```
//
//  DatabaseHandler.swift
//  Lab4MstTest
//
//  Created by Victor Pregén on 2019-01-18.
//  Copyright © 2019 Victor Pregén. All rights reserved.
//

import Foundation
import Firebase
import FirebaseAuth
import FirebaseFirestore

protocol DBHandlerDelegate: class {
    func registrationStateDBHandlerToDBController(registrationState: String)
    func authStateDBHandlerToDBController(authState: String, currentUser:
     UserModel)
    func dbStateChangedDBHandlerToDBController(threadId: String, partnerId:
     String, messageId: String, partner: UserModel, message: MessageModel)
    func setModelSearchList(users: [UserModel])


}

class DatabaseHandler  {
    weak var dBHandlerDelegate: DBHandlerDelegate?
    var dbRefUsers: DatabaseReference!
    var dbRefThreads: DatabaseReference!
    var userId: String!
    var currentUser: UserModel!

    init() {
        dbRefUsers = Database.database().reference().child("users")
        dbRefThreads = Database.database().reference().child("threads")

        //userId = Auth.auth().currentUser?.uid
        startFireBaseAuthStateChangeListener()
    }

    func notifyDBControllerAuthStateChange(currentUser: UserModel) {
        dBHandlerDelegate?.authStateDBHandlerToDBController(authState:
         "AUTHENTICATED", currentUser: currentUser)
    }

    func notifyDBControllerdbStateChanged(threadId: String, partnerId:
     String, messageId: String, partner: UserModel, message: MessageModel) {
        dBHandlerDelegate?.dbStateChangedDBHandlerToDBController(threadId:
         threadId, partnerId: partnerId, messageId: messageId, partner:
         partner, message: message)
    }

    func notifyDBControllerNewUsers(users: [UserModel]) {
        dBHandlerDelegate?.setModelSearchList(users: users)
    }
```

```
/** FINNS ÄVEN KALLA PÅ FUNKTIONER I FUNKTRIONERNA**/

//-------------------------------------------------------------------------
 ------------

func startFireBaseAuthStateChangeListener() {
    Auth.auth().addStateDidChangeListener() { auth, user in
        if user != nil {
            self.observeCurrentUserName()
        }
    }
}

// GET USER INFORMATION
func observeCurrentUserName() {
    let currentUserid = Auth.auth().currentUser?.uid
    let currentUserMail = Auth.auth().currentUser?.email

    dbRefUsers.child(currentUserid!).observeSingleEvent(of: .value,
     with: { (userSnapshot ) in
        if let userValues = userSnapshot.value as? NSDictionary {

            let currentUserName = userValues["name"] as? String
            let currentUser = UserModel(userId: currentUserid!, name:
             currentUserName!, email: currentUserMail!)

            DispatchQueue.main.async(execute: {
                //print("Observe current USER name kör Dump")
                self.currentUser = currentUser
                self.userId = currentUser.userId
                self.notifyDBControllerAuthStateChange(currentUser:
                 currentUser)
            })
        }
    }, withCancel: nil)
}

func register(email: String, password: String, name: String ) {
    /** AUTH USER HAR OCKSÅ EN VARIABEL FÖR DISPLAYNAME **/
    // FROM HANDLER TO CONTROLLER --->

    /*

      dBHandlerDelegate?.registrationStateDBHandlerToDBController(regist
      rationState: "TMP STATE")
     */

    Auth.auth().createUser(withEmail: email, password: password) { aRes,
     error in
        if let error = error {

            /*
             let alert = UIAlertController(title: "Registration failed",
             message: error.localizedDescription,
             preferredStyle: .alert)
```

```
            alert.addAction(UIAlertAction(title: "OK",
             style: .default))
            self.present(alert, animated: true, completion: nil)
            */

        } else {

            guard let user = aRes?.user else {
                return
            }
            let userValues = ["name": name, "email": email]

            Database.database().reference().child("users").child(user.u
            id).updateChildValues(userValues)
        }
    }
}

/* MAKES DELEGATE CALL BACK TO TO DB CONTROLLER IF WHEN SUCESS OR FAIL
 */
/** DET ÄR SKILLNAD PÅ SIGN IN TILL APPEN OCH SIGN IN (AUTH) TILL DB **/
func signIn(email: String, password: String) {

    Auth.auth().signIn(withEmail: email, password: password) { user,
     error in
        if let error = error, user == nil {

            /** SKICKA DENNA ALERT TILLBAKA TILL MAIN CONTROLLER **/
            /*
            let alert = UIAlertController(title: "Login Failed",
            message: error.localizedDescription,
            preferredStyle: .alert)

            alert.addAction(UIAlertAction(title: "OK",
             style: .default))

            self.present(alert, animated: true, completion: nil)
            */

        } else {
            //self.performSegue(withIdentifier: "loginToMain", sender:
             nil)
        }
    }
}

func searchForUsers() {
    dbRefUsers.observeSingleEvent(of: .value, with: { (snapshot ) in
        var users = [UserModel]()
        for rest in snapshot.children.allObjects as! [DataSnapshot] {
            users.append(UserModel(snapShot: rest))
        }

        DispatchQueue.main.async(execute: {
```

```
                self.notifyDBControllerNewUsers(users: users)
            })
        }, withCancel: nil)
    }

func sendMessage(threadId: String, MessageText: String) {

    let userId = Auth.auth().currentUser?.uid
    let timeStamp = String(NSDate().timeIntervalSince1970)


     dbRefUsers.child(userId!).child("threads").child(threadId).observeSingl
     eEvent(of: .value, with: { (threadSnapshot ) in

        let snapValues = threadSnapshot.value as? NSDictionary
        let partnerId = snapValues!["partnerId"] as? String
        let msgDict = ["msgText": MessageText, "sentToId": partnerId,
         "sentFromId": userId, "timeStamp": timeStamp]

        self.updateDbValues(threadId: threadId, msgDict: msgDict as! [String
         : String], userId: userId!, partnerId: partnerId!)

    }, withCancel: nil)
}

func updateDbValues(threadId: String, msgDict: [String: String], userId:
 String, partnerId: String) {

     self.dbRefThreads.child(threadId).child("messages").childByAutoId().upd
     ateChildValues(msgDict)  { (error, ref) in

        if error != nil {

             self.dbRefUsers.child(userId).child("threads").child(threadId).
             updateChildValues(["partnerId": partnerId])

             self.dbRefUsers.child(partnerId).child("threads").child(threadI
             d).updateChildValues(["partnerId": userId])
        }
    }
}

func startObserveThreads() {
    print("UserId \(userId)")
    dbRefUsers.child(userId).child("threads").observe(.childAdded, with:{
        (threadSnapshot) in

        let threadId = threadSnapshot.key
        let snapValues = threadSnapshot.value as? NSDictionary
        let partnerId = snapValues!["partnerId"] as? String

        self.observePartnerForThread(partnerId: partnerId!, threadId:
         threadId)

    }, withCancel: nil)
```

```
}

func observePartnerForThread(partnerId: String, threadId: String) {
    dbRefUsers.child(partnerId).observeSingleEvent(of: .value, with: {
        (snapshot) in

        let partner = UserModel(snapShot: snapshot)
        self.observeMessagesForThread(partnerId: partnerId, threadId:
         threadId, partner: partner)

    }, withCancel: nil)
}

func observeMessagesForThread(partnerId: String, threadId: String, partner:
 UserModel) {
    dbRefThreads.child(threadId).child("messages").observe(.childAdded,
     with:{
        (snapshot) in

        let message = MessageModel(snapShot: snapshot)
        DispatchQueue.main.async(execute: {
            self.notifyDBControllerdbStateChanged(threadId: threadId,
             partnerId: partnerId, messageId: message.messageId, partner:
             partner, message: message)
        })

    }, withCancel: nil)
}

func signout() {
    do {
        try Auth.auth().signOut()
    } catch (let error) {

    }
}
}
```