CommServe Help App

CIS 651- Mobile Application Programming

By Prof. Mina Jung

E&CS, Syracuse University

Author: Amritbani Sondhi, Spring 2018 SUID: 903888517

INTRODUCTION

The Community Service Help App, as the name suggests, is proposed to come to rescue of the people who need help. The areas can be related to getting books, food, or maybe a lift to college. All services will be free and its basic idea is just to offer help to anyone. Maybe for accompanying someone for a trek too, if people do get interested.

The application is designed to target the general audience, any users in the common public can be able to use the features of this application

Usefulness and Uniqueness

There are applications available for different services, such as, LiveSafe, ZimRide etc. LiveSafe app by DPS directly connects with the DPS emergency dispatch center from users' cell phones. Similarly, ZimRide shares the seats in your car or help you catch a ride. But, there are no apps for other services which can help the community as a whole. It can be done by offering your books of the previous semester to a student taking it now, or by sharing leftover food with the homeless. There are innumerable ways in which we can raise help and get connected with someone who can offer help with less efforts required.

TECHNICAL FEATURES

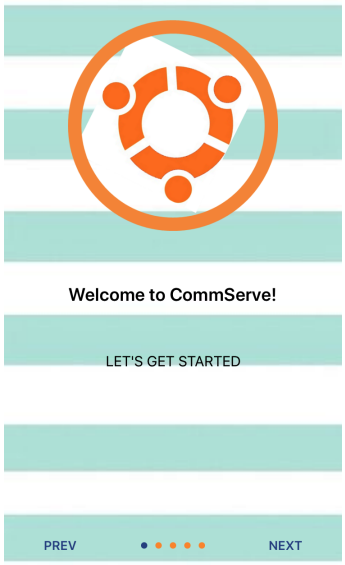
The app follows the MVC Architecture. It includes a number of features, each having their own, Views, Controllers and using different Models.

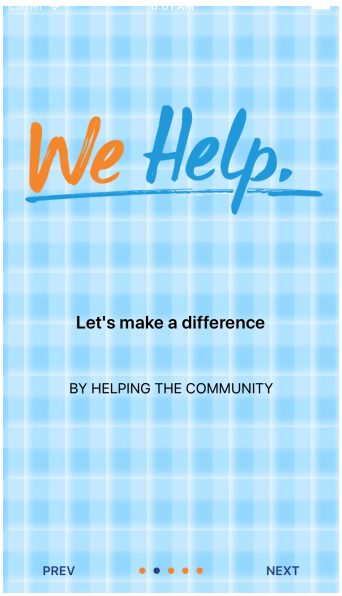
Features Covered

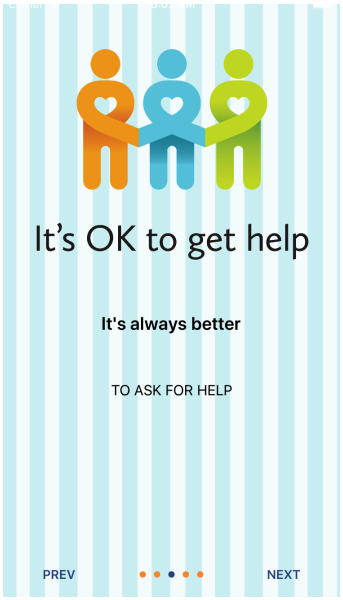
- Custom Views and UIViews
- UITableViews
- UIStackViews
- UICollectionViews and nested UICollectionViews
- Controllers
- Different models
- Camera App
- ImagePicker for Photos App
- Country Picker
- UIMapKit- current location and search bar
- Firebase- Authentication, Database and Storage
- Dropdown
- Alert Box
- Auto Layout and Constraints

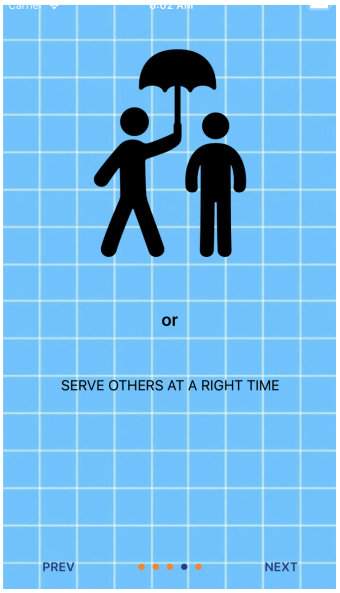
1. WELCOME CONTROLLER AND CELL COLLECTION

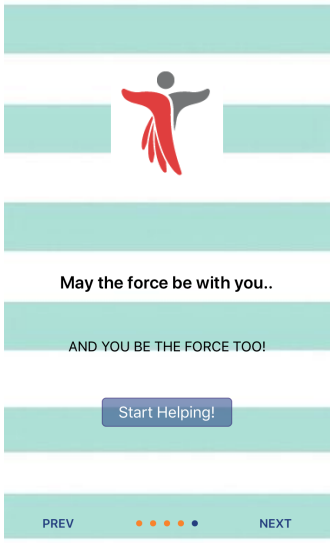
The Welcome screens show the objectives of the app to the User. These screens are only shown once in the lifetime of the app when the user is yet to sign-in. Later, even if he logs out, the welcome screens are not shown.









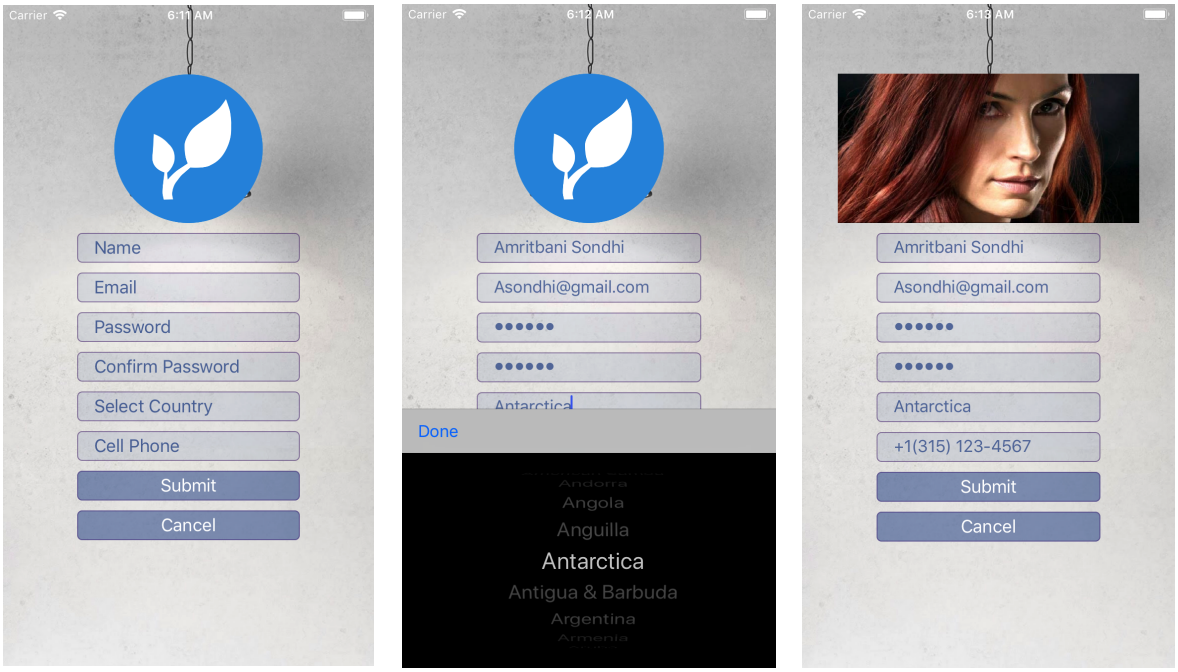


These are created with the help of horizontal collection views. Every view has a different background and paging controls with the previous and next buttons are also provided. It uses views like: UIImageView, UITextView, UIButton with their specific constraints.

```
class WelcomeController: UICollectionViewController,
    UICollectionViewDelegateFlowLayout
{
    var welcomeView : WelcomeViewCollection!

    // DATA
    let pages =
    [
        WelcomePage(imageName: "Community-Logo", header: "Welcome to
            CommServe!", body: "Let's get started"),
        WelcomePage(imageName: "Help2", header: "Let's make a difference", body:
            "By helping the Community"),
        WelcomePage(imageName: "Help3", header: "It's always better", body: "To
            ask for help"),
        WelcomePage(imageName: "help3-copy", header: "or", body: "Serve others
            at a right time"),
        WelcomePage(imageName: "logo2", header: "May the force be with you..",
            body: "And you be the force too!")
    ]
```

2. SIGN UP CONTROLLER AND CELL COLLECTION

This controller is called when the user wants to register for the app. It asks for the fields shown below.



This is a UIViewController, which is inherited from other classes like

ControllerDelegate and DataSource, etc. You can enter texts in your text fields provided, upload a picture from the Photos App, select your country from the CountryPicker and which converts your mobile number in an appropriate format as you enter the digits on the go. It uses an extension 'UITextFieldDelegate' method to convert the phone number format.

```
extension UITextFieldDelegate
{
    func phoneMask(phoneTextField: UITextField, textField: UITextField, _ range: NSRange, _
        string: String) -> (result: Bool, phoneNumber: String, maskPhoneNumber: String)
    {
        let oldString = textField.text!
        let newString = oldString.replacingCharacters(in: Range(range, in: oldString)!, with:
            string)
        //in numString only Numeric characters
        let components = newString.components(separatedBy: CharacterSet.decimalDigits.inverted)
        let numString = components.joined(separator: "")

        let length = numString.count
        let maxCharInPhone = 11

        if newString.count < oldString.count
        { //backspace to work
            if newString.count <= 2
            { //if now "+7(" and push backspace
                phoneTextField.text = ""
                return (false, "", "")
            } else {
                return (true, numString, newString) //will not in the process backspace
            }
        }

        if length > maxCharInPhone
        { // input is complete, do not add characters
            return (false, numString, newString)
        }
        var indexStart, indexEnd: String.Index
        var maskString = "", template = ""
        var endOffset = 0
```

The app uses Firebase authentication for creating a new user and Firebase Database for storing the other information apart from email and password.

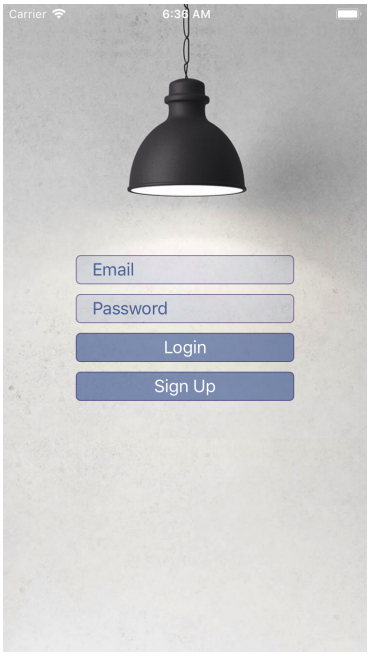
Firebase Authentication:

Identifier	Providers	Created	Signed In	User UID ↑
asondhi@gmail.com	✉	May 6, 2018	May 6, 2018	3zKOLJmT6nUR2ty7hBYKGJ933F...

Firebase Database:

3. LOGIN CONTROLLER AND CELL COLLECTION

When the user is successfully registered, he can enter his email id and password, in the login screen.



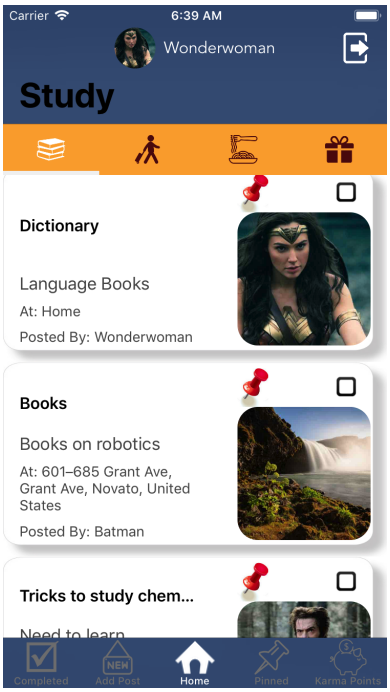
This controller uses Firebase for authenticating the user.

```
func loginPressed()
{
    guard let email = loginView.emailTextField.text else { return }
    guard let password = loginView.passwordTextField.text else { return }

    Auth.auth().signIn(withEmail: email, password: password) { (user, error) in
        if let err = error
        {
            print(err.localizedDescription)
        }
        else
        {
            self.defaults.set(true, forKey: "UserIsLoggedIn")

            // Call Tab Bar
            let customTabController = CustomTabBarController()
            customTabController.tabBar.backgroundColor = UIColor(red:0.19, green:0.32, blue:
                0.57, alpha:0.35)

            self.navigationController?.pushViewController(customTabController, animated:
                true)
        }
    }
}
```

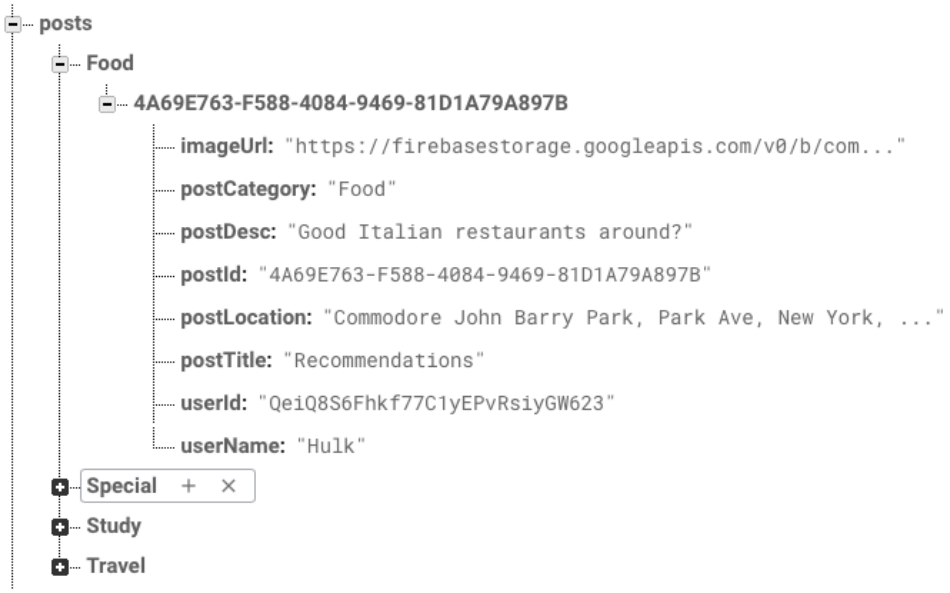
When the user is successfully logged in, the control gets transferred to the CustomTabBarController, to display the home screen.

4. HOME CONTROLLER AND CELL COLLECTION

The Home Screen brings in the NavigationBar, TabBar and the MenuBar. The image and name of the logged in user is displayed on top, and a logout button is present on the top right corner of the screen. The customized tab bar highlights the selected tab with a bright white, and the others are gray. These features are common for all the screens.



The Home Screen displays all the posts posted by the users using the app. There are 4 categories of posts, which can be selected accordingly by clicking on the respective menu bar item. The categories are: Study, Travel, Food and Special. The Posts can be pinned and unpinned and a checkbox is provided, to select if it has been completed. Separate databases are maintained for the 4 categories, and their respective details are stored.



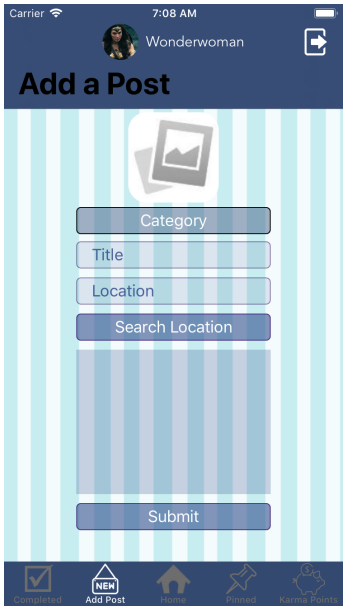
The posts are vertically scrollable collection views. The menu bar is a horizontally scrollable. Together the Home Controller forms a nested collection view structure. The menubar and posts collection views are synchronized so that if the user swipes the posts horizontally, the menubar swipes and highlights the respective category. Similarly, if the user clicks on the respective menu item, the posts collection moves accordingly.

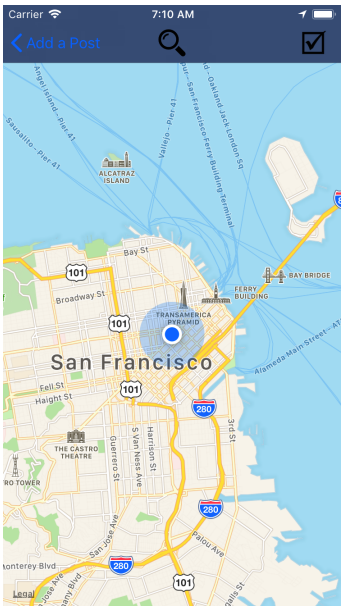
```
collectionView?.register(HomeViewCollection.self, forCellWithReuseIdentifier:
    studyCellId)
collectionView?.register(SpecialCell.self, forCellWithReuseIdentifier: specialCellId)
collectionView?.register(FoodCell.self, forCellWithReuseIdentifier: foodCellId)
collectionView?.register(TravelCell.self, forCellWithReuseIdentifier: travelCellId)


class MenuBar: UIView, UICollectionViewDataSource, UICollectionViewDelegate,
    UICollectionViewDelegateFlowLayout
{
    lazy var collectionView: UICollectionView =
    {
        let layout = UICollectionViewFlowLayout()
        let cv = UICollectionView(frame: .zero, collectionViewLayout: layout)
            cv.backgroundColor = UIColor(red:0.99, green:0.67, blue:0.22, alpha:1.0)
        cv.dataSource = self
        cv.delegate = self
        return cv
    }()

    let cellId = "cellId"
    let imageNames = ["Study", "Travel", "Food", "Special"]
```

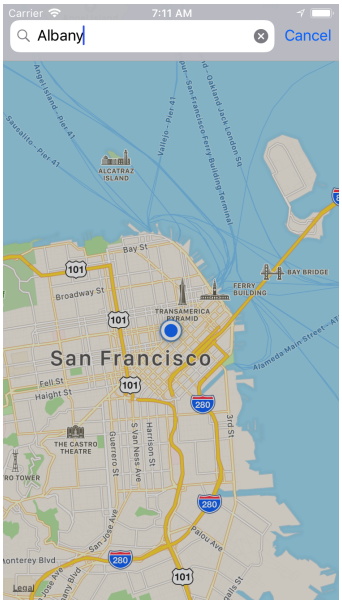
```
// for the white horizontalBar's left anchor when scrolled
override func scrollViewDidScroll(_ scrollView: UIScrollView)
{
    menuBar.horizontalBarLeftAnchorConstraint?.constant = scrollView.contentOffset.x / 4
}

// Synchronize selection with menuBar's collectionView
override func scrollViewWillEndDragging(_ scrollView: UIScrollView, withVelocity velocity:
    CGPoint, targetContentOffset: UnsafeMutablePointer<CGPoint>)
{
    let index = targetContentOffset.pointee.x / view.frame.width

    let indexPath = IndexPath(item: Int(index), section: 0)
    menuBar.collectionView.selectItem(at: indexPath, animated: true, scrollPosition:
        UICollectionViewScrollPosition.centeredHorizontally)

    setTitleForIndex(Int(index))
}

// synchronize collectionView with menuBar's scroll
func scrollToMenuIndex(_ menuIndex: Int)
{
    let indexPath = IndexPath(item: menuIndex, section: 0)
    collectionView?.scrollToItem(at: indexPath, at:
        UICollectionViewScrollPosition.centeredHorizontally, animated: true)

    setTitleForIndex(menuIndex)
}
```

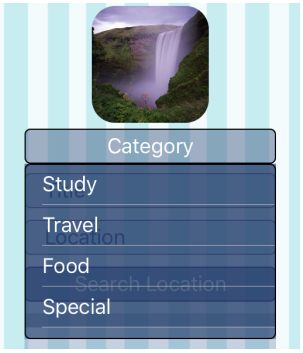
5. ADD POST CONTROLLER AND CELL COLLECTION

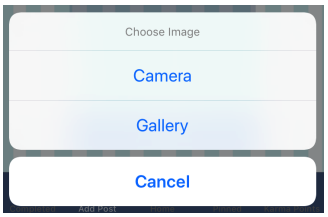
For creating a new post, the user should click the add post tab bar item. It gives the user, the following options to write about the post. The user can upload an image for a post, for which, it asks by a pop up to upload through a camera or the photos app. If the camera app is not present in the device then it gives an alert box.

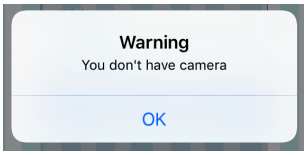












For selecting the category, a dropdown is listed with the available categories. This is created by using a button and a table view. For selecting the location, the user can click the search location button which brings in the mapkit view. Before determining the user's location, an alert is popped up for the user to allow access for the location permissions. If the user agrees, the current location is automatically shown. If the user wants to search any other location

then it can click on the search icon on the navigation bar, which locates the new entered place. The user can either select done (top right), or go back to type the location in the text field. He can provide a brief description about the post in the next text field. As soon as the user clicks on the Submit button, the post can be seen in the appropriate category in the Home tab.

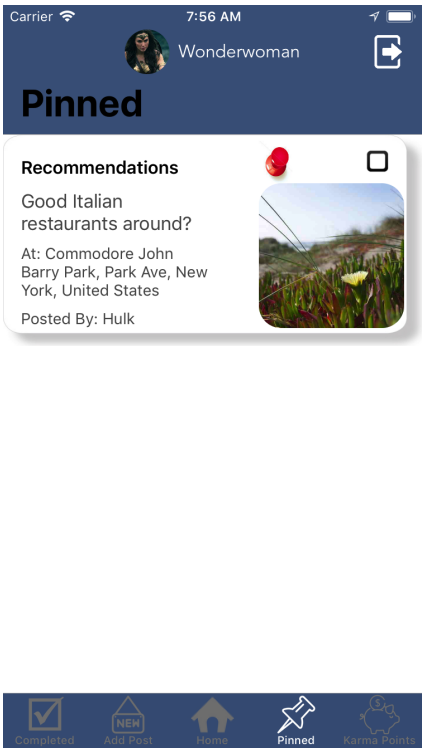
```
func openCamera()
{
    print("upload Image through Camera Pressed!")

        if(UIImagePickerController .isSourceTypeAvailable(UIImagePickerControllerSourceType.
        camera))
    {
        let picker = UIImagePickerController()
        picker.delegate = self
        picker.allowsEditing = true
        picker.sourceType = .photoLibrary

        // show Image Picker!!!! (Modally)
        present(picker, animated: true, completion: nil)
    }
    else
    {
        let alert  = UIAlertController(title: "Warning", message: "You don't have camera",
            preferredStyle: .alert)
        alert.addAction(UIAlertAction(title: "OK", style: .default, handler: nil))
        self.present(alert, animated: true, completion: nil)
    }
}
```

```
func uploadImagePressed()
{
    print("upload Image through Photos App Pressed!")

    let picker = UIImagePickerController()

    picker.delegate = self
    picker.allowsEditing = true
    picker.sourceType = .photoLibrary

    // show Image Picker!!!! (Modally)
    present(picker, animated: true, completion: nil)
}
```

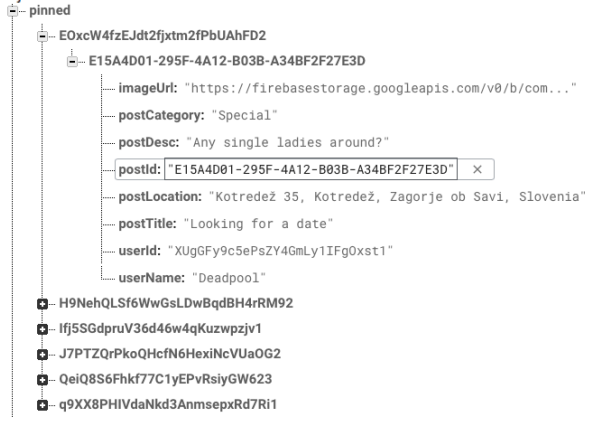
```
func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath)
{
    self.delegate.dropDownPressed(string: dropDownOptions[indexPath.row])
    self.tableView.deselectRow(at: indexPath, animated: true)
}

//Configure the button
button = dropDownBtn.init(frame: CGRect(x: 0, y: 0, width: 0, height: 0))
button.setTitle("Category", for: .normal)
button.translatesAutoresizingMaskIntoConstraints = false

//Set the drop down menu's options
button.dropView.dropDownOptions = ["Study", "Travel", "Food", "Special"]
```

6. PINNED CONTROLLER AND CELL COLLECTION

There are unpinned-pins available on the top right of each post. If the user pins any post, then that post is also available in the pin tab. The pinned posts are user specific. A separate database is maintained for this purpose.



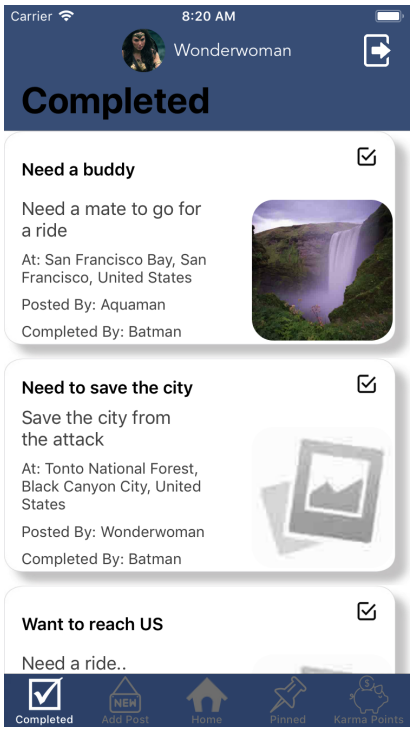

```
    let favReference = favRef.child(tempUserId).child(postId)
    favReference.updateChildValues(values, withCompletionBlock: { (err, ref) in
        if err != nil {
            print(err ?? "")
            return
        }
    })
}
else
{
    print("Button is Selected") // purpose is to unpin the post
    sender.isSelected = true

    // Reflect in the database
    let tempPostId = String(self.postArray[indexRow].postId!)

    favRef.child(tempUserId).child(tempPostId).removeValue()
    self.tableView.reloadData()
}
```

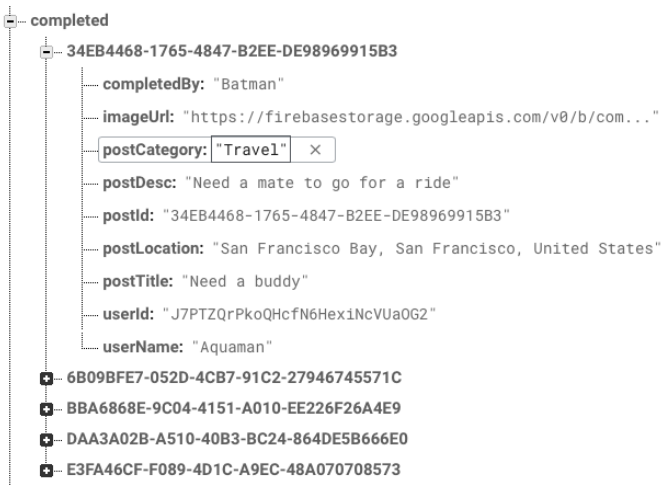


Unlike the Home tab, this is created using the UITableViews. The formatting is done in a way, that it's difficult to make out the difference.

7. CHECKED CONTROLLER AND CELL COLLECTION

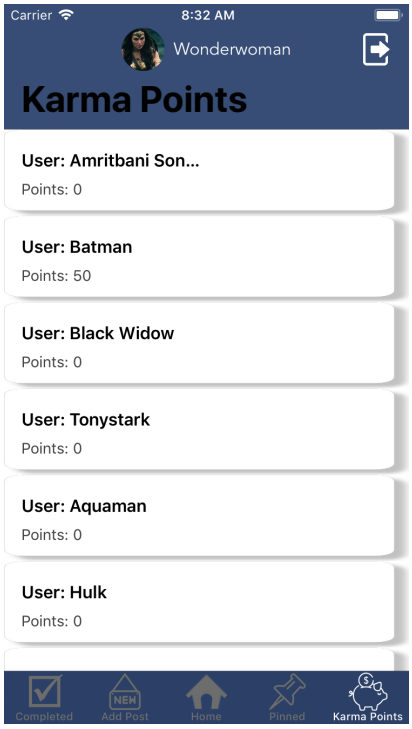
Like the pins, there are check boxes available too on top of each posts. These are provided to mark a task as done. On check, the post is deleted from the 'posts' database and added to 'completed' database. When a task is completed, the username of who completed the task is also added with the post information. Also, as the user has helped someone, 50 karma points are added to his account. The checked tab, recognizes the tasks completed and the people who posted and completed them.





Like the Pin Tab, the Checked Tab is also created using UITableViews and is scrollable vertically.

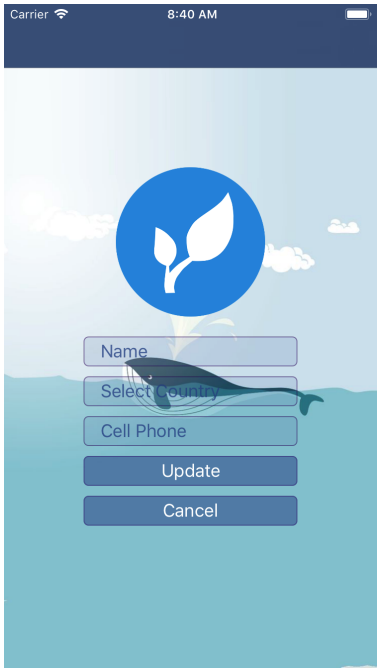
8. KARMA CONTROLLER AND CELL COLLECTION




```
point = point + 50

let value = ["karmaPoints": point]

usersRef.child(tempUserId).updateChildValues(value, withCompletionBlock: { (err,
    ref) in

    if err != nil {
        print(err ?? "")
        return
    }
})
```

This Tab recognizes the overall karma points of each user. Users can take it in a competitive way and help more people to increase their score. It is created using UITableViews

9. USER INFO CONTROLLER AND CELL COLLECTION



On clicking the user's profile picture or name, the user can edit and update their details. The Country text field provides with a list of countries to pick from.

10. OTHER DETAILS

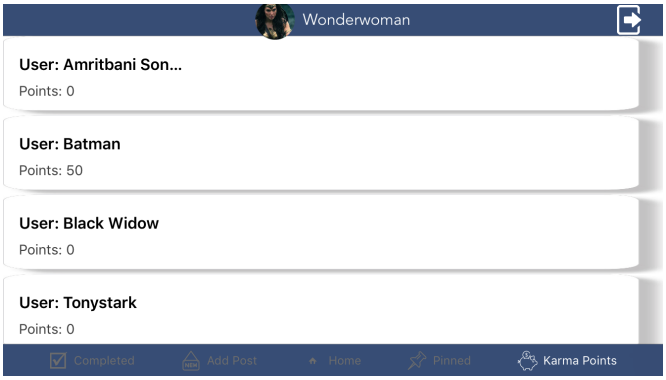
Session is maintained of the users when they are logging in. So, even if they close the app and come back, they don't have to log in again.

Users can logout from the app anytime by clicking on the logout button available at top right corner on the navigation bar. This is available on any screen the user is present on.

Also, the specific tab or menu item names are displayed on the Navigation Bar for the apps intuitiveness.

Constraints are set for mostly all the elements for it to function properly on any device and when the screen is rotated.

Constraints:




```
view.addSubview(bgView)
view.addConstraintsWithFormat(format: "H:|[v0]|", views: bgView)
view.addConstraintsWithFormat(format: "V:[v0(50)]", views: bgView)

view.addSubview(menuBar)
view.addConstraintsWithFormat(format: "H:|[v0]|", views: menuBar)
view.addConstraintsWithFormat(format: "V:[v0(50)]", views: menuBar)
```

Navigation Bar:

```
private func setupNavBarItems()
{
    guard let username = appUser?.username else { return }
    print(username)

    let titleView = UIButton()
    titleView.frame = CGRect(x:0, y:0, width:100, height: 40)

    let containerView = UIView()
    containerView.translatesAutoresizingMaskIntoConstraints = false
    titleView.addSubview(containerView)

    let profileImageView = UIImageView()
    profileImageView.translatesAutoresizingMaskIntoConstraints = false
    profileImageView.contentMode = .scaleAspectFill
    profileImageView.layer.cornerRadius = 20
    profileImageView.clipsToBounds = true

    if let profileImageUrl = appUser?.profileurl
    {

            profileImageView.loadImageUsingCacheWithUrlString(profileIma
            geUrl)
    }

    containerView.addSubview(profileImageView)
```

User Session:

```
userIsLoggedIn = defaults.bool(forKey: "UserIsLoggedIn")
```

```
let defaults = UserDefaults.standard
var userIsLoggedIn: Bool?
```

```
if userIsLoggedIn == true
{
    window?.rootViewController = CustomTabBarController()
}
else
{
    // Swiping Welcome CollectionView
    let layout = UICollectionViewFlowLayout()
    // for swiping!!!
    layout.scrollDirection = .horizontal
    let swipingController = WelcomeController(collectionViewLayout: layout)
    let NavSwipe = UINavigationController(rootViewController: swipingController)

    window?.rootViewController = NavSwipe
}
```

Logout:

```
@objc func handleLogout()
{
    do{
        try Auth.auth().signOut()
        defaults.set(false, forKey: "UserIsLoggedIn")
        let loginController = UINavigationController(rootViewController:
            LoginController())
        present(loginController, animated: true, completion: nil)
    }
    catch let err {
        print(err.localizedDescription)
    }
}
```

REFERENCES

- https://developer.apple.com/documentation/uikit/
- https://firebase.google.com/docs/
- https://www.youtube.com/channel/UCuP2vJ6kRutQBfRmdcI92mA
- https://www.youtube.com/user/Archetapp