

Project Report 2018-19

Royal Café
iOS Restaurant App

Tarun Verma

A00258754

B.Eng. (Hons) Software Engineering YEAR 4

Contents

Introduction.....	3
Title	3
Overview	3
About the project	3
Objective	3
Research.....	4
Technologies	4
Working	5
Requirements	6
Functional -.....	6
Non-Functional	6
Architecture	7
Implementation	10
Application screenshots	10
Code Snippets	16
Conclusion.....	18
References.....	19
Appendix A.....	20

Introduction

Title

Royal Cafe – iOS Restaurant App

Overview

The project mainly focuses on development of an iOS app from scratch. Having no prior knowledge of the language, the primary objective is to learn how to code in Swift. The application works on taking orders from user in a restaurant and managing it. The plan is to work from customer perspective in the first stage and then adding more features for restaurant manager. The application is connected with cloud using Google Firebase to store the database tables and authenticate users for signup and login which helped in understanding cloud computing as well. The benefit of having such an application for a restaurant is to automate its services and make it easy for customers to order the food of their choice, from the comfort of their home.

About the project

A customized restaurant application that will not only allow users to check the real-time menus, but also book a table for their desired time in the restaurant and order the food for dine-in or take-away and even pay bills from within the app. Users can also signup and login to the app. Features like edit cart, order history are also included.

Objective

The objective behind opting for this project is to learn a popular and one of the most demanding mobile application development language, Swift and one of the popular mobile application development platforms, Google Firebase.

Research

Technologies

SWIFT The application is developed for any iOS device, primary reason to select this is to learn an all-time demanding and popular Swift programming language. Though the market share of iOS is relatively low as of Android, there is always a demand of Swift developers due to their less number. Swift is a general-purpose programming language developed by Apple for devices like iOS, tvOS, watchOS, etc. It is very easy to use and expressive language with a simplified syntax and grammar. It is based on Objective-C, but very concise, which means less code is required to perform the same task, as compared to Objective-C.

AWS RDS (Relational Database System) was initially selected to be used as a database for the project, because AWS is the most popular choice for cloud services at this time, and provides a clear documentation of its services to learn and use them easily. It gives a good understanding of cloud services for first time learners. AWS RDS makes it easy to set up, operate, and scale a relational database in the cloud. It provides cost-efficient and resizable capacity while managing time-consuming database administration tasks itself, freeing us to focus on our application and business.

GOOGLE FIREBASE Firebase is a relatively new branch of cloud services, also known as "Back end as a service", traditionally where we have to build REST APIs in order to retrieve data from the server, Firebase makes it easier with just one line of code. Whether its storing data, verifying users or anything else, it does it instantly. Basically, Firebase is a technology that allows us to create mobile and web applications with no server-side programming so that the development turns out to be quicker and easier. There are lots of services provided by Firebase, the ones that I have used are:

Firebase Auth: It provides backend services, easy-to-use SDKs, and ready-made UI libraries to authenticate users to the app. It supports authentication from various methods like passwords, phone numbers and popular federated identity providers like Google, Facebook and Twitter, and more. It integrates tightly with other Firebase services. Knowing a user's identity allows an app to securely save user data in the cloud and provide some personalized experience.

Cloud Firestore: It is a fast, fully managed, serverless, cloud-native NoSQL document database that simplifies storing, syncing, and querying data from mobile, web, and IoT apps at global scale. Its client libraries provide live synchronization and offline support, while its security features and integrations with Firebase and Google Cloud Platform (GCP) accelerate building truly serverless apps. Cloud Firestore is a cloud-native database, which provides an automatically scaling solution. It gives the developer ability to directly talk to Cloud Firestore from mobile or

web clients for a truly serverless solution. No need to set up an intermediary server to manage access to the data.

COCOAPODS CocoaPods is a dependency management tool for Swift and Objective-C projects similar to what Maven is for Java projects, which is written in Ruby. It has over 30 thousand libraries. Essentially, it helps us to incorporate third-party libraries, frameworks, into our product without worrying about how to set them up and configure the project, which at times could be a huge pain. Also, once we add the third-party libraries to our project we'll no longer need to check if there's any newer versions. CocoaPods will handle that itself. It also makes reversing back to a certain version of the library super easy.

MVC Framework The Model-View-Controller is an architectural pattern that separates an application into three main logical components: the model, the view, and the controller. Each of these components are built to handle specific development aspects of an application. MVC is one of the most frequently used industry-standard web development frameworks to create scalable and extensible projects. The Model component corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data. The View component is used for all the UI logic of the application. Controllers act as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final output.

Working

Since both the technologies (iOS and AWS) were new for me, idea was to first get some basic working knowledge of both the technologies. Started with learning Swift programming language which is on demand and has very less no. of developers. I started learning from Udemy course and cleared my doubts online. AWS is chosen because it is the most popular and demanding cloud service provider at this time and is the best option to begin learning cloud computing, also available with several features for free along with proper documentation. I was also going through an online course to understand AWS, but when I started working on it, I had too many difficulties to get it started. I was involved in it for very long, then I planned to work with basic REST APIs to connect the application to the server. I created APIs in PHP and used PHPMyAdmin for the database. XAMPP is used to create the server and database on our local system. Since I wanted to use online services, I decided to go with Google Firebase, which provides many services for mobile application development. Finally, I created my application in Swift using Xcode IDE for Mac and integrated it with Firebase for authentication of users and storing data in Firestore NoSQL database.

Requirements

Functional - The application must be able to perform following tasks:

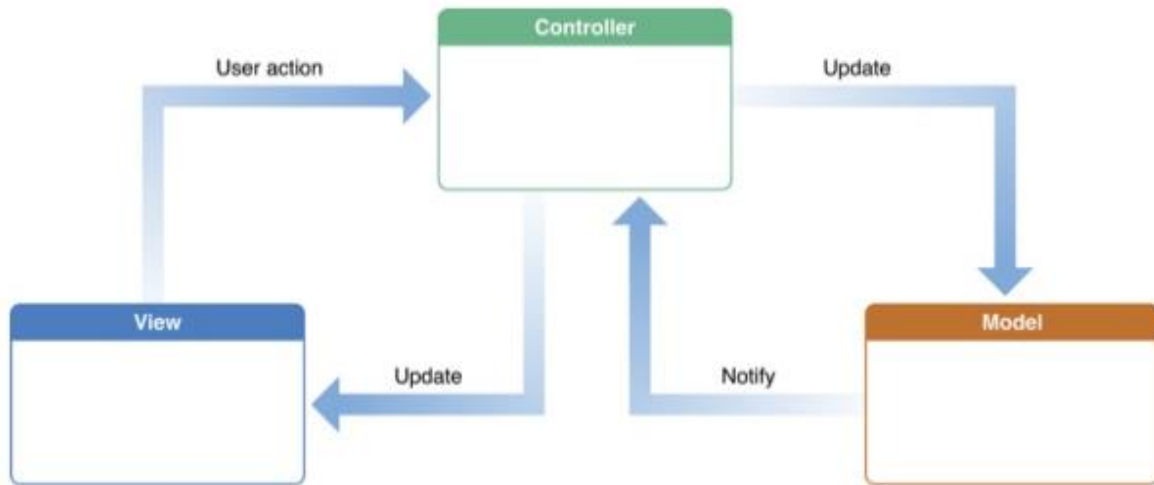
- Prompt the user for login.
- If the user has not signed up, provide an option for signing up.
- Provide the list of food items to the user, which is stored in Firestore.
- User must be able to explore the menu for different categories.
- User must be able to add any product in the cart.
- Cart should provide an option to delete, or change the quantity of the product added.

Non-Functional – The application should be able to perform given tasks:

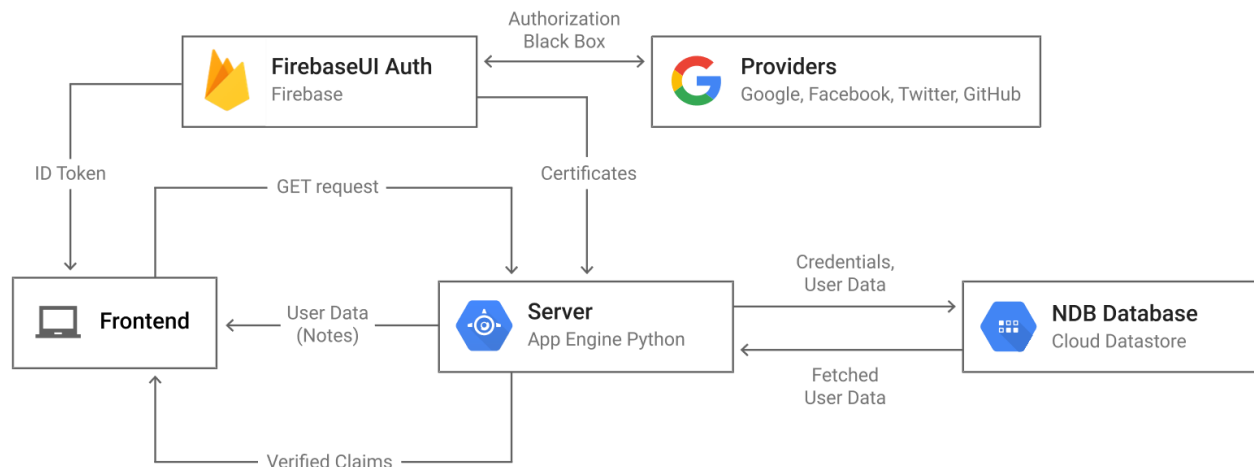
- Ask for valid email id and password for signing up and add the user in firebase authentication service.
- Check for user credentials from firebase for authentication while logging in.
- Menu shown to the user should be properly categorized.
- A separate Firestore collection should be created for each user to manage cart and order history.

Architecture

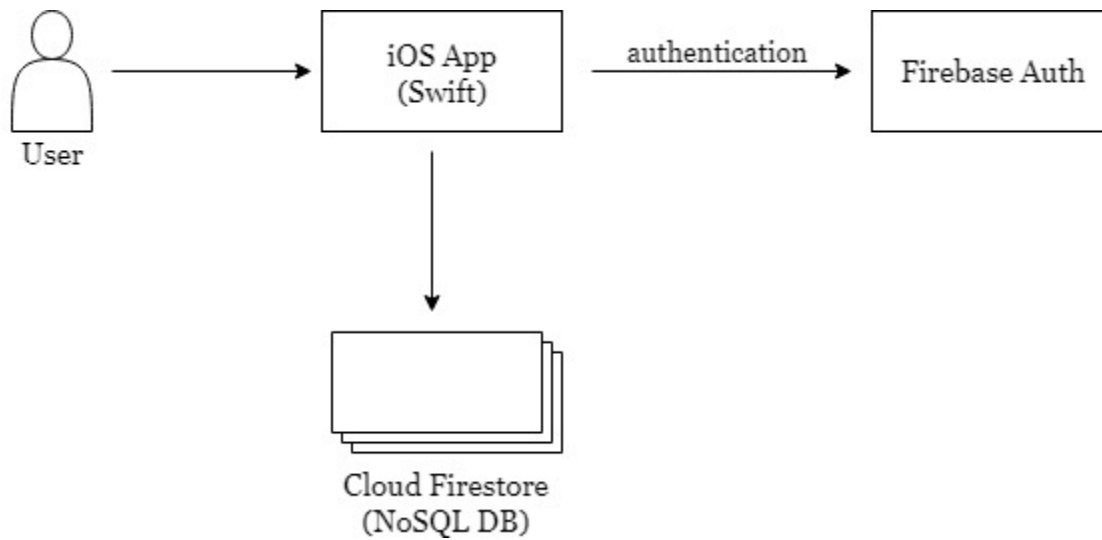
High-level architecture of Swift



High-level architecture of Firebase Authentication service

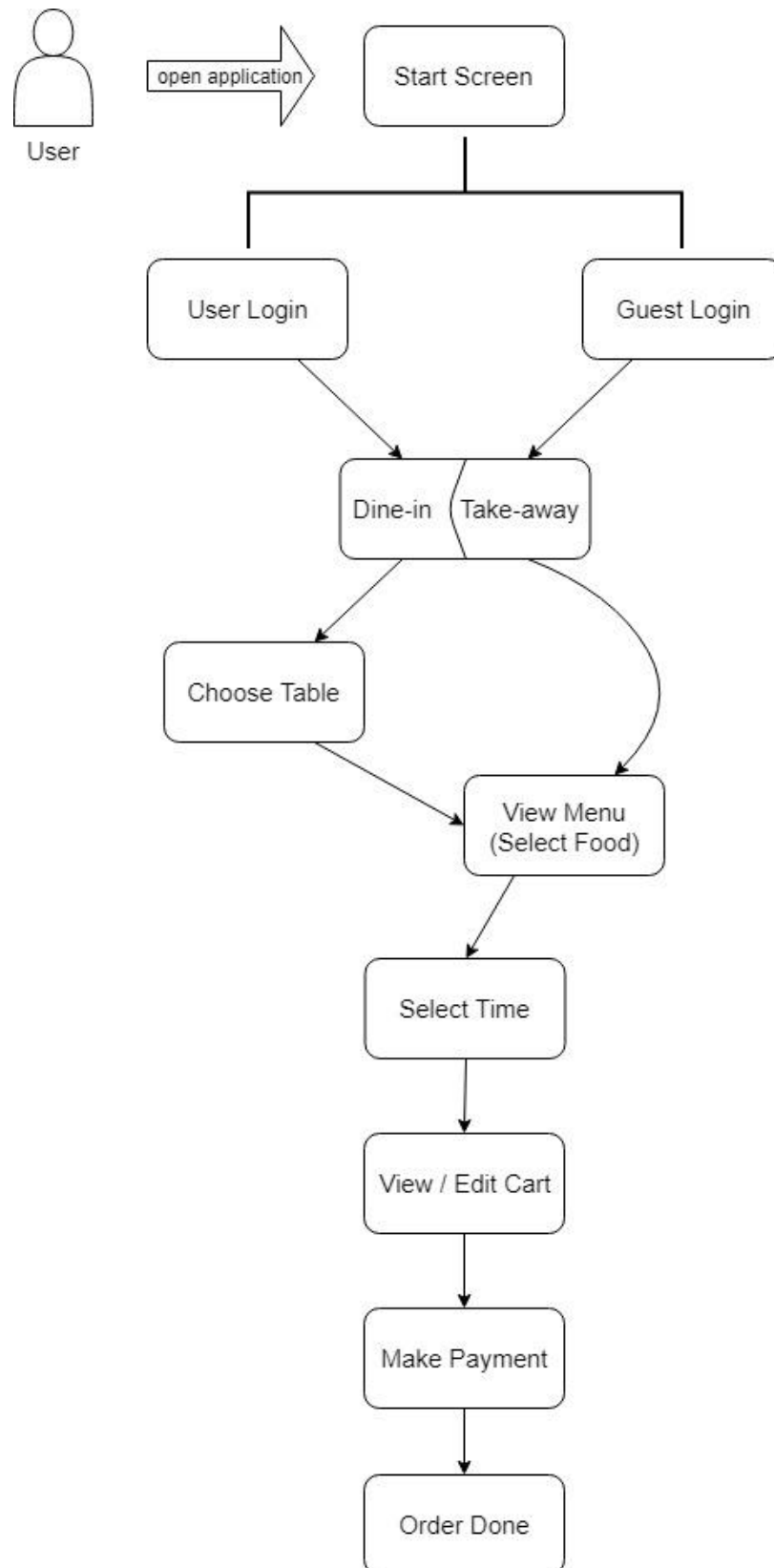


Application architecture



(Firebase helps to develop the application with no server-side programming, therefore no server APIs required to connect to the database.)

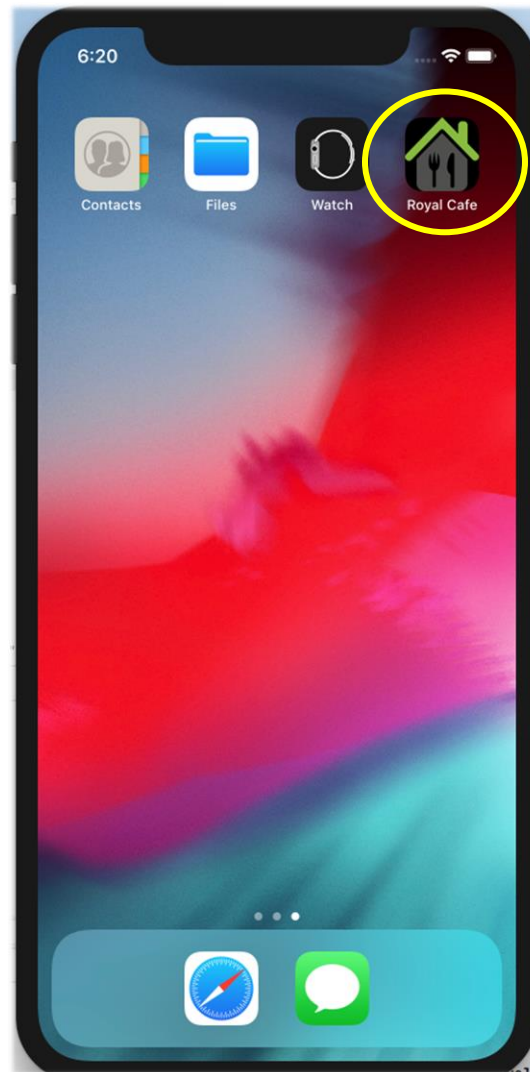
UML Diagram – Activity Diagram



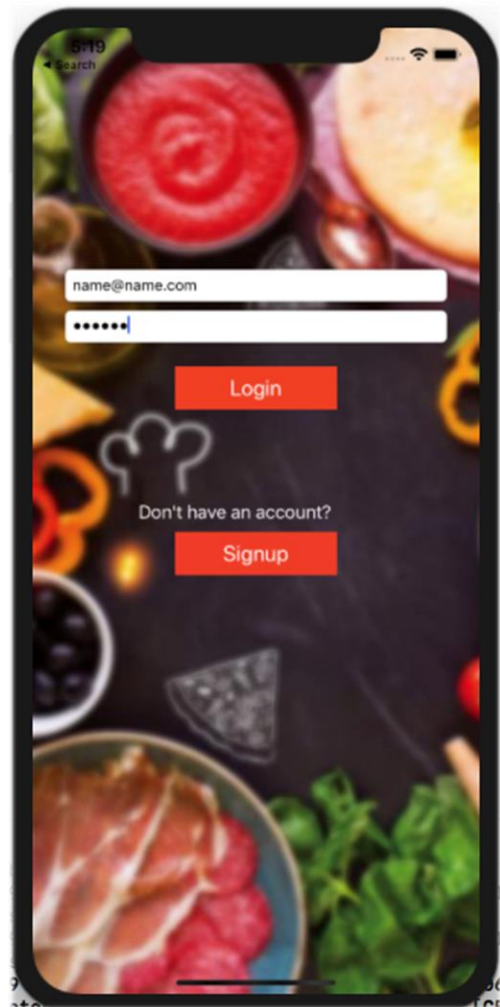
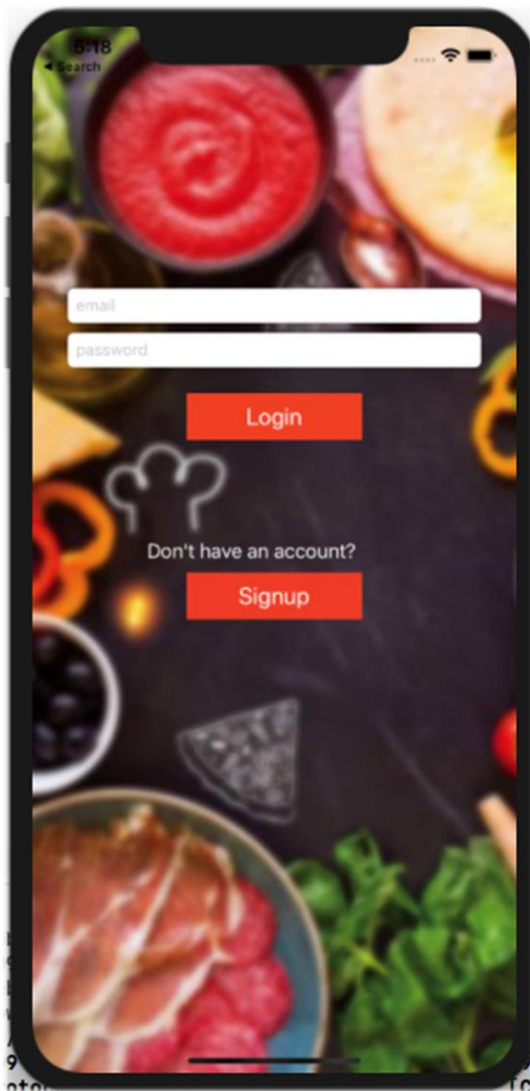
Implementation

Application screenshots

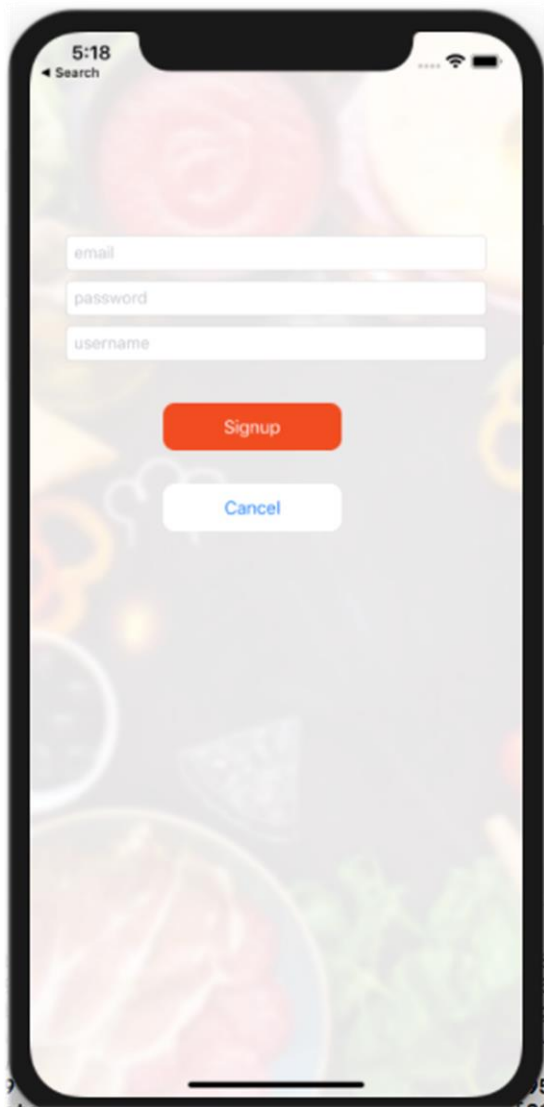
App icon on the home screen



Initial screen for Login or Signup



Screen for Signing up a new user

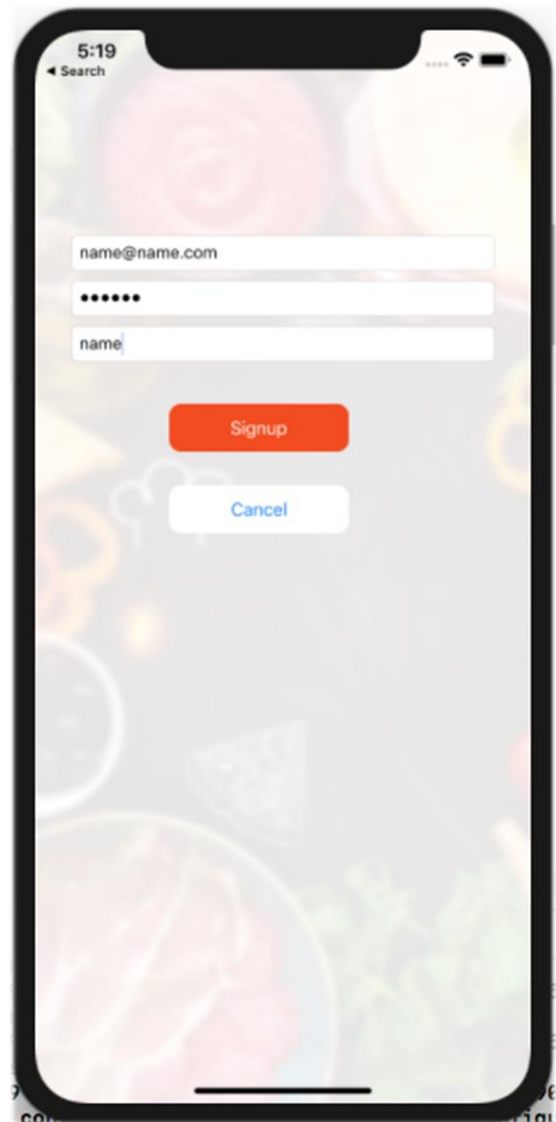


5:18
Search

email
password
username

Signup

Cancel



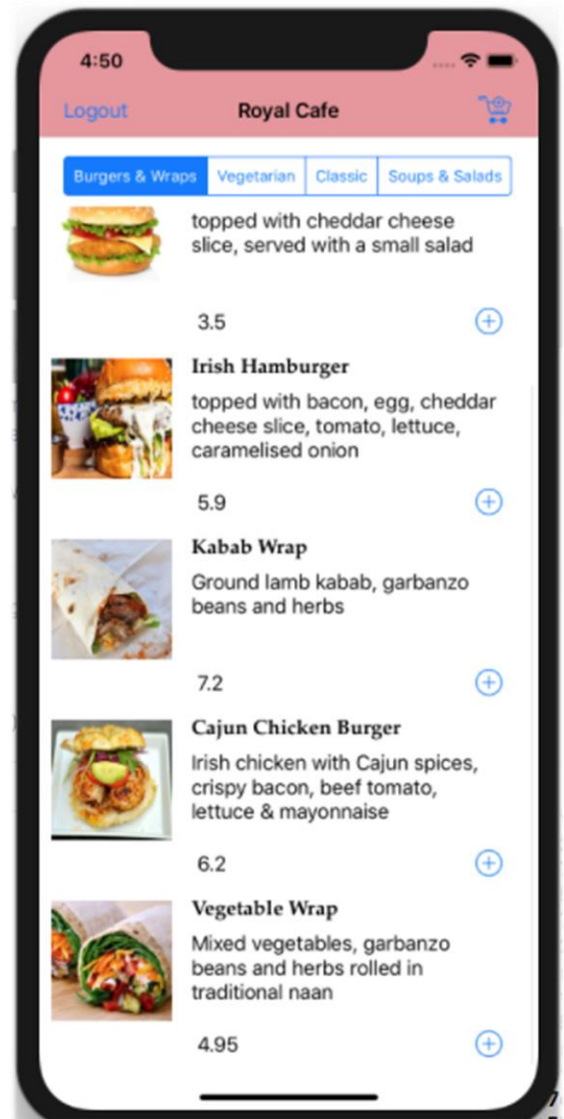
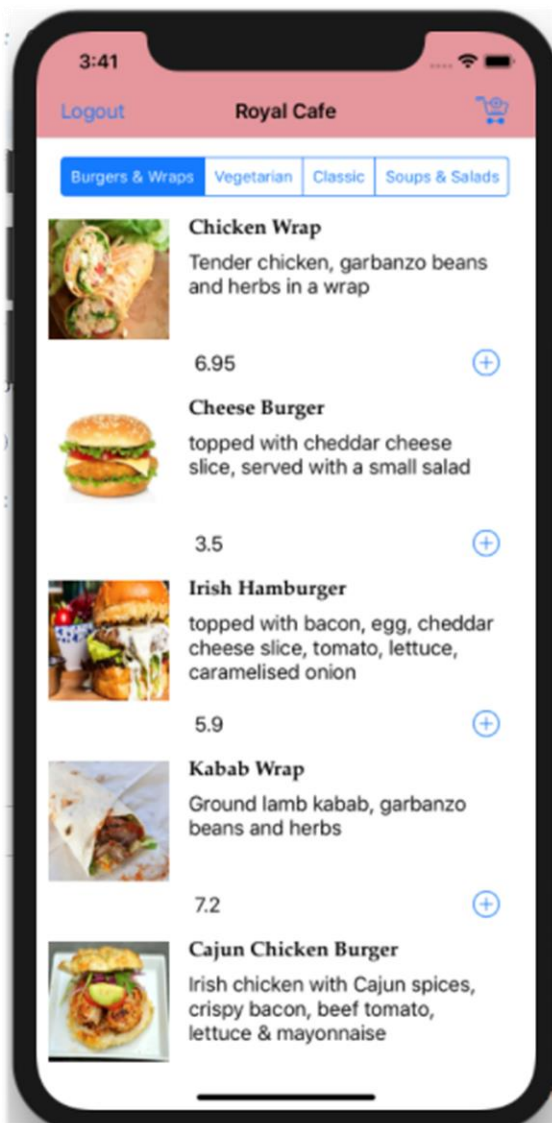
5:19
Search

name@name.com
.....
name

Signup

Cancel

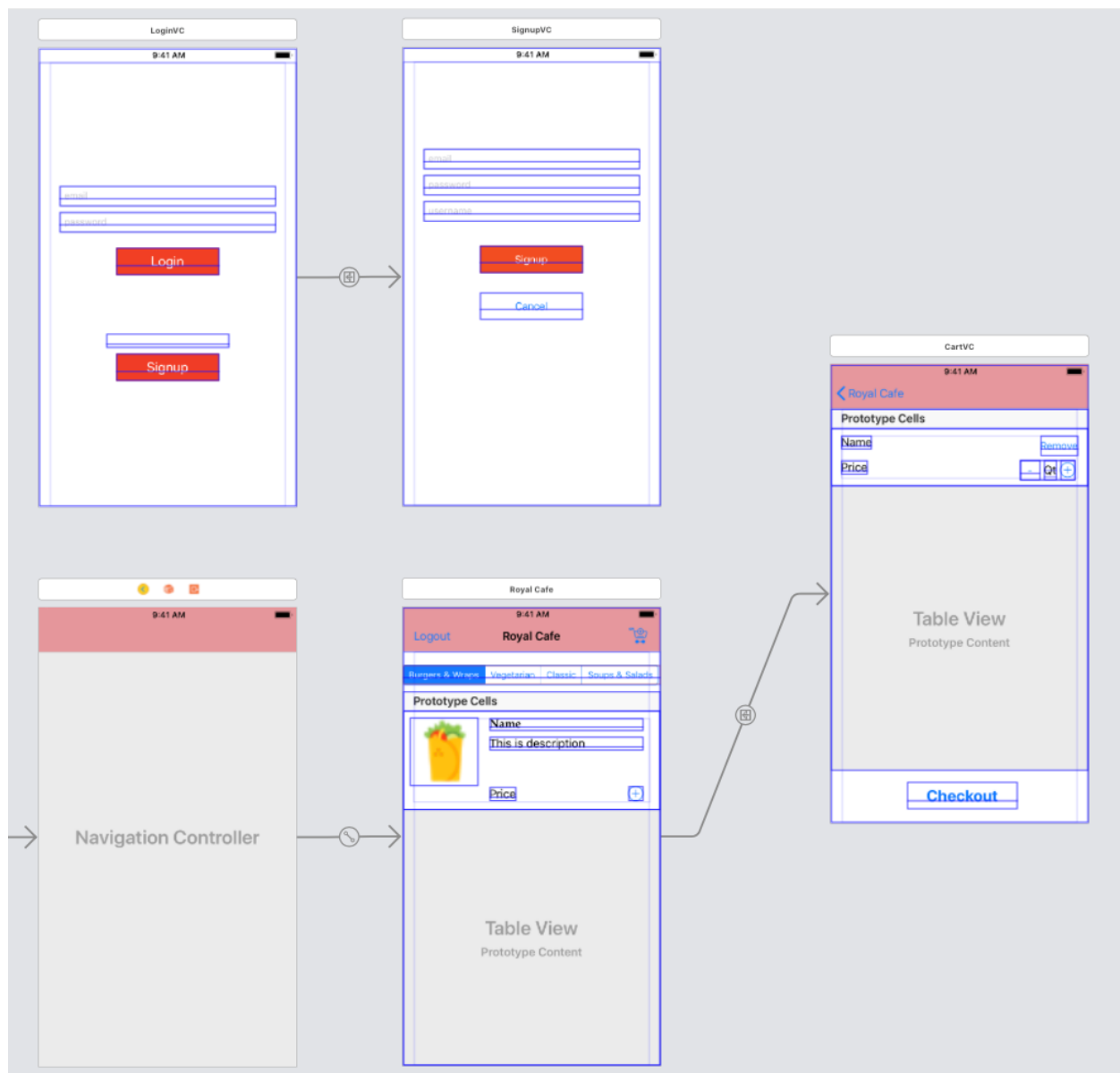
Categorized menu with infinite scrolling



Cart



Storyboard with all the screens in Xcode



Code Snippets

Login functionality with Firebase Auth using email and password entered

```
@IBAction func loginBtnTapped(_ sender: Any) {
    guard let email = emailTF.text,
          let password = passwordTF.text else { return }

    Auth.auth().signIn(withEmail: email, password: password) { (user, error) in
        if let error = error {
            debugPrint("Error signing in: \(error)")
        } else {
            self.dismiss(animated: true, completion: nil)
        }
    }
}
```

Signing up a new user with email, password using Auth and creating a new document in the users collection

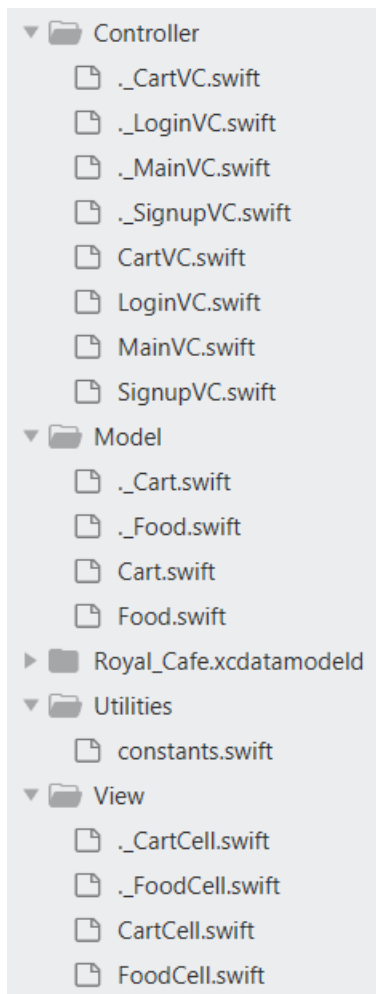
```
@IBAction func signupBtnTapped(_ sender: Any) {
    guard let email = emailTF.text,
          let password = passwordTF.text,
          let username = usernameTF.text else { return }

    Auth.auth().createUser(withEmail: email, password: password) { (user, error) in
        if let error = error {
            debugPrint("Error creating user: \(error.localizedDescription)")
        }

        let changeRequest = user?.user.createProfileChangeRequest()
        changeRequest?.displayName = username
        changeRequest?.commitChanges(completion: { (error) in
            if let error = error {
                debugPrint(error.localizedDescription)
            }
        })

        guard let userId = user?.user.uid else { return }
        Firestore.firestore().collection(USERS_REF).document(userId).setData([
            USERNAME : username ],
            completion: { (error) in
                if let error = error {
                    debugPrint(error.localizedDescription)
                } else {
                    self.dismiss(animated: true, completion: nil)
                }
            })
    })
}
```


MVC framework is used for easy development and modification



Full code is attached in Appendix A

Conclusion

The objective behind this project is to develop an application which will enhance the experience of customers while ordering food by providing them with a dedicated application of a restaurant, which allows them to order their favorite food online. The ability to sign up gives access to personalized features like managing cart and previous order history. This application can be used by any restaurant as all the features will work same for any given restaurant and all the products are stored online which can be modified according to the menu of that place.

The application is developed for iOS devices simply because of the eagerness to learn iOS application development using Swift. Having created apps for android gives an advantage to understand the architecture and application flow easily. Swift is a popular language used for building applications for apple devices and is quite easy from programming aspects. It was not a very difficult task to learn and develop swift application from scratch. I would recommend someone having interest in mobile application development to give it a try.

Database is an essential part for an application like this, so I decided to use AWS, which is undoubtedly very popular cloud service provider. But after learning it and started working on it, I realized it is taking more time. So, I shifted to Firebase which ends up to be very easy to work and provides several services needed for a mobile application like authentication, analytics, NoSQL database, etc. It makes it easy because there is no need to write APIs for server-side, which decreases a lot of work for a programmer.

Working on this project helped me to add another set of technologies in my portfolio including, Swift, Cloud services, Firebase and NoSQL database. The application created with these technologies is working as expected, to which I can say I would use the same tools next time for any similar project, though I will work with AWS to understand where I was stuck.

References

- <https://www.wikipedia.org/>
- <https://stackoverflow.com/>
- <https://www.udemy.com/>
- <https://cloud.google.com/firestore/>
- <https://firebase.google.com/docs/auth>
- <https://cocoapods.org/>
- <https://docs.aws.amazon.com/>
- <https://www.javatpoint.com/swift-tutorial>
- <https://www.tutorialspoint.com/mvc-framework/mvc-framework-introduction.htm>

Appendix A

Full application code

```
// MainVC.swift
// Royal Cafe

import UIKit
import Firebase
enum FoodCategory {
    case burger = "Burgers n Wraps"
    case classic = "Classic Dishes"
    case veg = "Vegetarian Delights"
    case soup = "Soups n Salads"
}
class MainVC: UIViewController, UITableViewDataSource, UITableViewDelegate {

    //outlets
    @IBOutlet weak var categorySegment: UISegmentedControl!
    @IBOutlet weak var tableView: UITableView!
    @IBOutlet weak var logoutBtn: UIBarButtonItem!

    //variables
    private var foods = [Food]()
    private var foodsCollectionRef: CollectionReference!
    private var foodsListener: ListenerRegistration!
    private var selectedCategory = FoodCategory.burger.rawValue
    private var handle: AuthStateDidChangeListenerHandle?

    override func viewDidLoad() {
        super.viewDidLoad()

        tableView.delegate = self
        tableView.dataSource = self
        // tableView.estimatedRowHeight = 150
        // tableView.rowHeight = UITableView.automaticDimension
        tableView.rowHeight = 150

        foodsCollectionRef = Firestore.firestore().collection(FOOD_REF)

        // Do any additional setup after loading the view.
    }

    override func viewWillAppear(_ animated: Bool) {
        //checking if the user is already logged in, else instantiate the login screen
```

```

handle = Auth.auth().addStateDidChangeListener({ (auth, user) in
    if user == nil {
        let storyboard = UIStoryboard(name: "Main", bundle: nil)
        let loginVC = storyboard.instantiateViewController(withIdentifier: "loginVC")
        self.present(loginVC, animated: true, completion: nil)
    } else {
        self.setListener()
    }
})
}

override func viewWillDisappear(_ animated: Bool) {
    if foodsListener != nil {
        foodsListener.remove()
    }
}

@IBAction func categoryChanged(_ sender: Any) {
    switch categorySegment.selectedSegmentIndex {
    case 0:
        selectedCategory = FoodCategory.burger.rawValue
    case 1:
        selectedCategory = FoodCategory.veg.rawValue
    case 2:
        selectedCategory = FoodCategory.classic.rawValue
    default:
        selectedCategory = FoodCategory.soup.rawValue
    }

    foodsListener.remove()
    setListener()
}

func setListener() {
    //retrieving food items from database with categories
    foodsListener = foodsCollectionRef
        .whereField(CATEGORY, isEqualTo: selectedCategory)
        .addSnapshotListener { (snapshot, error) in
            if let err = error {
                debugPrint("Error fetching docs: \(err)")
            } else {
                self.foods.removeAll()
                self.foods = Food.parseData(snapshot: snapshot)
                self.tableView.reloadData()
            }
        }
}

```

```

@IBAction func logoutBtnTapped(_ sender: Any) {
    let firebaseAuth = Auth.auth()
    do {
        try firebaseAuth.signOut()
    } catch let signoutError as NSError {
        debugPrint("Error signing out: \(signoutError)")
    }
}

func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
    //dynamically creating the number of rows in the table by counting the rows returned from
    database
    return foods.count
}

func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    if let cell = tableView.dequeueReusableCell(withIdentifier: "foodCell", for: indexPath) as? FoodCell {

        cell.configureCell(food: foods[indexPath.row])
        return cell
    } else {
        return UITableViewCell()
    }
}
}

```

```

// LoginVCViewController.swift
// Royal Cafe

```

```

import UIKit
import Firebase

```

```

class LoginVC: UIViewController {

    //outlets
    @IBOutlet weak var emailTF: UITextField!
    @IBOutlet weak var passwordTF: UITextField!
    @IBOutlet weak var loginBtn: UIButton!
    @IBOutlet weak var signupBtn: UIButton!

    override func viewDidLoad() {
        super.viewDidLoad()
        let backgroundImage = UIImage.init(named: "home_img")
        let backgroundImageView = UIImageView.init(frame: self.view.frame)
    }
}

```

```

    backgroundImageView.image = backgroundImage
    backgroundImageView.contentMode = .scaleAspectFill
    self.view.insertSubview(backgroundImageView, at: 0)

    // Do any additional setup after loading the view.
}

@IBAction func loginBtnTapped(_ sender: Any) {
    guard let email = emailTF.text,
          let password = passwordTF.text else { return }

    Auth.auth().signIn(withEmail: email, password: password) { (user, error) in
        if let error = error {
            debugPrint("Error signing in: \(error)")
        } else {
            self.dismiss(animated: true, completion: nil)
        }
    }
}
}
}

```

// SignupVC.swift

// Royal Cafe

```

import UIKit
import Firebase

```

```

class SignupVC: UIViewController {

```

```

    //outlets

```

```

    @IBOutlet weak var emailTF: UITextField!
    @IBOutlet weak var passwordTF: UITextField!
    @IBOutlet weak var usernameTF: UITextField!
    @IBOutlet weak var signupBtn: UIButton!
    @IBOutlet weak var cancelBtn: UIButton!

```

```

    override func viewDidLoad() {
        super.viewDidLoad()

```

```

        let backgroundImage = UIImage.init(named: "home_img")
        let backgroundImageView = UIImageView.init(frame: self.view.frame)
        backgroundImageView.image = backgroundImage
        backgroundImageView.contentMode = .scaleAspectFill
        backgroundImageView.alpha = 0.1
        self.view.insertSubview(backgroundImageView, at: 0)

```

```

        signupBtn.layer.cornerRadius = 10

```

```

cancelBtn.layer.cornerRadius = 10

// Do any additional setup after loading the view.
}

@IBAction func signupBtnTapped(_ sender: Any) {
    guard let email = emailTF.text,
          let password = passwordTF.text,
          let username = usernameTF.text else { return }

    Auth.auth().createUser(withEmail: email, password: password) { (user, error) in
        if let error = error {
            debugPrint("Error creating user: \(error.localizedDescription)")
        }

        let changeRequest = user?.user.createProfileChangeRequest()
        changeRequest?.displayName = username
        changeRequest?.commitChanges(completion: { (error) in
            if let error = error {
                debugPrint(error.localizedDescription)
            }
        })

        guard let userId = user?.user.uid else { return }
        Firestore.firestore().collection(USERS_REF).document(userId).setData([
            USERNAME : username ],
            completion: { (error) in
                if let error = error {
                    debugPrint(error.localizedDescription)
                } else {
                    self.dismiss(animated: true, completion: nil)
                }
            })
    })
}

@IBAction func cancelBtnTapped(_ sender: Any) {
    dismiss(animated: true, completion: nil)
}
}

```

```

// CartVC.swift
// Royal Cafe

```

```

import UIKit
import Firebase

```



```
class CartVC: UIViewController, UITableViewDataSource, UITableViewDelegate {

    //outlets
    @IBOutlet weak var tableView: UITableView!
    @IBOutlet weak var checkoutBtn: UIButton!

    //variables
    private var carts = [Cart]()
    private var cartsCollectionRef: CollectionReference!
    private var cartsListener: ListenerRegistration!
    var username: String!

    override func viewDidLoad() {
        super.viewDidLoad()
        tableView.delegate = self
        tableView.dataSource = self
        tableView.rowHeight = 80

        if let name = Auth.auth().currentUser?.displayName {
            username = name
        }
        print(username)
        cartsCollectionRef = Firestore.firestore().collection(CART_REF)

        // Do any additional setup after loading the view.
    }

    override func viewWillAppear(_ animated: Bool) {
        print(username)
        cartsListener = cartsCollectionRef
            .whereField("username", isEqualTo: "name")
            .addSnapshotListener { (snapshot, error) in
                if let err = error {
                    debugPrint("Error fetching docs: \(err)")
                } else {
                    self.carts.removeAll()
                    self.carts = Cart.parseData(snapshot: snapshot)
                    self.tableView.reloadData()
                }
            }
    }

    override func viewWillDisappear(_ animated: Bool) {
        if cartsListener != nil {
            cartsListener.remove()
        }
    }
}
```

```
func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
    return carts.count
}

func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    if let cell = tableView.dequeueReusableCell(withIdentifier: "cartCell", for: indexPath) as? CartCell {

        cell.configureCell(cart: carts[indexPath.row])
        return cell
    } else {
        return UITableViewCell()
    }
}
}
```

// Food.swift

// Royal Cafe

import Foundation

import Firebase

class Food {

//variables

private(set) var foodname: String!

private(set) var fooddesc: String!

private(set) var price: Double!

private(set) var img: String!

private(set) var foodId: String!

init(foodname: String, fooddesc: String, price: Double, img: String, foodId:String) {

self.foodname = foodname

self.fooddesc = fooddesc

self.price = price

self.img = img

self.foodId = foodId

}

class func parseData(snapshot : QuerySnapshot?) -> [Food]{

var foods = [Food]()

guard let snap = snapshot else { return foods }

for document in snap.documents {

let data = document.data()

```

    let name = data[NAME] as? String ?? "Anonymous"
    let desc = data[DESC] as? String ?? "This is description"
    let price = data[PRICE] as? Double ?? 0
    let img = data[IMG] as? String ?? "default"
    let foodId = document.documentID

    let newFood = Food(foodname: name, fooddesc: desc, price: price, img: img, foodId: foodId)
    foods.append(newFood)
  }
  return foods
}
}

```

```

// Cart.swift
// Royal Cafe

```

```

import Foundation
import Firebase

```

```

class Cart{

```

```

    //variables
    private(set) var foodname: String!
    private(set) var foodprice: Double!
    private(set) var foodquan: Int!
    private(set) var cartId: String!

```

```

    init(foodname: String, foodprice: Double, foodquan: Int, cartId: String ) {
        self.foodname = foodname
        self.foodprice = foodprice
        self.foodquan = foodquan
        self.cartId = cartId
    }

```

```

class func parseData(snapshot : QuerySnapshot?) -> [Cart]{
    var carts = [Cart]()
    guard let snap = snapshot else { return carts }
    for document in snap.documents {
        let data = document.data()

```

```

        let name = data[FOODNAME] as? String ?? "Coffee"
        let price = data[PRICE] as? Double ?? 0
        let quan = data[QUANTITY] as? Int ?? 3
        let cartId = document.documentID

```

```

        let newCart = Cart(foodname: name, foodprice: price, foodquan: quan, cartId: cartId)
        carts.append(newCart)
    }
}

```

```
    }  
    return carts  
  }  
}
```

```
// FoodCell.swift  
// Royal Cafe
```

```
import UIKit  
import Firebase
```

```
class FoodCell: UITableViewCell {
```

```
    //outlets  
    @IBOutlet weak var foodimg: UIImageView!  
    @IBOutlet weak var nameLbl: UILabel!  
    @IBOutlet weak var descLbl: UILabel!  
    @IBOutlet weak var priceLbl: UILabel!  
    @IBOutlet weak var addBtn: UIButton!
```

```
    //variables  
    private var food: Food!  
    var username: String!
```

```
    override func awakeFromNib() {  
        super.awakeFromNib()  
        // Initialization code  
    }
```

```
    @IBAction func addToCartTapped(_ sender: Any) {  
        if let name = Auth.auth().currentUser?.displayName {  
            username = name  
        }  
        Firestore.firestore().collection(CART_REF).addDocument(data: [  
            FOODNAME : nameLbl.text as Any,  
            PRICE : priceLbl.text!,  
            QUANTITY : 1,  
            USERNAME : username  
        ]) { (err) in  
            if let err = err {  
                debugPrint("Error adding document: \(err)")  
            } else {  
                //self.navigationController?.popViewController(animated: true)  
            }  
        }  
    }  
}
```

```
func configureCell(food: Food) {  
    self.food = food  
    nameLbl.text = food.foodname  
    descLbl.text = food.fooddesc  
    priceLbl.text = String(food.price)  
    foodimg.image = UIImage(named: food.img)  
}  
}
```

```
// CartCell.swift  
// Royal Cafe
```

```
import UIKit  
import Firebase
```

```
class CartCell: UITableViewCell {
```

```
    //outlets
```

```
    @IBOutlet weak var foodname: UILabel!
```

```
    @IBOutlet weak var foodprice: UILabel!
```

```
    @IBOutlet weak var removeBtn: UIButton!
```

```
    @IBOutlet weak var minusBtn: UIButton!
```

```
    @IBOutlet weak var foodquan: UILabel!
```

```
    @IBOutlet weak var plusBtn: UIButton!
```

```
    //variables
```

```
    private var cart: Cart!
```

```
    var cartRef: DocumentReference!
```

```
    var oldQuan: Int!
```

```
    override func awakeFromNib() {
```

```
        super.awakeFromNib()
```

```
        // Initialization code
```

```
    }
```

```
    func configureCell(cart: Cart) {
```

```
        self.cart = cart
```

```
        foodname.text = cart.foodname
```

```
        foodprice.text = String(cart.foodprice)
```

```
        foodquan.text = String(cart.foodquan)
```

```
    }
```

```
    @IBAction func plusBtnTapped(_ sender: Any) {
```

```
        Firestore.firestore().runTransaction({ (transaction, errorPointer) -> Any? in
```

```

let cartDocument: DocumentSnapshot
do {
    try cartDocument = transaction.getDocument(Firestore.firestore()
        .collection(CART_REF).document(self.cart.cartId))
} catch let error as NSError {
    debugPrint("Fetch error: \(error.localizedDescription)")
    return nil
}
self.oldQuan = cartDocument.data()![QUANTITY] as? Int
transaction.updateData([QUANTITY : self.oldQuan + 1], forDocument: self.cartRef)

return nil
}) { (object, error) in
    if let error = error {
        debugPrint("Transaction failed: \(error)")
    } else {
        self.foodquan.text = String( self.oldQuan + 1)
    }
}
}

@IBAction func minusBtnTapped(_ sender: Any) {

    Firestore.firestore().runTransaction({ (transaction, errorPointer) -> Any? in

        let cartDocument: DocumentSnapshot
        do {
            try cartDocument = transaction.getDocument(Firestore.firestore()
                .collection(CART_REF).document(self.cart.cartId))
        } catch let error as NSError {
            debugPrint("Fetch error: \(error.localizedDescription)")
            return nil
        }

        self.oldQuan = cartDocument.data()![QUANTITY] as? Int

        if(self.oldQuan == 0){
            transaction.updateData([QUANTITY : self.oldQuan - 1], forDocument: self.cartRef)
        }
        else
        {
            transaction.updateData([QUANTITY : 0], forDocument: self.cartRef)
        }

        return nil
    }) { (object, error) in
        if let error = error {
            debugPrint("Transaction failed: \(error)")

```

```
        } else {  
            self.foodquan.text = String(self.oldQuan + 1)  
        }  
    }  
}  
}
```

```
// constants.swift
```

```
// Royal Cafe
```

```
import Foundation
```

```
let FOOD_REF = "allfood"
```

```
let CART_REF = "cart"
```

```
let USERS_REF = "users"
```

```
let CATEGORY = "category"
```

```
let DESC = "desc"
```

```
let IMG = "img"
```

```
let NAME = "name"
```

```
let PRICE = "price"
```

```
let FOODNAME = "food"
```

```
let QUANTITY = "quantity"
```

```
let USERID = "userid"
```

```
let USERNAME = "username"
```
