iFlat

Object Design

1.0

02/01/2017

Tolga Taner

Eren Ay

Ö. Meva Demiröz

Alican Yilmaz

Prepared for

SE301 Software Engineering



Table of Contents

[1. Introduction 1](#_Toc436772639)

[1.1. Object Design Trade-offs 1](#_Toc436772640)

[1.2. Interface Documentation Guidelines 1](#_Toc436772641)

[1.3. Definitions, Acronyms, and Abbreviations 1](#_Toc436772642)

[1.4. References 1](#_Toc436772643)

[2. Packages 1](#_Toc436772644)

[3. Class Interfaces 1](#_Toc436772645)

OBJECT DESIGN DOCUMENT

# Introduction

## Object Design Trade-offs[2]

**Organization and Complexity tradeoffs**

We are totally Serverless! But this is does not mean Codeless! Our back-end is running on client-side. We’ve are using client-side back-end but we are writing lots of back-end code in our client-side and its hard to organize. But we have no a API layer which connected to a server. E.g. Rest API connected with a relational DB server.

**Consistency and Speed tradeoffs**

We are using **Firebase** Back-End as a Service which is a persistent data storage service which built by **Nosql** apporach. We have lots of trade-offs with our persistent-data storage layer due to Nosql Apporach. As we are using nosql, absolutely no traditional relational database motor, our tradeoffs are for data storage;

In our database, we have no tables, so we have no physical relations. This does not let us to get complex queries. Firebase hosts data as key-value form. So physically have no relations. This excludes relations, relatively relationships. So without relationships we have no joins which effects performance of querying directly. So our database is running out with [single web socket] this causes a perfect speed con. Our database is really very swift.

**Reliability and Speed tradeoffs**

Firebase has no a powerful querymotor. So we can not get complex queries. But we are swift.

**Functionality and Usability tradeoffs**

Our app is a iPhone iOS App. In this project we are aimed to deliver functionality rather than usability. We have not used any UI design apporach like Material design or another apporaches. We are directly designed ui of our app with apple stock ui components inherited from UIKit framework. So in our project exactly we are followed a touchable screen rules. This decreases usability a bit but functionality is more important.

In our app we are used phone-sensors like camera this increases usability but this slows our app.

**Space and Time tradeoffs**

In our project we are using image caching solutions, so caching the image includes memory space and disk space but our images is loading and downloading very fast. Including 100,000+ records on DB. Our project uses abit memory and disk space but our images is loading very fast. Generally in our app we are tradeoff space vs time.

**Other tradeoffs**

Firebase has no DBMS. This excludes, dbms related tools i.e. reporting, migration tools.

Our data model is lightweight with Json format. So reading of Json is hard. But Json is more fast and lightweight. Our data **space** is lighther than a classic relational db table data **space**.

Firebase has a sync and realtime database features, we trade consistency with a perfect observer design pattern implementations. In our app there is no need to refresh any page to get news.

## Interface Documentation Guidelines[1]

**Naming Guideline**

|  |  |  |
| --- | --- | --- |
| Identifier Type | Rules for Naming | Examples |
| Classes | Class names should be nouns, in mixed case with the first letter of each internal word capitalized. Try to keep your class names simple and descriptive. Use whole words-avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL). | class Reservation |
| Protocols | Interface names should be capitalized like class names. | protocol TableViewDelegate |
| Methods | Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized. | validate()  getCurrentUser() |
| Variables | Except for variables, all instance, class, and class constants are in mixed case with a lowercase first letter. Internal words start with capital letters. Variable names should not start with underscore \_ or dollar sign $ characters, even though both are allowed.  Variable names should be short yet meaningful. The choice of a variable name should be mnemonic- that is, designed to indicate to the casual observer the intent of its use. One-character variable names should be avoided except for temporary "throwaway" variables. Common names for temporary variables are i, j, k, m, and n for integers; c, d, and e for characters. | int i  var c  Double myWidth; |
| Constants | The names of variables declared class constants and of ANSI constants should be all uppercase with words separated by underscores ("\_"). (ANSI constants should be avoided, for ease of debugging.) | static let int MIN\_WIDTH = 4;  static let int MAX\_WIDTH = 999; |

**Using FIR… classes in Model classes Guideline**

If any developer want to use Firebase manipulation methods which are responsible to constructing data manipulation relationship with Google Firebase server, developers must get an instance of the manipulation classes in their model classes like;

func getFlatsWhichAreNotReserved(){

self.DB\_ENDPOINT.getFlatsWhichAreNotReserved()

...

the DB\_ENDPOINT is the bridge instance which is instance of firebase related manipulation methods. Developers must user this instance to call manipulation methods in their models. Direct usage of this class instance in control classes is **not allowed.**

**Exceptions Guideline**

FIR… methods does not throws exceptions directly. But generally developers handle optional chaning feature of Swift API. Swift includes nil pointer handling mechanism. If developers are exactly knows there is a nilpointer, then unwrap the variable. If any variable is may be nil. Then developers must tag it as optional.When FIR… class methods gets crashed, they returns error’s debugdescription for inform the developers about what is going wrong. So in completion block, developers must ensure if the completion block parameter is nil or not.

}

## Definitions, Acronyms, and Abbreviations

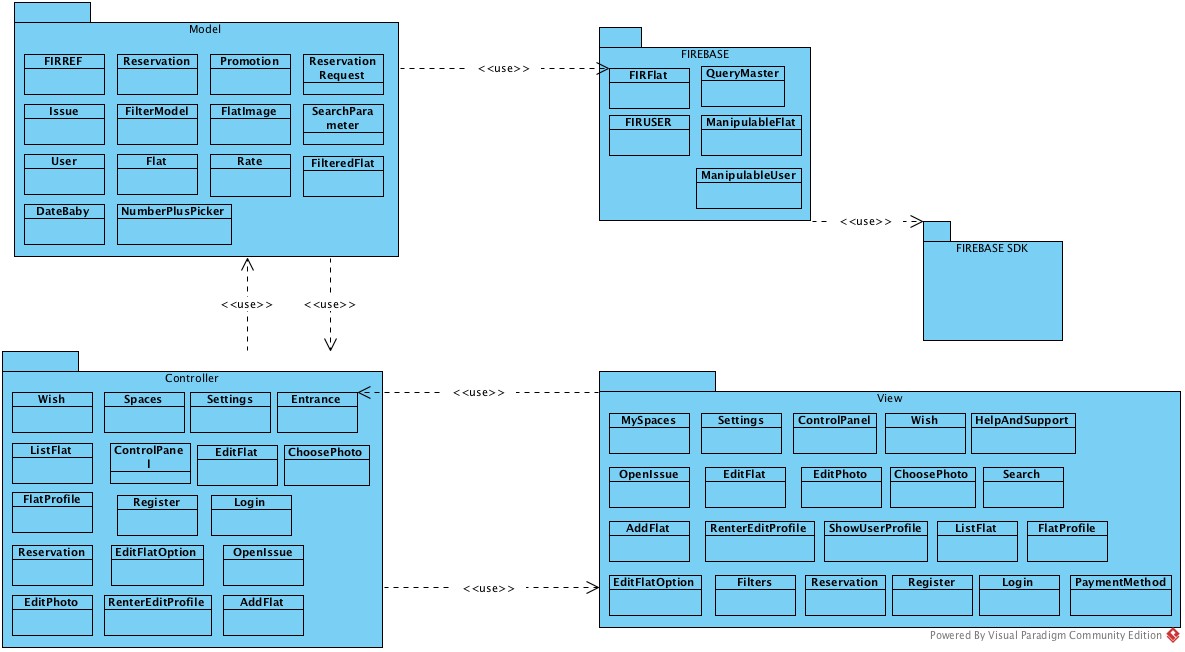
|  |  |
| --- | --- |
| Model | A Model class of MVC Architectural pattern. |
| API | Application Programming Interface |
| Swift | Apple Swift Native Language |
| Optional | A variable or object may become nil anytime |
| Firebase | Firebase Backend as a Service |
| Controller | Controller classes which interacts with Models and View classes of MVC pattern. |
| DBMS | Database Management System |
| Nosql | Not only SQL apporach |
| Protocol | Protocol which can be thinked like Interfaces in Java |
| DB | Database |
| App | Application |
| Backend | A layer of a application, which is responsible for organising data storage and organization. |

## References

[1]<http://www.oracle.com/technetwork/java/codeconventions-135099.html>

[2]<https://firebase.google.com/docs/ios>

# Packages



In our packages, in general, we are using Model View Controller apporach which is approved by Swift and Apple iOS API. Our MVC packages contains related classes. For instance, in model package there is model classes which interacts only Firebase layer to manipulate persistent data. In controller package our classes including use only UI and Model integration and lifecycle of UI with data manipulation with Model layer. In our View layer, we only wroted our view classes. But in iOS we’ve used Storyboards which uses markup language XML, storyboard technology let developers finish UI with Interface Builder. For application lifecycle, we use Apple’s Cocoa and Cocoa Touch app lifecycle and UI lifecycle methods and functionalities. We didn’t used all the storyboard features, in some cases we have to design custom Uis with UIKIT framework so there is some related swift code in our View classes also.

# Class Interfaces

In our project we’ve used for class interfaces a tool like javadoc named “Jazzy”.

Jazzy is a tool which converts and builds the code with comment and documentation marks and tags. For beeing example, there is some examples below, which is our class interfaces, the full documentation is on this URL as Appledoc:

<https://iflat-3b499.firebaseapp.com/>

//

// FIRFlat.swift

// iFlat

//

// Created by Alican Yilmaz on 12/12/2016.

// Copyright © 2016 SE 301. All rights reserved.

//

import Foundation

import Firebase

import FirebaseStorage

internal protocol FIRFlatDelegate : class {

internal func edit(oldcity: String, newFlt: @autoclosure @escaping ManipulableFlat!, completion: @autoclosure (String?) -> ()) -> <<error type>>

internal func disable(disablingFlat: ManipulableFlat!, completion: @autoclosure @escaping (String?) -> ()) -> <<error type>>

internal func getFlatsofUser(userID: String, completion: @autoclosure @escaping ([ManipulableFlat]?) -> ()) -> <<error type>>

internal func getFlatofUser(userID: String, flatID: @autoclosure @escaping String, completion: @autoclosure (ManipulableFlat?) -> ()) -> <<error type>>

internal func getFlatImages(flatID: String, completion: @autoclosure @escaping ([FlatImageDownloaded]?) -> ()) -> <<error type>>

internal func setOwnerID() -> String!

internal func insertFlat(flt: ManipulableFlat, completion: @autoclosure @escaping (String?) -> ()) -> <<error type>>

internal func insertTimeSlot(flt: ManipulableFlat, timeslot: @autoclosure @escaping [Int], completion: @autoclosure (String?) -> ()) -> <<error type>>

internal func addWishList(flt: ManipulableFlat, completion: @autoclosure @escaping (String?) -> ()) -> <<error type>>

internal func deleteWish(flt: ManipulableFlat, completion: @autoclosure @escaping (String?) -> ()) -> <<error type>>

internal func getAvailableTimeSlots(flt: ManipulableFlat, completion: @autoclosure @escaping ([Int?]) -> ()) -> <<error type>>

internal func deleteFlat(flt: ManipulableFlat, oldcity: String) -> <<error type>>

}

internal class FIRFlat : FIRFlatDelegate {

/// This function deletes flat

///

/// - Parameter flt: (ManipubleFlat) flat that is wanted to be deleted

/// - Parameter oldcity: (String) flats city

/// - returns void

/// - throws: FIRERROR

internal func deleteFlat(flt: ManipulableFlat, oldcity: @autoclosure @escaping String) -> <<error type>>

/// This function gets available time slots of flat from DB

///

/// - Parameter flt: (ManipubleFlat) Requesting flat

/// - Parameter completion: Completion Block

/// - returns void

/// - throws: FIRERROR

internal func getAvailableTimeSlots(flt: ManipulableFlat, completion: @autoclosure @escaping ([Int?]) -> ()) -> <<error type>>

/// This function deletes wish of flat from DB

///

/// - Parameter flt: (ManipubleFlat) Requesting flat

/// - Parameter completion: Completion Block

/// - returns void

/// - throws: FIRERROR

internal func deleteWish(flt: ManipulableFlat, completion: @autoclosure @escaping (String?) -> ()) -> <<error type>>

/// This function adds wish for flat to DB

///

/// - Parameter flt: (ManipubleFlat) Requesting flat

/// - Parameter completion: Completion Block

/// - returns void

/// - throws: FIRERROR

internal func addWishList(flt: ManipulableFlat, completion: @autoclosure @escaping (String?) -> ()) -> <<error type>>

/// This function inserts time slot to DB for flat

///

/// - Parameter flt: (ManipubleFlat) Requesting flat

/// - Parameter timeslot: ([Int]) Time slot array for flat

/// - Parameter completion: Completion Block

/// - returns void

/// - throws: FIRERROR

internal func insertTimeSlot(flt: ManipulableFlat, timeslot: @autoclosure @escaping [Int], completion: @autoclosure (String?) -> ()) -> <<error type>>

/// This function inserts flat to DB

///

/// - Parameter flt: (ManipubleFlat) Flat to insert

/// - Parameter completion: Completion Block

/// - returns void

/// - throws: FIRERROR

internal func insertFlat(flt: ManipulableFlat, completion: @autoclosure @escaping (String?) -> ()) -> <<error type>>

/// This function sets ownerID by currentLoggedUser

///

/// - Parameter completion: Completion Block

/// - returns void

/// - throws: FIRERROR

internal func setOwnerID() -> String!

/// This function pulls all images of flat

///

/// - Parameter flatID: (String) Requesting flat

/// - Parameter completion: Completion Block

/// - returns void

/// - throws: FIRERROR

internal func getFlatImages(flatID: String, completion: @autoclosure @escaping ([FlatImageDownloaded]?) -> ()) -> <<error type>>

/// This function gets requesting flat by UserID and FlatID

///

/// - Parameter userID: (String) Flat's owner ID

/// - Parameter flatID: (String) Flat ID

/// - Parameter completion: Completion Block

/// - returns void

/// - throws: FIRERROR

internal func getFlatofUser(userID: String, flatID: @autoclosure @escaping String, completion: @autoclosure (ManipulableFlat?) -> ()) -> <<error type>>

/// This function pulls all flats of user

///

/// - Parameter userID: (String) Requesting User ID

/// - Parameter completion: Completion Block

/// - returns void

/// - throws: FIRERROR

internal func getFlatsofUser(userID: String, completion: @autoclosure @escaping ([ManipulableFlat]?) -> ()) -> <<error type>>

/// This function disables flat

///

/// - Parameter disablingFlat: (ManipubleFlat) Disabling flat

/// - Parameter completion: Completion Block

/// - returns void

/// - throws: FIRERROR

internal func disable(disablingFlat: ManipulableFlat!, completion: (String?) -> ()) -> <<error type>>

/// This function edits flat

///

/// - Parameter oldCity: (String) Old city of flat

/// - Parameter newFlt: (ManipubleFlat) Edited flat

/// - Parameter completion: Completion Block

/// - returns void

/// - throws: FIRERROR

internal func edit(oldcity: String, newFlt: @autoclosure @escaping ManipulableFlat!, completion: @autoclosure (String?) -> ()) -> <<error type>>

}

//

// UserDelegate.swift

// iFlat

//

// Created by Alican Yilmaz on 02/12/2016.

// Copyright © 2016 SE 301. All rights reserved.

//

//This class is wroted though FIREBASE authendication framework!

//Handle Optionals

import Foundation

import Firebase

import FirebaseStorage

///This protocol is a delegation bridge. When any coder wants to use DB manipulation methods, then the coder must use this bridge instance.

///Coder also need to conform the protocol which is ManipulableUser. If want to use the manipulation methods. Manipulation methods only takes objects which conforms ManipulableUser.

internal protocol FIRUSERDelegate : class {

internal func insert(usr: ManipulableUser!, completion: @autoclosure @escaping (String?) -> ()) -> <<error type>>

internal func edit(newUsr: ManipulableUser!, completion: @autoclosure @escaping (String?) -> ()) -> <<error type>>

internal func loginByEmailAndPassword(email: String, password: String, completion: @escaping (String?) -> ())

internal func logout(completion: @escaping (String?) -> ())

internal func getCurrentLoggedIn(completion: (ManipulableUser?) -> ()) -> <<error type>>

internal func getByEmail(email: String, completion: @autoclosure @escaping (ManipulableUser?) -> ()) -> <<error type>>

internal func sendverificationEmail(completion: @escaping (String?) -> ())

internal func isUserVerified(completion: @escaping (Bool) -> ())

internal func changePassword(newPassword: String, completion: @escaping (String?) -> ())

internal func changeEmail(newEmail: String, completion: @escaping (String?) -> ())

internal func insertUserProfileImage(user: ManipulableUser, completion: @autoclosure @escaping (String?) -> ()) -> <<error type>>

internal func getUserProfileImg(user: ManipulableUser, completion: (String?) -> ()) -> <<error type>>

internal func rateUser(toUserID: String, rate: @autoclosure @escaping Rate, completion: @autoclosure (String?) -> ()) -> <<error type>>

internal func getUserRates(userID: String, completion: @autoclosure @escaping ([Rate]?) -> ()) -> <<error type>>

internal func changeUserProfileImage(user: ManipulableUser, img: @autoclosure @escaping UIImage, completion: @autoclosure (String?) -> ()) -> <<error type>>

internal func getCities(completion: @escaping ([String]) -> ())

internal func forgotPassword(email: String, completion: @escaping (String?) -> ())

internal func openIssue(toUser: ManipulableUser, issue: @autoclosure @escaping Issue, completion: @autoclosure (String?) -> ()) -> <<error type>>

internal func getISsue(user: ManipulableUser, completion: @autoclosure @escaping ([Issue]) -> ()) -> <<error type>>

internal func getUserByID(id: String, completion: @autoclosure @escaping (ManipulableUser?) -> ()) -> <<error type>>

internal func sendReservationRequest(req: ReservationRequest, completion: @autoclosure @escaping (String?) -> ()) -> <<error type>>

internal func acceptReservationRequest(req: ReservationRequest, completion: @autoclosure @escaping (String?) -> ()) -> <<error type>>

internal func rejectReservationRequest(req: ReservationRequest, completion: @autoclosure @escaping (String?) -> ()) -> <<error type>>

internal func getFlatByID(id: String, completion: (ManipulableFlat?) -> ()) -> <<error type>>

internal func getUsersReservationRequests(usr: ManipulableUser, completion: @autoclosure @escaping ([ReservationRequest]) -> ()) -> <<error type>>

internal func getWishes(usrID: String, completion: @escaping ([String : Bool]?) -> ())

}

/\*\*

This class is the object which connects coder to Db for manipulation.

Developers can use this class at their controller classes. This class \*\*usage\*\* directly in model is not \*\*allowed.\*\*

This class' methods only takes parameters of User,Flat and other models which conforms related protocols i.e. ManipulableUser

\*/

internal class FIRUSER : FIRUSERDelegate {

/\*\*

This method returns its completion closure paramter, the users which have wishes. The returning wishes returns as flatID.

- parameters:

- usrID: The userID which is ID of user whose wishes requested.

- completion: This parameter is a callback closure block which returns the wishes as dictionary String:Bool.

key of the dictionary is the id of user, value is always true.

\*/

internal func getWishes(usrID: String, completion: @escaping ([String : Bool]?) -> ())

/\*\*

Get flat by its id as ManipulableFlat object

- parameters:

- id: The id which is ID of flat whose requested.

- completion: This parameter is a callback closure block which returns flats as ManipulableFlat object.

- returns:Void

- throws:FIRERROR

\*/

internal func getFlatByID(id: String, completion: @autoclosure @escaping (ManipulableFlat?) -> ()) -> <<error type>>

/// This functions loads logged user's reservation request

///

/// - Parameter usr: (ManipubleUser)

/// - Parameter completion: Completion Block

/// - throws: FIRERROR?

/// - returns void

internal func getUsersReservationRequests(usr: ManipulableUser, completion: @autoclosure @escaping ([ReservationRequest]) -> ()) -> <<error type>>

/// This function makes reservation request accepted

///

/// - Parameter req: (ReservationRequest)

/// - Parameter completion: Completion Block

/// - throws: FIRERRIR

/// - returns: void

internal func acceptReservationRequest(req: ReservationRequest, completion: @autoclosure @escaping (String?) -> ()) -> <<error type>>

/// This function rejects reservation request

///

/// - Parameter req: (ReservationRequest)

/// - Parameter completion: Completion Block

/// - returns void

/// - throws: FIRERROR

internal func rejectReservationRequest(req: ReservationRequest, completion: @autoclosure @escaping (String?) -> ()) -> <<error type>>

/// This function adds reservation request to DB

///

/// - Parameter req: (ReservationRequest)

/// - Parameter completion: Completion Block

/// - returns void

/// - throws: FIRERROR

internal func sendReservationRequest(req: ReservationRequest, completion: @autoclosure @escaping (String?) -> ()) -> <<error type>>

/// This functions gets user from DB with userID

///

/// - Parameter id: (String) UserID

/// - Parameter completion: Completion Block

/// - returns void

/// - throws: FIRERROR

internal func getUserByID(id: String, completion: @autoclosure @escaping (ManipulableUser?) -> ()) -> <<error type>>

/// This function opens issue to specific user.

///

/// - Parameter toUser: (ManipulableUser) User

/// - Parameter completion: Completion Block

/// - returns void

/// - throws: FIRERROR

internal func openIssue(toUser: ManipulableUser, issue: @autoclosure @escaping Issue, completion: @autoclosure (String?) -> ()) -> <<error type>>

/// This function gets issue of user.

///

/// - Parameter user: (ManipulableUser) User object

/// - Parameter completion: Completion Block

/// - returns void

/// - throws: FIRERROR

internal func getISsue(user: ManipulableUser, completion: @autoclosure @escaping ([Issue]) -> ()) -> <<error type>>

/// This function sends password reset email to user

///

/// - Parameter email: (String) UserEmail

/// - Parameter completion: Completion Block

/// - returns void

/// - throws: FIRERROR

internal func forgotPassword(email: String, completion: @escaping (String?) -> ())

/// This function gets cities from the db.

///

/// - Parameter completion: Completion Block

/// - returns void

/// - throws: FIRERROR

internal func getCities(completion: @escaping ([String]) -> ())

/// This function changes user's profile Image

///

/// - Parameter user: (ManipubleUser) that currently logged

/// - Parameter img: (UIImage) new profile image

/// - Parameter completion: Completion Block

/// - returns void

/// - throws: FIRERROR

internal func changeUserProfileImage(user: ManipulableUser, img: @autoclosure @escaping UIImage, completion: @autoclosure (String?) -> ()) -> <<error type>>

/// This function retrieves rates of User from DB

///

/// - Parameter userID: (String) User ID of requesting one

/// - Parameter completion: Completion Block

/// - returns void

/// - throws: FIRERROR

internal func getUserRates(userID: String, completion: @autoclosure @escaping ([Rate]?) -> ()) -> <<error type>>

/// This function adds rate to user

///

/// - Parameter toUserID: (String) User ID of rating user

/// - Parameter completion: Completion Block

/// - returns void

/// - throws: FIRERROR

internal func rateUser(toUserID: String, rate: @autoclosure @escaping Rate, completion: @autoclosure (String?) -> ()) -> <<error type>>

/// This function retrieves user's profile image's URL

///

/// - Parameter user: ManipubleUSer

/// - Parameter completion: Completion Block

/// - returns void

/// - throws: FIRERROR

internal func getUserProfileImg(user: ManipulableUser, completion: @autoclosure @escaping (String?) -> ()) -> <<error type>>

/// This fuction inserts User Profile Image to storage and DB

///

/// - Parameter user: ManipubleUser

/// - Parameter completion: Completion Block

/// - returns void

/// - throws: FIRERROR

internal func insertUserProfileImage(user: ManipulableUser, completion: @autoclosure @escaping (String?) -> ()) -> <<error type>>

/// This function changes Email of user

///

/// - Parameter newEmail: UserEmail

/// - Parameter completion: Completion Block

/// - returns void

/// - throws: FIRERROR

internal func changeEmail(newEmail: String, completion: @escaping (String?) -> ())

/// This function changes password of user

///

/// - Parameter newPassword: UserEmail

/// - Parameter completion: Completion Block

/// - returns void

/// - throws: FIRERROR

internal func changePassword(newPassword: String, completion: @escaping (String?) -> ())

/// This function checks is user verified

///

/// - Parameter completion: Completion Block

/// - returns void

/// - throws: FIRERROR

internal func isUserVerified(completion: @escaping (Bool) -> ())

/// That function sends verification email to user

///

/// - Parameter completion: Completion Block that gets String

/// - Returns: Void

/// - returns void

/// - throws: FIRERROR

internal func sendverificationEmail(completion: @escaping (String?) -> ())

///Returns a Manipulableuser Instance for a given email. If email exists in DB.

///If the user exists, returns it. Otherwise, returns nil.

///Returning parameters are in completion block.

///

/// - Parameter email: UserEmail

/// - Parameter completion: Completion Block that gets ManipubleUser

/// - returns void

/// - throws: FIRERROR

internal func getByEmail(email: String, completion: (ManipulableUser?) -> ()) -> <<error type>>

///Returns currently logged user if any.

///If the user exists, returns it. Otherwise, returns nil.

///Returning parameters are in completion block.

///

/// - Parameter completion: Completion Block that gets ManipubleUser

/// - returns void

/// - throws: FIRERROR

internal func getCurrentLoggedIn(completion: (ManipulableUser?) -> ()) -> <<error type>>

///Logouts user who is logged in already.

///If the user exists, returns true. Otherwise, returns false.

///Returning parameters are in completion block.

///

/// - Parameter completion: Completion Block that gets String

internal func logout(completion: @escaping (String?) -> ())

///Logouts user who is logged in already.

///If the user exists, returns true. Otherwise, returns false.

///Returning parameters are in completion block.

///

/// - Parameter email: UserEmail

/// - Parameter password: USerPassword

/// - Parameter completion: Completion Block that gets String

internal func loginByEmailAndPassword(email: String, password: String, completion: @escaping (String?) -> ())

///Edit user info. The parameter newUsr is the new user whos info will replaced by the user which is passed by its email.

///If the user exists, returns true. Otherwise, returns false.

///Returning parameters are in completion block.

///

/// - Parameter newUsr: ManipubleUser object that edited

/// - Parameter completion: Completion Block that gets String

/// - returns void

/// - throws: FIRERROR

internal func edit(newUsr: ManipulableUser!, completion: @autoclosure @escaping (String?) -> ()) -> <<error type>>

///Insert user which is manipulableuser.

///If the operation is OK, returns true. Otherwise, returns false.

///Returning parameters are in completion block.

///This func also adds id to the inserted user object.

///

/// - Parameter usr: New Manipuble User

/// - Parameter completion: Completion Block that get String

/// - returns void

/// - throws: FIRERROR

internal func insert(usr: ManipulableUser!, completion: @autoclosure @escaping (String?) -> ()) -> <<error type>>

}