// AppDelegate

```
import UIKit
import Firebase

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?


    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {

        FirebaseApp.configure()

        return true
    }

    func applicationWillResignActive(_ application: UIApplication) {
        // Sent when the application is about to move from active to inactive state. This can occur for certain types of temporary interruptions (such as an incoming phone call or SMS message) or when the user quits the application and it begins the transition to the background state.
        // Use this method to pause ongoing tasks, disable timers, and invalidate graphics rendering callbacks. Games should use this method to pause the game.
    }

    func applicationDidEnterBackground(_ application: UIApplication) {
        // Use this method to release shared resources, save user data, invalidate timers, and store enough application state information to restore your application to its current state in case it is terminated later.
        // If your application supports background execution, this method is called instead of applicationWillTerminate: when the user quits.
    }

    func applicationWillEnterForeground(_ application: UIApplication) {
        // Called as part of the transition from the background to the active state; here you can undo many of the changes made on entering the background.
    }

    func applicationDidBecomeActive(_ application: UIApplication) {
        // Restart any tasks that were paused (or not yet started) while the application was inactive. If the application was previously in the background, optionally refresh the user interface.
    }

    func applicationWillTerminate(_ application: UIApplication) {
        // Called when the application is about to terminate. Save data if appropriate. See also applicationDidEnterBackground:.
    }
```

// LoginViewController

```
import Foundation
import UIKit

class LoginViewController : UIViewController {

    @IBOutlet weak var userNameTextField :UITextField!

    override func viewDidLoad() {
        super.viewDidLoad()
    }

    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

        let userName = self.userNameTextField.text

        let chatRoomsTVC = segue.destination as! ChatRoomsTableViewController
        chatRoomsTVC.userName = userName
    }

}
```

// UIImage extension

```
import Foundation
import UIKit

extension UIImage {

    func resizeImage(newWidth :CGFloat) -> UIImage {

        let scale = newWidth / self.size.width
        let newHeight = self.size.height * scale

        UIGraphicsBeginImageContext(CGSize(width: newWidth, height: newHeight))

        self.draw(in: CGRect(x: 0, y: 0, width: newWidth, height: newHeight))
        let newImage = UIGraphicsGetImageFromCurrentImageContext()
        UIGraphicsEndImageContext()
        return newImage!

    }

}
```

// chatrooms table view controller

```
import Foundation
import UIKit

class ChatRoomsTableViewController : UITableViewController {

    let chatRooms = ["iOS","Android",".NET","JavaScript"]
    var userName :String!

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

    override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {

        let cell = tableView.dequeueReusableCell(withIdentifier: "Cell", for: indexPath)
        cell.textLabel?.text = chatRooms[indexPath.row]
        return cell
    }

    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

        guard let indexPath = self.tableView.indexPathForSelectedRow else {
            return
        }

        let chatRoom = self.chatRooms[indexPath.row]

        let chatVC = segue.destination as! ChatViewController
        chatVC.userName = self.userName
        chatVC.chatRoom = chatRoom

    }

}
```

// chatViewController

```
import Foundation
import UIKit
import JSQMessagesViewController
import Firebase

class ChatViewController : JSQMessagesViewController, UIImagePickerControllerDelegate, UINavigationControllerDelegate {

    var userName :String!
    var chatRoom :String!

    private var chatRoomRef :DatabaseReference!
    private var storage :StorageReference!

    private var photoMessageMap = [String:JSQPhotoMediaItem]()

    private var messages :[JSQMessage] = [JSQMessage]()

    private var imagePicker :UIImagePickerController!

    lazy var outgoingBubbleImageView :JSQMessagesBubbleImage = self.setupOutgoingBubble()
    lazy var incomingBubbleImageView :JSQMessagesBubbleImage = self.setupIncomingBubble()

    private func setupOutgoingBubble() -> JSQMessagesBubbleImage {
        let bubbleImageFactory = JSQMessagesBubbleImageFactory()
        return bubbleImageFactory!.outgoingMessagesBubbleImage(with: UIColor.jsq_messageBubbleBlue())
    }

    private func setupIncomingBubble() -> JSQMessagesBubbleImage {

        let bubbleImageFactory = JSQMessagesBubbleImageFactory()
        return bubbleImageFactory!.incomingMessagesBubbleImage(with: UIColor.jsq_messageBubbleLightGray())
    }

    override func viewDidLoad() {
        super.viewDidLoad()

        self.title = self.chatRoom
        self.senderId = self.userName
        self.senderDisplayName = self.userName

        self.storage = Storage.storage().reference(forURL: "gs://whatsup-281aa.appspot.com")
        self.chatRoomRef = Database.database().reference(withPath: self.chatRoom)

        populateMessages()
        setupObserverForPhotoMessage()
    }

    private func setupObserverForPhotoMessage() {

        self.chatRoomRef.observe(.childChanged) { (snapshot) in

            let messageDictionary = snapshot.value as! [String:Any]
            let imageURL = messageDictionary["imageURL"] as! String
            let senderId = messageDictionary["senderId"] as! String

            self.fetchImageDataAtURL(imageURL: imageURL, key: snapshot.key, senderId: senderId)

        }

    }
```

```
private func addPhotoMessage(id :String, key :String, mediaItem :JSQPhotoMediaItem) {

    if let message = JSQMessage(senderId: id, displayName: self.senderDisplayName, media: mediaItem) {

        self.messages.append(message)

        if mediaItem.image == nil {
            self.photoMessageMap[key] = mediaItem
        }

        collectionView.reloadData()
    }
}

private func fetchImageDataAtURL(imageURL :String, key :String, mediaItem :JSQPhotoMediaItem? = nil, senderId :String) {

    let storageRef = Storage.storage().reference(forURL: imageURL)

    storageRef.getData(maxSize: INT64_MAX) { (data, error) in

        if let error = error {
            print(error.localizedDescription)
            return
        }

        if mediaItem == nil && senderId != self.senderId {
            let img = UIImage(data: data!)

            self.addPhotoMessage(id: senderId, key: key, mediaItem: JSQPhotoMediaItem(image: img))

        } else {
            if let img = UIImage(data :data!) {
                mediaItem?.image = img
                self.photoMessageMap.removeValue(forKey: key)
            }
        }

        DispatchQueue.main.async {
            self.collectionView.reloadData()
        }

    }

}
```

```
private func populateMessages() {

    self.chatRoomRef.observe(.childAdded) { snapshot in

        let messageDictionary = snapshot.value as? [String:Any] ?? [:]

        guard let senderId = messageDictionary["senderId"] as? String,
            let senderDisplayName = messageDictionary["senderDisplayName"] as? String else {
                return
        }

        let text = messageDictionary["text"] as? String
        let imageURL = messageDictionary["imageURL", default :"NOIMAGE"] as? String

        if text != nil {
            self.addMessage(senderId: senderId, senderDisplayName: senderDisplayName, text: text!)
        } else if imageURL != "NOIMAGE" {

            if let mediaItem = JSQPhotoMediaItem(maskAsOutgoing: senderId == self.senderId) {

                self.addPhotoMessage(id :senderId, key: snapshot.key, mediaItem :mediaItem)
                self.fetchImageDataAtURL(imageURL: imageURL!, key: snapshot.key, mediaItem :mediaItem, senderId: senderId)

            }

        }
    }

}

private func addMessage(senderId :String, senderDisplayName :String, text :String) {

    if let message = JSQMessage(senderId: senderId, displayName: senderDisplayName, text: text) {
        self.messages.append(message)

        DispatchQueue.main.async {
            self.collectionView.reloadData()
        }
    }

}

override func collectionView(_ collectionView: JSQMessagesCollectionView!, avatarImageDataForItemAt indexPath: IndexPath!) -> JSQMessageAvatarImageDataSource! {
    return nil
}

override func collectionView(_ collectionView: JSQMessagesCollectionView!, messageBubbleImageDataForItemAt indexPath: IndexPath!) -> JSQMessageBubbleImageDataSource! {

    let message = self.messages[indexPath.row]
    if message.senderId == self.senderId {
        return setupOutgoingBubble()
    } else {
        return setupIncomingBubble()
    }

}

override func collectionView(_ collectionView: JSQMessagesCollectionView!, messageDataForItemAt indexPath: IndexPath!) -> JSQMessageData! {
    return self.messages[indexPath.item]
}
```

```
override func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {
    return self.messages.count
}

override func didPressAccessoryButton(_ sender: UIButton!) {

    self.imagePicker = UIImagePickerController()
    self.imagePicker.delegate = self

    if UIImagePickerController.isSourceTypeAvailable(.camera) {
        self.imagePicker.sourceType = .camera
    } else {
        self.imagePicker.sourceType = .photoLibrary
    }

    self.present(self.imagePicker, animated: true, completion: nil)

}

private func uploadPhoto(img :UIImage, key :String) {

    let meta = StorageMetadata()
    meta.contentType = "image/png"

    if let imageData = UIImagePNGRepresentation(img) {

        let storageRef = self.storage.child(UUID().uuidString)
        storageRef.putData(imageData, metadata: meta) { (meta, error) in

            if let error = error {
                print(error.localizedDescription)
            }

            guard let meta = meta,
                let downloadURL = meta.downloadURL() else {
                    fatalError("Error uploading media!")
            }

            let messageRef = self.chatRoomRef.child(key)
            messageRef.updateChildValues(["imageURL": downloadURL.absoluteString])

            print(downloadURL.absoluteString)

        }
    }

}

private func sendPhotoMessage() -> String {

    let messageData = [
        "senderId" : self.senderId,
        "senderDisplayName" : self.senderDisplayName,
        "imageURL" : "NOIMAGE"
    ]

    let messageRef = self.chatRoomRef.childByAutoId()
    messageRef.setValue(messageData)
    return messageRef.key

}

func imagePickerController(_ picker: UIImagePickerController, didFinishPickingMediaWithInfo info: [String : Any]) {

    picker.dismiss(animated: true, completion: nil)

    if let originalImage = info[UIImagePickerControllerOriginalImage] as? UIImage {

        // save photo message to firebase database
        let key = sendPhotoMessage()

        let resizedImage = originalImage.resizeImage(newWidth: 200)
        self.uploadPhoto(img :resizedImage, key :key)

        let mediaItem = JSQPhotoMediaItem(image: originalImage)

        if let message = JSQMessage(senderId: self.senderId, displayName: self.senderDisplayName, media: mediaItem!) {

            self.messages.append(message)

            DispatchQueue.main.async {
                self.collectionView.reloadData()
            }

        }

    }

}


override func didPressSend(_ button: UIButton!, withMessageText text: String!, senderId: String!, senderDisplayName: String!, date: Date!) {

    let messageData = [
        "senderId" : self.senderId,
        "senderDisplayName" : self.senderDisplayName,
        "text" : text
    ]

    let chatMessageRef = self.chatRoomRef.childByAutoId()
    chatMessageRef.setValue(messageData)
    finishSendingMessage()
}

}
```

↳ part of the JSQ

/* let message = JSQMessage(senderId: self.senderId, displayName: self.senderDisplay
, text: text)
self.messages.append(message!)
finishSendingMessage()
*/DispatchQueue.main.async {
self.collectionView.reloadData()