Minyx's Dinner Food Ordering App

Michael J. Mendoza Navarro
Master in Software Engineering
Dr. Jeffrey Duffany
Computer Sciences Department
Polytechnic University of Puerto Rico

Abstract *— Smartphone applications are now a massive and valid field for exposure for companies, expansion of consumers' needs, and convenience for quick and easy access to multiple activities, including gaming, sales, and just viewing maps. Smartphones are now more a need than a convenience, and as such their applications are now staples of life. There are apps for calendars, music, e-books, streaming video, ordering food, and many other functions. What is most important is that applications work well and are easy for consumers to use. The main problem of many applications is how buggy they can be, fail to work as expected, or crash in inopportune moments. Moreover, apps need to be designed to work well in different types of smartphones like Android, iPhones, or Google phones. This project's goal is to make an iPhone application to order, pay, and receive food; in other words, a food ordering app.*

Key terms *— apps, food ordering app, smartphone applications, software design*

INTRODUCTION

What are apps and how important are they? Nowadays, in 2022, there are approximately 5 million apps in the mobile market. This alone shows the importance given to mobile apps worldwide and their effect on our daily life. Mobile apps have changed the way many things work, such as travelling, shopping, gaming, banking, and ordering food. They have brought a new and ever-changing aspect to our lives, and every day more and more apps are added to app stores.

Apps started around July 2008, when Apple and Google launched their app stores. Apple started with just 552 apps, and in just one week Apple apps had had over 10 million downloads. [1] Meanwhile, the Google App store had over 50 million downloads. These numbers showed that the new concept had been an instant hit.

With mobile phones becoming a necessity in present day, apps followed them in being part of that necessity. Now, apps are not as important as the mobile phone itself; the main purpose of a phone is to make calls and send text messages. But apps offer conveniences that make life much easier. Need a map? Download Google Maps. Need to read some books? Download Kindle. Want to listen to music? Download Spotify. That is, phones stopped being just phones and are now a way to access everything one might need: maps, music, games, banks, cameras, and a lot of other things.

What this means for users is that they depend more on apps and want good apps. What this means for business owners and companies is that apps are now expected. What this means for programmers is that apps are now a very good area to be knowledgeable on. Businesses rely a lot on apps for many things [2]; just as ads used to be very important to make a business known, now apps are important for business convenience and offers.

Maybe a business wants to offer coupons; a shopper is no longer needed to distribute them, as it may be done through an app. Or maybe a store wants to keep track of customers' loyalty points, which may also be done through an app. That is what brought this project along: the idea of an app for a business, in this case a restaurant. Restaurants were at first only in person, then came fast food, then ordering through a call, but now we can and should be able to order via an app.

BACKGROUND

Software is a huge field: there is software for the World Wide Web, computers, all manners of hardware, and phones, one of the most popular hardware used today. In the past, the phone was simply a piece of technology used to make calls, later sending text messages, and now it is a whole computer system in a miniature box. Its popularity is something to take advantage of, and this project aims to show possible phone software and how it may improve an aspect of a person's life as many apps do today.

The COVID-19 pandemic brought huge changes. Jobs, especially, were greatly affected and now many are done remotely, online. Restaurants and fast foods were also affected negatively. However, more and more people started using apps such as Uber Eats, Grubhub, and Doordash to order food. These apps are generic: they can be used to order any food and a random person delivers it. There is great potential in apps to make the most of current events by creating something new, useful, and economical.

In addition to general apps like the ones aforementioned, there are specialized apps that work for one specific thing, and as such are better fit for that one thing. This is the reason why apps rarely do a lot of things, but they are rather specialized. This is an opportunity to show what a specialized app for a fast-food restaurant could be, which a customer can order from, receive an estimate, and get food delivered straight to their location by an employee from said establishment. This would also represent an opportunity to learn to program an app for the Apple App Store and learn their programming language.

PROBLEM

Programming poses multiple challenges. Programming has multiple languages. This means that specialized software will probably use a different programming language. In our specific case, programming for Apple and Android use different languages. A programming software like Visual Studio could be used and a cross-platform app could be created both for Apple and Android, but this would make the app very thick in code, which would make the app take up more storage than needed in a phone. Therefore, it is generally better to simply make an app for either Apple or Android and translating the program into the other system afterwards. This does mean that the developer would need to learn a new programming language and ensure it works as expected.

Another challenge is mixing and matching the different sections of an app. An app may depend on a server, which would entail working on programming language to create a server and then attach it to the app so that they can work together. This is the same challenge for other attachments; if an app needed to use Google Maps, it would have to be attached to the app and its functionality ensured while being used in the software.

The different types of phones is another challenge. Different iPhones work with different systems, and the less said about the number of different Android phones, the better. Therefore, although an app should work well in all the phones of one type, this is not always guaranteed, as nothing is when it comes to programming.

In addition to keeping these things in mind to make an app as good as possible, it is also important to know that an app is never truly finished. Once released, an app should be monitored and updated as needed. This is not an issue in this project, but it should be kept in mind in actual commissioned apps.

EQUIPMENT AND MATERIAL

Software Components

There are multiple options for programming applications, but the one we will be using is Android Studio. The programming languages used for iOS in Visual Studio would be either C++ or F#; it is a matter of preference and needs. Android Studio uses Java or Kotlin, with Kotlin being the never language.

The OS used for the project is Windows 10. Mac OS used to be the only option to design iOS apps, and there is also the Apple developer app, with which a developer can test apps through iPhone or iPad for a $100 subscription. Android has a smoother process. Visual Studio allows for the designing of apps for Android and the emulation of many types of Android phones to test the app. Android Studio specializes in anything Android.

- **Visual Studio 2022:** This software serves as a standalone source code editor that works on Windows, Mac OS, and Linux. It is used to develop computer programs for anything from websites and computer programs to mobile apps. To design apps, we would specifically use Xamarin, which is a cross-platform iOS and Android mobile phone app designer, or mobile development using C++.
- **Android Studio:** This software, released in 2014, is a source code editor for all Android devices. Unlike Visual Studio, this is specialized software for that section of Google's company. It uses the languages of Java, and now the never Kotlin. This software makes it is easy to emulate Android phones and tablets and offers a smoother Android app developing process.
- **Firebase:** This software is a database system designed to work well with Android Studio. In fact, Android Studio has a tab dedicated to Firebase. With this software, we can emulate what a database would be like for our app and, for a fee, it can be used officially once the app is released so that it works beyond testing. Firebase can work as an authenticator, database, Realtime database, Storage, hosting, and testing, among other functions.

METHODOLOGY

The approach used for this project is one purely of experimentation. Software engineering is widely taught in classes, but programming languages tends to be a neglected topic, probably because once one understands how software programming works, it can be applied to multiple programming languages and it is only a matter of getting used to how each type of language works. So this project required a lot of experimentation while learning the language used to create the app. This approach implied that designing the app would require making many mistakes and having to restart it due to complications.

Software engineer keeps getting more and more complicated every year. Some programming languages are outdated, some new ones come out. Software programmers need to remain up to date. For example, not long ago a programmer could not design an app for iOS in Windows 10; this is relatively new to Windows 10. An Apple computer had to be purchased just to be able to program an app for iOS. Meanwhile, it has been possible to develop Android in Windows 10 for a long time and there are even emulations to test different Android phones.

Apple used what is known as Xcode, along with the Swift software, to be able to make apps for iOS. These were Mac-exclusive software. Now Swift works on Windows 10, or Visual Studio can be used to make apps for iOS. Moreover, apps can work for both iOS and Android.

In Visual Studio, an app that would work for both iOS and Android can be created using Xamarin. It would be more code-intensive, but it would be a quicker way to design an app that would work for both types of OS. Visual Studio is a competitor of Android Studio, which focuses solely on Android. Visual Studio is a more complete application than Android Studio, since the latter focuses solely on Android.

In Android Studio, apps for phones, tablets, and more specialized devices like Android watches can be made. Android Studio was created for a better programming experience for Android.

To get a better idea of what a programmer wants when making a program, it is usually a good approach to first make designs, such as Object-Oriented Designs. In this case, we would be making a food ordering app, so we would have to make an object-oriented design based on that. This works not only in the programmer's favor, but also for others such as the customer itself or other programmers.

In this case, two designs were made: a use case diagram, to have a good idea of how the app will interact between users and the database, and a design showing the program structure, to have a good idea of the classes, XMLs, and interfaces that would be used.

The use case diagram will show how a user and a database interact with the shopping cart, menu, or About section (figure 1).

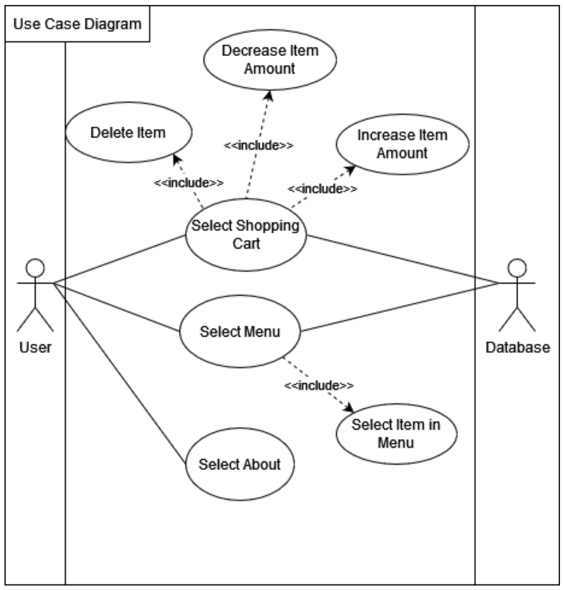


Figure 1
Use case diagram for Minyx's Dinner

Figure 2 shows how the program should be built in code, a program structure of sorts. This design shows the program's expected end result, how many layouts or XMLs it should have, the number of classes, and the resulting Interfaces.

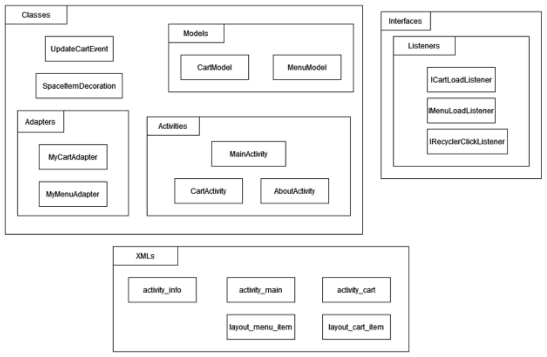


Figure 2
Program structure

The designs are, of course, examples to help with the code procedures that are to be made. Ultimately, how well the code is done is most important. That is why designs are so helpful. A good picture of how everything works and connects eases carrying out the idea in coding. While making the code, it is also good to keep commentary alongside the code, so that any confusion in the code can be cleared up easily. Usually, programmers differ greatly in what the code ends up looking like. There are, after all, many ways to obtain the desired results with different ways of programming.

The step that follows finishing a program is testing. A programmer should constantly test an app while making it to fix any mistakes. After this, other users will also test the app and more mistakes will be often found that will need to be fixed. This is why software have frequent updates: to fix mistakes that had been left inside undiscovered.

Then follows simply maintaining an app and making future updates. An app is very easy to update, and active apps are expected to be constantly updated as needed.

This process of designing, coding, debugging, and future updating is the standard. There will never be perfect software at the end.

This project in particular is about a food ordering app called the Minyx's Dinner App. Designs have been previously worked on for, and now it is time to find where to design it, what programming language to use, and how to make it work. The app will have a login screen, a menu to order from, checkout, and estimated time, among other details.

RESULTS AND DISCUSSION

The results differed greatly from the original goal, which was to make an iOS app for ordering food, with some added features that included a map for the driver and servers to hold the app and work with.

The first challenge arose when the design of the app for iOS had been worked on for a while. Online research said that today iOS could be designed in Windows 10 and would not require an Apple computer, which we lacked. Research failed to show that testing the app would not only require an iPhone or iPad, but also a $100 subscription to the Apple developer. This caused the change to making the app for Android instead.

While doing the program for iOS, Visual Studios 2022 was used, with Xamarin. Generally, Xamarin has the option of making an app for both iOS and Android at the same time. However, that code would be more complex and data-heavy, so initially we worked only for iOS. After learning about the conditions to test iOS, the switch to Android backtracked all our progress at the time.

Android is iPhone's competition, and there are fewer Android apps than Apple apps, but Android is no less relevant. The change to Android only implied a difference in OS and code and not in the project's objective. It did come with a drawback of having to restart the app in another programming language and software, but it was not a total loss. The general idea of the app was already there; it just had to adapted to the new software.

This showed the that, although programming languages are similar in idea, they are different enough to require additional training.

When the decision to change the app to Android mas made, the software was changed from Visual Studio to Android Studio. Visual Studio can be used to make Android apps, but we determined a software made specifically for Android would be better. In Android Studio, Android phones can be emulated easily to test the app while being made, making the process much smoother. During this process, the app received its official name, changing it form the original temporary name of McMichael's to Minyx's Dinner, representing a possible business would.

The Minyx's Dinner app will open by default on the menu page (figure 3) so that the users see all the different food and drink items they can order.

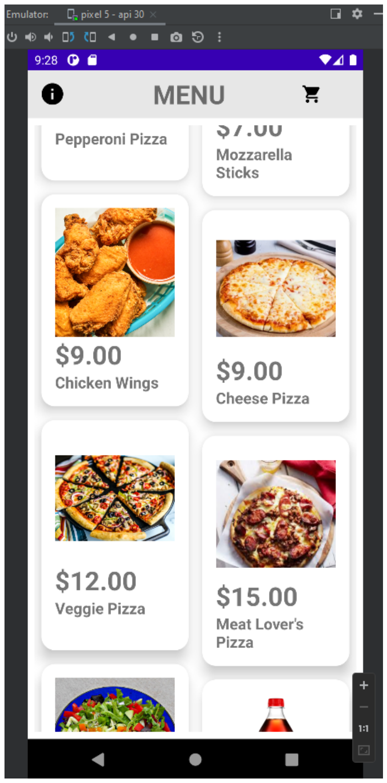


Figure 3
App's menu

Once a user has selected something from the menu by clicking on any of the items, the items will be added to the shopping cart. The app will display a message and the shopping cart will show the number of items in it (figure 4).

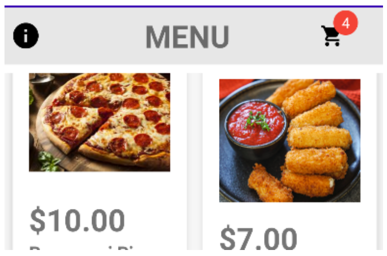


Figure 4
App's shopping cart notification

Once the user has finished making their selection, they may go to the cart to view it, add or decrease any item quantities, and see the total cost (figure 5).

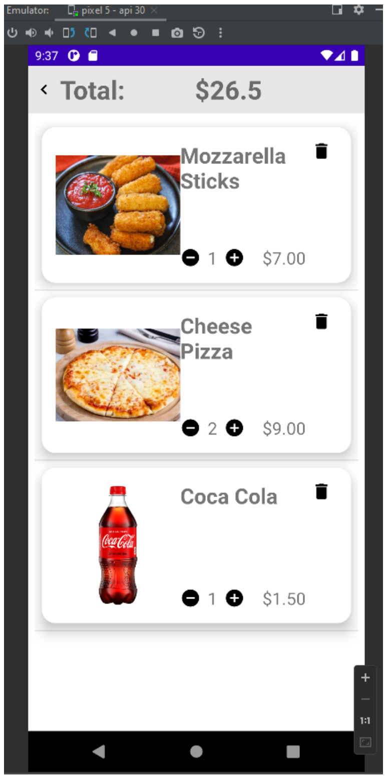
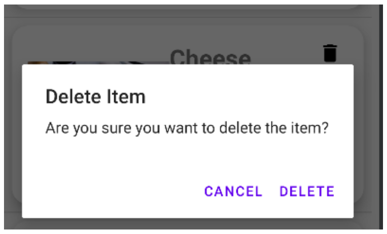


Figure 5
App's shopping cart

If item quantities are changed, the total cost changes in real time. If an item is deleted from the shopping, a warning message will be displayed (figure 6).

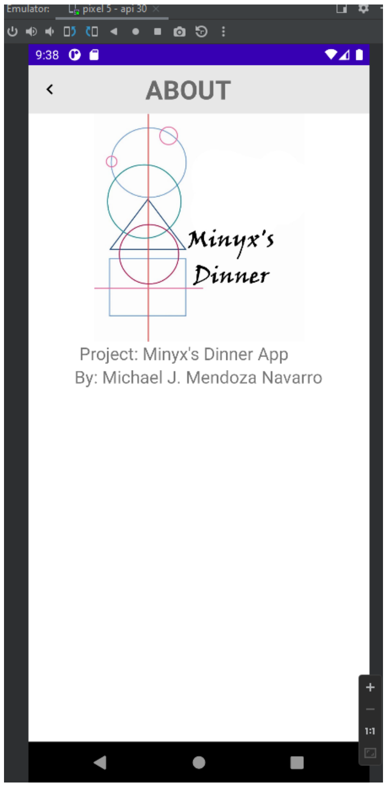


Delete Item

Are you sure you want to delete the item?

Figure 6
App's warning message for deleting item from cart

The last section of the app is the About section (figure 7), showing the app's name, logo, and the creator's name.



Project: Minyx's Dinner App
By: Michael J. Mendoza Navarro

Figure 7
App's About section

The app was tested on different Android phones and an Android tablet (figure 8).

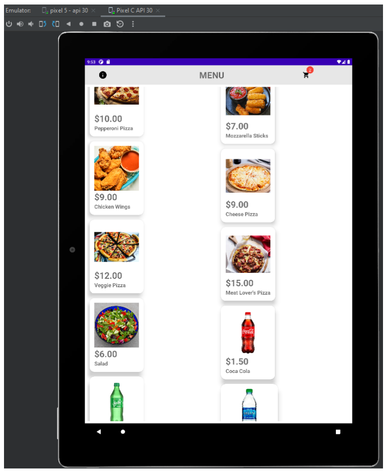
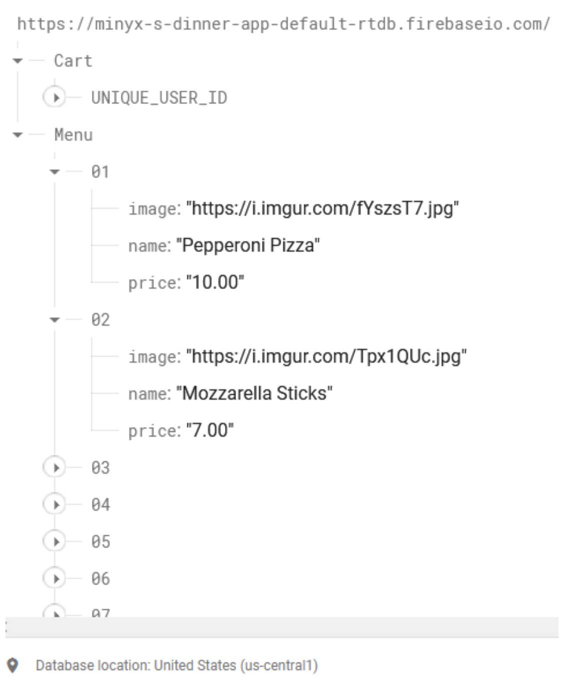


Figure 8
App tested on Android tablet

Now that we have shown how the app works for users, we can now approach its more technical side: the code within the app. We will begin by discussing the database. The database used was Firebase, which works using the formatting of JSON. All the menu items with their images, names, and cost were uploaded in the database. The Firebase database (figure 9) was then connected to the app using Android Studio. This way, if an item needs to be added or removed from the menu, it can simply be adjust in the database and the app will be automatically updated through the Internet.

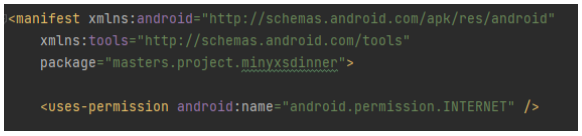


```
https://minyx-s-dinner-app-default-rtdb.firebaseio.com/
▾ — Cart
    ▸ — UNIQUE_USER_ID
▾ — Menu
    ▾ — 01
        — image: "https://i.imgur.com/fYszsT7.jpg"
        — name: "Pepperoni Pizza"
        — price: "10.00"
    ▾ — 02
        — image: "https://i.imgur.com/Tpx1QUc.jpg"
        — name: "Mozzarella Sticks"
        — price: "7.00"
    ▸ — 03
    ▸ — 04
    ▸ — 05
    ▸ — 06
    ▸ — 07
```

Database location: United States (us-central1)

Figure 9
Firebase database

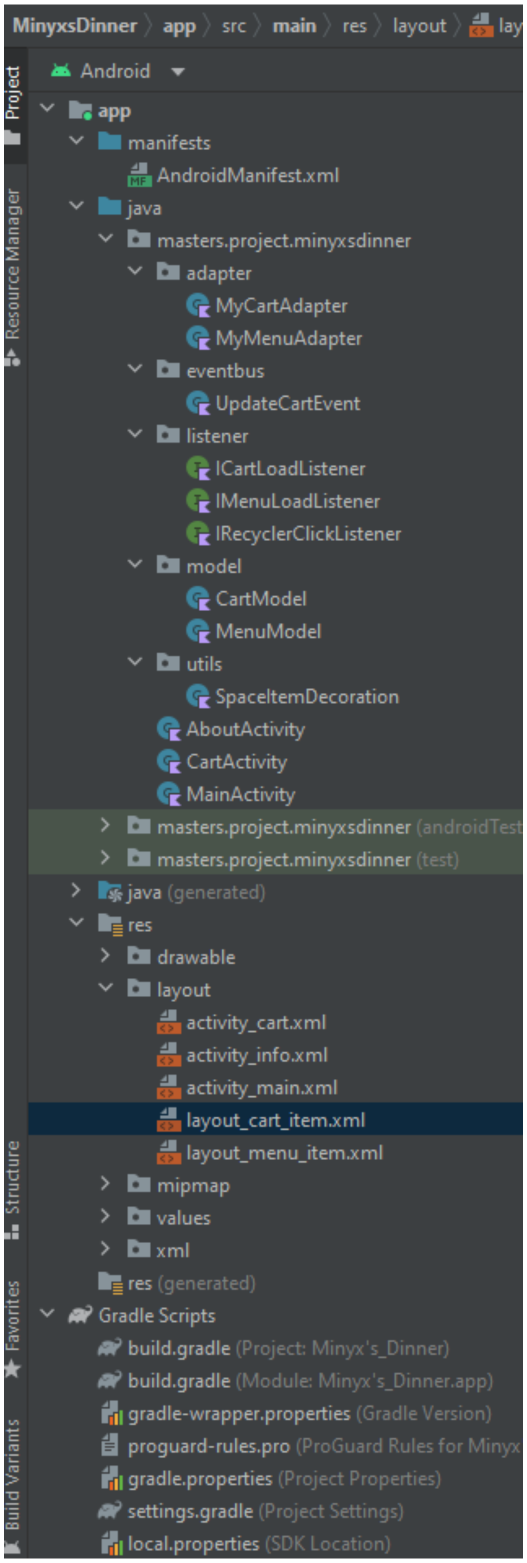
For the app to utilize the Firebase database, the app must be able to connect to the Internet. This is allowed for by adding a piece of code (figure 10) to the manifest of the program.



```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="masters.project.minyxsdinner">

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

Figure 10
Manifest's Internet code

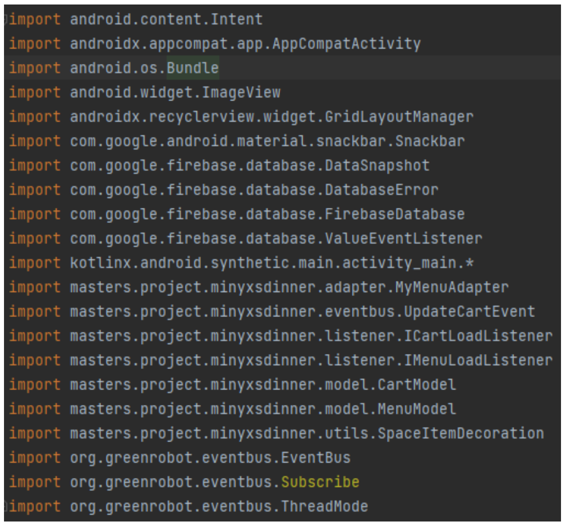
Even the simplest program tends to have many sections. The larger the program, the more its parts. Figure 5 already showed the parts the program will have, but figure 11 shows them in action.



```
MinyxsDinner 〉 app 〉 src 〉 main 〉 res 〉 layout 〉 lay
Android ▾
∨ app
    ∨ manifests
        AndroidManifest.xml
    ∨ java
        ∨ masters.project.minyxsdinner
            ∨ adapter
                MyCartAdapter
                MyMenuAdapter
            ∨ eventbus
                UpdateCartEvent
            ∨ listener
                ICartLoadListener
                IMenuLoadListener
                IRecyclerClickListener
            ∨ model
                CartModel
                MenuModel
            ∨ utils
                SpaceItemDecoration
            AboutActivity
            CartActivity
            MainActivity
        > masters.project.minyxsdinner (androidTest
        > masters.project.minyxsdinner (test)
    > java (generated)
    ∨ res
        > drawable
        ∨ layout
            activity_cart.xml
            activity_info.xml
            activity_main.xml
            layout_cart_item.xml
            layout_menu_item.xml
        > mipmap
        > values
        > xml
      res (generated)
∨ Gradle Scripts
    build.gradle (Project: Minyx's_Dinner)
    build.gradle (Module: Minyx's_Dinner.app)
    gradle-wrapper.properties (Gradle Version)
    proguard-rules.pro (ProGuard Rules for Minyx
    gradle.properties (Project Properties)
    settings.gradle (Project Settings)
    local.properties (SDK Location)
```

Figure 11
Minyx's Dinner app files

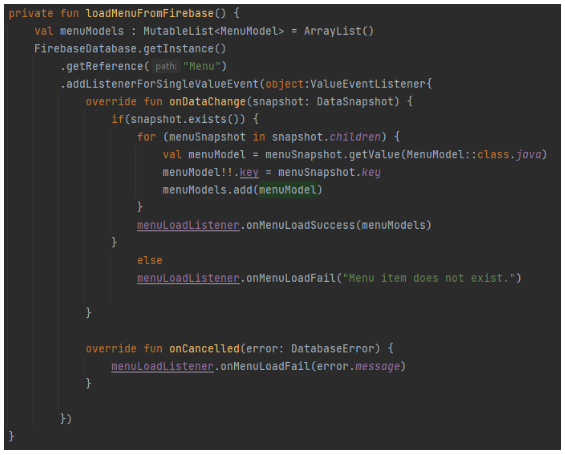
When coding, the classes are connected by using imports into a class. Imports also bring into the program pre-made classes or external connections. The main page tends to be the one with the most imports. In this app, data for Firebase had to be imported (figure 12).



```
import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.ImageView
import androidx.recyclerview.widget.GridLayoutManager
import com.google.android.material.snackbar.Snackbar
import com.google.firebase.database.DataSnapshot
import com.google.firebase.database.DatabaseError
import com.google.firebase.database.FirebaseDatabase
import com.google.firebase.database.ValueEventListener
import kotlinx.android.synthetic.main.activity_main.*
import masters.project.minyxsdinner.adapter.MyMenuAdapter
import masters.project.minyxsdinner.eventbus.UpdateCartEvent
import masters.project.minyxsdinner.listener.ICartLoadListener
import masters.project.minyxsdinner.listener.IMenuLoadListener
import masters.project.minyxsdinner.model.CartModel
import masters.project.minyxsdinner.model.MenuModel
import masters.project.minyxsdinner.utils.SpaceItemDecoration
import org.greenrobot.eventbus.EventBus
import org.greenrobot.eventbus.Subscribe
import org.greenrobot.eventbus.ThreadMode
```

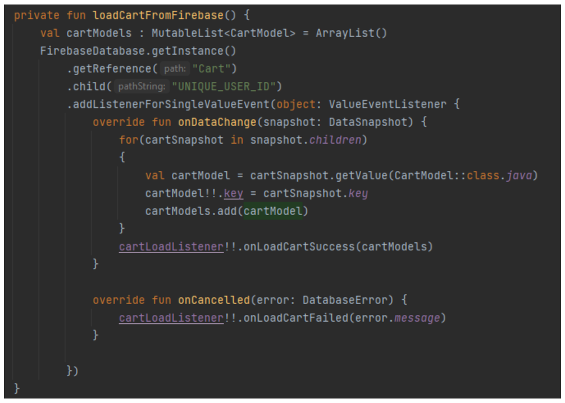
Figure 12
Minyx's Dinner App Imports

Firebase also needed a section of the code dedicated to loading the information from Firebase into the required parts of the program (figures 13 and 14). This piece of code required, among other bits, models, to be able to use the correct objects; arrays, because, since the menu has multiple items, arrays allow for the use of the same values over and over again for different items; and listeners, so that the code knows how to react to certain inputs.



```
private fun loadMenuFromFirebase() {
    val menuModels : MutableList<MenuModel> = ArrayList()
    FirebaseDatabase.getInstance()
        .getReference( path: "Menu")
        .addListenerForSingleValueEvent(object:ValueEventListener{
            override fun onDataChange(snapshot: DataSnapshot) {
                if(snapshot.exists()) {
                    for (menuSnapshot in snapshot.children) {
                        val menuModel = menuSnapshot.getValue(MenuModel::class.java)
                        menuModel!!.key = menuSnapshot.key
                        menuModels.add(menuModel)
                    }
                    menuLoadListener.onMenuLoadSuccess(menuModels)
                }
                    else
                    menuLoadListener.onMenuLoadFail("Menu item does not exist.")

            }

            override fun onCancelled(error: DatabaseError) {
                menuLoadListener.onMenuLoadFail(error.message)
            }

        })
}
```

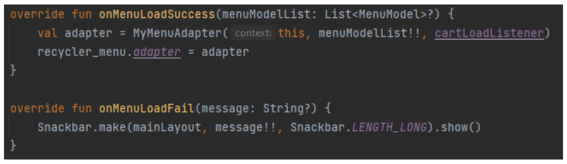
Figure 13
Menu loaded using Firebase code



```
private fun loadCartFromFirebase() {
    val cartModels : MutableList<CartModel> = ArrayList()
    FirebaseDatabase.getInstance()
        .getReference( path: "Cart")
        .child( pathString: "UNIQUE_USER_ID")
        .addListenerForSingleValueEvent(object: ValueEventListener {
            override fun onDataChange(snapshot: DataSnapshot) {
                for(cartSnapshot in snapshot.children)
                {
                    val cartModel = cartSnapshot.getValue(CartModel::class.java)
                    cartModel!!.key = cartSnapshot.key
                    cartModels.add(cartModel)
                }
                cartLoadListener!!.onLoadCartSuccess(cartModels)
            }

            override fun onCancelled(error: DatabaseError) {
                cartLoadListener!!.onLoadCartFailed(error.message)
            }

        })
}
```

Figure 14
Shopping cart loaded using Firebase

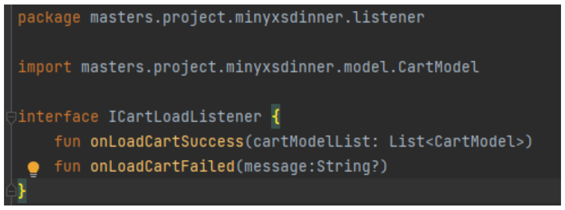
Those pieces of codes needed redundancies in case they did not load as expected, for which we have the functions in case they fail or succeed and what would happen in both situations (figure 15). The code that works in a successful condition will give the steps the program will take. There are two versions of this code in the program: one for the menu and another for the shopping cart. In case of failure, either code will send out a message so the user knows.



```
override fun onMenuLoadSuccess(menuModelList: List<MenuModel>?) {
    val adapter = MyMenuAdapter( context: this, menuModelList!!, cartLoadListener)
    recycler_menu.adapter = adapter
}

override fun onMenuLoadFail(message: String?) {
    Snackbar.make(mainLayout, message!!, Snackbar.LENGTH_LONG).show()
}
```

Figure 15
Loading success or failure codes

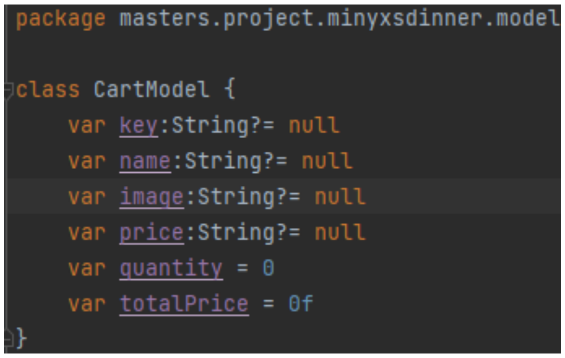
The previous code used listeners. Listeners are interfaces that will react to the input and either give a positive or negative result to it. In this case, it will show if the results of the input were either successful or a failure. This code bit is shown in figure 16.



```
package masters.project.minyxsdinner.listener

import masters.project.minyxsdinner.model.CartModel

interface ICartLoadListener {
    fun onLoadCartSuccess(cartModelList: List<CartModel>)
    fun onLoadCartFailed(message:String?)
}
```

Figure 16
Listener code

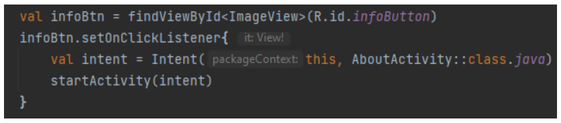
We mentioned that multiple lines of this code use models, which are basically the classes that declare the objects and how they work. There will be models for both the shopping cart (figure 17) and the menu.



```
package masters.project.minyxsdinner.model

class CartModel {
    var key:String?= null
    var name:String?= null
    var image:String?= null
    var price:String?= null
    var quantity = 0
    var totalPrice = 0f
}
```

Figure 17
Model class code

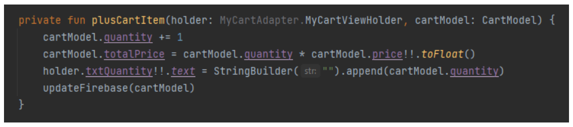
The program has buttons in multiple areas. There are buttons to switch from the menu, shopping cart, and About section. There are also buttons to add, subtract, or delete items in the shopping cart. Those buttons require code to actually work as buttons and not as mere decorations. Figure 18 shows the code used to go from the menu to the About section when clicking on the info button at the top left of the menu page.



```
val infoBtn = findViewById<ImageView>(R.id.infoButton)
infoBtn.setOnClickListener{
    val intent = Intent( this, AboutActivity::class.java)
    startActivity(intent)
}
```

Figure 18
Button use code

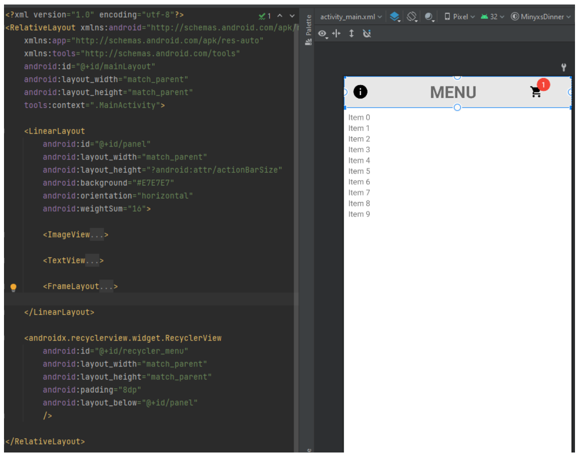
The button code in figure 19 is simple, since because it is a code to switch from one page to another. However, the plus and minus buttons in the shopping cart required something more. They are supposed to add or subtract to the total number of items wanted and we had to make functions for both adding and subtracting. Figure 19 shows a function that adds to the number of items in the shopping cart.



```
private fun plusCartItem(holder: MyCartAdapter.MyCartViewHolder, cartModel: CartModel) {
    cartModel.quantity += 1
    cartModel.totalPrice = cartModel.quantity * cartModel.price!!.toFloat()
    holder.txtQuantity!!.text = StringBuilder( "").append(cartModel.quantity)
    updateFirebase(cartModel)
}
```

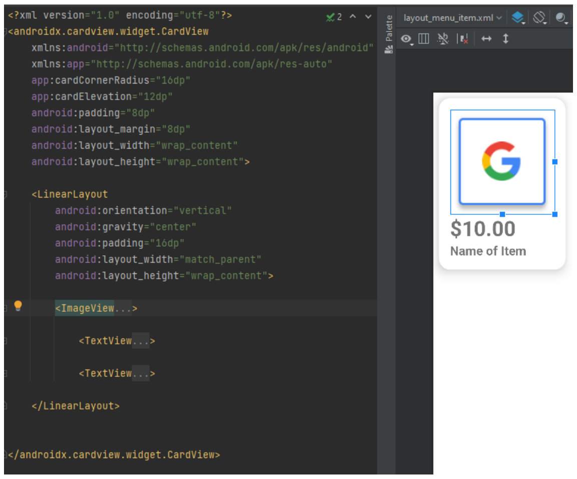
Figure 19
Addition function in cart item code

Now we arrive to one of the code's trickiest parts: the Firebase database data is downloaded into the program, but this information needs to be presented to the user. After three different attempts, it was ultimately decided to make it using the recycler function and the card widget in Android Studio. The recycler function is a code in which an area is designated and the data is supplied, and that data will be created dynamically, meaning as many items as desired may be added, and the recycler will keep adding them and fitting them without issue. Meanwhile, the card widget would fit in perfectly with the recycler. With the card widget, a "card" is designed and its values designated. Once a card is created, it will be filled with information directly from the database as needed. These cards are then sent to the recycler and added dynamically. The recycler for the menu is shown in figure 20; the card for the menu is shown in figure 21. However, both the shopping cart and the menu have recyclers and cards.



```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/mainLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <LinearLayout
        android:id="@+id/panel"
        android:layout_width="match_parent"
        android:layout_height="?android:attr/actionBarSize"
        android:background="#E7E7E7"
        android:orientation="horizontal"
        android:weightSum="16">

        <ImageView...>

        <TextView...>

        <FrameLayout...>

    </LinearLayout>

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/recycler_menu"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:padding="8dp"
        android:layout_below="@+id/panel"
        />

</RelativeLayout>
```

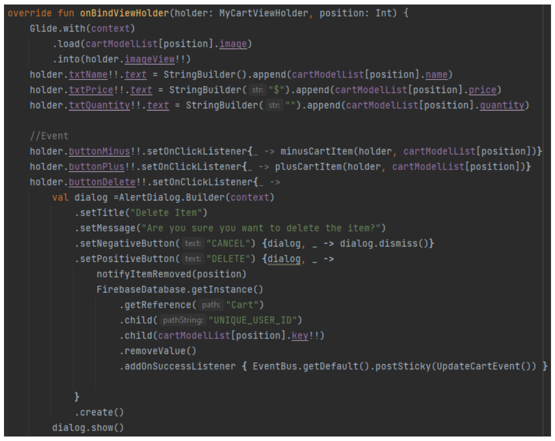
Figure 20
Recycler view and layout code for the menu



```
<?xml version="1.0" encoding="utf-8"?>
<androidx.cardview.widget.CardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    app:cardCornerRadius="16dp"
    app:cardElevation="12dp"
    android:padding="8dp"
    android:layout_margin="8dp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">

    <LinearLayout
        android:orientation="vertical"
        android:gravity="center"
        android:padding="16dp"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <ImageView...>

            <TextView...>

            <TextView...>

    </LinearLayout>


</androidx.cardview.widget.CardView>
```

Figure 21
Card widget and layout code for the menu

Lastly, we have the piece of code that would manage the cards for the shopping cart. These cards must have a name, price, image, and quantity of items in the shopping cart. It should also be taken in consideration the uses that will be given to the plus, minus, and delete buttons (figure 22).



```
override fun onBindViewHolder(holder: MyCartViewHolder, position: Int) {
    Glide.with(context)
        .load(cartModelList[position].image)
        .into(holder.imageView!!)
    holder.txtName!!.text = StringBuilder().append(cartModelList[position].name)
    holder.txtPrice!!.text = StringBuilder( str: "$").append(cartModelList[position].price)
    holder.txtQuantity!!.text = StringBuilder( str: "").append(cartModelList[position].quantity)

    //Event
    holder.buttonMinus!!.setOnClickListener{_ -> minusCartItem(holder, cartModelList[position])}
    holder.buttonPlus!!.setOnClickListener{_ -> plusCartItem(holder, cartModelList[position])}
    holder.buttonDelete!!.setOnClickListener{_ ->
        val dialog =AlertDialog.Builder(context)
            .setTitle("Delete Item")
            .setMessage("Are you sure you want to delete the item?")
            .setNegativeButton( text: "CANCEL") {dialog, _ -> dialog.dismiss()}
            .setPositiveButton( text: "DELETE") {dialog, _ ->
                notifyItemRemoved(position)
                FirebaseDatabase.getInstance()
                    .getReference( path: "Cart")
                    .child( pathString: "UNIQUE_USER_ID")
                    .child(cartModelList[position].key!!)
                    .removeValue()
                    .addOnSuccessListener { EventBus.getDefault().postSticky(UpdateCartEvent()) }

            }
            .create()
        dialog.show()
```

Figure 22
Shopping cart card managing code

Undergoing processes such as needing servers, the multiple hoops one has to jump through to program for Apple, and the different programming software that can be used for both iOS and Android, including Swift, Visual Studio, and Android Studio, are useful to any potential developer.

The results show that, despite some bumps, the project yielded experiences instead of losses.

CONCLUSION

The experience of working with multiple software, learning about them, and the process to make them work during this project will be evident when these products are used in the future. Knowledge has been acquired on Swift, Visual Studio, and Android Studio; the process behind the creation of apps; the hurdles in making them; and programming languages that each platform uses. The impact of apps in today's world is massive. Knowledge on apps is critical for computer engineers and any computer-related specialization. People depend on technologies that use apps more and more. Therefore, this project was critical to gain experience in this area.

Apps designed to deliver food is not an unknown idea, as discovered in the research for this project. There are many apps like this, and although most do not work appropriately, some are exceptionally good. The fact that most apps barely function appropriately reinforces the importance of knowledge on app developing. It is very disappointing that apps for companies like Pizza Hut or Domino's Pizza often crashed and did not even allow for a smooth login. This is a field that needs great improvement.

FUTURE WORK

When coming up with the idea for this app, there were many possibilities. It could be bigger, with a map showing the drivers, menus, shopping cart, and many other features. Not all features that were though of were included.

An app will never be perfect and will have to be constantly updated. Apps depend on the system they are on, so if the OS of a smartphone is updated, an app might not work as well anymore, or something in the app might make it stop working.

Future work would include adding usernames and passwords to the app and including a login screen, so that the app could save credentials for an easy login, as having to enter their credentials every time they want to use the app may be an inconvenience to users. The server should also contain information on sales and receipts. Usually, fast-food restaurants have their own server, so it would merely require connecting the app to their servers and making sure they work well together.

Another future improvement would be splitting the app for customers and employees.

REFERENCES

[1] M. Srivastava, (2022, January 10). "Era of mobile apps: How they have changed the way we live," Mobile App Daily, Jan 10, 2022 [Online]. Available: https://www.mobileappdaily.com/how-mobile-apps-are-changing-our-lives

[2] A. McKeon, "The rise of 'modern applications': Why you need them," TechTarget, July 24, 2020 [Online]. Available: https://www.techtarget.com/searchcio/feature/The-rise-of-modern-applications-Why-you-need-them