Додаток 3

Лістинг програми

```
@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?


    func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[NSObject: AnyObject]?) -> Bool {
        UIApplication.sharedApplication().setStatusBarStyle(UIStatusBarStyle.LightContent,
animated: false)

        AFNetworkActivityLogger.sharedLogger().level = .AFLoggerLevelDebug
        AFNetworkActivityLogger.sharedLogger().startLogging()

            println(NSFileManager.defaultManager().URLsForDirectory(.DocumentDirectory,
inDomains: .UserDomainMask))

        return FBSDKApplicationDelegate.sharedInstance().application(application,
didFinishLaunchingWithOptions: launchOptions)
    }

    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 throttle down OpenGL ES
frame rates. 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 inactive state; here you can
undo many of the changes made on entering the background.
    }

    func applicationDidBecomeActive(application: UIApplication) {
        FBSDKAppEvents.activateApp()

    }

    func applicationWillTerminate(application: UIApplication) {
        // Called when the application is about to terminate. Save data if appropriate. See also
applicationDidEnterBackground:.
        // Saves changes in the application's managed object context before the application
terminates.
        self.saveContext()
    }

    func application(application: UIApplication, openURL url: NSURL, sourceApplication: String?,
```

```
annotation: AnyObject?) -> Bool {
        return FBSDKApplicationDelegate.sharedInstance().application(application, openURL: url,
sourceApplication: sourceApplication, annotation: annotation)
    }

    // MARK: - Core Data stack

    lazy var applicationDocumentsDirectory: NSURL = {
        // The directory the application uses to store the Core Data store file. This code uses a
directory named "com.trustsourcing.LetsHookah" in the application's documents Application
Support directory.
        let urls = NSFileManager.defaultManager().URLsForDirectory(.DocumentDirectory,
inDomains: .UserDomainMask)
        return urls[urls.count-1] as NSURL
    }()

    lazy var managedObjectModel: NSManagedObjectModel = {
        // The managed object model for the application. This property is not optional. It is a fatal
error for the application not to be able to find and load its model.
        let modelURL = NSBundle.mainBundle().URLForResource("LetsHookah", withExtension:
"momd")!
        return NSManagedObjectModel(contentsOfURL: modelURL)!
    }()

    lazy var persistentStoreCoordinator: NSPersistentStoreCoordinator? = {
        // The persistent store coordinator for the application. This implementation creates and
return a coordinator, having added the store for the application to it. This property is optional
since there are legitimate error conditions that could cause the creation of the store to fail.
        // Create the coordinator and store
        var coordinator: NSPersistentStoreCoordinator? =
NSPersistentStoreCoordinator(managedObjectModel: self.managedObjectModel)
        let url =
self.applicationDocumentsDirectory.URLByAppendingPathComponent("LetsHookah.sqlite")
        var error: NSError? = nil
        var failureReason = "There was an error creating or loading the application's saved data."
        if coordinator!.addPersistentStoreWithType(NSSQLiteStoreType, configuration: nil, URL: url,
options: nil, error: &error) == nil {
            coordinator = nil
            // Report any error we got.
            var dict = [String: AnyObject]()
            dict[NSLocalizedDescriptionKey] = "Failed to initialize the application's saved data"
            dict[NSLocalizedFailureReasonErrorKey] = failureReason
            dict[NSUnderlyingErrorKey] = error
            error = NSError(domain: "YOUR_ERROR_DOMAIN", code: 9999, userInfo: dict)
            // Replace this with code to handle the error appropriately.
            // abort() causes the application to generate a crash log and terminate. You should not
use this function in a shipping application, although it may be useful during development.
            NSLog("Unresolved error \(error), \(error!.userInfo)")
            abort()
        }

        return coordinator
    }()

    lazy var managedObjectContext: NSManagedObjectContext? = {
        // Returns the managed object context for the application (which is already bound to the
```

```
persistent store coordinator for the application.) This property is optional since there are
legitimate error conditions that could cause the creation of the context to fail.
        let coordinator = self.persistentStoreCoordinator
        if coordinator == nil {
            return nil
        }
        var managedObjectContext = NSManagedObjectContext()
        managedObjectContext.persistentStoreCoordinator = coordinator
        return managedObjectContext
    }()

    // MARK: - Core Data Saving support

    func saveContext () {
        if let moc = self.managedObjectContext {
            var error: NSError? = nil
            if moc.hasChanges && !moc.save(&error) {
                // Replace this implementation with code to handle the error appropriately.
                // abort() causes the application to generate a crash log and terminate. You should not
use this function in a shipping application, although it may be useful during development.
                NSLog("Unresolved error \(error), \(error!.userInfo)")
                abort()
            }
        }
    }

}


import Foundation

enum commandOfferState {
    case Initial, Executing, Stopped
}

protocol GetOffersDelegate {
    func operationFinished(operation: GetOffersBase)
}

class GetOffersBase: NSObject, GetOffersCommandProtocol {

    private var state : commandOfferState = commandOfferState.Initial

    var delegate: GetOffersDelegate?

    internal var parameters: [String : AnyObject]?
    internal var response: (([Offer]) -> ())?
    internal var currentOperation: AFHTTPRequestOperation?
    internal var cancelFlag = false

    init(parameters: [String : AnyObject], response: ([Offer] -> ())) {
        super.init()
        self.parameters = parameters
        self.response = response
    }
```

```
    internal func operationFinished() {
        if delegate != nil {
            delegate!.operationFinished(self)
        }
    }

    func execute() {
        println("super execute")
        if state == commandOfferState.Executing {
            return
        }
        state = commandOfferState.Executing
    }

    func stop() {
        println("super stop")
        state = commandOfferState.Stopped
    }

        }
class MainMenuTableViewController: UITableViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        // Uncomment the following line to preserve selection between presentations
        // self.clearsSelectionOnViewWillAppear = false

        // Uncomment the following line to display an Edit button in the navigation bar for this view
controller.
        // self.navigationItem.rightBarButtonItem = self.editButtonItem()
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

    // MARK: - Table view data source

    override func numberOfSectionsInTableView(tableView: UITableView) -> Int {
        // #warning Potentially incomplete method implementation.
        // Return the number of sections.
        return 1
    }

    override func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        // #warning Incomplete method implementation.
        // Return the number of rows in the section.
        return 4
    }


    override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
        if segue.identifier == "specialOffer" {
```

```
            var backImage = UIImage(named: "back-icon")!
            var size = CGSizeMake(backImage.size.width, backImage.size.height + 8)
            UIGraphicsBeginImageContext(size)
            backImage.drawInRect(CGRectMake(0, 4, backImage.size.width,
backImage.size.height))
            var resultImage = UIGraphicsGetImageFromCurrentImageContext()
            UIGraphicsEndImageContext()
public class Reachability {

    class func isConnectedToNetwork()->Bool{

        var Status:Bool = false
        let url = NSURL(string: "http://google.com/")
        let request = NSMutableURLRequest(URL: url!)
        request.HTTPMethod = "HEAD"
        request.cachePolicy =
NSURLRequestCachePolicy.ReloadIgnoringLocalAndRemoteCacheData
        request.timeoutInterval = 10.0

        var response: NSURLResponse?

        var data = NSURLConnection.sendSynchronousRequest(request, returningResponse:
&response, error: nil) as NSData?

        if let httpResponse = response as? NSHTTPURLResponse {
            if httpResponse.statusCode == 200 {
                Status = true
            }
        }

        return Status
    }
}

class NetworkManager {

    class var sharedManager : NetworkManager {
        struct Static {
            static let instance : NetworkManager = NetworkManager()
        }
        return Static.instance
    }

    func receivePostDataFor(command:String, parameters:AnyObject?, success:(json: AnyObject)
-> (), failure:(json: NSError) -> ()) -> AFHTTPRequestOperation {
        return AFHTTPRequestOperationManager().POST(domain + command, parameters:
parameters, success: {(operation: AFHTTPRequestOperation!, responseObject: AnyObject!) in
            success(json: responseObject)
            }, failure: {(operation: AFHTTPRequestOperation!, error: NSError!) in
                UIAlertView(title: "Error", message: "There is no internet connection", delegate: nil,
cancelButtonTitle: "Cancel").show()
                failure(json: error)
        })
    }

    func receiveGetDataFor(command:String, parameters:AnyObject?, success:(json: AnyObject) -
```

```
> Void, failure:(json: AnyObject?) -> Void) -> AFHTTPRequestOperation {
        return AFHTTPRequestOperationManager().GET(domain + command, parameters:
parameters, success: {(operation: AFHTTPRequestOperation!, responseObject: AnyObject!) in
            success(json: responseObject)
            }, failure: {(operation: AFHTTPRequestOperation!, error: NSError!) in
                failure(json: ["error":error])
                UIAlertView(title: "Error", message: "There is no internet connection", delegate: nil,
cancelButtonTitle: "Cancel").show()
        })
    }

}

class ExploreTableViewCell: UITableViewCell {

@IBOutlet weak var customView: UIImageView!

    override func awakeFromNib() {
        super.awakeFromNib()



        distanceLabel.textColor = UIColor(red: 0.44, green: 0.75, blue: 0.66, alpha: 1)
    }

    override func setSelected(selected: Bool, animated: Bool) {
        super.setSelected(selected, animated: animated)


        // Configure the view for the selected state
    }

}

class Gallery: NSManagedObject {

    @NSManaged var originalLink: String
    @NSManaged var thumbLink: String
    @NSManaged var hookah: Hookah


    class func createInManagedObjectContext(moc: NSManagedObjectContext, originalLink:
String, thumbLink: String, hookah: Hookah) -> Gallery {

        var gallery = NSEntityDescription.insertNewObjectForEntityForName("Gallery",
inManagedObjectContext: moc) as? Gallery

        gallery?.originalLink = originalLink
        gallery?.thumbLink = thumbLink
        gallery?.hookah = hookah

        return gallery!
    }


}
```

```
class MenuViewController: UIViewController {

    @IBOutlet weak var avatarView: UIImageView!
    @IBOutlet weak var userNameLabel: UILabel!
    @IBOutlet weak var userSurnameLabel: UILabel!


    override func viewDidLoad() {
        super.viewDidLoad()
        setupAvatar(avatarView.layer)

    }

    func setupAvatar(layer : CALayer) {
        layer.masksToBounds = true
        layer.borderColor = UIColor(red: 0.44, green: 0.75, blue: 0.66, alpha: 1).CGColor
        layer.borderWidth = 1
        layer.cornerRadius = layer.frame.height / 2

    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

}

//
//  GetHookahFilterCommand.swift
//  LetsHookah
//
//  Created by Pavel Zagorskyy on 12.06.15.
//  Copyright (c) 2015 TrustSourcing. All rights reserved.
//

import Foundation
import CoreData


class GetOffersFilterCommand : GetOffersBase, GetOffersCommandProtocol {

    //    var locationManager:OneShotLocationManager?

    override func execute() {
        super.execute()
        println(__FUNCTION__ + "filter")
        //        locationManager = OneShotLocationManager()

        LocationManager.sharedManager.fetchWithCompletion({ (location) -> () in

            var amount = self.parameters!["amount"] as Int
            var longitude = location.coordinate.longitude
            var latitude = location.coordinate.latitude
            var radius = self.parameters!["radius"] as Double
```

```
            var parameters : [String : AnyObject] = [
                "longitude" : 30.508278,//longitude
                "latitude" : 50.441734,//latitude
                //              "radius" : radius
            ]

            self.currentOperation = NetworkManager.sharedManager.receiveGetDataFor("offers",
parameters: parameters, success: { (json) -> Void in //GetByLocation
                println(json)

                var data = json["data"]! as [[String : AnyObject]]
                var result : [Offer] = Array<Offer>()

                var backgroundContext = NSManagedObjectContext(concurrencyType:
NSManagedObjectContextConcurrencyType.PrivateQueueConcurrencyType)
                var mainContext = CoreDataManager.sharedManager.managedObjectContext
                //              backgroundContext.parentContext = mainContext
                var fetchRequest = NSFetchRequest(entityName: "Offer")
                fetchRequest.includesPendingChanges = true

var sortDescriptor = NSSortDescriptor(key: "id", ascending: false)
                fetchRequest.sortDescriptors = [sortDescriptor]
                fetchRequest.fetchLimit = amount
                fetchRequest.resultType = NSFetchRequestResultType.ManagedObjectIDResultType

                var error: NSError?
                var resultForDelete = mainContext?.executeFetchRequest(fetchRequest, error: &error)
                //              var resultForDelete = mainContext.executeFetchRequest(fetchRequest,
error: &error)
                if (error != nil) {
                    println(error!.localizedDescription)
                }
                if !self.cancelFlag {
                    var resultArray : [Offer] = Array<Offer>()
                    for offerId in resultForDelete as [NSManagedObjectID] {
                        var offer = mainContext?.objectWithID(offerId) as Offer
                        resultArray.append(offer)
                    }

                    resultArray.filter({ (offer) -> Bool in
                        return Bool(pow(longitude - (offer as Offer).hookah.longitude, 2) + pow(latitude -
(offer as Offer).hookah.latitude, 2) <= pow(radius, 2))
                    })

                    for offer in resultArray as [Offer] {
                        mainContext?.deleteObject(offer as Offer)
                    }
                }

                for dictionary in data {
                    var offer = SpecialOffersTableViewController().addOfferToChache(dictionary)
                    result.append(offer)
                }

                self.response!(result)
                self.operationFinished()
```

```
            }, failure: { (json) -> Void in


dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), { ()
-> Void in

                    var backgroundContext = NSManagedObjectContext(concurrencyType:
NSManagedObjectContextConcurrencyType.PrivateQueueConcurrencyType)
                    var mainContext = CoreDataManager.sharedManager.managedObjectContext
                    backgroundContext.parentContext = mainContext
                    var fetchRequest = NSFetchRequest(entityName: "Offer")
                    fetchRequest.includesPendingChanges = true

                    var longPlus = longitude + radius
                    var longMinus = longitude - radius
                    var latPlus = latitude + radius
                    var latMinus = latitude - radius

                    var squarePredicate = NSPredicate(format: "hookah.longitude
>=\(longMinus)","hookah.latitude>=\(latMinus)" , "hookah.longitude
<=\(longPlus)","hookah.longitude>=\(longMinus)")
                    fetchRequest.predicate = squarePredicate
                    var sortDescriptor = NSSortDescriptor(key: "id", ascending: false)
                    fetchRequest.sortDescriptors = [sortDescriptor]
                    fetchRequest.fetchLimit = amount
                    fetchRequest.resultType =
NSFetchRequestResultType.ManagedObjectIDResultType

                    var error: NSError?
                    var result = backgroundContext.executeFetchRequest(fetchRequest, error: &error)
                    if (error != nil) {
                        println(error!.localizedDescription)
                    }
                    if !self.cancelFlag {
                        dispatch_sync(dispatch_get_main_queue(), { () -> Void in

                            var resultArray : [Offer] = Array<Offer>()
                            for offerId in result as [NSManagedObjectID] {
                                var offer = mainContext?.objectWithID(offerId) as Offer
                                resultArray.append(offer)
                            }

                            resultArray.filter({ (offer) -> Bool in
                                return Bool(pow(longitude - (offer as Offer).hookah.longitude, 2) +
pow(latitude - (offer as Offer).hookah.latitude, 2) <= pow(radius, 2))
                            })

                            self.response!(resultArray)
                            self.operationFinished()
                        })
                    }
                })
            })

            }, error: { (error) -> () in
```

```
                self.response!(Array<Offer>())
                self.operationFinished()
                //                test
                UIAlertView(title: "Error", message: error.userInfo!["error"] as? String ?? "Unknown
error.", delegate: nil, cancelButtonTitle: "OK").show()
            })

        }
        override func stop() {
            super.stop()
            println(__FUNCTION__ + "filter")
            if self.currentOperation != nil {
                self.currentOperation!.cancel()
                self.currentOperation = nil
            }
            self.cancelFlag = true
        }
    }
```