs your application that it is going to move from active to inactive state. This can happen in following cases:
 - a. If the user quits the application
 - b. In case of any interruption like a phone call alert
 - c. The best use case for this method to pause any ongoing tasks or disable alarms etc.
5. *applicationDidEnterBackground* - This method is called when there is a brief period of inactivity on the application. It informs your app that it is not running in the foreground. You have around five seconds to perform any action and return back. Your application will be terminated by system if you do not perform any activity during that time.
6. *applicationWillEnterForeground* - This method is called when app is doing a transition from background state to active state. The best use case for this method is to undo any changes you made to your app upon entering the background state. *applicationDidBecomeActive* method is called immediately after this method has finished its execution. After that, it transitions the app from inactive to active state.
7. *applicationWillTerminate* - This method is called to inform that your app is just going to be terminated. The best use case for this method is to perform any final cleanup tasks. You have around five seconds to perform any action and return back. System may kill the process if the method does not return before this time expires. This method may also be

called in scenarios where the app is running in the background state and system needs to terminate it for memory or other reasons. You shouldn't rely on *applicationWillTerminate* to be called in order to perform your mission critical tasks. There are few scenarios when *applicationWillTerminate* will not be called at all before app termination. One of the examples is, the system will not call *applicationWillTerminate* when the device is restarted.

3.5 Drawbacks of Mobile Apps

As mentioned in (Mariana, 2020), Although the mobile application market is still going strong and new apps are constantly being added to both Google and Apple stores, however there is also a trend that users are gradually declining to install new apps to their phones. It is true that there are other alternates of mobile apps in market, including the Progressive Web Apps (PWA) however, there are certain factors which limit the potential of mobile apps. Find below some of the disadvantages of mobile apps:

3.5.1 Mobile apps cannot replace a website

Native applications are a great tool for many businesses to engage customers by providing rewards programs, shopping applications, location finders, and more. Other companies are built entirely around a native application that is core of their business – and this can be a game, a utility, a tool like the Uber, or another form of entertainment. However, no matter what the use case for a native application, a successful business always needs a website too. So planning, building, and creating both a website and a mobile application is double the effort. And of course, double the money! That's one of the drawbacks of mobile apps that not all businesses can overcome. Certain apps which require large screen to operate, including the drag and drop UI widgets, managing a visual seat booking plan etc., these are example use cases for which a web application is a better choice as compared to a mobile app.

3.5.2 Developing for Both Android and iOS

To take maximum advantage of mobile applications, you will have to develop a native application for Android and a separate one for iOS, and even for Microsoft store as well

if your application supports desktop application too. Although you have option to go for the famous hybrid approach which is cross platform mobile application development, but you lose some of the benefits of native application development. This doubled cost will be a major factor in deciding whether you can go for native mobile application or not.

3.5.3 Listing for Both Apple and Google Stores

Whether you develop your application in native or in cross platform like reactNative, you will need not only to compile and build both types of apps on their native hardware (MacOS for iOS build), but you will also need to submit both apps to their respective stores. And Google might be a little forgiving in terms of application acceptance, Apple is not. Apple has strict rules for accepting applications into their Apple store. They have strong and comprehensive guidelines for accepting application into their store. It will need effort, knowledge, skill, and little bit cost to successfully get your application into Apple store.

3.5.4 Other Drawbacks

Compared to a website, which can be presented by just a URL, mobile app presentation to stakeholders for testing is not as easy. Yes, there are solution available like Diawi or Apple's test flight but there have always been technical challenges in adopting these technologies. Take the example of distributing iOS build. If the application is under development and you want to distribute it to many persons for testing, you cannot do it without using test flight, or you have to include the device UUID of every person in the iOS build. That is a big hassle compared to a website testing through just a URL.

It is not just about investing in development of website or a native mobile application. Both it also needs to be updated regularly and as needed whenever there is a product or change in platform libraries/SDK. Consider the Android and iOS updates, and that could be three different developers who need to be tasked with three sets of updates. Not only that, but application stores take time to approve updated applications and for consumers to benefit from an update, they must either download software updates or re-download an entire application.

There would have been many times, when you could not update your iOS mobile app on Apple store just because Apple wants you to accept an updated license agreement. Unless and until you visit the Apple iTunes, website and accept the new license agreement you cannot manage your application on store.

Undoubtedly, one of the biggest disadvantages of mobile apps is that their content is not indexed by search engines. It means they can't be optimized for organic traffic – and consequently, users won't be able to find your app through Google or any other search engine. Bots or crawlers cannot find your mobile application. As a result, driving traffic to a mobile application comes from mainly the marketing efforts of the application link, and store listings only. This is different than the website, where a website properly configured for SEO can be found by crawlers and will be indexed by search engines almost automatically.

Most of the mobile applications submitted to app stores, never make to their first download. Without downloads, consumer reviews to improve ratings and positioning are not possible, leading to a vicious cycle of no downloads, no reviews, no visibility, and again – no downloads. If marketing efforts fail or there is not enough budget to drive visitors to an application store link, a new application could fall through the cracks.

The chart below illustrates the potential dropout rate and point of drop-out of 1,000 consumer clicks that could result in an application download. Out of 1,000 prospective consumers, only 800 will actually land on the app store, only 640 will successfully find an application of their choice, more will abandon at the “Accept Permissions” stage, and only 262 out of 1,000 are likely to use an application.



Figure 7 - Mobile App Usage (Mariana, 2020)

With the passage of time, many consumers around the world are less inclined to download new applications, preferring instead to conserve their device storage, even though the phone storage is becoming cheaper with the advancement in phone hardware. According to Zipwhip 2019 State of Texting Report, around 61% of consumers will not install a new application at all, this is a big number.

Although the focus of the report is, the use of applications for communication, its findings indicate a growing consumer trend. Zipwhip believes, a typical user struggles with data management and **21% of** consumers abandon new applications after their first download. Not only that, a more than 76% of consumers will never use an application again 3 days after installing it.

3.6 Types of Mobile Apps

Mobile application development has come a long way during last 10 years. There was time where only native mobile application existed. Then came the hybrid apps with PhoneGap and other hybrid mobile app development frameworks. ReactNative and flutter gained lot of popularity for cross platform mobile application development. Another rather simple and easy way of mobile apps was progress web apps (PWA). Find below we will discuss various types of mobile applications and what purpose do they serve.

According to (Valdellon, n.d.) There are three basic types of mobile apps based on the technology and how we code the apps:

1. Native apps are developed specifically for one platform or operating system.
2. Web applications are responsive versions of websites that can work on any mobile device or operating system as they are rendered using the mobile phone browser.
3. Hybrid or cross platform apps are combination of both native and web applications, but wrapped within a native app, which makes it capable of having its own icon or available on app store. Hybrid app is basically another layer built on top of native application backbone.

3.6.1 Native Apps

Native mobile applications are called native because they are developed for native operating system of a mobile device whether it is a phone, tablet, or a wearable. It is perfectly fine to develop an android native application as well as native iOS application, however both must be developed separately having their separate codebase. Since they are developed targeting just one platform, you cannot mix both. i.e., you cannot use an android application on apple phone and vice versa.

Technology Used: Native apps are developed using various different programming languages. Some examples include: Kotlin, Java, Swift, Objective-C, React, C++ etc.

Pros: Native apps are generally faster, and they have much better performance as compared to non-native apps. This is because they are closer to device's native operating system. They use mobile device's resources more efficiently as compared to other types of apps. Also, Native apps utilize the native device user interface, giving users a much smoother and elegant user experience. And because the native apps connect with device's hardware directly without any intermediate layer, they can take advantage of device's features like GPS, Bluetooth, phonebook contacts, camera roll, NFC etc.

Cons: As the native app is specific to one platform so you have to double the efforts to make it available to another platform. The code you write for one platform cannot be reused on another platform at all. This has impact on the costs. On top of that, one has to put efforts to maintain and update the codebase for each version. Last but not least, whenever there is an update to the app, the user has to download the new version and reinstall on the phone. This also means that native apps occupy more space in the device's storage.



Figure 8 - Pros and Cons of Native Apps

3.6.2 Web Apps

Web apps operate similarly to native apps but are accessed through your mobile device’s web browser. They are actually responsive websites that adjust its user interface according to the screen size, aspect ratio and the device the user is on. In fact, when a user comes across the option to “install” a web application, the URL of the website simple added as a bookmark on your device.

A recent progress in mobile web apps is the progressive web app (PWA), which is basically a native app running inside a browser. An icon is placed on the mobile app desktop through which user can access this web app.

Technologies Used: As they are web apps, so they are developed using HTML5, CSS, JavaScript, Python, and similar programming languages used for web development.

Pros: Since it is web-based, so you do not need to customize it for a particular platform or operating system. This reduces the development costs. Not to mention, you do not have to go to app store and download anything. Also, it will not take up space on your device's memory like a native application does. That makes maintenance much easier as well.

Cons: As the web application runs in the browser, that means, the cross-browser compatibility will be an issue. There will be some features of HTML or CSS which will be available in one browser perfectly, however the same feature might not work properly on another browser on same phone. Also, unlike the native apps which can run in offline mode, the web apps cannot work offline. They are just shells for website, so they do not have capability to work offline.



Figure 9 - Pros and Cons of Web Apps

3.6.3 Hybrid Apps

Hybrid apps have come a long way. In the early stages of hybrid apps, the strong integration with hardware was missing. Using features of phone camera, GPS, Bluetooth was not easy. However, with the passage of time, the gap between hybrid apps and native apps reduced. Hybrid apps are web apps that look and feel just like native apps. In fact, a non-technical user might not be able to confirm if this is a native app or hybrid app. They have all the features which hybrid apps have, like, home screen, app icon, responsive design, fast performance, offline mode etc. but actually they are web applications which are developed to look like native apps.

Latest advancement like Xamarin and flutter has changed the way hybrid apps are developed. It is one step forward towards closing the gap between hybrid mobile apps and native apps.

Technology Used: Hybrid apps use a mixture of web technologies and native APIs. They're developed using: React, C#, Dart, Swift, HTML5, and others.

Pros: Development of a hybrid application is faster and more economical than a native app. A hybrid app is a perfect use case for developing an MVP (minimum viable product) of a product. If you need quick turnaround time to develop an app and to showcase it to investors, hybrid app is your way to go. They also load rapidly, and they are ideal for usage in countries with slower internet connections. They give users a consistent user experience, although they might lack the finesse in animations etc. Also, because they use a single codebase, there is much less code to maintain.

Cons: Hybrid apps might lack in power and speed, which is a common drawback across all hybrid apps. Since they are relatively new in the market, so there is also lack of support for integration with 3rd party SDK's or API's. If you prefer a high-quality application having complex integrations, then hybrid app is not the best choice. Note that hybrid apps are one layer on top of the native platform, so it might not run as smoothly as the native apps do, however this difference will be minimal in small, simple applications.



Figure 10 - Pros and Cons of Hybrid Apps

3.7 iOS Native Development Vs Flutter Vs React Native

Cross platform application development has come a long way. React Native and Flutter are two of the most popular choices for developing cross platform hybrid applications. Both have their pros and cons. As discussed in (Sharma, 2021), Here is a brief comparison of these three options:

3.7.1 Installation & Architecture

After Apple launched the swift language, developing native iOS applications has become more exciting. Swift is a versatile, fast, and type-safe programming language. In order to develop an iOS app, all you need to do is, download and install XCode as the IDE and install iOS SDK for development. Swift provides a wide range of interesting features to developers, and it is used not only for iphone and iPad app development but also for development of apple watches, apple TV etc. Every Apple device e.g., Apple TV, watches, iPod, iPad etc. uses an application developed in swift.

React Native is an open-source cross platform mobile application framework created by Facebook. It comes under the category of hybrid mobile app development platform. It is used to develop iOS and Android apps using the single code base. React Native use JavaScript, HTML and CSS for development. React Native is an extra layer on top of the native iOS platform, it creates a bridge to communicate between native iOS and JavaScript. Json messages are used to carry out this communication.

Flutter is an open-source, cross platform mobile platform from Google. Just like react native, it can be used to develop iOS and Android apps from the single code base instead of maintaining two separate code bases. Flutter uses Dart language which is gaining popularity due to its powerful features, ease of use and it is also based on OOPS principles. The good thing about Flutter is, that it does not require the bridging concept to communicate with native components as it already contains everything inside it. Not only that, but it also provides full support to native features.

3.7.2 User Interface

As discussed in (Langer, 2012, p. 24), User interface plays a prominent role in any business application and proper business analysis of target business audience is very important. Although using the bridging concept, you can develop native UI easily and react native also provides few native UI features. However, sometimes it becomes difficult to replicate complex native UI components.

Whereas Flutter provides a UI package which facilitates in using native UI features and develop application having good user experience. Native iOS outperforms both flutter and react native when it comes to complex UI components like animations.

3.7.3 App Reloading

Reloading the app is very easy using SwiftUI. And developers can use non-native solutions to add new functionalities as well.

React Native uses hot reloading to update the changes in hybrid application. React Native uses virtual DOM which compares the current changes with last changes and updates only latest changes in code.

Like React Native, Dart also uses hot reload feature, and it is considered even faster when incorporating changes in your code.

3.7.4 CI/CD Integration

For CI/CD integration, we use tools and libraries specific to each development framework. For swift, we use either Fastlane or Jenkins to automate the build and integration process. React Native can also make use of Jenkins or Fastlane.

Flutter framework uses Nevercode that uses Codemagic CI/CD tool for implementation of continuous integration and continuous deployment.

3.7.5 Size of Application Build

Swift based iOS app size is usually smaller because it uses built-in tools and libraries for most of the work. For React Native, app size is little bit more than Swift because React Native makes use of lot of third-party libraries. Flutter iOS app is even more than React Native & Swift due to the size of Dart Engine.

	Flutter	React Native	Native
 Performance	Better	Good	Best
UI and graphics	Best	Best	Best
Compatibility & features	Better	Good	Best
Time to market	Better	Best	Good
Engineering cost	Best	Better	Good

Figure 11 - Comparison of Flutter Vs React Native Vs Native

Source: <https://nix-united.com/blog/flutter-vs-react-native/>

4. Part 2 – Development of Multitasker app

4.1 Introduction to XCode

XCode is the default IDE for developing mobile applications in swift. Since its launch in 2003, it is the first choice for iOS native development among the beginners and veterans alike. XCode is a tool through which developers create applications for different apple platforms like iPhone, iPad, Apple TV, and apple wearables.

Before swift programming language came into being, Objective-C language was used to develop applications through XCode. However, after swift's launch in 2014, Objective-C is not used in any new application development. There are many legacy iOS apps, which were developed on Objective-C, and they are still in use.

According to (softwaretestinghelp, n.d.) , Through XCode, developers can develop a complete application from scratch and submit it to Apple store. Here are some of the core features of XCode which are used by developers every day:

- Designing user interface
- Writing application code
- Compiling the code and resolving any compilation errors
- Testing the code
- Deploying the application to simulator
- Submit the application to apple store

Following are the minimum requirements which must be met in order to download and use the XCode.

1. Minimum macOS version i.e., macOS Big Sur 11.3
2. Minimum SDK's i.e., iOS 15.2, macOS 12.1, tvOS 15.2, watchOS 8.3
3. Supported Swift Versions Swift 4, Swift 4.2, Swift 5.5

Find below minimum the table for minimum requirements, taken from Apple website (Apple, n.d.)

Minimum requirements and supported SDKs

Xcode Version	Minimum OS Required	SDK	Architecture	Deployment Targets	Simulator	Swift
Xcode 13.2	macOS Big Sur 11.3	iOS 15.2 macOS 12.1 tvOS 15.2 watchOS 8.3 DriverKit 21.2	x86_64 armv7 armv7s armv7k arm64 arm64e arm64_32	iOS 9-15.2 iPadOS 13-15.2 macOS 10.9-12.2 tvOS 9-15.2 watchOS 2-8.3 DriverKit 19-21.2	iOS 10.3.1-15.2 tvOS 10.2-15.2 watchOS 3.2-8.3	Swift 4 Swift 4.2 Swift 5.5

Figure 12 - XCode Requirements

As mentioned in (Chris, n.d.), XCode is only supported on Mac OS. If you want to develop iOS applications on Windows OS, then there are few works around options. One of the workarounds is make use of virtualization (e.g., virtualbox, VMware etc.). Another option is to rent the Mac online. There are other options too, but these two are the most commonly used options. Please note that the level of performance, flexibility, reliability, and ease which is available on Mac OS, is not available on any other OS.

Every new version of XCode brings some exciting new features and XCode 13 is no exception. (Allen, 2021) has described lot of useful features in XCode 13 and here are some of the important highlights of this XCode:

- Source Code Editor Improvements

(Hudson, 2021) has discussed that the auto-completion of code editor has improved a lot. Not only that, XCode is also able to detect if you are trying to unwrap an optional and it will complete the code block for you, interesting, isn't it?

Below is an example code block.

```
Struct CloseContract {  
func close(contractId: Int?) {  
if let cont  
}  
}
```

As you're writing `if let contrac`, XCode will offer the correct completion: `if let contractId = contractId`.

- Design Improvements

XCode 13 brings with itself an improved project navigator design. It now has icons for different file types and file extension names are not shown by default. This results in a clean and compact look.

- XCode Cloud

With the rapid growth in devops and continuous integration, XCode cloud is a remarkable step towards CI/CD. It offers parallel testing across different device types, automatic push your application to Apple's test flight so that your builds could be tested easily.

- DocC

This was a long-awaited feature. Now developers can create the documentation directly from the code. It improves the user experience of your codebase. Through *DocumentationCompiler(DocC)*, developers can create not only documentation, but tutorials and articles for the code base as well. And it has native integration with Apple's default documentation viewer too.

Following are some of the common issues faced by junior developers, as mentioned on Apple website (Apple, n.d.) :

- 1- If you are unable to grant XCode Cloud access to your code repository, kindly ensure required permissions are assigned to connect XCode Cloud for your code repository.
- 2- If you are facing issues related to dependencies, please make sure to review project dependencies. If you are using cocoa pods to manage dependencies, ensure you have committed your Podfile and Podfile.lock files to code repository and installed CocoaPods correctly.
- 3- If your build fails with an error about a missing app capability, make sure that your app ID has all the required capabilities added. This is the same app ID which you used, when you configured your first XCode Cloud workflow.

- 4- If you are facing build errors, and you are using new build system, switch to the default build system and see if it works.
- 5- Make sure that there is enough free disk space left in the system.
- 6- Sometimes using “clean” and then build resolves the build issues.

4.2 Introduction to Swift

Swift is a powerful language, and it offers some very useful features which facilitates developers in many aspects. As discussed in (Wilson, 2020) and (Lim, 2020, p. 522), Some of the salient features include Closures, Tuples, Generics, builtin error handling, classes/structs. Extensions etc. Let’s discuss these in detail.

4.2.1 Closures

As per (tutlane, n.d.) , Closure is a block of code which is self-contained and can be passed to another method as parameter. Closures can capture and store a reference to any constant or variable. Closures are designed for variables and constants; means we assign the value in it and then pass it to the function parameter.

Swift has introduced a special syntax for passing the closure. It is called trailing closure syntax. Instead of passing the closure as a parameter, you can pass it right after the function inside the curly braces (). It's an easy and flexible way for developers, that’s why more and more developers are adopting this practice.

Generally, in swift, functions are regarded as a special type of closures, and it can take any one of three forms

- **Global Functions:** These are considered closures having a name but do not capture a value.
- **Closure Expressions:** These are unnamed closures that are written in lightweight context and can capture values from its surrounding context.
- **Nested Functions:** These are the types of closures which can capture values from the functions encapsulated in another function.

Swift has a shortcut syntax that lets you go even shorter. Instead of typing string (variable) in, we can let Swift provide automatic names for the closure’s parameters. These are named with a dollar sign and a number starting with 0.

4.2.2 Tuples and multiple return values

Although Tuple is not considered as an official collection type by apple, however it is an important data structure to be used in swift programming. Tuples are the new collection-like type found in Swift. Tuples fit in on odd place between structs and arrays, but allow for quite remarkable flexibility in code, especially when returning multiple values in a function.

Just like an array is a collection type containing elements of the same type, A tuple in swift is a collection type which contains values of different types. Although This does not make them an alternate for arrays, but a temporary way of moving several values around simultaneously. For understanding of Collection types, inspiration was taken from (raywenderlich.com Team, 2017).

Declaring a tuple is super easy. Here are some examples:

```
let mySwiftCourse = ("Udemy", 5)
```

You can add names for the parts of the tuple.

```
var mySwiftCourse = (vendor:"UdmeY",rating:5, complexity:"Easy")
```

There are various ways to retrieve values from a tuple. One of the easiest ways is to do the reverse of assigning the tuple, creating two variables with data of the tuple, like below:

```
var (vendor,rating) = mySwiftCourse  
println("Your course is from \ \(vendor) having rating of \ \(rating)")
```

4.2.3 Generics

As discussed in (Hudson, Pro Swift Break Out of Beginner's Swift, 2016, p. 106), Generics help you write the code once and then reuse it later. Swift is a type of safe language which means you need to explicitly specify the type when passing to any function. If you define a variable of type Integer, then you cannot pass a string to it. However, sometimes we need to have a function that can handle more than one type, or we need to work with types which should not be strict, that's where generics come into play.

Let's take an example based on (hackingwithswift, n.d.), if you have to implement a specific protocol called *XProtocol* and you want to create a function that takes few parameters and it return it. The function below can only work with the *XProtocol* type.

```
func hello(first: XProtocol, second: XProtocol) -> XProtocol {  
//more code..  
}
```

What if you have a protocol of different type e.g., *ZProtocol*? then we probably have to make another function which accepts *ZProtocol* of as a parameter. This will result in code repetition, and it is against the DRY (Don't repeat yourself) code principle. Let's solve this problem through generics. Generics allow you to create one single method that is tailor-made for the type that invokes it. Let's modify the above example to incorporate through generics:

```
function hello<A: XProtocol>(first: A, second: A) -> A {  
//more code..  
}
```

In the code above, the placeholder type *A* is an example of a *type parameter*. A *type parameter* specifies and names a placeholder type and is written immediately after the function name between the two angle brackets (<A>). So, *A* will be replaced with whatever type you pass in at runtime.

4.2.4 Class and Structs

According to (Swanner, 2020), both class and structs(structures) are very similar. They are basic building blocks of Swift language, and every good swift program is based on reusable and solid code chunks of class and struct.

Find below some of the similarities between both class and struct.

- They both store values.
- Both allow access across your codebase.
- Both struct and class define initializers.
- Both can be extended further in your code.
- They comply to standard functionality protocols.

Classes are more intelligent and more suited to complex business logic. Whereas structs are more suited to static logic which will not be modified later. Here are some of the differences between class and struct:

- Class can use make sure of inheritance but struct cannot.
- Classes can use type casting at runtime while struct cannot.
- Classes can make use of de-initializers to free up resources.
- Classes allow reference counting for multiple class references.

Here's an official example of Class and struct from Apple's Swift website (Apple, n.d.)

```
struct Resolution {  
    var width = 0  
    var height = 0  
}
```

As you can see, the struct contains the static data. Width and height are normal methods for identifying resolution, and they are stored as variables so they can be altered as needed later on.

```
class VideoMode {  
    var resolution = Resolution()  
    var interlaced = false  
    var frameRate = 0.0  
    var name: String?  
}
```

Now let's analyze the Class. It is not only accessing struct's data for resolution but also adds other features like interlacing, a frame-rate count, name of the apple device etc.

4.2.5 Extensions

As the name suggests, extensions in swift extend the functionality of an existing class, structure or enumeration type. Please note that you can add type functionality with extensions, but you cannot override existing functionality with extensions.

According to (tutorialspoint, n.d.), Some of the powerful features of extensions are as follows:

- You can add functions and computer properties
- You can define instance and type methods
- It allows to use new initializers i.e., constructor functions
- You can define subscripts with subscript() function
- You can define and use new nested types
- You can make an existing type conform to a protocol

Extensions are declared with the keyword 'extension'. Here is an example:

```
extension MyType {
    //extend and existing function.
}
```

4.2.6 Swift UI

Swift UI is the new framework from Apple, and it was launched with iOS 13. Gradually it is replacing the previous UI framework of UIKit. Through SwiftUI, you can design and developer highly powerful user interfaces declaratively and without the need to write too much code. Not only the syntax of SwiftUI is easy to understand but you can also preview SwiftUI project in automatic preview easily.

(steelkiwi Inc, n.d.) and (Yu, 2021) has discussed SwiftUI in detail, here are some of the core features of SwiftUI are following:

- Drag-and-drop components: Using SwiftUI lets you drag a button or other UI component from the object library and drop it onto the canvas. Swift UI will automatically write the relevant code. This drag-and-drop method even applies to attributes like font weight etc.
- Reusable UI components: After you have successfully created layouts in SwiftUI, you can further reuse them anywhere in your application. For example, if you've created a photo album carousel which shows images thumbnail and clicking on a particular image shows enlarged version of image, that component can be reused by extracting a new subview.
- Vertical-Horizontal-Z Axis Stack: This is an interesting feature of SwiftUI. Through VHZ stack, developers can create complex designs by dragging and dropping elements to any orientation either vertical or horizontal or even the Z-

axis of other elements. It is just like building within rows or columns, with no manual coding required.

- Build across Apple platforms: With the rise in cross platform app development, this is not a surprising feature. With SwiftUI, it is very easy for developers to build across Apple platforms like WatchOS, TV OS, and macOS by using the subview components made in one app across other apps.
- You can use a hybrid approach of using both SwiftUI and UIKit using `UIHostingController`.
- SwiftUI provides mechanisms for reactive programming. Developer can use `ObjectBinding`, `BindableObject` and the whole `Combine` framework.

However, there are some disadvantages of SwiftUI as well. Here are some of those:

- As a relatively new entrant, it needs minimum iOS 13 and minimum XCode 11. If you decide to use SwiftUI then you are abandoning the users of the older versions of the iOS. But more and more applications are now using SwiftUI for UI development and new apps on apple store are now using it.
- As it is relatively new, so the community support is limited as compared to UIKit. However, the number of SwiftUI posts on Stack overflow are now increasing and with the passage of time, the support for SwiftUI related issues is growing rapidly.
- It is difficult to examine the view hierarchy in XCode Previews. In XCode 13, developers do not have the ability to debug XCode Previews.

4.3 Built in Error Handling in Swift

According to (journaldev, n.d.), Error handling is defined as the process of catching and handling various errors in the application. Swift has a robust system of error handling. Generally speaking, we can divide the errors into three categories:

- 1- Informational Errors. Which are intentionally shown to user. E.g., Incorrect Pin code.
- 2- Errors which must be displayed to help other developers e.g., Code merge errors
- 3- Errors which stop application from performing it optimally

A basic way to handle errors is to use *If else* statements but it results in too many nested conditions and redundant code. Swift treats errors as values of a certain type, however checked exceptions are not supported in Swift. Find below some of the ways provided by Swift to handle errors:

4.3.1 Throws and Throw

If a function (or initializer) might throw an error, the *throws* keyword must be added in the definition itself right after the brackets and just before the return type. See a simple example below.

```
func userTest() throws -> <Return Type> {  
    }  
}
```

The *throws* keyword will propagate the error from the function back to its calling code. The code which will call this function, must add a try catch block so that it could handle any error thrown by this function. The keyword *throws* indicates that this function might throw an error whereas the keyword *throw* will actually throw an error.

Let's look at an example demonstrating *throws* and *throw* in a function:

```
func validateUserName() throws {  
    if <condition_matches> {  
        //Add your function code here  
    }  
    else{  
        throw UserNameError.noSpecialCharaterAllowed  
    }  
}
```

In Error Handling, *guard let* is useful in the sense that we can replace the return statement in the else block with the throwing error. This prevents too many if else conditions. Find below another simple example:

```
func validateUser(invoiceCode: Int, accessCode: String) throws {  
  
    guard invoiceCode > 0 else{
```

```

    throw UserDetailError.invoiceCodeNotValid
}

guard accessCode.count > 0 else{
    throw UserDetailError.accessCodeNotValid
}
}

```

In the above-mentioned code, if any of the condition becomes true then it will throw an error and the function would return the control to its calling function.

4.3.2 Try, do-catch

Just like try-catch is used in java and many other languages, Swift uses do-catch block to handle errors. Each function that has *throws* keyword must set in the try statement since it might throw an error.

Note that the *try* statement will execute only when it is inside the do-catch block. Find below a brief example:

```

do{
    try userValidate(pin: 0, name: "")
} catch let error {
    print("Error: \(error)")
}

```

Here is another way to handle it, using multiple *catch* statements:

```

do{
    try userValidate(pin: 0, name: "")
}
catch UserDetailError.noValidName
{
    print("The name is not valid")
}
}

```

```

catch UserDetailError.noValidpin
{
    print("The pin is not valid")
}
catch let error {
    print("Unspecified Error: \(error)")
}

```

4.3.3 Try? and try!

Try? was a relatively new keyword which came with XCode 7. You can use *try?* keyword handle errors by converting them into an optional value. As a result, when an error actually occurs, the function will return a nil which is a valid value for an optional. That will eliminate the need to add the *do-catch* block.

Try! is used to declare that the error will not occur. Use it only when you are 100% sure that the function will not throw an error. Just Like *try?*, *try!* works without a *do-catch* block. Here is an example of both keywords.

```
var n1 = try? Patient(name: nil)
```

```
var n2 = try! Patient (name: "John")
```

please not that using *try!* In your code will disable error handling at all and it will stop propagating the error. If error occurs, the application will crash.

When you use *try?* you are ignoring the actual error which took place. You should use it in scenarios where overall success or failure is more important than the error itself.

4.4 Common mistakes made by swift developers

It is not uncommon for junior developers to face issues when adopting new programming language. (Agrawal, 2017) has evaluated these issues in detail. Following are some of the programming mistakes commonly made by developers who are new to swift:

4.4.1 Unwrapping Optionals

Forced unwrapping of optional is one of the frequent mistakes by beginner Swift developers. Optionals are a very powerful feature of Swift. They are just types similar to `int` and `String`. As discussed in detail in (Keur, 2015, p. 87), Optionals are annotated with a question mark after the type declaration. Here is an example which show how to declare an optional string:

```
var swiftVariable: String?
```

This will let the compiler know that either there can be a valid value or no value at all. Please note that `String` and `String?` Are two different types, they are not just a variation of same type.

In order to extract the value of an optional, you must first unwrap the optional. There are many ways of doing this. The incorrect way to do is, is by using the bang operator. The exclamation sign `!` is the bang operator which is used to perform the operation of unwrapping. The problem occurs, when you try to unwrap an optional which does not hold a value (`nil`). This results in crashing your code. Here is an example below:

```
var xVariable: String?  
var yVariable: String = "hello world"
```

```
func executeMethod() {  
self.yVariable = self.xVariable!  
}
```

In the above example, the app will crash because the value for `xVariable` was never defined, and we are trying to assign it to a variable of type `String`. This kills the whole purpose of optionals, which were introduced to protect from errors like this!

Here is one of the correct ways to do it.

```
var someVariable: String?  
var somethingElse: String = "hello world"
```

```
func executeMethod() {
```

```
    if let theThing = someVariable {
        self.somethingElse = self.someVariable!
    } else {
        print("error")
    }
}
```

The only thing new to this example is the optional binding. As a result, instead of crashing, the code enters the else statement and prints "error."

4.4.2 Too much use of *self* keyword

As discussed in (Neuburg, 2017, p. 19), it is not mandatory to use *self* to access a class' or struct's properties inside a method. It is needed only inside a closure where it needs to capture *self*. Frequent use of *self* is not an error; however, it will result in unnecessary and inconsistent code.

4.4.3 Not using new features like Generics, Protocol Oriented Programming, Enums etc.

Developers usually do not take advantage of new features introduced in Swift. Take the example of Enums. Enums is not just a simple list of related constants, it is much more than that. you can attach a value to each enum case. Enums can also have methods and computed properties that can be used to add more details to each case.

4.4.4 Not using functional programming features in swift

Swift offers many methods which are basis for functional programming. Lot of functionality is encapsulated inside these methods. Instead of using writing lengthy code to achieve something, you can use methods like map, filter, reduce etc. to achieve the same purpose using very few lines of codes.

4.5 TestFlight by Apple

TestFlight is an Apple product through which you can invite users to test your iOS, iPadOS, watchOS and tvOS applications before you release them to the App Store. It is one of the most popular beta testing applications from Apple.

4.5.1 Benefits of Conducting TestFlight

TestFlight is a powerful tool and some of its major features including following;

- The process is simple, and you can easily test all your new applications.
- Through its built-in dashboard, developers can manage and track all the tests.
- You can easily distribute your applications over-the-air to the testers.
- You can perform testing on various different devices at the same time so you can easily find test failures.
- Through TestFlight, you can get many meaningful metrics and reports about the OS versions, device models etc.
- It enables you to collect feedback early in the testing, that way, all the feedback can be incorporated into the app before its release.
- It enables you to receive application crash reports too.

4.5.2 The Pre-Requisites of TestFlight Beta Testing

Setting up TestFlight is not complex at all. Find below some of the pre-requisites for get it up and running:

- Application ID.
- A certificate for distributing apps.
- Device UDID.
- A developer ID for accessing the Apple developer account.
- An Ad Hoc provisioning profile so that the application could be distributed to the tester's devices.

Note: The Multitasker application was not tested through the TestFlight.

4.6 Design Patterns Used in iOS App

According to (wikipedia, n.d.), *a software design pattern is a general, reusable solution to a commonly occurring problem within a given context in software design. It is not a finished design that can be transformed directly into source or machine code. Rather, it is a description or template for how to solve a problem that can be used in many different situations.*

Two major design patterns used in the iOS mobile app development include following:

4.6.1 MVVM (Model - View – View Model)

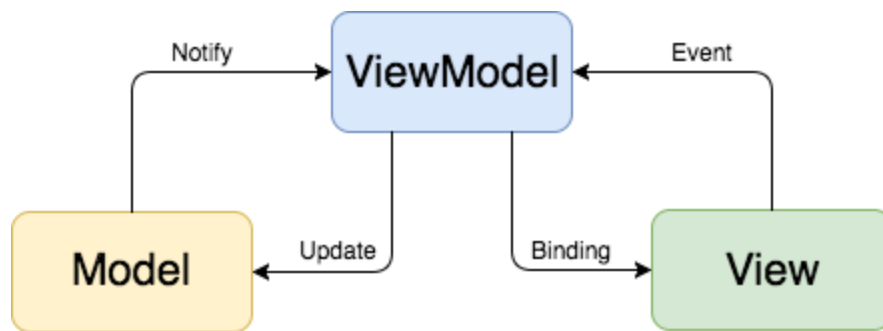


Figure 13 - MVVM Diagram

Source: (Benoit Pasquier, n.d.) <https://benoitpasquier.com/ios-swift-mvvm-pattern/>

MVVM is also a very common design pattern used in the iOS development. As above diagram depicts, there are three main components just like in MVC. For understanding of this pattern, inspiration was taken from (Hudson, Swift Design Patterns, 2018, p. 37)

4.6.2 MVC (Model – View – Controller)

Model view controller is very famous and most commonly used design pattern in many modern programming languages. And swift is no exception.

In the Multitasker application, the modules of General Knowledge and Side Menu utilized the MVC pattern. According to (Laso-Marsetti, 2019):

- Model is where your data is residing
- View is the face of your application
- Controller acts as a bridge between view and controller through the delegation pattern

Find below a high-level diagram explaining the flow of the MVC pattern in iOS:

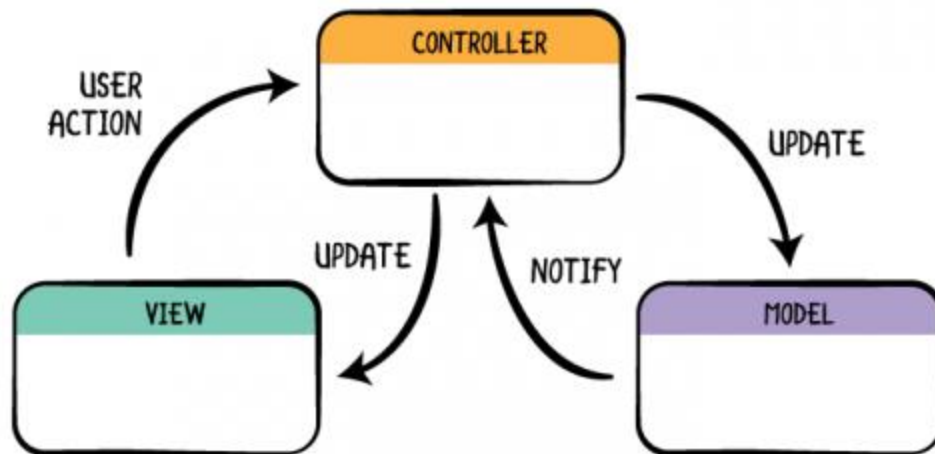


Figure 14 - MVC Diagram

Source: <https://www.raywenderlich.com/1000705-model-view-controller-mvc-in-ios-a-modern-approach>

4.7 Technical Details of Multitasker Modules

Find below the technical details of each section of the application. Technical details include following:

- Which built in library/framework was used in this module?
- Which data structures were used in this module?

Technical Details of Maps

Inspired from (Wenderlich, n.d.), Following component/libraries were used to develop the maps module in the application.

- Google Maps. It is Used to display maps, mark location etc. For using Google Maps, inspiration was taken from (Jakob Iversen, 2013, p. 146)
- Core Location. It is used to determine user's phone's geographic location, altitude, and orientation

No Major data structures were used in this module.

Technical Details of Music Player

Following component/libraries were used to develop the music player module in the application.

- AVFoundation. It was used to play music files from the audio files present in the application. Note that you can use any audio /video feature using AVFoundation whether it is video playing, music playing, camera app etc.
- UITableViewDataSource
- UITableViewDelegate
- UISlider

Technical Details of Recipe Collection

Following component/libraries were used to develop the recipes module in the application.

- UITableViewDelegate
- UIDataTableViewDataSource
- UIImageView
- UITextView
- UILabel

The main data structure used was Arrays.

Technical Details of General Knowledge Section

Following component/libraries were used to develop the general knowledge module in the application.

- UICollectionView
- UIDataTableViewDataSource
- UIImageView
- UITextView
- UILabel

The main data structure was dictionary, where country name was stored as key while the questions answers were added its value. To understand how dictionary works, help was taken from (Feiler, 2017, p. 116).

Navigation controllers were used to navigate from one screen to another.

Technical Details of Shopping List Section

Following component/libraries were used to develop the shopping list module in the application.

- UITableViewDelegate
- UITableViewDataSource
- Core Data (For saving and retrieving data)

Data Model included entities, attributes and its type like String, Integer etc.

4.7.1 Details of plist.info

Configuration of a mobile application hold a key place in the overall application development. It is configuration where you mention everything related to configuration including following:

- Permissions in your application (e.g., location, Bluetooth etc.)
- How will your application run?
- App name
- App version
- Build number
- And many more

It is actually a data file which stores information in the form of key value pair. Inspiration was taken from (raywenderlich Team, 2016, p. 105), to learn about how plistinfo work. Find below info.plist file for the Multitasker application.

Key	Type	Value
Information Property List	Dictionary	(5 items)
Privacy - Location When In Use Usage Description	String	Allow Location
Privacy - Location Always and When In Use Usage Description	String	We need your location
LSApplicationQueriesSchemes	Array	(2 items)
Item 0	String	googlechromes
Item 1	String	comgooglemaps
Required background modes	Array	(2 items)
Item 0	String	App plays audio or streams audio/video using AirPlay
Item 1	String	App registers for location updates
Application Scene Manifest	Dictionary	(2 items)
Enable Multiple Windows	Boolean	NO
Scene Configuration	Dictionary	(1 item)
Application Session Role	Array	(1 item)
Item 0 (Default Configuration)	Dictionary	(3 items)

Figure 15 - Plist.Info

As you can see from the screenshot, this file is just a list of properties, and its type is dictionary. There are many important configurations added in the above file. Some of these include following:

- Location access permissions. This will be used in the Maps section of this application
- Features which require background modes. In current application, location and Music players are part of the background operation.
- Configuration of LSApplicationQueriesSchemes which was a security feature introduced in iOS 9. Any application which is built with SDK 9 or above must provide a LSApplicationQueriesSchemes entry in the plist file, declaring which schemes it will try to query.

4.7.2 Source Code Management

Source code of iOS application development can be managed through GIT (Git Hub, n.d.) based free repository hosting. Desktop version of GitHub (Git Hub, n.d.) can be used for CLI based management and GUI based management. Another popular option is (Bit Bucket, n.d.).

5. How Did I Develop Multitasker iOS App

5.1 Introduction to Multitasker application

This is a simple but useful iOS native application developed using Swift version. It is compatible with iOS versions and iPhone versions. The app provides most useful features all in one central place. The major business modules of the application include:

- Music Player
- Recipe Collection
- Map
- General Knowledge Section
- Shopping List

Find below the core functionalities of this application.

Music Player

This is a basic music player which plays the audio files from the phone. In this application, default and built-in music player from Apple has been used. Through the music player, you can play music while your application is running.

Recipe Collection

This is an interesting feature for food lovers. This module shows various categories of food, and you can click on any category to see the recipe of a particular food dish. Some of the categories include North Indian, Pizza, cake, noodles, etc.

Maps

This is Google's map (Google, n.d.) integrated into the system and user can see his current location on the map. User's current location will be shown in the form of location pin and location details like city, country etc. Phone must have GPS installed and enabled to use this feature.

General Knowledge Section

This is section to display the general knowledge information about various countries of the world. When you enter this section, you will see list of many countries. When you click on a particular country, the application will show general knowledge in the form of questions and answers specific to that country.

Shopping List

This is section to display as a quick notes list on shopping list items when going out. User can add new items as well. That way, you can just open the list in the app, and it will ensure you did not miss anything.

Application Architecture of App Architecture Diagram

Find below a high-level architecture diagram of the Multitasker application.

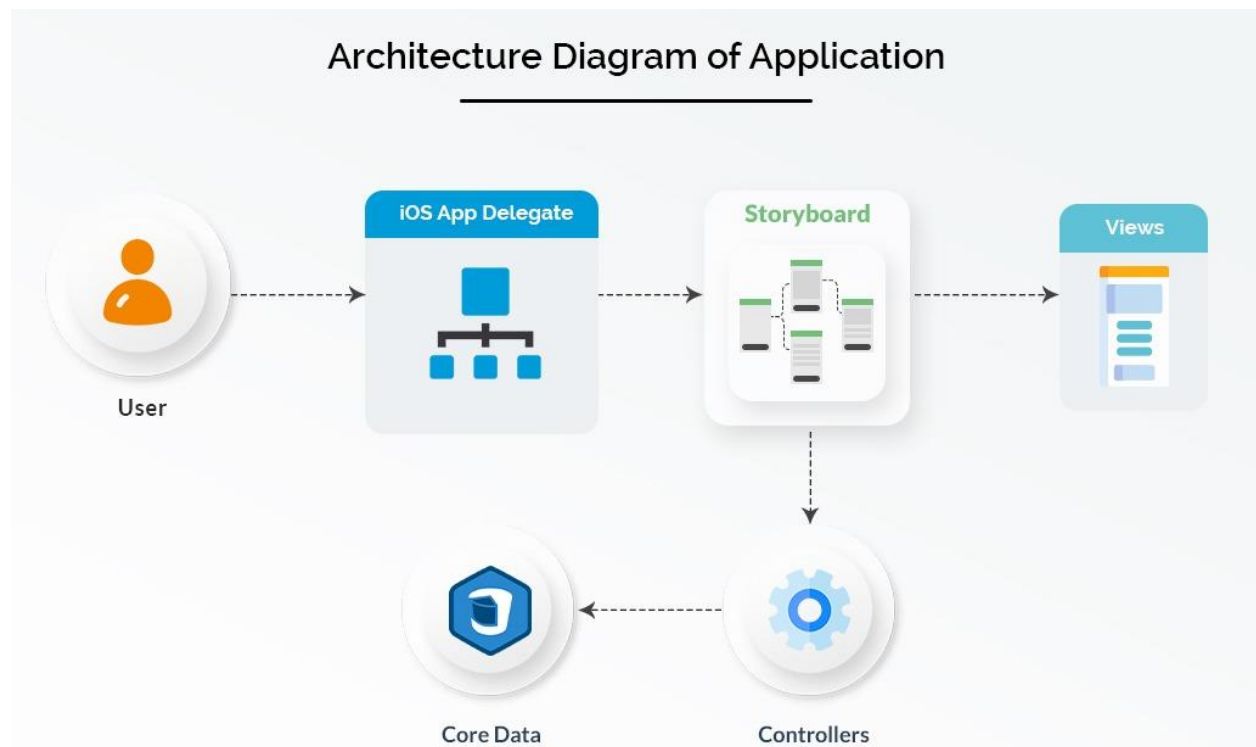


Figure 16 - Application Architecture Diagram

Find below details of the architecture diagram flow and sequence.

- 1- User starts interacting with the application. App delegates manages the call. App delegate is the main entry point of the application and will work every time application will start.
- 2- Next, the story board comes into action. The story board has laid out the visual flow and sequence of navigation. So, story board will redirect the control to the relevant controller.
- 3- In the next step, controller does its job. Based on the need, it will retrieve data from the Core Data module.
- 4- Core Data is a framework for managing object graph. It is not a database itself but can use SQLite as the database. Core Data will contain all the required entities/class which will be needed.
- 5- After getting the data from Core Data, controller will then delegate the request to story board again.
- 6- Story board will then redirect to the relevant view.
- 7- View will be rendered along with the data retrieved from the Core Data and passed along by controller.

5.2 Application Screenshots

Find below application screenshots from iPhone 11

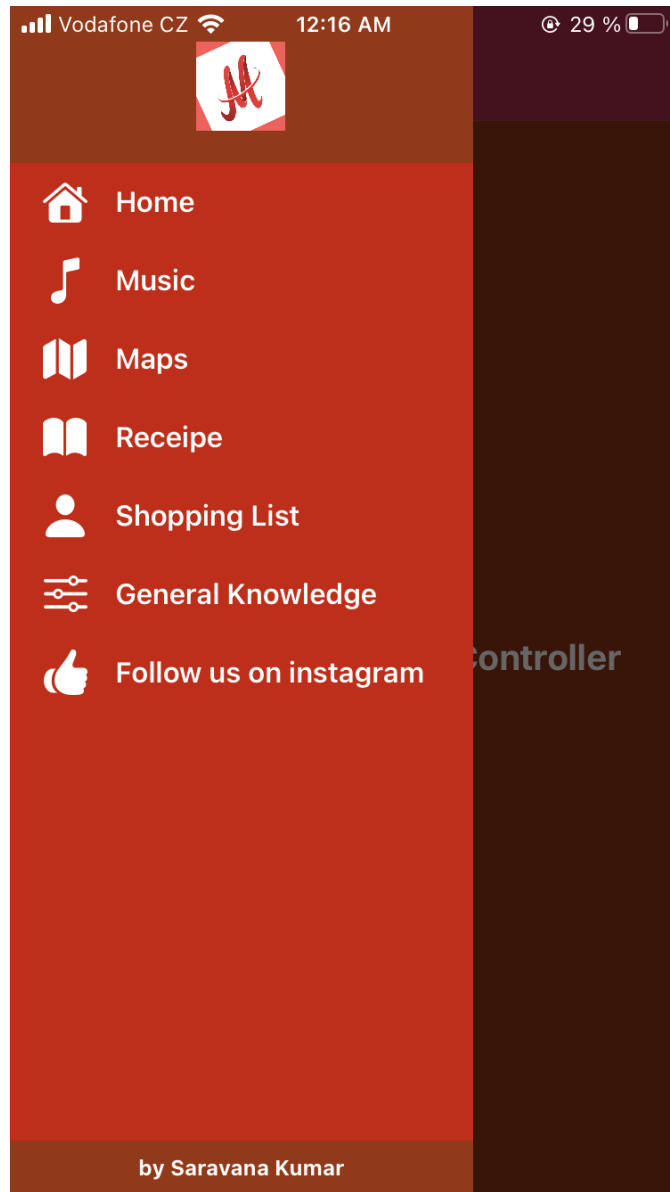


Figure 17 - Application Side Menu

This is the main menu of the application. It shows all the sections of the application accessible from this menu. Whatever screen the user is on, this menu will be available to user so that used can access any option directly from any screen of the application.

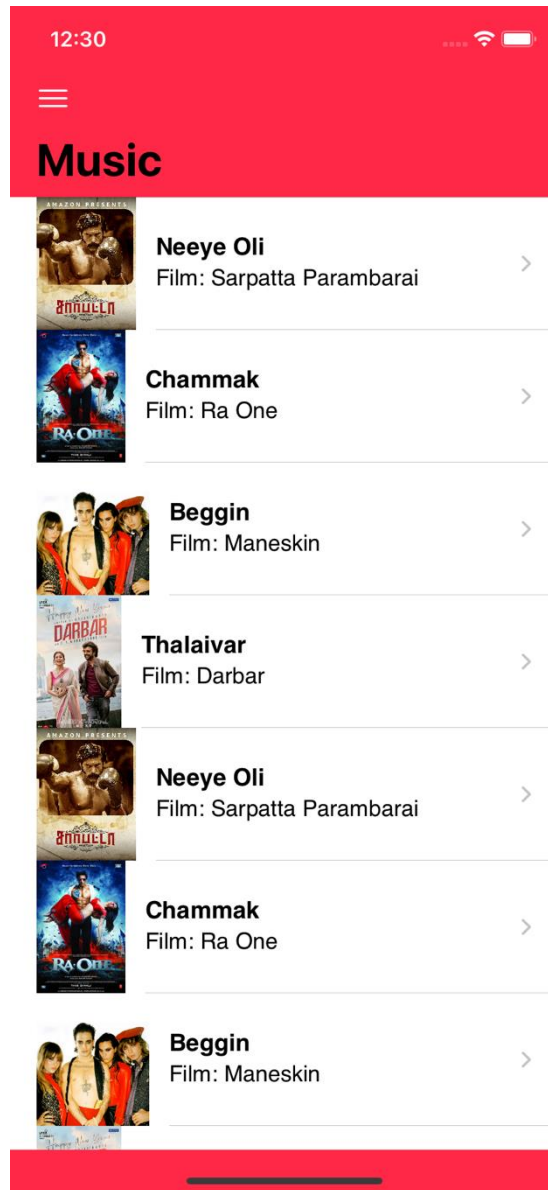
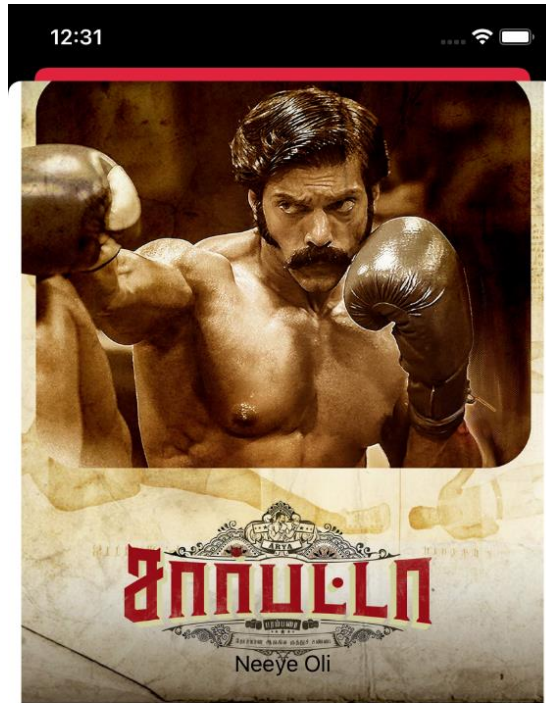


Figure 18 - Music Player Home Page

This is the home screen of the music section. As you can see, it contains the list of the music items to be played. Each item has its following information:

- Name of the music track
- Name of the film which contains this track
- Icon of the music track



Film: Sarpatta Parambarai

Music by: Santhosh

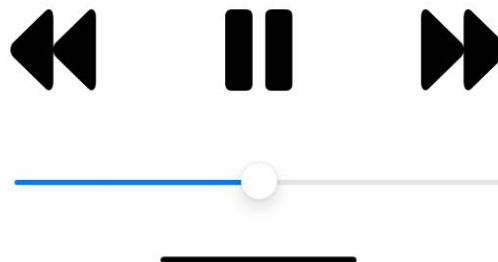


Figure 19 - Music Track Playing

This is the default media player from Apple, embedded into the application. User can play any of the songs which are present in the library. Currently following options are available for music player:

- Play song
- Pause song
- Go to next song in the collection
- Go to previous song in the collection

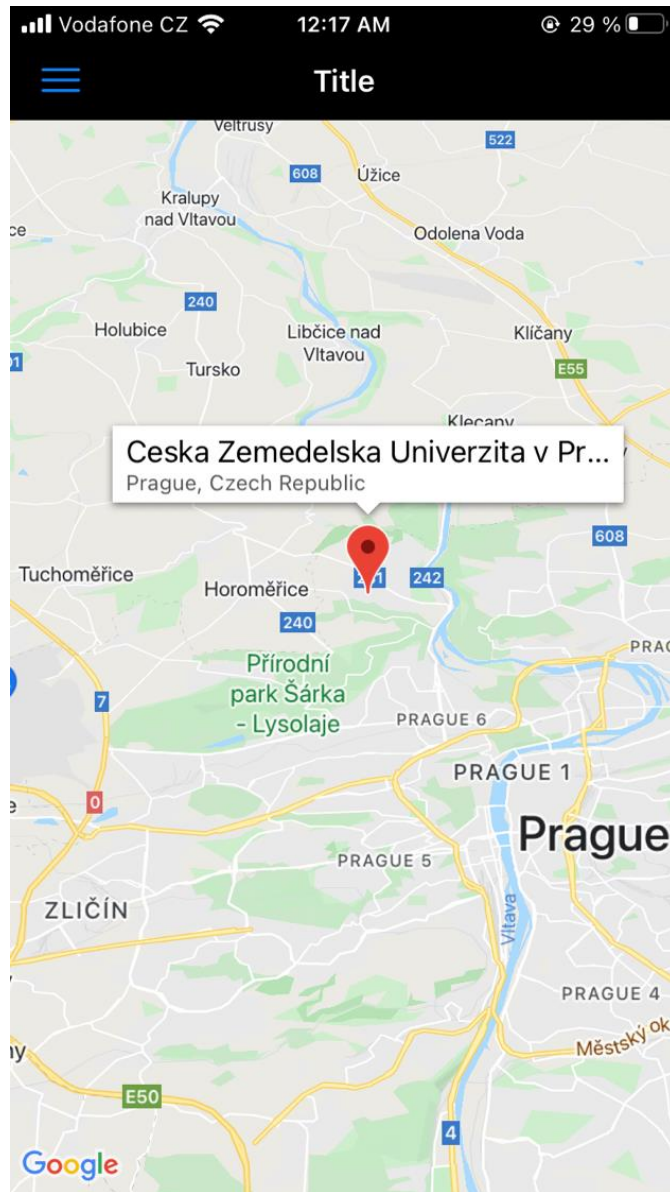


Figure 20 - Map showing current location

This is the maps section of the application which makes use of Google maps. As you can see above, it shows not only the map, but it also displays current location pin of the user's phone and the pin displays public name of the current location including city, country etc.

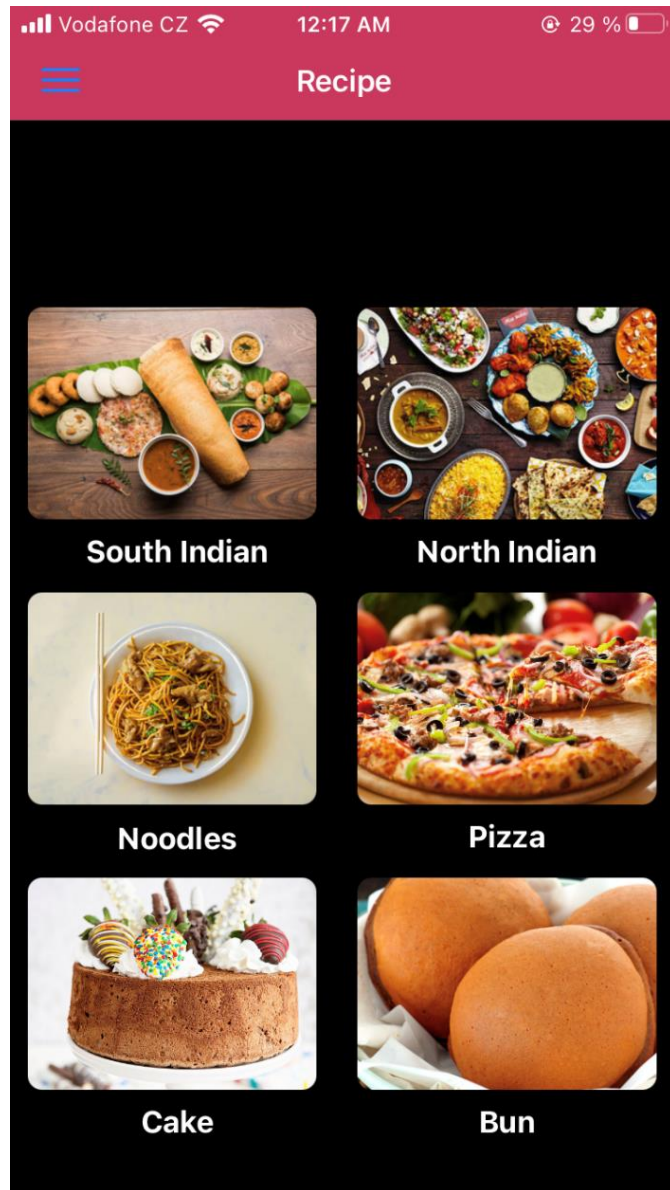


Figure 21 - Recipe Home Page

This is the recipes section. As you can see, the recipes are categorized into various categories. When user clicks on a particular category then the recipes related to that food category section will display.

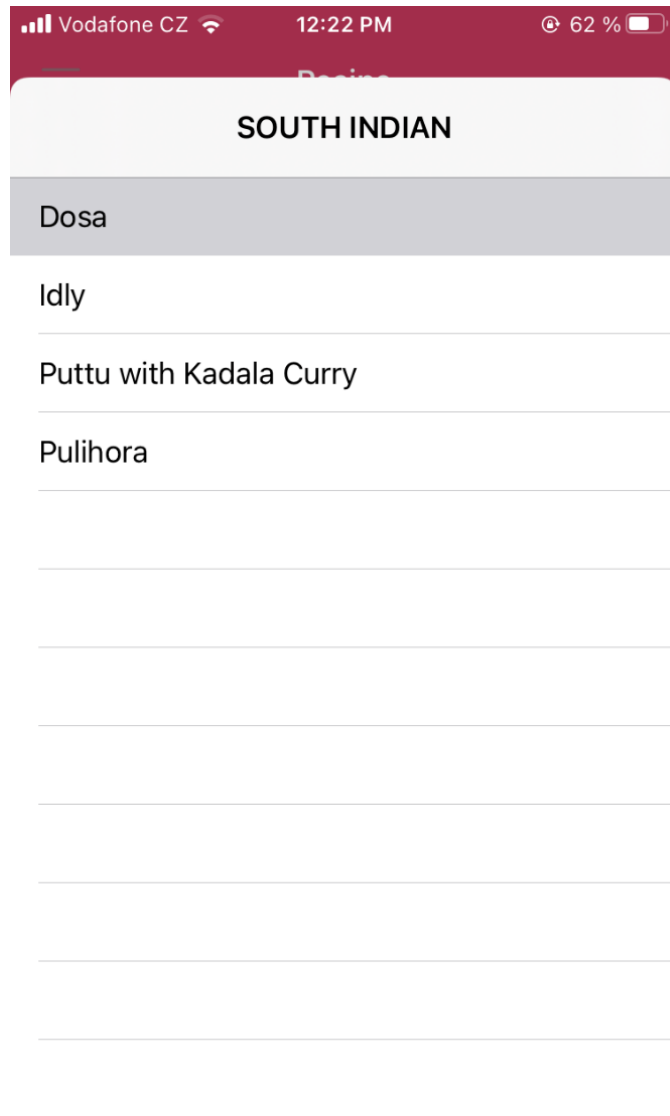
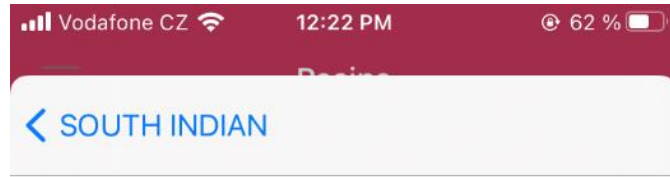


Figure 22 - Recipes List

When user clicks on a particular recipes category, then all the recipes belonging to that category will show up. Above example shows the recipes list screen if user clicks on “South Indian” recipe category.



Dosa



Dosa also called as dosai (in Tamil language) is a famous and popular South Indian breakfast or snack in India as well in the rest of the world. Dosa are basically crispy or soft crepes made with ground and fermented lentil and rice batter. To make the batter, first the lentils and rice are soaked in water for 4 to 5 hours. They are then ground separately to a fine consistency. Then both the lentil batter and

Figure 23 - Recipe Details

Above screen shows the food item belonging to a particular recipe category on which user clicked. An image along with the description of the food item will be displayed.

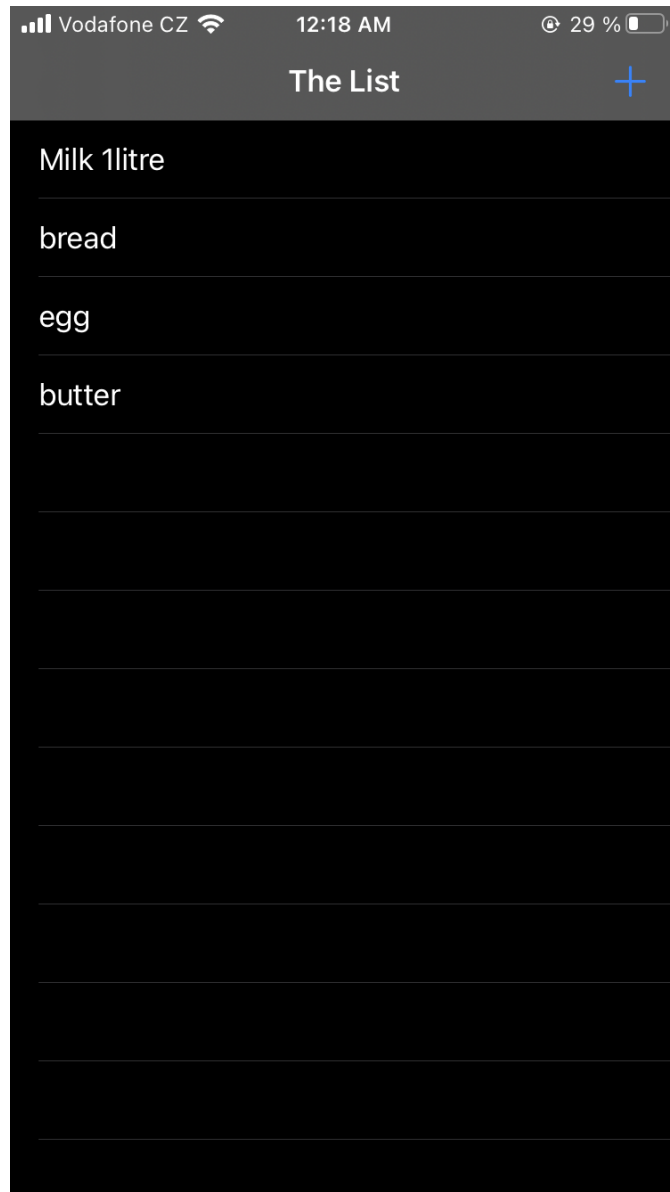


Figure 24 - Shopping List Screen

This is the screen of Shopping list. User can add items to shop here. This will serve as a quick reminder to make sure you do not miss any item when going out for shopping.



Figure 25 - General Knowledge Home Screen

This is the home page of the general knowledge section. As you can see, this screen shows list of various countries along with their flags. When user will click on a particular country name/image, the application will show the questions and answers related to that particular country.

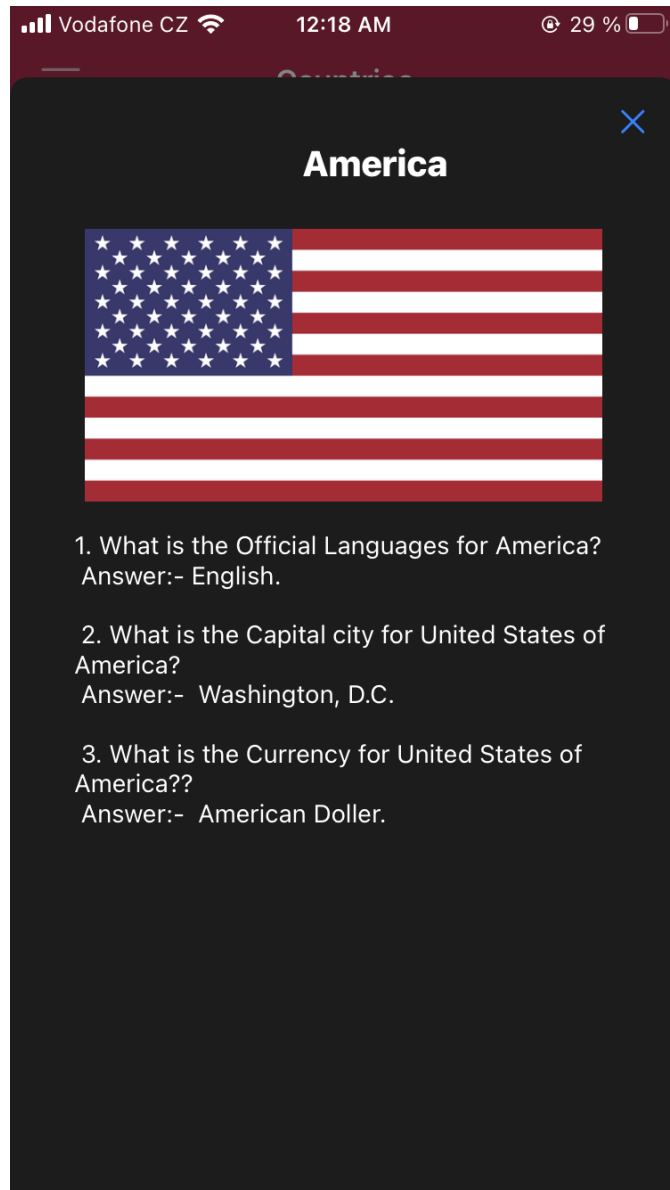


Figure 26 - General Knowledge Q&A

The application screen above was displayed when user clicked on the country “America”. The country’s flag shows up on the top followed by the questions and answers related to general knowledge about America.

5.3 Storyboard & Technical Components

In iOS, storyboarding is defining the user journey through series of UI screens. A storyboard is a visual representation of the user interface of an iOS application, which not only displays content on screens, but also the connection between them. As mentioned in (Ramnath, 2014, p. 193), A storyboard consists of a sequence of scenes, each representing a view controller and its views; scenes are connected by segue objects, which represent transition between two view controllers.

XCode provides a built-in visual editor for creating story boards. Through this editor, developers you can lay out and design the screens of application by adding various UI components like text boxes, buttons, table views, and text views onto scenes. Not only that, through storyboard, you link a view to its controller, and to manage the transfer of data between different view controllers. Using storyboarding is the preferred way to design the user interface of your application because it enables you to visualize the appearance and overall flow of your user interface on one place.

Find below the storyboarding steps of Multitasker application.

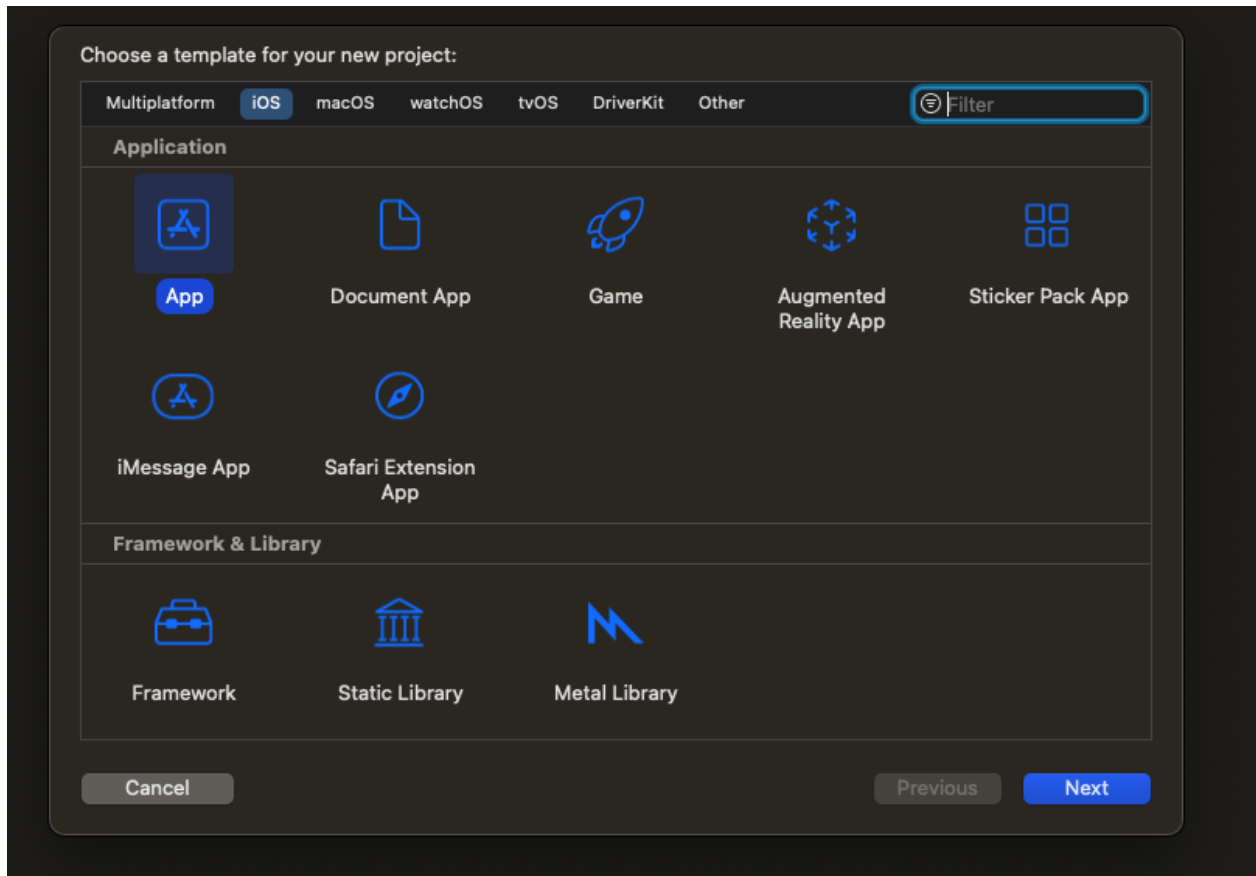


Figure 27 - XCode Choose Template

As you see, this is the very first initial screen in XCode when you start building an application. It is like the first step in a wizard to create an application. It lets you choose if you want to develop an app, game, an AR app, an iMessage app or some other kind of iOS application.

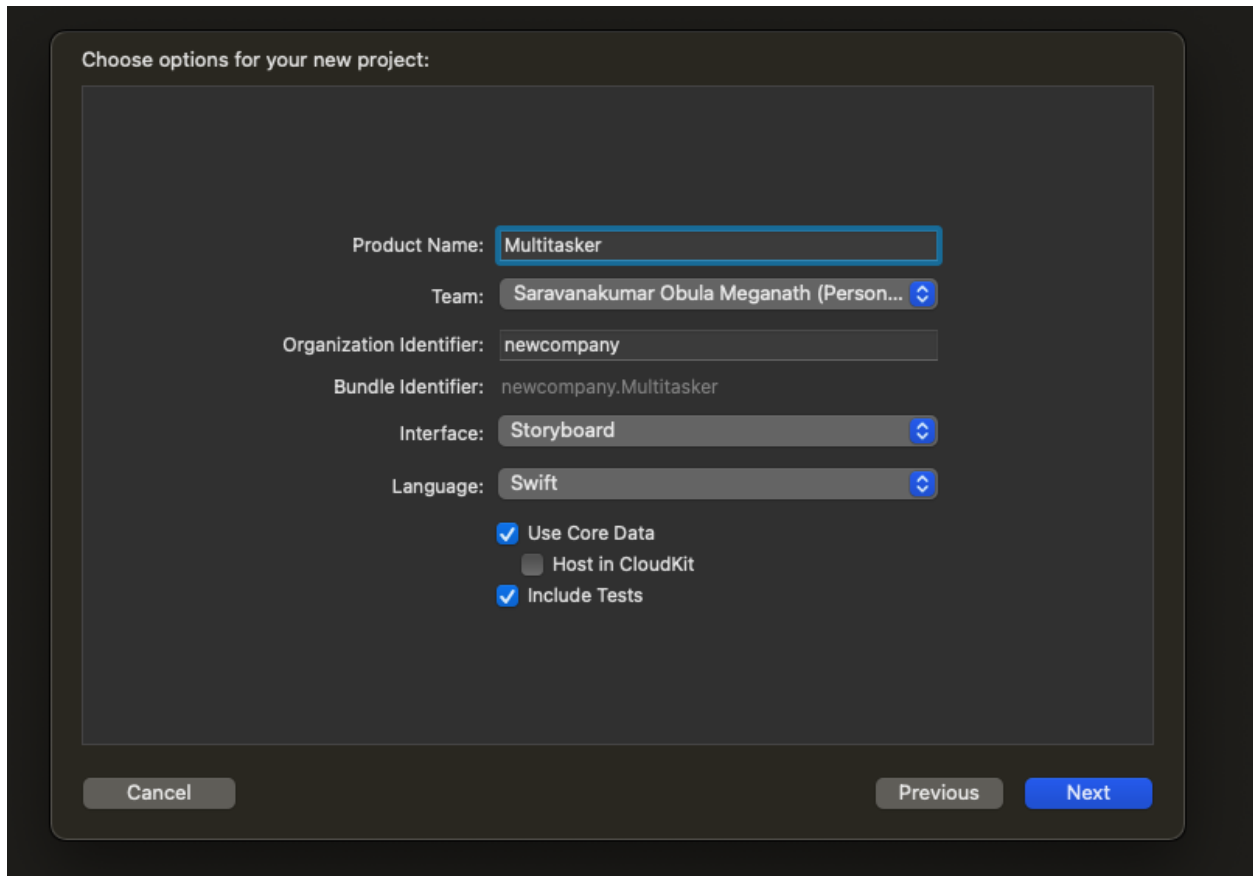


Figure 28 - XCode Project Details

This screens in the wizard are about the details of the application project. You have to fill in following details:

- Name of the product
- Team name
- Identifier of your organization
- Bundle identifier
- Interface (we selected storyboard, but you can custom code UI or utilize SwiftUI as well)
- Programming language (we are using swift, but objective-C has been another option for older projects)
- Enable option of "Use Core Data"
- Enable option of tests in the code

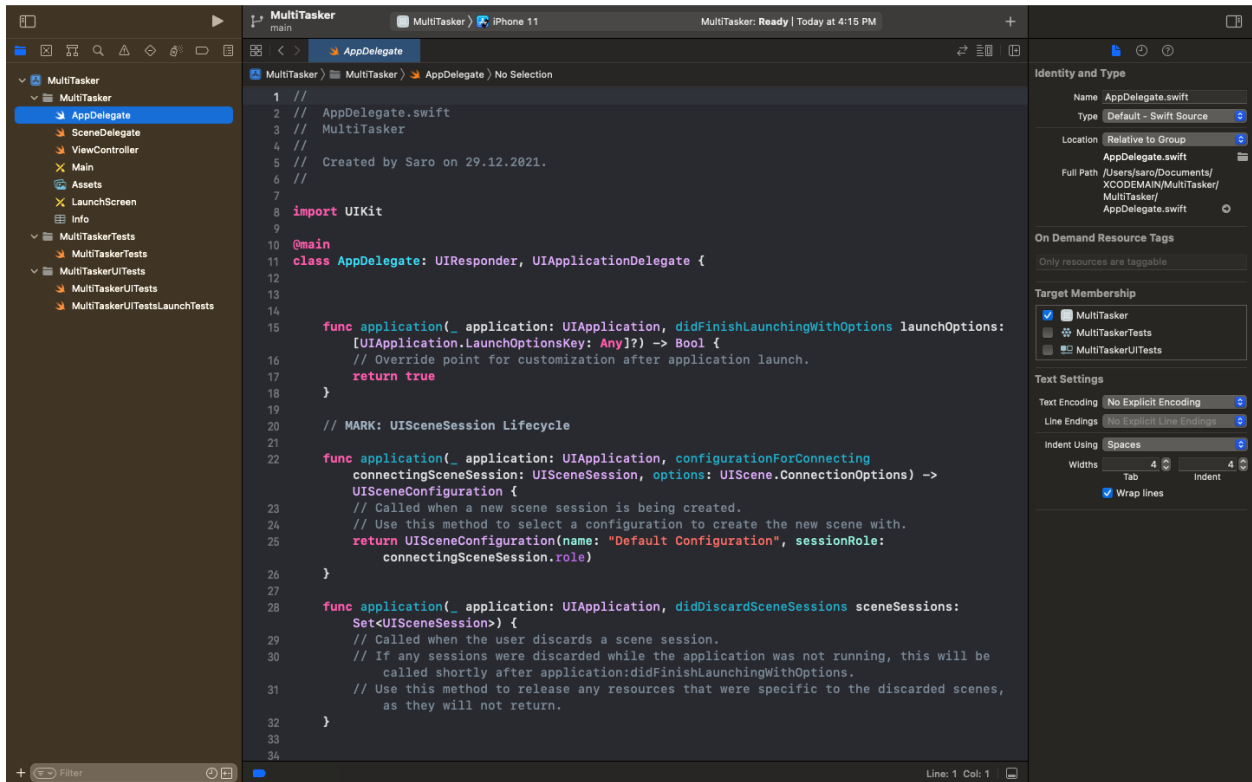


Figure 29 - App Delegate

Above screen shows the AppDelegate. From iOS 13, the responsibilities of AppDelegate have been changed. Now, the AppDelegate only takes care of the application lifecycle and setup. However, AppDelegate is still the main entry point for the application.

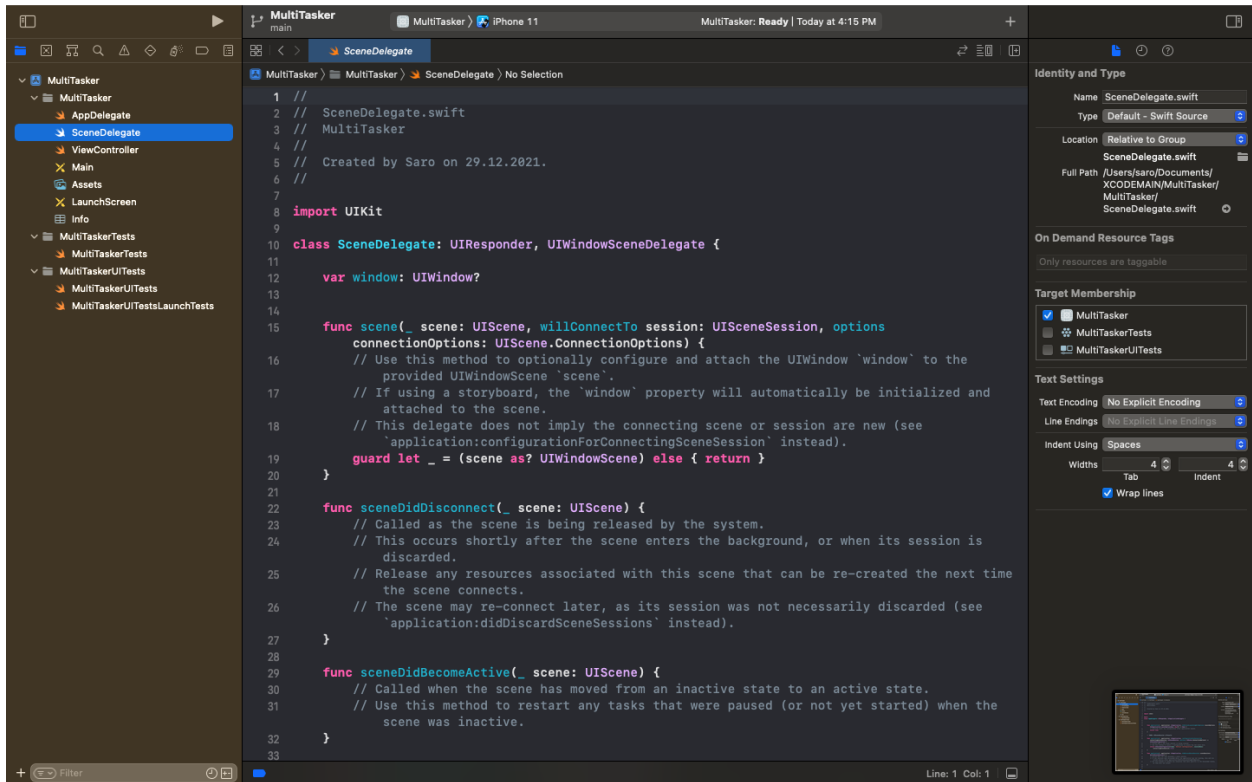


Figure 30 - Scene Delegate

This is the screen showing SceneDelegate. As mentioned previously, the role of AppDelegate was split between AppDelegate and SceneDelegate since iOS 13. The SceneDelegate now takes care of for what is displayed on the handle and manage the way content is displayed on your app.

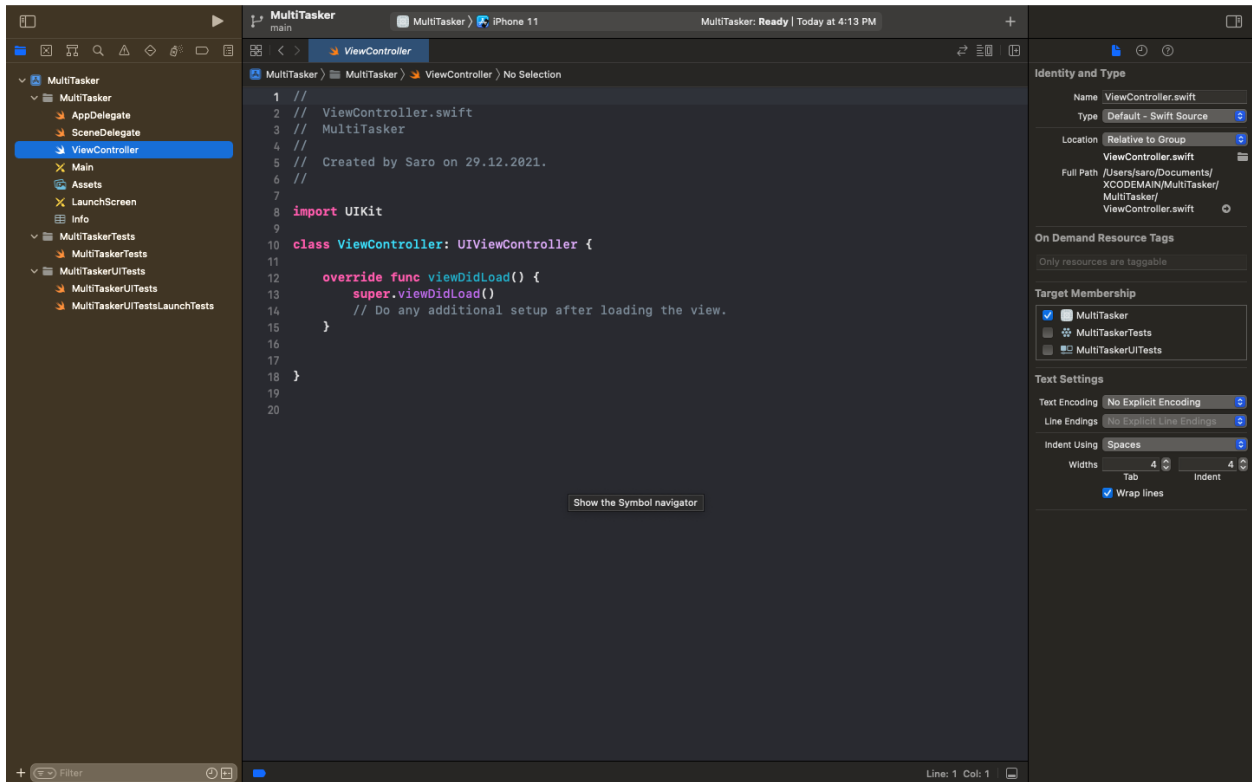


Figure 31 - View Controller

This screen shows the basic behavior of ViewController. The View Controller is parent of all the views present on the storyboard, assuming the application UI is developed through Story board. There will be at least one ViewController in each application. The job of ViewController is to manage the transition between various portions of the user interface.

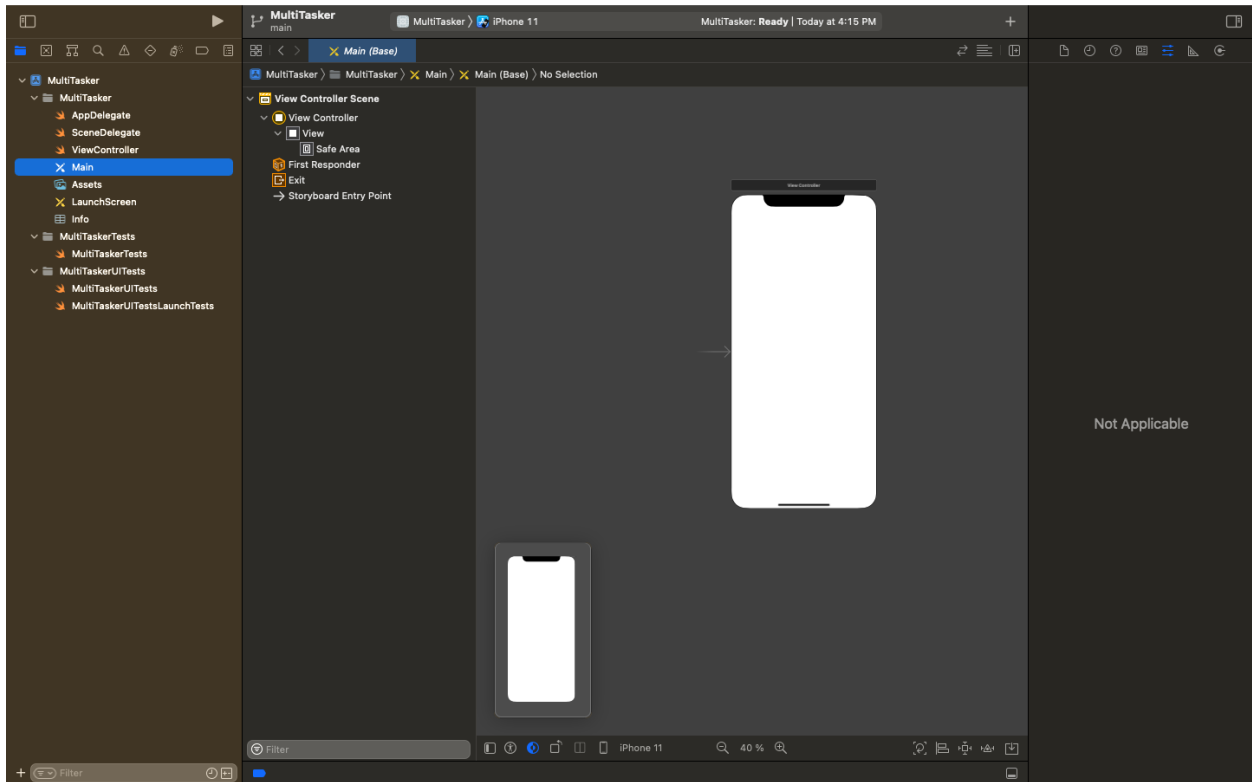


Figure 32 - Initial Story Board

This is the initial storyboard designer. Storyboard is the visual layout of all the screens in the app and how the user journey is carried out on these screens. Please note that, during the initial setup of this application in XCode, the option of “Storyboard” was selected from the available options for the interface type. Many new applications are being developed using SwiftUI, however many beginners still prefer to use storyboard because it is simpler, easier, and faster to implement screens/views through storyboard as compared to SwiftUI.

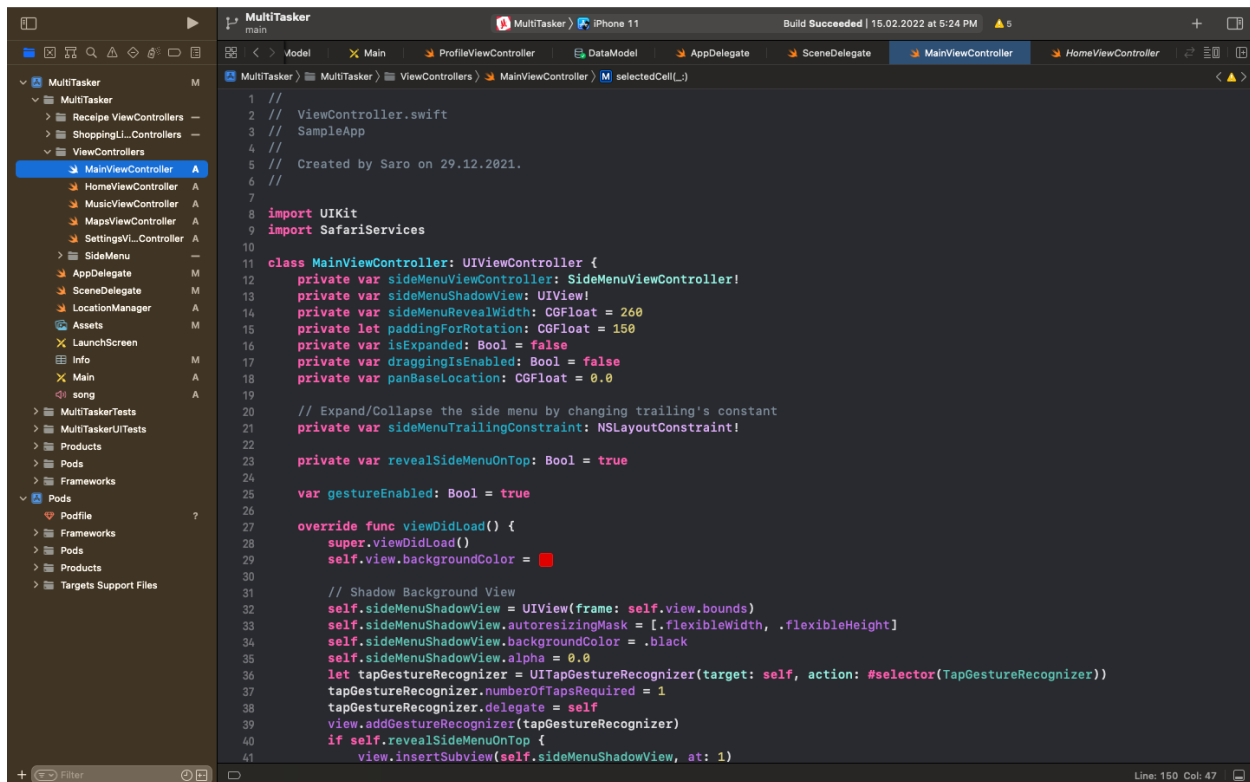


Figure 33 - Main View Controller

This screen shows MainViewController which is root of all view controllers. It is derived from UIViewController. The UIViewController defines the behavior for managing your views, handling events, transitioning from one view controller to another etc.

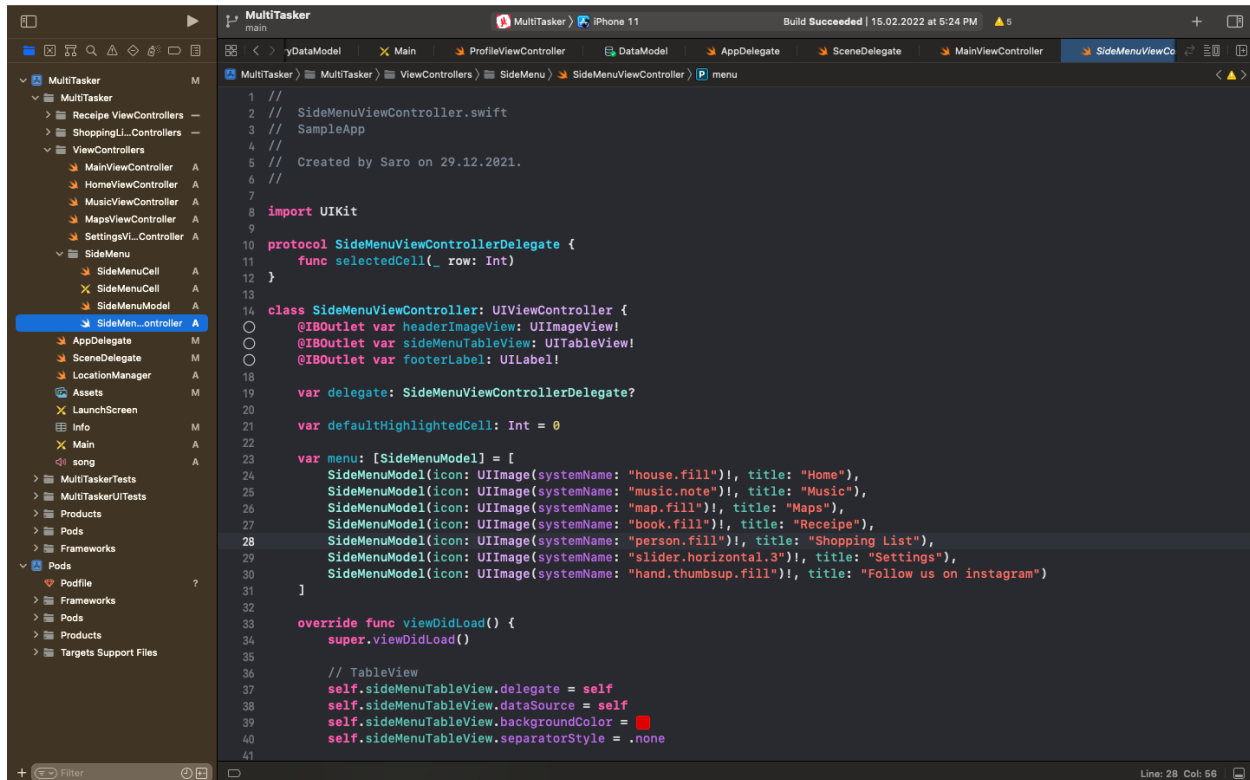


Figure 34 - Side Menu View Controller

This is the ViewController related to the side menu of the application. It shows all the menu items in the form of a list. It also defines how the transitioning from this view to other views will be managed. It is perfectly fine to have multiple views associated with a single ViewController. And it is also possible to have only one view associated with a ViewController.

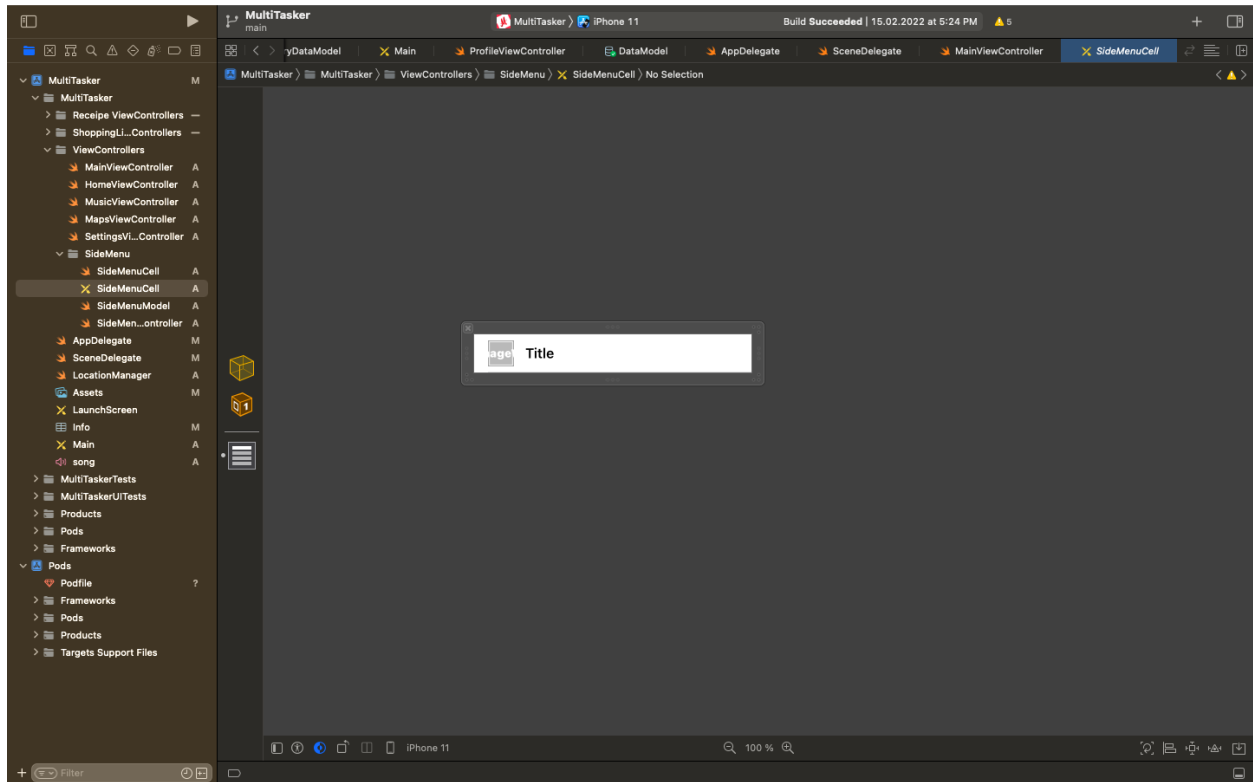


Figure 35 - Building Menu Item

This screen shows how to build each individual menu item cell in the side menu view. You can not only design the look and feel of how it will look but you can also define its behavior as well. Note that that after creating the cell, you will have to register it with the view of the side menu as well.

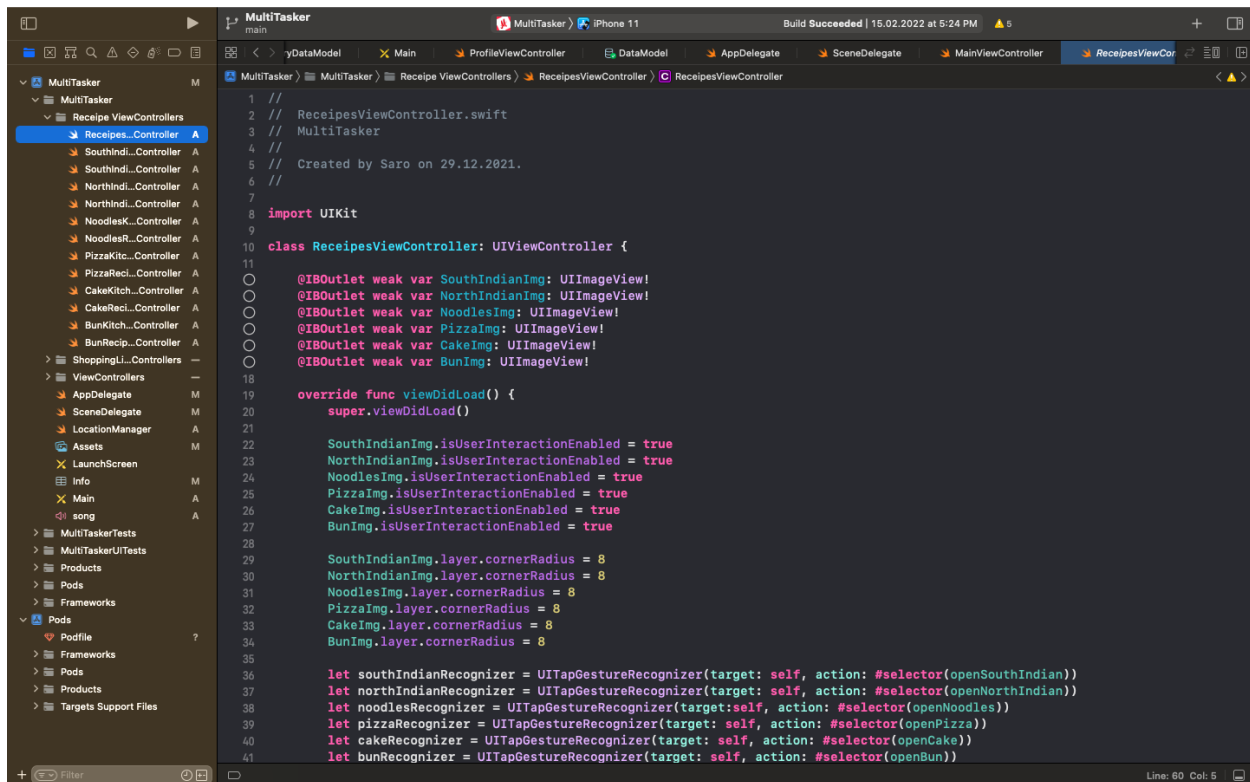


Figure 36 - Recipes View Controller

This is the ViewController for recipes section. You can see that all the sections of recipes are being developed in this class. Again, you can not only define the UI of the recipes section but also the behavior, transitioning to other views as well.

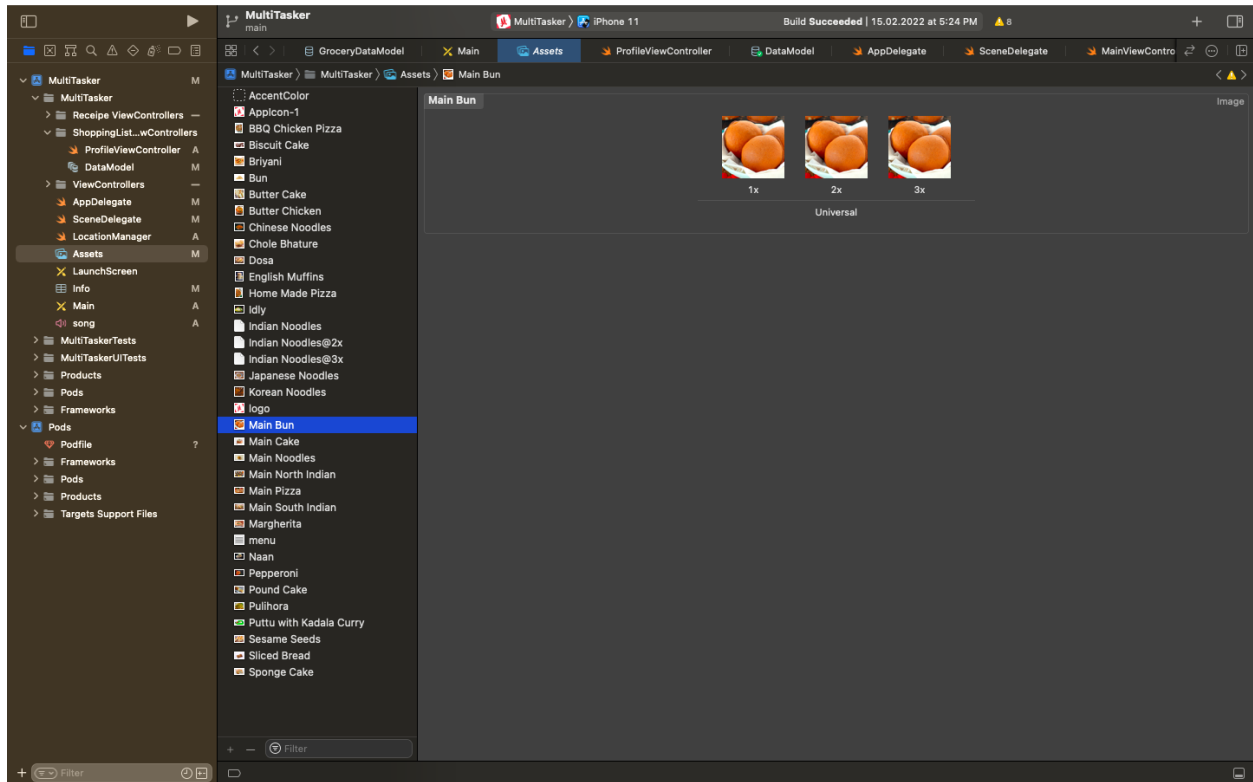


Figure 37 - Assets

This screen shows one of the image assets in the XCode. Assets can be images, videos etc. Let's touch upon the image versions 1x, 2x, 3x briefly. As we know, Apple supports multiple devices with different screen resolutions and screen sizes, and apps developed by developers should be compatible with as many of these devices as possible. 1x, 2x, and 3x images allow developers to optimize rendering of application UI based on the user's device, regardless of the phone on which the app is running. In a nutshell, 1x, 2x, and 3x images are the same image, but at different sizes.

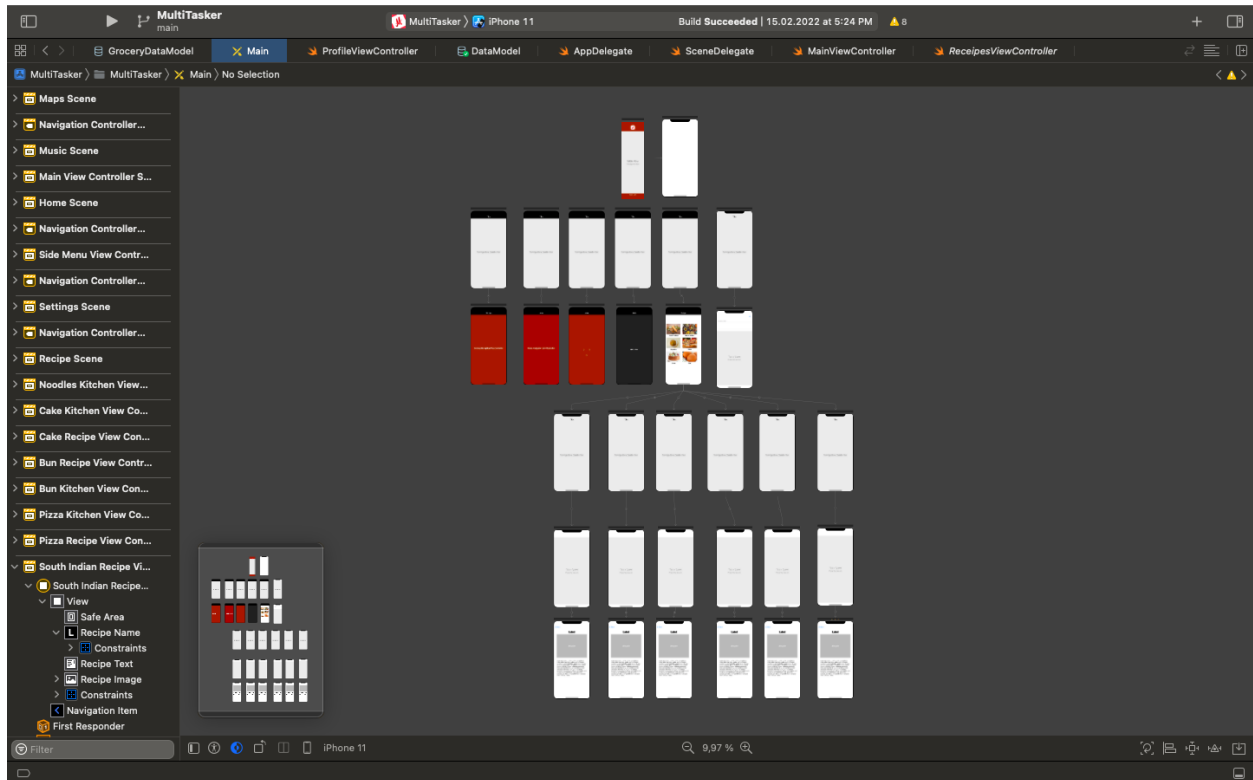


Figure 38 - Story Board

This is the overall storyboard of the application in XCode. As mentioned previously, storyboard visually lays out the overall user journey across all the views in the application. It will contain all the views in the application, and it will also visually assemble the navigation path through all those screens. For creating story boards, inspiration was taken from (Gary Bennett, 2019, p. 107)

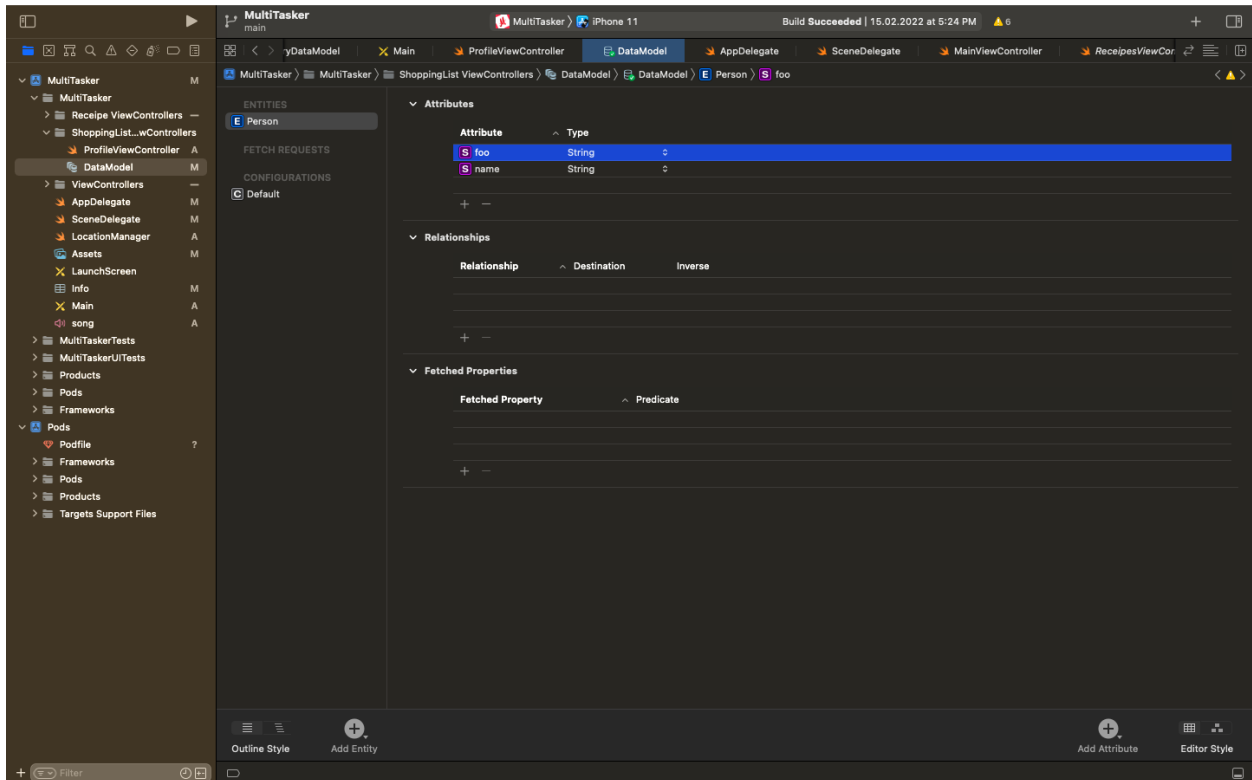


Figure 39 - Data Model

This screenshot shows initial stages of the Core Data. Core Data is actually a framework provided by Apple for iOS apps. It is used to manage the model layer object in iOS application. Through Core Data you can track, save, modify, and filter the data inside your iOS application, however, Core Data should not be considered or used as a database.

5.4 UI Components used in App

How the user interface of an application is developed, is an important factor in any iOS application. Multitasker application though simple in nature, has used some core and important UI components on its views. Find below details of such UI components:

Following are some of the basic UI components which were used in overall application:

- Navigation bar
- Buttons
- Labels
- Alerts
- Icons

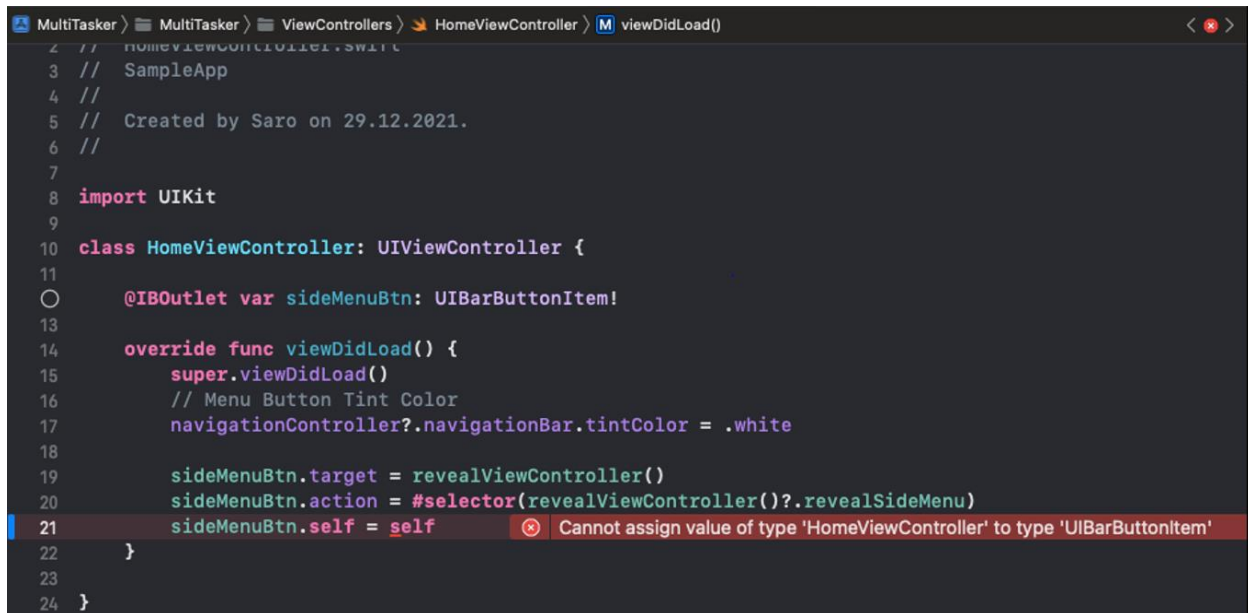
Image view (*UIImageView*) was used in many modules of this application. Some of the modules where this component was used, include General knowledge, Recipe section etc. Scroll view (*UIScrollView*) was used in the modules of recipe, shopping list and countries.

Table view (*UITableView*) was used in the modules of recipe, shopping list and countries. To understand the working of Table Views, help was taken from (Maurice Sharp, 2013, p. 275)

UIPanGesture was used in the module of side menu.

5.5 Errors faced during development

Following are some of the compilation errors faced during the development of this application.



```
MultiTasker > MultiTasker > ViewControllers > HomeController > viewDidLoad()
2 // HomeController.swift
3 // SampleApp
4 //
5 // Created by Saro on 29.12.2021.
6 //
7
8 import UIKit
9
10 class HomeController: UIViewController {
11
12     @IBOutlet var sideMenuBtn: UIBarButtonItem!
13
14     override func viewDidLoad() {
15         super.viewDidLoad()
16         // Menu Button Tint Color
17         navigationController?.navigationBar.tintColor = .white
18
19         sideMenuBtn.target = revealViewController()
20         sideMenuBtn.action = #selector(revealViewController()?.revealSideMenu)
21         sideMenuBtn.self = self
22     }
23
24 }
```

Figure 40 - Compile Time Error 1

Compile time error “Cannot assign value of type ‘HomeController’ to type ‘UIBarButtonItem’

```
MultiTasker
main
MultiTasker } iPhone
Build Failed | Today at 11:58 PM 2 9
GoogleMap...wController MusicViewController SideMenuViewController MapsViewController
MultiTasker > MultiTasker > ViewControllers > MusicViewController > No Selection
48     super.viewWillAppear(animated)
49     self.revealViewController()?.gestureEnabled = true
50 }
51
52
53 @IBAction func play (_ sender: Any){
54     audioPlayer.play()
55 }
56
57 @IBAction func pause (_ sender: Any){
58     audioPlayer.pause()
59 }
60
61 @IBAction func replay (_ sender: Any){
62     audioPlayer.currentTime = 0
63 }
64 }
65 }
66
```

2 Extraneous \'}' at top level

Figure 41 - Compile Time Error 2

Compile time error “Extraneous ‘}’ at top level’ This is self-explanatory. There is an extra parenthesis in this code block.

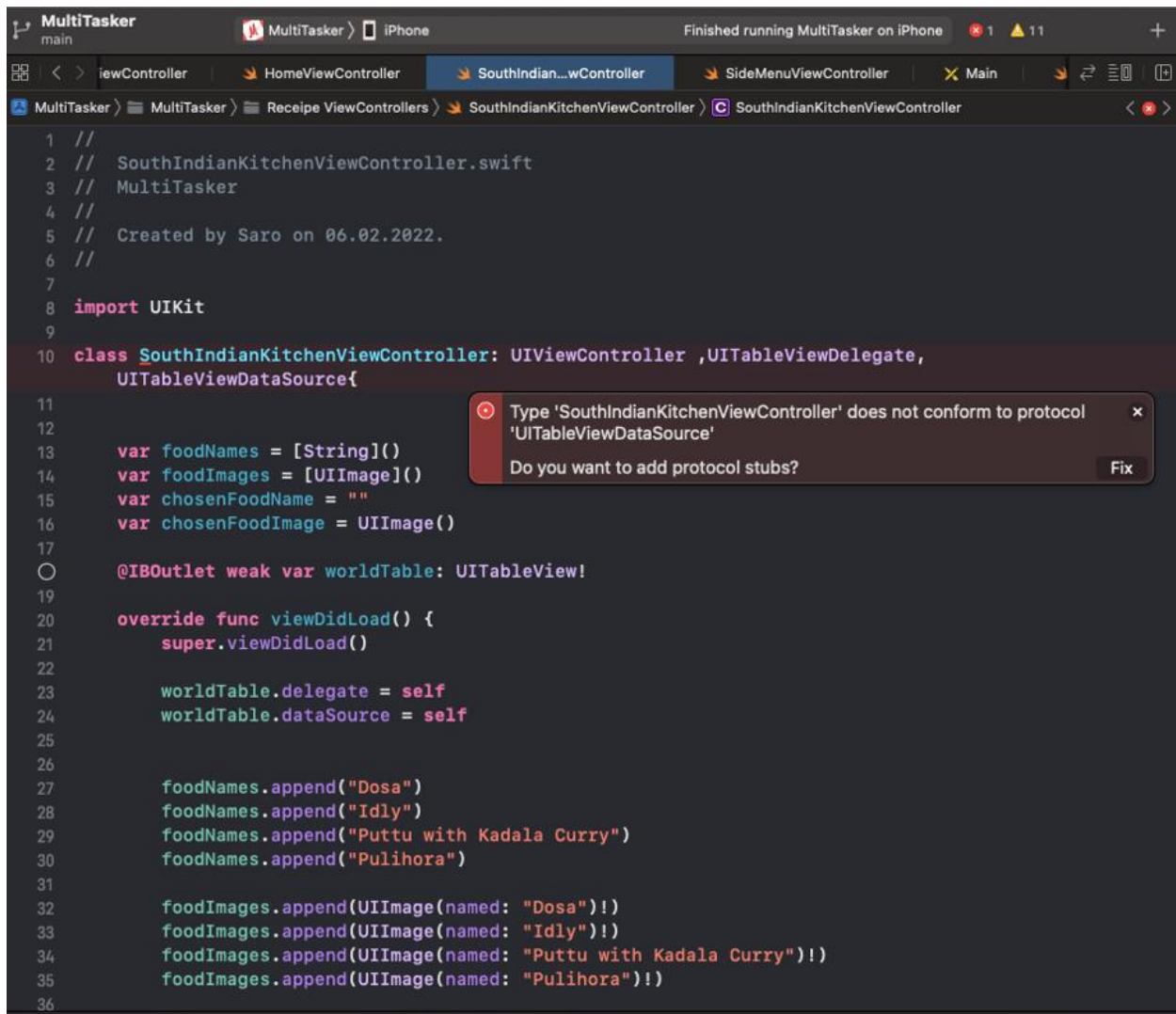


Figure 42 - Compile Time Error 3

Compile time error “Type ‘SountIndianKitchenViewController’ does not conform to protocol ‘UITableViewDataSource’, Do you want to add protocol stubs?”

```
6 //
7
8 import UIKit
9
10 class SouthIndianKitchenViewController: UIViewController, UITableViewDelegate{
11
12
13     var foodNames = [String]()
14     var foodImages = [UIImage]()
15     var chosenFoodName = ""
16     var chosenFoodImage = UIImage()
17
18     @IBOutlet weak var worldTable: UITableView!
19
20     override func viewDidLoad() {
21         super.viewDidLoad()
22
23         worldTable.delegate = self
24         worldTable.dataSource = self
25
26         foodNames.append("Dosa")
27         foodNames.append("Idly")
28         foodNames.append("Puttu with Kadala Curry")
29         foodNames.append("Pulihora")
30
31
32         foodImages.append(UIImage(named: "Dosa")!)
33         foodImages.append(UIImage(named: "Idly")!)
34         foodImages.append(UIImage(named: "Puttu with Kadala Curry")!)
35         foodImages.append(UIImage(named: "Pulihora")!)
36
37         navigationItem.title = "SOUTH INDIAN"
38
39     }
40 }
```

Cannot assign value of type 'SouthIndianKitchenViewController' to type 'UITableViewDataSource?'

Figure 43 - Compile Time Error 4

Compile time error “Type ‘SountIndianKitchenViewController’ does not conform to type ‘UITableViewDataSource?’ As discussed on (stackoverflow, n.d.), this happens when one of the functions is missing in the implementation.

During the application development, there are some errors which are received when application code is executed on the simulator or on the actual physical device. Find below some of the errors which were faced during the application execution on actual device.

```
MultiTasker
main
MultiTasker > iPhone 11
Running MultiTasker on iPhone 11 14
SouthIndian...wController
NorthIndian...wController
SideMenuViewController
Main
AppDelegate
MultiTasker > MultiTasker > AppDelegate > AppDelegate
8 import UIKit
9 import CoreData
10 import SwiftUI
11 import GoogleMaps
12 import GooglePlaces
13
14 let googleApiKey = "AIzaSyANuAR-3bcvxx11qk7DfPrFd5z3BchD0C0"
15
16 @main = Thread 1: "Google Maps SDK for iOS must be initialized via [GSMSServices provideAPIKey:...] prior to use"
17 class AppDelegate: UIResponder, UIApplicationDelegate {
18     var window: UIWindow?
19
20
21
22     func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
    [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
23
24
25
26
27     return true
28 }
29
30 func applicationWillTerminate(_ application: UIApplication) {
31     self.saveContext()
32 }
33
34 // MARK: - Core Data stack
MultiTasker > Thread 1 > 15 static AppDelegate.$main()
Line: 18 Col: 5
.simruntime/Contents/Resources/RuntimeRoot/usr/lib/libMTLCapture.dylib
DYLD_FRAMEWORK_PATH=/Users/saro/Library/Developer/Xcode/DerivedData/MultiTasker-gxcbpexuquxkgceqwsznhrk
kxagb/Build/Products/Debug-iphonesimulator
*** Terminating app due to uncaught exception 'GSMSServicesException', reason: 'Google Maps SDK for iOS
must be initialized via [GSMSServices provideAPIKey:...] prior to use'
terminating with uncaught exception of type NSError
CoreSimulator 776.4 - Device: iPhone 11 (72603051-1754-4F9F-8AE1-084437CA3987) - Runtime: iOS 15.0
(19A339) - DeviceType: iPhone 11
(lldb)
```

Figure 44 - Runtime Error 1

Runtime Error “Thread 1: Google Maps SDK must be initialized via [GSMSServices provideAPIKey...] prior to use”.

```
10
11 class MusicViewController: UIViewController, UITableViewDelegate, UITableViewDataSource {
12
13     @IBOutlet var table: UITableView!
14
15     var songs = [Song]()
16
17     @IBOutlet var sideMenuBtn: UIBarButtonItem!
18
19     override func viewDidLoad() {
20         super.viewDidLoad()
21         self.sideMenuBtn.target = revealViewController()
22         self.sideMenuBtn.action = #selector(self.revealViewController()?.revealSideMenu)
23
24         configureSongs()
25         table.delegate = self
26         table.dataSource = self
27
28     }
29
30     func configureSongs() {
31         songs.append(Song(name: "Background music",
32                           albumName: "123 Other",
33                           artistName: "Rnado",
34                           imageName: "cover1",
35                           trackName: "Neeeye"))
36         songs.append(Song(name: "Havana",
37                           albumName: "Havana album",
38                           artistName: "Cuba Gooding Jr.",
39                           imageName: "cover2",
40                           trackName: "Havana"))
41     }
42 }
```

MultiTasker/MusicViewController.swift:25: Fatal error: Unexpectedly found nil while implicitly unwrapping an Optional value

2022-02-28 16:25:45.956672+0100 MultiTasker[3809:104471] MultiTasker/MusicViewController.swift:25: Fatal error: Unexpectedly found nil while implicitly unwrapping an Optional value (lldb)

Figure 45 - Runtime Error 2

Runtime error “Thread 1: Fatal error Unexpectedly found nil while implicitly unwrapping an optional value...”. This is a common error while unwrapping the optional. This point has been discussed in detail in the earlier section of “Common mistakes made by swift developers”. As discussed in (Cui, 2020) , before forced unwrapping, we need to make sure that the optional contains a value.

```

1 //
2 // AppDelegate.swift
3 // MultiTasker
4 //
5 // Created by Saro on 29.12.2021.
6 //
7
8 import UIKit
9 import CoreData
10 import SwiftUI
11 import GoogleMaps
12 import GooglePlaces
13
14 let googleApiKey = "AIzaSyANuAR-3bcvxx11qk7DfPrFd5z3BchDOC0"
15
16 @main = Thread 1: "unable to dequeue a cell with identifier cell - must register a nib or a class for the identifier or connect a proto...
17 class AppDelegate: UIResponder, UIApplicationDelegate {
18
19     var window: UIWindow?
20
21
22     func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
    [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
23
24         GMSServices.provideAPIKey("AIzaSyANuAR-3bcvxx11qk7DfPrFd5z3BchDOC0")
25         GMSPplacesClient.provideAPIKey("AIzaSyANuAR-3bcvxx11qk7DfPrFd5z3BchDOC0")
26         //GMSServices.provideAPIKey(googleApiKey)
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

```

DYLD_FRAMEWORK_PATH=/Users/saro/Library/Developer/Xcode/DerivedData/MultiTasker-gxcpexuqxkgceqwsznhrk
kxagb/Build/Products/Debug-iphonesimulator
*** Terminating app due to uncaught exception 'NSInternalInconsistencyException', reason: 'unable to
dequeue a cell with identifier cell - must register a nib or a class for the identifier or connect a
prototype cell in a storyboard'
terminating with uncaught exception of type NSError
CoreSimulator 776.4 - Device: iPhone 11 (72603051-1754-4F9F-8AE1-084437CA3987) - Runtime: iOS 15.0
(19A339) - DeviceType: iPhone 11

```

Figure 46 - Runtime Error 3

Runtime error “Unable to dequeue a cell with identifier cell – must register a nib or a class for the identifier or connect a...”

5.6 Used Hardware & Software

Following hardware was used as part of the development of the Multitasker application.

- MacBook Air 2015 - 1.6 GHz Dual - Core Intel i5, 8GB Ram
- iPhone 8 rose gold 64GB
- iPhone 11 Black 128GB

Following libraries, frameworks, component were used during development of Multitasker application.

- XCode Version 13.1 (13A1030d)
- For iPhone 8 - Software Version 14.8.1
- For iPhone 11 - Software Version 15.3.1
- CocoaPods (www.cocoapods.org)
- Canva (www.canva.com)
- Microsoft Word, Microsoft Excel

6. Future Vision

The Multitasker application can be further enhanced with many features. Some of the desired features, which can be included in the future releases are following:

1. API's for music, recipes, general knowledge etc.: The current implementation has all the data residing inside the app itself. For the future version, API's should be integrated so that the data is pulled from the API's. Many free rest-based API's are available related to almost everything. Whether it is music, food recipes, general knowledge, location-based data, there is an API for everything.
2. Publishing the app to the Apple store: After making some tweaks and some new features, the application should be submitted to the store for review. Not only, it will be a learning experience in terms of Apple's required guidelines for app submission, but there would be a sense of achievement as well.
3. Device Compatibility: The Multitasker application should be enhanced to have more support for iPads and other iPhones as well. This will require change to the views and how they are rendered. Most of the UI components will remain same and the layout for views will need to be modified for device compatibility.
4. Addition of user login: Adding a user's login will result in a customized experience for consumer. User will be able to mark favorite tracks, recipes, locations etc. And we can have a feature similar to wish list where users can ask to add their favorite recipes, music tracks etc.

7. Conclusion

We have successfully completed everything described earlier in the objectives. This diploma thesis has proved the effectiveness of literature review. The topics discussed in literature review shows that it is certainly possible to develop an iOS application by following the user guides and tutorials about the iOS development. This can serve a quick guide for a beginner iOS developer so that he could understand the foundation of the iOS development and various areas of swift programming language which are pre-requisites for developing an actual native iOS application.

The thesis also described in detail, how was the Multitasker mobile application was developed using Swift programming language. All the major components, life cycle, error handling, core libraries etc. were discussed which are very helpful for any iOS developer who intends to develop a native mobile app from scratch.

If the features mentioned in the section of “Future Vision” are also included in the future versions of the application, then it can be a good addition to the Apple app store and it can catch eyes of many youngsters who want to see various different features in a single mobile application.

8. List of Figures Used in this Document

Figure 1 - History of iPhones (Source (Proulx, n.d.)).....	16
Figure 2 - Mobile OS Market Share.....	17
Figure 3 - Mobile Vendor Market Share.....	18
Figure 4 - Android vs iOS Market Share UK	19
Figure 5 - Life Cycle Management.....	21
Figure 6 - Life Cycle Methods.....	22
Figure 7 - Mobile App Usage (Mariana, 2020)	26
Figure 8 - Pros and Cons of Native Apps	29
Figure 9 - Pros and Cons of Web Apps	30
Figure 10 - Pros and Cons of Hybrid Apps.....	32
Figure 11 - Comparison of Flutter Vs React Native Vs Native	34
Figure 12 - XCode Requirements	36
Figure 13 - MVVM Diagram.....	50
Figure 14 - MVC Diagram.....	51
Figure 15 - Plist.Info.....	54
Figure 16 - Application Architecture Diagram.....	56
Figure 17 - Application Side Menu.....	58
Figure 18 - Music Player Home Page	59
Figure 19 - Music Track Playing	60
Figure 20 - Map showing current location.....	61
Figure 21 - Recipe Home Page	62
Figure 22 - Recipes List.....	63
Figure 23 - Recipe Details	64
Figure 24 - Shopping List Screen	65
Figure 25 - General Knowledge Home Screen	66
Figure 26 - General Knowledge Q&A.....	67
Figure 27 - XCode Choose Template	69
Figure 28 - XCode Project Details.....	70
Figure 29 - App Delegate.....	71
Figure 30 - Scene Delegate	72
Figure 31 - View Controller.....	73
Figure 32 - Initial Story Board.....	74
Figure 33 - Main View Controller	75
Figure 34 - Side Menu View Controller	76
Figure 35 - Building Menu Item	77

Figure 36 - Recipes View Controller	78
Figure 37 - Assets	79
Figure 38 - Story Board	80
Figure 39 - Data Model.....	81
Figure 40 - Compile Time Error 1	82
Figure 41 - Compile Time Error 2	83
Figure 42 - Compile Time Error 3	84
Figure 43 - Compile Time Error 4	85
Figure 44 - Runtime Error 1.....	86
Figure 45 - Runtime Error 2.....	87
Figure 46 - Runtime Error 3.....	88

9. Abbreviations/Acronyms Used

Following are some of the abbreviations and acronyms used in this document.

iOS – iPhone, iPad, iPod Touch Operating system

Android – Operating system made by Google

SDK – Software Development Kit

UX – User Experience

UI – User Interface

MVC – Model View Controller

MVVM – Model – View – View Model

CSS – Cascade Style Sheet

GPS – Global Positioning System

10. References

- Jones, M. (2022, 02 16). Retrieved from <https://historycooperative.org/the-history-of-the-iphone/>
- Prasad, L. (2018, July 15). Retrieved from <https://hackernoon.com/application-life-cycle-in-ios-12b6ba6af78b>
- Mariana. (2020, July). Retrieved from www.beezer.com: <https://www.beezer.com/disadvantages-of-mobile-apps/>
- Valdellon, L. (n.d.). Retrieved from <https://clevertap.com/blog/types-of-mobile-apps/>
- Sharma, N. (2021, February 02). Retrieved from <https://nehaiosdeveloper.medium.com/swift-vs-react-native-vs-flutter-dart-which-one-to-choose-78553970c794>
- Apple. (n.d.). Retrieved from <https://developer.apple.com/support/xcode/>
- Chris. (n.d.). Retrieved from <https://codewithchris.com/xcode-for-windows/>
- Allen, S. (2021, June 10). Retrieved from <https://wwdcbysundell.com/2021/whats-new-xcode-13/>
- Hudson, P. (2021, June 10). Retrieved from <https://www.hackingwithswift.com/articles/236/whats-new-in-xcode-13>
- Apple. (n.d.). Retrieved from <https://developer.apple.com/documentation/xcode/resolving-common-configuration-and-build-issues>
- Wilson, T. (2020, March 18). Retrieved from <https://levelup.gitconnected.com/introduction-to-closures-in-swift-ea75477e8f0b>
- Smyth, N. (2018). SwiftUI Essentials - iOS Edition. Packt.
- Swanner, N. (2020, april 1). Retrieved from <https://insights.dice.com/2020/04/01/swift-tutorial-structs-classes-beginners/>
- Yu, A. (2021, November 17). Retrieved from <https://builtin.com/software-engineering-perspectives/swift-ui>

Agrawal, V. (2017, August 21). Retrieved from <https://code.tutsplus.com/articles/three-terrible-mistakes-of-ios-developers--cms-29355>

7t.co. (n.d.). *Mobile Market Share 2021 – Android vs iOS, Apple vs Samsung*. Retrieved from <https://7t.co/blog/mobile-market-share-2021-android-vs-ios-apple-vs-samsung/>

wikipedia. (n.d.). *Software design pattern*. Retrieved from wikipedia: https://en.wikipedia.org/wiki/Software_design_pattern

Benoit Pasquier. (n.d.). *How to implement MVVM pattern in Swift from scratch*. Retrieved from benoitpasquier: <https://benoitpasquier.com/ios-swift-mvvm-pattern/>

developer.apple.com. (n.d.). *Model-View-Controller*. Retrieved from Apple Developers: <https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html>

stackoverflow. (n.d.). *see why "type does not conform to protocol" in Xcode (swift)*. Retrieved from stackoverflow: <https://stackoverflow.com/questions/36872667/see-why-type-does-not-conform-to-protocol-in-xcode-swift>

Git Hub. (n.d.). *GitHub Desktop*. Retrieved from GitHub: <https://desktop.github.com/>

Wenderlich, R. (n.d.). *SwiftUI Maps & Location Fundamentals*. Retrieved from raywenderlich: <https://www.raywenderlich.com/14255236-swiftui-maps-location-fundamentals>

Bit Bucket. (n.d.). *Bit Bucket Home Page*. Retrieved from BitBucket: bitbucket.org

matchboard. (n.d.). *10 Great Reasons to build a Mobile App*. Retrieved from matchboard: <https://www.matchboard.com.au/10-great-reasons-to-build-a-mobile-app/>

Git Hub. (n.d.). *Git Hub Home Page*. Retrieved from GitHub: <https://github.com>

Apple. (n.d.). *Use fall detection with Apple Watch*. Retrieved from Apple Support: <https://support.apple.com/en-us/HT208944>

Cui, Y. (2020, February 13). *Best Practices for Using Optionals in Swift*. Retrieved from betterprogramming: <https://betterprogramming.pub/best-practices-for-using-optionals-in-swift-e9ac093ad50d>

Laso-Marsetti, F. (2019, April 15). *Model-View-Controller (MVC) in iOS – A Modern Approach*. Retrieved from raywenderlich: <https://www.raywenderlich.com/1000705-model-view-controller-mvc-in-ios-a-modern-approach>

Wise, J. (2022, 02 12). *Mobile & Desktop Operating System Market Share Stats For 2022*. Retrieved from earthweb: <https://earthweb.com/operating-system-market-share/>

Verizon. (n.d.). *A timeline: Notable milestones in the history of iPhone from Apple*. Retrieved from Verizon: <https://www.verizon.com/articles/milestones-in-history-of-apple-iphone>

Proulx, D. (n.d.). *iPhone Size Comparison Chart: Ranking Them ALL By Size*. Retrieved from pinterest: <https://www.pinterest.com/pin/642396334341225553/>

softwaretestinghelp. (n.d.). *Xcode Tutorial – What Is Xcode And How To Use It*. Retrieved from softwaretestinghelp: <https://www.softwaretestinghelp.com/xcode-tutorial/>

tutlane. (n.d.). *Swift Closures*. Retrieved from tutlane: <https://www.tutlane.com/tutorial/swift/swift-closures>

hackingwithswift. (n.d.). *Understanding generics – part 1*. Retrieved from hackingwithswift: <https://www.hackingwithswift.com/plus/intermediate-swift/understanding-generics-part-1>

Apple. (n.d.). *Structures and Classes*. Retrieved from Swift Docs: <https://docs.swift.org/swift-book/LanguageGuide/ClassesAndStructures.html>

tutorialspoint. (n.d.). *Swift Extensions*. Retrieved from tutorialspoint: https://www.tutorialspoint.com/swift/swift_extensions.htm

steelkiwi Inc. (n.d.). *SwiftUI vs UIKit: Benefits and Drawbacks*. Retrieved from steelkiwi: <https://steelkiwi.medium.com/swiftui-vs-uikit-benefits-and-drawbacks-6a540cced684>

journaldev. (n.d.). *Swift Error handling – Swift try, do catch, throws*. Retrieved from journaldev: <https://www.journaldev.com/19651/swift-error-handling-swift-try>

Hudson, P. (2018). *Swift Design Patterns*. In P. Hudson, *Swift Design Patterns*. Paul Hudson.

raywenderlich.com Team, J. C. (2017). *Swift Apprentice Third Edition: Beginning Programming with Swift 4*. Razeware LLC.

Feiler, J. (2017). *5. Learn Computer Science with Swift_ Computation Concepts, Programming Paradigms, Data Management, and Modern Component Architectures with Swift and Playgrounds* . Apress.

Hudson, P. (2016). *Pro Swift Break Out of Beginner's Swift*. Paul Hudson.

Ramnath, R. (2014). *Beginning iOS Programming For Dummies*. For Dummies.

Keur, C. (2015). *iOS Programming: The Big Nerd Ranch Guide*. Addison-Wesley Professional.

Neuburg, M. (2017). *iOS 11 Programming Fundamentals with Swift: Swift, Xcode, and Cocoa Basics*. O'Reilly Media.

Maurice Sharp, E. S. (2013). *Learning iOS Development: A Hands-on Guide to the Fundamentals of iOS Programming*. Addison-Wesley Professional.

Google. (n.d.). *Google Maps*. Retrieved from Google Maps: maps.google.com

Jakob Iversen, M. E. (2013). *Learning Mobile App Development: A Hands-on Guide to Building Apps With Ios and Android*. Addison-Wesley Professional.

Lim, G. (2020). *Beginning iOS 14 & Swift App Development*. Packt.

raywenderlich Team, M. H. (2016). *iOS Apprentice Fifth Edition: Beginning iOS development with Swift 3*. Razeware LLC.

Langer, A. M. (2012). *Guide to Software Development: Designing and Managing the Life Cycle* . Springer.

Gary Bennett, B. L. (2019). *Swift 5 for Absolute Beginners: Learn to Develop Apps for iOS*. Apress;.