```
import Foundation
import CoreLocation
import Firebase

class DataManager: ObservableObject {
    @Published var accounts: [Account] = []
    @Published var users: [User] = []
    @Published var jobPostings: [JobPosting] = []
    @Published var geocodingResult: Result<CLLocationCoordinate2D, Error>?
    @Published var geocodedJobPostings: [GeocodedJobPosting] = []

    private let geocoder = CLGeocoder()
    private var db = Firestore.firestore()

    init() {
        fetchUsers()
    }


    func addUser(userProfession: String, usersName: String, emailAdd:
     String) {
            let id = UUID().uuidString
            let ref = db.collection("Users").document(id)
            ref.setData(["name": usersName, "profession": userProfession,
             "id": id, "email": emailAdd, "userId":
             Auth.auth().currentUser?.uid ?? ""]) { error in
                if let error = error {
                    print("Error adding user:", error.localizedDescription)
                }
            }
        }

    func fetchUsers() {
            accounts.removeAll()
            let db = Firestore.firestore()
            let ref = db.collection("Users")
            ref.getDocuments { snapshot, error in
                guard error == nil else {
                    print("Error fetching users:",
                     error!.localizedDescription)
                    return
                }
                if let snapshot = snapshot {
                    for document in snapshot.documents {
                        let data = document.data()

                        let id = data["id"] as? String ?? ""
                        let profession = data["profession"] as? String ?? ""
                        let name = data["name"] as? String ?? ""
                        let email = data["email"] as? String ?? ""
                        let userId = data["userId"] as? String ?? ""

                        let account = Account(id: id, userId: userId, name:
                         name, profession: profession, email: email)
                        self.accounts.append(account)
```

```
                    }
                    print("Fetched \(self.accounts.count) users")
                }
            }
        }

//    func fetchJobPosts() {
//        accounts.removeAll()
//        let db = Firestore.firestore()
//        let ref = db.collection("Jobs")
//        print("Number of jobs found: \(jobPostings.count)")
//        ref.getDocuments { snapshot, error in
//            guard error == nil else {
//                print(error!.localizedDescription)
//                return
//            }
//            if let snapshot = snapshot {
//                for document in snapshot.documents {
//                    let data = document.data()
//
//                    let id = data["id"] as? String ?? ""
//                    let userID = data["userID"] as? String ?? ""
//                    let companyName = data["companyName"] as? String ?? ""
//                    let jobDescription = data["jobDescription"] as?
 String ?? ""
//                    let postcode = data["postcode"] as? String ?? ""
//                    // Coordinates should be a dictionary, not a string
//                    let coordinates = data["coordinates"] as? [String:
 Any] ?? [:]
//                    // Latitude and longitude should be Double, not String
//                    let latitude = coordinates["latitude"] as? Double ??
 0.0
//                    let longitude = coordinates["longitude"] as? Double
 ?? 0.0
//                    let location = CLLocation(latitude: latitude,
 longitude: longitude)
//
//                    let jobPosting = JobPosting(id: id, userID: userID,
 companyName: companyName, jobDescription: jobDescription, postcode:
 postcode)
//                    self.jobPostings.append(jobPosting)
//
//                }
//            }
//        }
//    }


    func fetchUsersByUserId(userId: String, completion: @escaping
     ([Account]) -> Void) {
        accounts.removeAll()
        let db = Firestore.firestore()
        let ref = db.collection("Users").whereField("userId", isEqualTo:
         userId)
        ref.getDocuments { snapshot, error in
```

```
                guard error == nil else {
                    print("Error fetching users by userID:",
                     error!.localizedDescription)
                    completion([])
                    return
                }
                if let snapshot = snapshot {
                    var userAccounts: [Account] = []
                    for document in snapshot.documents {
                        let data = document.data()

                        let id = data["userId"] as? String ?? ""
                        let profession = data["profession"] as? String
                         ?? ""
                        let name = data["name"] as? String ?? ""
                        let email = data["email"] as? String ?? ""

                        let account = Account(id: id, userId: userId,
                         name: name, profession: profession, email:
                         email)
                        userAccounts.append(account)
                    }
                    print("Fetched \(userAccounts.count) users by
                     userID")
                    completion(userAccounts)
                }
            }
        }
func fetchUserData(userId: String, completion: @escaping (Account?) ->
 Void) {
    let db = Firestore.firestore()
    let ref = db.collection("Users").document(userId)
    ref.getDocument { document, error in
        guard error == nil else {
            print(error!.localizedDescription)
            completion(nil)
            return
        }
        if let document = document, document.exists {
            let data = document.data()

            let id = data?["id"] as? String ?? ""
            let name = data?["name"] as? String ?? ""
            let profession = data?["profession"] as? String ?? ""
            let email = data?["email"] as? String ?? ""
            let accounts = Account(id: id, userId: userId, name: name,
             profession: profession, email: email)
            completion(accounts)
        } else {
            completion(nil)
        }
    }
}
func fetchJobs() {
        jobPostings.removeAll()
```

```
        let ref = db.collection("Jobs")
        ref.getDocuments { snapshot, error in
            guard error == nil else {
                print("Error fetching jobs:",
                 error!.localizedDescription)
                return
            }
            if let snapshot = snapshot {
                for document in snapshot.documents {
                    let data = document.data()

                    let id = data["id"] as? String ?? ""
                    let userID = data["userID"] as? String ?? ""
                    let companyName = data["companyName"] as? String ??
                     ""
                    let jobDescription = data["jobDescription"] as?
                     String ?? ""
                    let postcode = data["postcode"] as? String ?? ""

                    // Coordinates should be a dictionary, not a string
                    let coordinates = data["coordinates"] as? [String:
                     Any] ?? [:]
                    // Latitude and longitude should be Double, not
                     String
                    let latitude = coordinates["latitude"] as? Double
                     ?? 0.0
                    let longitude = coordinates["longitude"] as? Double
                     ?? 0.0
                    let location = CLLocation(latitude: latitude,
                     longitude: longitude)

                    let jobPosting = JobPosting(id: id, userID: userID,
                     companyName: companyName, jobDescription:
                     jobDescription, coordinates: location.coordinate,
                     postcode: postcode)
                    self.jobPostings.append(jobPosting)
                }
                print("Fetched \(self.jobPostings.count) jobs")
            }
        }
    }
func fetchJobPostings(completion: @escaping ([JobPosting]) -> Void) {
    jobPostings.removeAll()
    let db = Firestore.firestore()
    let ref = db.collection("Jobs")

    ref.getDocuments { snapshot, error in
        guard error == nil else {
            print("Error fetching job postings:",
             error!.localizedDescription)
            completion([])
            return
        }

        if let snapshot = snapshot {
```

```
        print("Found \(snapshot.documents.count) job documents")

        let dispatchGroup = DispatchGroup()
        var geocodedJobs: [JobPosting] = []

        for document in snapshot.documents {
            let data = document.data()

            let id = data["id"] as? String ?? ""
            let userID = data["userID"] as? String ?? ""
            let companyName = data["companyName"] as? String ?? ""
            let jobDescription = data["jobDescription"] as? String
             ?? ""
            let postcode = data["postcode"] as? String ?? ""
            let coordinates = data["coordinates"] as? [String: Any]
             ?? [:]
            let latitude = coordinates["latitude"] as? Double ?? 0.0
            let longitude = coordinates["longitude"] as? Double ??
             0.0
            let location = CLLocation(latitude: latitude,
             longitude: longitude)

            dispatchGroup.enter()

            self.geocodeLocation(location, forJobId: id) { result in
                switch result {
                case .success(let placemark):
                    let jobPosting = JobPosting(
                        id: id,
                        userID: userID,
                        companyName: companyName,
                        jobDescription: jobDescription,
                        coordinates: placemark.location?.coordinate
                         ?? CLLocationCoordinate2D(),
                        postcode: postcode
                    )
                    geocodedJobs.append(jobPosting)
                    print("Geocoded Job - ID: \(id), Company:
                     \(companyName), Postcode: \(String(describing:
                     placemark.postalCode))")
                case .failure(let error):
                    print("Geocoding error:
                     \(error.localizedDescription)")
                }

                dispatchGroup.leave()
            }
        }

        dispatchGroup.notify(queue: .main) {
            print("Geocoding of job postings completed")

            // Now, geocodedJobs array contains all the geocoded
             JobPosting instances
```

```
                print("Fetched \(geocodedJobs.count) geocoded job
                 postings")
                completion(geocodedJobs)
            }
        }
    }
}
func geocodeLocation(_ location: CLLocation, forJobId jobId: String,
 completion: @escaping (Result<CLPlacemark, Error>) -> Void) {
    print("Before geocoding - Job ID: \(jobId)")
    geocoder.reverseGeocodeLocation(location) { placemarks, error in
        if let error = error {
            let geocodingError = NSError(domain:
             "GeocodingErrorDomain", code: 1, userInfo:
             [NSLocalizedDescriptionKey: "Geocoding failed:
             \(error.localizedDescription)"])
            completion(.failure(geocodingError))
            print("Geocoding error: \(error.localizedDescription)")
            return
        }

        guard let placemark = placemarks?.first else {
            let customError = NSError(domain: "GeocodingErrorDomain",
             code: 2, userInfo: [NSLocalizedDescriptionKey: "No
             placemark found"])
            completion(.failure(customError))
            print("Geocoding error: No placemark found")
            return
        }

        completion(.success(placemark))
        print("After geocoding - Job ID: \(jobId)")
    }
}

func fetchUserNames(for userIDs: [String], completion: @escaping
 ([String: String]) -> Void) {
    var userNames: [String: String] = [:]
    let dispatchGroup = DispatchGroup()

    for userID in userIDs {
        dispatchGroup.enter()
        let db = Firestore.firestore()
        let ref = db.collection("Users").document(userID)

        ref.getDocument { document, error in
            defer {
                dispatchGroup.leave()
            }

            guard error == nil else {
                print("Error fetching user name:
                 \(error!.localizedDescription)")
                return
            }
```

```
            if let document = document, document.exists {
                let data = document.data()
                if let name = data?["name"] as? String {
                    userNames[userID] = name
                }
            }
        }
    }

    dispatchGroup.notify(queue: .main) {
        completion(userNames)
    }
}

func postJobWithGeocoding(jobPosting: JobPosting, coordinates:
 CLLocationCoordinate2D?) {
    guard let coordinates = coordinates else {
        print("Invalid coordinates")
        return
    }

    let geocoder = CLGeocoder()

    let location = CLLocation(latitude: coordinates.latitude,
     longitude: coordinates.longitude)

    geocoder.reverseGeocodeLocation(location) { (placemarks, error) in
        guard let placemark = placemarks?.first, let location =
         placemark.location?.coordinate else {
            print("Geocoding error: \(error?.localizedDescription ??
             "Unknown error")")
            return
        }

        var updatedJobPosting = jobPosting
        updatedJobPosting.coordinates = location

        self.postJob(jobPosting: updatedJobPosting)
    }
}

func geocodeAddress(_ address: String, completion: @escaping
 (Result<(CLLocationCoordinate2D, String), Error>) -> Void) {
        let geocoder = CLGeocoder()
        geocoder.geocodeAddressString(address) { placemarks, error in
            if let error = error {
                completion(.failure(error))
                return
            }

            if let location = placemarks?.first?.location?.coordinate,
               let postcode = placemarks?.first?.postalCode {
                let result: (CLLocationCoordinate2D, String) =
                 (location, postcode)
```

```
                completion(.success(result))
            } else {
                let customError = NSError(domain:
                 "GeocodingErrorDomain", code: 1, userInfo:
                 [NSLocalizedDescriptionKey: "Invalid coordinates"])
                completion(.failure(customError))
            }
        }
    }

func postJob(jobPosting: JobPosting) {
    let db = Firestore.firestore()

    let coordinates: [String: Double] = [
        "latitude": jobPosting.coordinates?.latitude ?? 0.0,
        "longitude": jobPosting.coordinates?.longitude ?? 0.0
    ]

    let ref = db.collection("Jobs").document(jobPosting.id)

    ref.setData([
        "id": jobPosting.id,
        "userID": jobPosting.userID,
        "companyName": jobPosting.companyName,
        "jobDescription": jobPosting.jobDescription,
        "coordinates": coordinates, // Fix here: Use the coordinates
         dictionary
        "postcode": jobPosting.postcode
        // ... other properties you may have
    ]) { error in
        if let error = error {
            print("Error posting job:", error.localizedDescription)
        } else {
            print("Job posted successfully")
        }
    }
}
func postJobWithGeocoding(jobPosting: JobPosting, address: String) {
    // Use a dispatch group to wait for geocoding task to finish
    let dispatchGroup = DispatchGroup()

    // Create a placeholder for the geocoded location and postcode
    var geocodedLocation: CLLocationCoordinate2D?
    var geocodedPostcode: String?

    // Enter the dispatch group for the geocoding task
    dispatchGroup.enter()

    // Perform geocoding for the provided address
    geocodeAddress(address) { result in
        switch result {
        case .success(let (location, postcode)):
            geocodedLocation = location
            geocodedPostcode = postcode
        case .failure(let error):
```

```
            print("Geocoding error: \(error.localizedDescription)")
        }

        // Leave the dispatch group when the geocoding task is done
        dispatchGroup.leave()
    }

    // Notify when the geocoding task is finished
    dispatchGroup.notify(queue: .main) {
        guard let geocodedLocation = geocodedLocation, let
         geocodedPostcode = geocodedPostcode else {
            print("Failed to geocode the address. Job not posted.")
            return
        }

        // Update the job posting with the geocoded location and
         postcode
        var updatedJobPosting = jobPosting
        updatedJobPosting.coordinates = geocodedLocation
        updatedJobPosting.postcode = geocodedPostcode

        // Continue with posting the job to the database
        self.postJob(jobPosting: updatedJobPosting)
    }
}
func geocodeJobPostings(completion: @escaping () -> Void) {
    // Use a dispatch group to wait for all geocoding tasks to finish
    let dispatchGroup = DispatchGroup()

    var updatedGeocodedJobPostings: [GeocodedJobPosting] = []

    for jobPosting in jobPostings {
        // Assuming 'coordinates' is a property in your JobPosting model
        guard let coordinates = jobPosting.coordinates else {
            continue // Skip geocoding if coordinates are missing
        }

        dispatchGroup.enter()

        // Create a CLLocation instance using the coordinates
        let location = CLLocation(latitude: coordinates.latitude,
         longitude: coordinates.longitude)

        geocodeLocation(location, forJobId: jobPosting.id) { result in
            switch result {
            case .success(let placemark):
                // Assuming you have a GeocodedJobPosting initializer
                let geocodedJob = GeocodedJobPosting(
                    jobPosting: jobPosting,
                    id: jobPosting.id,
                    userID: jobPosting.userID,
                    companyName: jobPosting.companyName,
                    jobDescription: jobPosting.jobDescription,
                    coordinates: coordinates,
                    postcode: placemark.postalCode
```

```
                    )

                    // Append the geocoded job within the context of the
                     dispatch group
                    updatedGeocodedJobPostings.append(geocodedJob)
                case .failure(let error):
                    print("Geocoding error for Job ID \(jobPosting.id):
                     \(error.localizedDescription)")
                }

                dispatchGroup.leave()
            }
        }

        // Notify when all geocoding tasks are finished
        dispatchGroup.notify(queue: .main) {
            // Update the geocodedJobPostings array after all tasks are done
            self.geocodedJobPostings = updatedGeocodedJobPostings

            // Call the completion block
            completion()
        }
    }
}
```