ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Департамент программной инженерии

СОГЛАСОВАНО
Преподаватель департамента программной инженерии факультета компьютерных наук

_____ М. К. Горденко
«_____» _____ 2019 г.

УТВЕРЖДАЮ
Академический руководитель образовательной программы «Программная инженерия» профессор департамента программной инженерии, канд. техн. наук

_____ В. В. Шилов
«_____» _____ 2019 г.

IOS-ПРИЛОЖЕНИЕ «СОЦИАЛЬНАЯ СЕТЬ ДЛЯ СОТРУДНИКОВ НИУ ВШЭ»

Текст программы

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.04.03-01 12 01-1

Исполнитель: студент группы БПИ174
_____ Д. Ю. Редникина
«_____» _____ 2019 г.

Москва 2019

УТВЕРЖДЕН
RU.17701729.04.03-01 51 01-1

Инв. № подл	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

IOS-ПРИЛОЖЕНИЕ «СОЦИАЛЬНАЯ СЕТЬ ДЛЯ СОТРУДНИКОВ НИУ ВШЭ»

Программа и методика испытаний

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.04.03-01 51 01-1

Листов 175

Москва 2019

Содержание

1 Текст программы

```
//
//  Constants.swift
//  GDproject
//
//  Created by cstore on 13/02/2019.
//  Copyright  2019 drHSE. All rights reserved.
//

import UIKit
/// constant for cell in posts NewsController.swift
let postCellId = "PostCell"

///  constant for segmented controll in NewsController.swift and
   ChannelsController.swift
let headerNewsChannelsVC = "HeaderSegmentedControll"

///  for transitioning to full post view
let fullPostControllerId = "FullPostController"

/// constant for dequing basic cell
let basicInfoCellId = "BasicInfoCell"

/// constant for dequing info cell
let infoCellId = "InfoCell"

/// constant for switching to view controller for invitations
let inviteVC = "InviteViewController"

/// constant for dequing basic cell
let channelCellId = "ChannelCell"

/// constant for switching to channel list view controller for choosing
   channel
let channelListControllerId = "ChannelListController"

/// constant for dequing channel item cell
let itemCellId = "ItemCell"

/// constant for switching to view controller to add or edit one channel
let channelControllerId = "ChannelController"

/// constant for switching to news controller
let newsController = "NewsController"

/// constant for switching to view controller for log in app
let logInController = "LogInController"

/// constant for getting to messages
let messagesViewControllerId = "MessagesViewController"

/// constant for displaying list of people to send message to
let peopleToWriteVC = "PeopleToWriteViewController"

let dialogVC = "DialogViewController"

let profileViewController = "ProfileViewController"
```

```
let simplifiedChannelsList = "SimplifiedChannelsList"

let channelViewControllerId = "ChannelViewController"

let addToChannelVCId = "AddToChannelVC"

let taggsSelectionViewController = "TaggsSelectionViewController"

let dialogViewController = "DialogViewController"

let tagsSuggestionController = "TagsSuggestionController"

let chatInfoViewController = "ChatInfoViewController"

let blueSystemColor = UIColor(red: 0, green: 137/255, blue: 249/255, alpha
   : 0.5)

//
//  DataStorage.swift
//  GDproject
//
//  Created by cstore on 21/02/2019.
//  Copyright  2019 drHSE. All rights reserved.
//

import Foundation
import UIKit

/**
 _DataStorage_ class is aimed to store user private settings
 to display correct data after user leaving the app
 */
class DataStorage{

    private init(){}

    //weak var coordinator: LogInCoordinator?
    static let standard = DataStorage()

    /**
      Function for setting log in status of user
     */
    func setIsLoggedIn(value: Bool, with id: Int){
        UserDefaults.standard.set(value, forKey: UserDefaultsKeys.loggedIn
           .rawValue)
        setUserKey(with: id)
        isLoggedIn = UserDefaults.standard.bool(forKey: UserDefaultsKeys.
           loggedIn.rawValue)
    }

    func setUserKey(with id: Int){
        UserDefaults.standard.set(id, forKey: UserDefaultsKeys.id.rawValue
           )
    }

    func getUserId() -> Int {
        return UserDefaults.standard.integer(forKey: UserDefaultsKeys.id.
           rawValue)
```

```
    }

    func setEmail(email: String) {
        UserDefaults.standard.set(email, forKey: UserDefaultsKeys.email.
           rawValue)
    }

    func getEmail() -> String? {
        return UserDefaults.standard.string(forKey: UserDefaultsKeys.email
           .rawValue)
    }
    /**
     Function to determine is user logged in already or not
    */
    var isLoggedIn: Bool = UserDefaults.standard.bool(forKey:
       UserDefaultsKeys.loggedIn.rawValue)
}

/**
 Enum to store UserDefaults keys for getting and setting values
 */
enum UserDefaultsKeys: String{
    case loggedIn
    case id
    case email
}
//
//  Extentions.swift
//  GDproject
//
//  Created by cstore on 15/02/2019.
//  Copyright  2019 drHSE. All rights reserved.
//

import UIKit

extension UIImage{
    var roundedImage: UIImage {
        let rect = CGRect(origin:CGPoint(x: 0, y: 0), size: self.size)
        UIGraphicsBeginImageContextWithOptions(self.size, false, 1)
        UIBezierPath(
            roundedRect: rect,
            cornerRadius: self.size.height
            ).addClip()
        self.draw(in: rect)
        return UIGraphicsGetImageFromCurrentImageContext()!
    }
}

extension UIView
{
    func addConstraintsWithFormat(format: String, views: UIView...){
        var viewsDictionary = [String: UIView]()
        for (index, view) in views.enumerated(){
            let key = "v\(index)"
            viewsDictionary[key] = view
            view.translatesAutoresizingMaskIntoConstraints = false
        }
```

```
        addConstraints(NSLayoutConstraint.constraints(withVisualFormat:
            format, options: NSLayoutConstraint.FormatOptions(), metrics:
            nil, views: viewsDictionary))
    }
}


extension UIStoryboard {

    private static let userEdit = UIStoryboard(name: "Main", bundle: nil)

    class func makeChannelsListController() -> ChannelListController {
        return UIStoryboard.userEdit.instantiateViewController(
            withIdentifier: channelListControllerId) as!
            ChannelListController
    }

    class func makeNewsController() -> NewsController {
        return UIStoryboard.userEdit.instantiateViewController(
            withIdentifier: newsController) as! NewsController
    }

    class func tabBarController() -> UITabBarController{
        return UIStoryboard.userEdit.instantiateViewController(
            withIdentifier: "TabBar") as! UITabBarController
    }

    class func navRoot() -> UINavigationController{
        return UIStoryboard.userEdit.instantiateViewController(
            withIdentifier: "root") as! UINavigationController
    }

    class func makeLogIn() -> LogInViewController {
        return UIStoryboard.userEdit.instantiateViewController(
            withIdentifier: logInController) as! LogInViewController
    }
}


extension String {
    func getDate() -> String
    {
        let dateFormatter = DateFormatter()
        dateFormatter.dateFormat = "yyyy-MM-dd'T'HH:mm:ss.SSSZ"//this your
            string date format
        dateFormatter.timeZone = TimeZone(abbreviation: "UTC")
        let date = dateFormatter.date(from: self)

        dateFormatter.dateFormat = "MMM d, yyyy HH:mm" ///this is what you
            want to convert format
        dateFormatter.timeZone = NSTimeZone.local
        let timeStamp = dateFormatter.string(from: date!)

        return timeStamp
    }
}

extension UIViewController {
    func showAlertOn(result: ResultR) {
```

```
        let message: String

        switch result {
        case .impossibleContent:
            message = "Impossible content"
        case .exceededContent:
            message = "The content exceeded"
        case .longContent:
            message = "The content is too long"
        case .badAccess:
            message = "Try reloading the page again"
        case .alreadyRegistered:
            message = "User is already registered"
        case .tooMuchToAdd:
            message = "Limit of channels exceeded"
        case .success, .success1:
            return
        default:
            message = "Something went wrong"
        }

        let alert = UIAlertController(title: "Error", message: message,
           preferredStyle: .alert)
        alert.addAction(UIAlertAction(title: "OK", style: .default,
           handler: nil))
        self.present(alert, animated: true, completion: nil)
    }
}
//
//  MyStackView.swift
//  GDproject
//
//  Created by cstore on 14/02/2019.
//  Copyright  2019 drHSE. All rights reserved.
//

import UIKit

class MyStackView: UIStackView {

    var isFull: Bool = true


    override var bounds: CGRect {
        didSet{
            if frame.height == 300 && !isFull {
                let gradientLayer = CAGradientLayer()
                gradientLayer.colors = [UIColor(displayP3Red: 255, green:
                   255, blue: 255, alpha: 0).cgColor, UIColor.white.
                   cgColor]
                gradientLayer.startPoint = CGPoint(x: 0.5, y: 0.5)
                gradientLayer.frame = self.bounds
                self.layer.addSublayer(gradientLayer)
            }
        }
    }
}
//
```

```
//  MainController.swift
//  GDproject
//
//  Created by cstore on 21/02/2019.
//  Copyright  2019 drHSE. All rights reserved.
//

import UIKit

class MainController: UINavigationController {

    override func viewDidLoad() {
        super.viewDidLoad()

//        if false {
//            perform(#selector(showLogInController), with: nil,
   afterDelay: 0.01)
//        } else {
//            perform(#selector(fhfh), with: nil, afterDelay: 0.01)
//
//        }
    }

//    @objc func showLogInController(){
//        let vc = storyboard!.instantiateViewController(withIdentifier:
   logInController) as! LogInViewController
//        present(vc, animated: true) {
//            print("opportunity to log in")
//        }
//    }

//    @objc func fhfh(){
//
//        let newsController = NC()
//        viewControllers = [newsController]
//    }
//    override func viewDidAppear(_ animated: Bool) {
//        super.viewDidAppear(animated)
//
//
//    }
}



//
//  ViewController.swift
//  GDproject
//
//  Created by cstore on 13/02/2019.
//  Copyright  2019 drHSE. All rights reserved.
//

import UIKit
import TinyConstraints

// MARK:- Controller with posts and channels availiable.
// Search is availiable within every table (posts and channels). Has
   button-functionality for boths post and chnnels
```

```
class NewsController: UIViewController
{

    @IBOutlet weak var tableView: UITableView!

    var searchController = UISearchController(searchResultsController: nil
        )



    override func viewDidLoad()
    {
        super.viewDidLoad()
        tableView.register(HeaderNewsChannels.self, forCellReuseIdentifier
            : headerNewsChannelsVC)
        tableView.register(PostViewCell.self, forCellReuseIdentifier:
            postCellId)

        tableView.delegate = self
        tableView.dataSource = self
        tableView.reloadData()

        setUpNavigationItemsforPosts()
        searchController.searchBar.placeholder  = "Search anything"

        navigationItem.searchController = searchController
        definesPresentationContext = true

        setUpBanner()
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        navigationController?.navigationBar.prefersLargeTitles = true
    }

    func setUpNavigationItemsforPosts(){
        navigationItem.title = "Posts"
        navigationItem.rightBarButtonItems = [UIBarButtonItem(
            barButtonSystemItem: .compose, target: self, action: #selector(
            self.writePost(_:)))]
    }

    func setUpNavigationItemsForChannels(){
        navigationItem.title = "Channels"
        navigationItem.rightBarButtonItems = [UIBarButtonItem(
            barButtonSystemItem: .add, target: self, action: #selector(
            addChannel))]
    }

    @objc func writePost(_ barItem: UIBarButtonItem)
    {
        let vc = storyboard?.instantiateViewController(withIdentifier: "
            NewPostViewController") as! NewPostViewController

        vc.navigationItem.rightBarButtonItem = UIBarButtonItem(title: "
            Post", style: .plain, target: vc, action: #selector(vc.newPost)
            )
```

```
        vc.navigationItem.leftBarButtonItem = UIBarButtonItem(title: "
            Close", style: .plain, target: vc, action: #selector(vc.
            closeView))

        vc.parentVC = self

        let transition = CATransition()
        transition.duration = 0.5
        transition.timingFunction = CAMediaTimingFunction(name:
            CAMediaTimingFunctionName.easeInEaseOut)
        transition.type = CATransitionType.moveIn
        transition.subtype = CATransitionSubtype.fromTop
        navigationController?.view.layer.add(transition, forKey: nil)
        navigationController?.pushViewController(vc, animated: false)
    }

    @objc func addChannel(){
        print("Add channel")
    }

    // for animating the banner
    var topConstraint: NSLayoutConstraint?

    let bannerView: UIView = {
        let view = UIView()
        view.backgroundColor = .blue
        view.layer.cornerRadius = 25.0
        view.clipsToBounds = true
        return view
    }()

    let statusLabel: UILabel = {
        let label = UILabel()
        label.text = "-"
        label.textColor = .white
        label.textAlignment = .center
        return label
    }()

    private func setUpBanner()
    {
        view.addSubview(bannerView)
        bannerView.addSubview(statusLabel)
        statusLabel.edgesToSuperview()

        let tap = UITapGestureRecognizer(target: self, action: #selector(
            handleTap(sender:)))

        bannerView.addGestureRecognizer(tap)
        topConstraint = NSLayoutConstraint(item: bannerView, attribute: .
            top, relatedBy: .equal, toItem: view.safeAreaLayoutGuide,
            attribute: .top, multiplier: 1, constant: -300)
        view.addConstraint(topConstraint!)

        bannerView.edgesToSuperview(excluding: [.bottom, .top, .left],
            insets: .right(20), usingSafeArea: true)
        bannerView.height(50)
        bannerView.width(50)
    }
```

```
    // when table is scrolling no deletion is availiable
    @objc func handleTap(sender: UITapGestureRecognizer? = nil) {
        let indexPath = IndexPath(row: 0, section: 0)
        self.tableView.scrollToRow(at: indexPath, at: .top, animated: true
            )
//        isBannerVisible = false
//        changeConstraint(isVisible: false)
    }

    // animation for banner
    func changeConstraint(isVisible: Bool){
        topConstraint?.constant =  isVisible ? 50 : -300

        UIView.animate(withDuration: 0.5, delay: 0, options: .curveEaseOut
            , animations: {
             self.view.layoutIfNeeded()
        }) { (completed) in

        }
    }

    var isBannerVisible: Bool = false

     func scrollViewDidScroll(_ scrollView: UIScrollView) {
        print(scrollView.contentOffset.y)
        if scrollView.contentOffset.y >= 50 && !isBannerVisible{
            isBannerVisible = true
            changeConstraint(isVisible: true)
        }

        if isBannerVisible && scrollView.contentOffset.y == 0{
            isBannerVisible = false
            changeConstraint(isVisible: false)
        }
    }
}

// MARK:- tableView delegate
extension NewsController: UITableViewDelegate{
    func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
        IndexPath)
    {
        tableView.deselectRow(at: indexPath, animated: true)

        let vc = storyboard!.instantiateViewController(withIdentifier:
            fullPostControllerId) as! FullPostController

        vc.post = dataSourse[indexPath.row]
        navigationController!.pushViewController(vc, animated: true)
    }
}

// MARK:- tableView dataSourse
extension NewsController: UITableViewDataSource
{
    func tableView(_ tableView: UITableView, numberOfRowsInSection section
        : Int) -> Int {
        return dataSourse.count
```

```
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath:
       IndexPath) -> UITableViewCell
    {
        let cell = tableView.dequeueReusableCell(withIdentifier:
           postCellId, for: indexPath) as! PostViewCell

        cell.fill(with: dataSourse[indexPath.row].dataArray, false)
        cell.selectionStyle = .none
        return cell
    }

    func tableView(_ tableView: UITableView, heightForRowAt indexPath:
       IndexPath) -> CGFloat {
        return UITableView.automaticDimension
    }

    func tableView(_ tableView: UITableView, estimatedHeightForRowAt
       indexPath: IndexPath) -> CGFloat {
        return 100.0
    }

    func tableView(_ tableView: UITableView, heightForHeaderInSection
       section: Int) -> CGFloat {
        return 46.0
    }

    func tableView(_ tableView: UITableView, viewForHeaderInSection
       section: Int) -> UIView?
    {
        let cell = tableView.dequeueReusableCell(withIdentifier:
           headerNewsChannelsVC) as! HeaderNewsChannels
        cell.vc = self

        let view = UIView(frame: CGRect(x: 0, y: 0, width: UIScreen.main.
           bounds.width, height: 46.0))
        view.addSubview(cell)
        cell.edgesToSuperview()

        return view
    }


}
//
//  ChannelsCoordinator.swift
//  RxSwift
//
//  Created by cstore on 03/03/2019.
//  Copyright  2019 drHSE. All rights reserved.
//

import Foundation
import UIKit
import Pulley

class ChannelsCoordinator: BaseCoordinator{
    let storyboard = UIStoryboard(name: "Main", bundle: nil)
```

```
    private weak var navigationController: UINavigationController?

    init(nc: UINavigationController) {
        self.navigationController = nc
        navigationController?.navigationItem.largeTitleDisplayMode = .
            always
    }

    override func start() {
        show()
    }

    private func show() {

        let channels = storyboard.instantiateViewController(withIdentifier
            : channelListControllerId) as! ChannelListController

        channels.onChannelSelected = { [unowned self]
            (channel) in
            self.navigationController?.pushViewController(self.
                presentNewsController(with: channel), animated: true)
        }

        channels.askForUpdates()

        channels.onEditingModeBegins = { [unowned self] (channel,
            indexPath) in
            let vc = self.presentChannelController(with: channel)
            self.navigationController?.pushViewController(vc, animated:
                true)
            vc.tabBarController?.tabBar.isHidden = true
        }

        let nc = presentNewsController()
        navigationController?.setViewControllers([channels, nc], animated:
            false)
    }

    /// Function that presents channel controller
    ///
    /// - Parameter channel: channel that needs to be displayed
    func presentChannelController(with channel: Model.Channels? = nil) ->
        ChannelViewController {
        let mainContentVC = storyboard.instantiateViewController(
            withIdentifier: channelViewControllerId) as!
            ChannelViewController

        // to show preview we need to instantiate newsController but with
            different functionality
        mainContentVC.onShowingPreview = { [weak mainContentVC, unowned
            self] ch in
            let vc = self.presentNewsController(with: ch, previewMode:
                true)
            mainContentVC?.navigationController?.pushViewController(vc,
                animated: true)
        }

        // to go for choosing hashtags
```

```
        mainContentVC.onChoosingHashTags = { [weak mainContentVC, unowned
            self] ch in
            let vc = self.storyboard.instantiateViewController(
                withIdentifier: taggsSelectionViewController) as!
                TaggsSelectionViewController

            vc.currentTags = ch
            vc.receiver = mainContentVC

            self.navigationController?.pushViewController(vc, animated:
                true)
        }

        mainContentVC.onChoosingPeople = { [weak mainContentVC, unowned
            self] channel in
            let vc = self.storyboard.instantiateViewController(
                withIdentifier: addToChannelVCId) as! AddToChannelVC

            vc.channel = channel
            vc.update = mainContentVC

            self.navigationController?.pushViewController(vc, animated:
                true)
        }

        mainContentVC.channel = channel
        return mainContentVC
    }

    func presentNewsController(with channel: Model.Channels? = nil,
        previewMode: Bool = false) -> NewsController
    {
        let mainContentVC = storyboard.instantiateViewController(
            withIdentifier: newsController) as! NewsController
        mainContentVC.channel = channel

        if !previewMode {
            mainContentVC.news.onFullPost = {
                [weak self] (type,post) in

                let vc = self?.storyboard.instantiateViewController(
                    withIdentifier: fullPostControllerId) as!
                    FullPostController
                vc.type = type
                vc.post = post
                self?.navigationController!.pushViewController(vc,
                    animated: true)
            }

            mainContentVC.news.onChannelDidChange = {
                print("anon with \($0.0.count) users")
            }

            mainContentVC.navigationItem.rightBarButtonItems = [
                UIBarButtonItem(barButtonSystemItem: .compose, target: self
                , action: #selector(writePost(_:)))
            ]
        }
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.03-01 ТЗ 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

```
        mainContentVC.changedChannelName = {
            [weak mainContentVC] (title) in mainContentVC?.navigationItem.
               title = title
        }

        return mainContentVC
    }

    @objc private func writePost(_ barItem: UIBarButtonItem)
    {
        let vc = storyboard.instantiateViewController(withIdentifier: "
           NewPostViewController") as! NewPostViewController

        vc.navigationItem.rightBarButtonItem = UIBarButtonItem(title: "
           Post", style: .plain, target: vc, action: #selector(vc.newPost)
           )
        vc.navigationItem.leftBarButtonItem = UIBarButtonItem(title: "
           Close", style: .plain, target: vc, action: #selector(vc.
           closeView))

        let transition = CATransition()
        transition.duration = 0.5
        transition.timingFunction = CAMediaTimingFunction(name:
           CAMediaTimingFunctionName.easeInEaseOut)
        transition.type = CATransitionType.moveIn
        transition.subtype = CATransitionSubtype.fromTop
        navigationController?.view.layer.add(transition, forKey: nil)
        navigationController?.pushViewController(vc, animated: false)

        vc.moveBackToParentVC = {
            [weak self] in

            let transition = CATransition()
            transition.duration = 0.5
            transition.timingFunction = CAMediaTimingFunction(name:
               CAMediaTimingFunctionName.easeInEaseOut)
            transition.type = CATransitionType.reveal
            transition.subtype = CATransitionSubtype.fromBottom
            self?.navigationController?.view.layer.add(transition, forKey:
                nil)
            self?.navigationController?.popViewController(animated: false)
        }
    }
}
//
//  ProfileCoordinator.swift
//  RxSwift
//
//  Created by cstore on 02/03/2019.
//  Copyright  2019 drHSE. All rights reserved.
//

import Foundation
import UIKit

class ProfileCoordinator: BaseCoordinator {

    var didEndSession: (()->())?
```

```
    private weak var navigationController: UINavigationController?

    init(nc: UINavigationController) {
        self.navigationController = nc
    }

    override func start() {
        show()
    }

    private func show() {
        let storyboard = UIStoryboard(name: "Main", bundle: nil)
        let profile = storyboard.instantiateViewController(withIdentifier:
            "ProfileViewController") as! ProfileViewController

        profile.logOut  = {
            Model.logout() {
                [weak self] in

                DataStorage.standard.setIsLoggedIn(value: false, with: 0)
                self?.didEndSession?()
            }
        }

        profile.deleteAllSessions = {
            Model.deactivateAll() {
                [weak self] in

                DataStorage.standard.setIsLoggedIn(value: false, with: 0)
                self?.didEndSession?()
            }
        }

        profile.onSettings = { [weak self, weak storyboard, weak profile]
           in

            let vc = storyboard?.instantiateViewController(withIdentifier:
                "RegisterTableViewController") as!
               RegisterTableViewController
            vc.delegate = profile
            vc.userActive = profile?.user

            self?.navigationController?.pushViewController(vc, animated:
               true)
        }


        navigationController?.setViewControllers([profile], animated:
           false)
    }
}
//
//  ApplicationCoordinator.swift
//  RxSwift
//
//  Created by cstore on 04/03/2019.
//  Copyright   2019 drHSE. All rights reserved.
//
```

```
import Foundation
import UIKit



fileprivate enum LaunchInstructor {
    case main, auth

    static func configure(
        isAutorized: Bool = DataStorage.standard.isLoggedIn) ->
           LaunchInstructor {

        if isAutorized{
            return .main
        } else {
            return .auth
        }
    }
}

final class ApplicationCoordinator: BaseCoordinator{
    private var window: UIWindow!

    init(window: UIWindow) {
        self.window = window
    }

    private var instructor: LaunchInstructor {
        return LaunchInstructor.configure()
    }

    override func start() {
        switch instructor {
        case .auth: runAuthFlow()
        case .main: runMainFlow()
        }
    }

    private func runAuthFlow() {
        let coordinator = LogInCoordinator(window: window)
        coordinator.didEndFlow = { [weak self, weak coordinator] in
            self?.start()
            self?.removeDependency(coordinator)
        }
        addDependency(coordinator)
        coordinator.start()
    }

    private func runMainFlow() {
        let storyboard = UIStoryboard(name: "Main", bundle: nil)
        let tabBar = storyboard.instantiateViewController(withIdentifier:
           "TabbarController") as! TabbarController
        let coordinator = TabBarCoordinator(tabbarView: tabBar, window:
           window!)

        coordinator.didEndFlow = { [weak self, weak coordinator] in
            self?.start()
            self?.removeDependency(coordinator)
        }
```

```
        addDependency(coordinator)
        coordinator.start()
    }
}
//
//  LogInCoordinator.swift
//  RxSwift
//
//  Created by cstore on 01/03/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit

class LogInCoordinator: BaseCoordinator {

    let storyboard = UIStoryboard(name: "Main", bundle: nil)

    var didEndFlow: (()->())?
    var window: UIWindow!
    var navigationController: UINavigationController?

    init(window: UIWindow) {
        self.window = window
        self.window?.rootViewController = UINavigationController()
        self.navigationController = window.rootViewController as?
           UINavigationController
    }

    override func start(){
        let logInVC = LogInViewController()
        logInVC.authenticate = { [weak self, weak logInVC] (email) in

            DataStorage.standard.setEmail(email: email)
            Model.authenticate(with: email) { [weak self]
                (authStatus) in

                print(authStatus.userStatus)

                if (authStatus.userStatus == "invalid") {
                    logInVC?.presentAlertInvalidCode()
                }
                else if (authStatus.userStatus == "canRegister") { //
                   register form
                    self?.presentRegisterVC()
                } else { // validation code
                    self?.presentViewControllerForCodeInput()
                }
            }
        }

        navigationController?.pushViewController(logInVC, animated: false)
    }

    func presentViewControllerForCodeInput()
    {
        let vc = storyboard.instantiateViewController(withIdentifier: "
           CodeViewController") as! CodeViewController
```

```
        self.navigationController?.pushViewController(vc, animated: true)
        vc.onSuccessLogIn = {
            [weak self] in self?.didEndFlow?()
        }
    }

    func presentRegisterVC(){

        let vc1 = storyboard.instantiateViewController(withIdentifier: "
           RegisterTableViewController") as! RegisterTableViewController

        vc1.onRegistration = { [weak self] in
            self?.presentViewControllerForCodeInput()
        }

        self.navigationController?.pushViewController(vc1, animated: true)
    }


}
//
//  TabBarCoordinator.swift
//  RxSwift
//
//  Created by cstore on 02/03/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import Foundation
import UIKit


class TabBarCoordinator: BaseCoordinator {

    var didEndFlow: (()->())?

    private let tabbarView: TabbarView
    private weak var window: UIWindow?

    init(tabbarView: TabbarView, window: UIWindow) {
        self.tabbarView = tabbarView
        self.window = window
    }

    override func start() {
        tabbarView.onViewDidLoad = runChannelsFlow()
        tabbarView.onChannelsFlowSelect = runChannelsFlow()
        tabbarView.onProfileFlowSelect = runProfileFlow()
        tabbarView.onMessagesFlowSelect = runMessagesFlow()
        window?.rootViewController = tabbarView as! TabbarController
    }

    private func runChannelsFlow() -> ((UINavigationController) -> ())
    {
        return { [unowned self] navController in
            if navController.viewControllers.isEmpty == true {
                let channelCoordinator = ChannelsCoordinator(nc:
                   navController)
                self.addDependency(channelCoordinator)
```

```
                channelCoordinator.start()
            }
        }
    }

    private func runMessagesFlow() -> ((UINavigationController) -> ()){
        return { [unowned self] navController in
            if navController.viewControllers.isEmpty == true {
                let messagesCoordinator = MessagesCoordinator(nc:
                   navController)
                self.addDependency(messagesCoordinator)
                messagesCoordinator.start()
            }
        }
    }

    private func runProfileFlow() -> ((UINavigationController) -> ())
    {
        return { [unowned self] navController in
            if navController.viewControllers.isEmpty == true {
                let profileCoordinator = ProfileCoordinator(nc:
                   navController)
                profileCoordinator.didEndSession = { [weak self, weak
                   profileCoordinator] in
                    self?.removeDependency(profileCoordinator)
                    self?.didEndFlow?()
                }
                self.addDependency(profileCoordinator)
                profileCoordinator.start()
            }
        }
    }
}
//
//  MessagesCoordinator.swift
//  GDproject
//
//  Created by cstore on 01/05/2019.
//  Copyright   2019 drHSE. All rights reserved.
//


import Foundation
import UIKit

class MessagesCoordinator: BaseCoordinator {

    var didEndSession: (()->())?

    let storyboard = UIStoryboard(name: "Main", bundle: nil)
    private weak var navigationController: UINavigationController?

    init(nc: UINavigationController) {
        self.navigationController = nc
    }

    override func start() {
        show()
    }
```

```
    private func show(){
        let vc = storyboard.instantiateViewController(withIdentifier:
           messagesViewControllerId) as! MessagesViewController

        // choose person to write message to
        vc.onUserDisplayList = { [weak vc, unowned self] in

            let newVC = self.storyboard.instantiateViewController(
               withIdentifier: peopleToWriteVC) as!
               PeopleToWriteViewController

            newVC.whatToDoWithSelection = { [weak newVC, weak self] people
                in
                newVC?.navigationController?.popViewController(animated:
                   true)

                // detect is it a user or group dialog
                let count = people.count
                if count == 1 {

                    let user = people.first!.key
                    let createdDialog = Model.Dialog.userChat(Model.
                       UserChat(user: user))

                    let vc = DialogViewController()
                    vc.users = Model.Channels.fullPeopleDict
                    vc.dialog = createdDialog
                    self?.navigationController?.pushViewController(vc,
                       animated: true)

                } else {
                    var group = Model.Group(users: people, name: "Untitled
                       ", id: 0)
                    group.users[DataStorage.standard.getUserId()] = Model.
                       UserPermission(isAdmin: true)

                    Model.createGroupChat(from: group, completion: { [weak
                        self] id in

                            let newVC1 = DialogViewController()
                            group.id = id
                            newVC1.users = Model.Channels.fullPeopleDict
                            newVC1.dialog = Model.Dialog.groupChat(Model.
                               GroupChat(group: group))
                            self?.navigationController?.pushViewController
                               (newVC1, animated: true)
                    })
                }
            }

            vc?.navigationController?.pushViewController(newVC, animated:
               true)
        }

        vc.onDialogDisplay = { [weak vc]  in

            let newVC = DialogViewController()
            newVC.dialog = $0.dialog
```

```
            newVC.users = $0.users
            newVC.onUserDisplay = { [weak self] id in

                let vc = self?.storyboard.instantiateViewController(
                    withIdentifier: profileViewController) as!
                    ProfileViewController
                vc.idProfile = id
                self?.navigationController!.pushViewController(vc,
                    animated: true)

            }

            vc?.navigationController?.pushViewController(newVC, animated:
                true)
        }

        navigationController?.viewControllers = [vc]
    }
}
//
//  BaseCoordinator.swift
//  RxSwift
//
//  Created by cstore on 02/03/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import Foundation
import UIKit

protocol Coordinator: class {
    func start()
}

class BaseCoordinator: Coordinator {

    var childCoordinators: [Coordinator] = []

    func start() {

    }

    // add only unique object
    func addDependency(_ coordinator: Coordinator) {
        guard !childCoordinators.contains(where: { $0 === coordinator })
            else {  return }
        childCoordinators.append(coordinator)
    }

    func removeDependency(_ coordinator: Coordinator?) {
        guard
            childCoordinators.isEmpty == false,
            let coordinator = coordinator
            else { return }

        // Clear child-coordinators recursively
        if let coordinator = coordinator as? BaseCoordinator, !coordinator
            .childCoordinators.isEmpty {
            coordinator.childCoordinators
```

```
                .filter({ $0 !== coordinator })
                .forEach({ coordinator.removeDependency($0) })
        }
        for (index, element) in childCoordinators.enumerated() where
           element === coordinator {
            childCoordinators.remove(at: index)
            break
        }
    }
}
//
//  TabbarController.swift
//  RxSwift
//
//  Created by cstore on 03/03/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import Foundation
import UIKit

protocol TabbarView: class {
    var onChannelsFlowSelect: ((UINavigationController) -> ())? { get set
       }
    var onProfileFlowSelect: ((UINavigationController) -> ())? { get set }
    var onMessagesFlowSelect: ((UINavigationController) -> ())? { get set
       }
    var onViewDidLoad: ((UINavigationController) -> ())? { get set }
}

final class TabbarController: UITabBarController,
   UITabBarControllerDelegate, TabbarView {


    var onChannelsFlowSelect: ((UINavigationController) -> ())?
    var onProfileFlowSelect: ((UINavigationController) -> ())?
    var onMessagesFlowSelect: ((UINavigationController) -> ())?

    var onViewDidLoad: ((UINavigationController) -> ())?

    override func viewDidLoad() {
        super.viewDidLoad()

        delegate = self
        if let controller = customizableViewControllers?.first as?
           UINavigationController {
            onViewDidLoad?(controller)
        }
    }

    func tabBarController(_ tabBarController: UITabBarController,
       didSelect viewController: UIViewController)
    {

        guard let controller = viewControllers?[selectedIndex] as?
           UINavigationController else { return }

        if selectedIndex == 0 {
            onChannelsFlowSelect?(controller)
```

```
        } else if selectedIndex == 2 {
            onProfileFlowSelect?(controller)
        } else {
            onMessagesFlowSelect?(controller)
        }
    }
}
//
//  MessageViewCell.swift
//  GDproject
//
//  Created by cstore on 13/05/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit

class MessageViewCell: UITableViewCell {

    @IBOutlet weak var dialogName: UILabel!

    @IBOutlet weak var lastMessagePreview: UILabel!

    @IBOutlet weak var dateLabel: UILabel!

    override func awakeFromNib() {
        super.awakeFromNib()
        // Initialization code
    }

    func fill(with dialog: Model.Dialog, user: Model.Users? = nil)
    {
        switch dialog {

        case .groupChat(let group):
            dialogName.text = group.group.name
            lastMessagePreview.text = group.lastMessage?.body.markdown
            dateLabel.text = group.lastMessage?.time.getDate()
        case .userChat(let userChat):
            dialogName.text = " \(user!.fullName())"
            lastMessagePreview.text = userChat.lastMessage?.body.markdown
            dateLabel.text = userChat.lastMessage?.time.getDate()
        }
    }
}
//
//  PeopleToWriteViewController.swift
//  GDproject
//
//  Created by cstore on 01/05/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit

class PeopleToWriteViewController: UITableViewController {

    // TODO: - edit button when it's used for selection
    var whatToDoWithSelection: (([Int: Model.UserPermission])->())?
```

```
let searchC = UISearchController(searchResultsController: nil)

let users = Model.Channels.fullPeople

var chosenUsers: [Int: Model.UserPermission] = [:]

override func viewDidLoad() {
    super.viewDidLoad()

    tableView.isEditing = true
    self.navigationItem.title = "People"

    self.navigationItem.rightBarButtonItem = UIBarButtonItem(
       barButtonSystemItem: .done, target: self, action: #selector(
       newMessage))

    self.navigationItem.largeTitleDisplayMode = .never
    self.navigationItem.searchController = searchC
    self.navigationItem.hidesSearchBarWhenScrolling = false
}

override func setEditing(_ editing: Bool, animated: Bool) {
    super.setEditing(true, animated: animated)
}

override func numberOfSections(in tableView: UITableView) -> Int {
    return 1
}

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

override func tableView(_ tableView: UITableView, didSelectRowAt
   indexPath: IndexPath) {
    chosenUsers[users[indexPath.row].id] = Model.UserPermission(
       isAdmin: false)
}

override func tableView(_ tableView: UITableView, cellForRowAt
   indexPath: IndexPath) -> UITableViewCell
{
    let cell = tableView.dequeueReusableCell(withIdentifier: "
       peopleToWriteCell", for: indexPath)

    cell.textLabel?.text = users[indexPath.row].fullName()

    return cell
}

@objc func newMessage()
{
    if chosenUsers.count != 0
    {
        whatToDoWithSelection?(chosenUsers)
    }
}
```

```
}
//
//  DialogViewController.swift
//  GDproject
//
//  Created by cstore on 01/05/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit
import TinyConstraints
import Marklight

protocol UpdatableGroup:class  {
    func updateGroup(with group: Model.Group)
}

class DialogViewController: UIViewController, UpdatableGroup,
   UITableViewDelegate, UITableViewDataSource
{
    var onUserDisplay: ((Int)->())?

    var tableView: UITableView = UITableView()

    func updateGroup(with group: Model.Group){
        self.groupChat?.group = group
        setTitleForGroup(groupChat: groupChat!)
    }

    let cellId = "cell3"
    var onInfoShow: (()->())?

    var dialog: Model.Dialog? {
        didSet{
            switch dialog! {
            case .groupChat(let chat):
                self.groupChat = chat
            case .userChat(let chat):
                self.userChat = chat
            }
        }
    }

    var groupChat: Model.GroupChat?
    var userChat: Model.UserChat?

    var users: [Int: Model.Users]?

    var groupId: Int?

    var cellData: [PostCellData] = [] {
        didSet {
            tableView.reloadData()
            canBePaginated = true
        }
    }

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

```
        setConstraints()
        magicForKeyboardChanges()

        self.tableView.delegate = self
        self.tableView.dataSource = self

        navigationItem.largeTitleDisplayMode = .never

        tableView.keyboardDismissMode = .onDrag
        tableView.separatorStyle = .none
        tableView.register(DialogCell.self, forCellReuseIdentifier: cellId
            )
        tableView.transform = CGAffineTransform(scaleX: 1, y: -1)
        tableView.contentInsetAdjustmentBehavior = .never
        tableView.contentOffset = CGPoint(x: 0, y: 30)

        if let groupChat = groupChat {
            setTitleForGroup(groupChat: groupChat)
        } else if let userChat = userChat{
            setTitleForChat(userChat: userChat)
        }
    }

    var messageSendView: UIView = {
        let view = UIView()
        view.backgroundColor = #colorLiteral(red: 1.0, green: 1.0, blue:
            1.0, alpha: 1.0)
        return view
    }()

    var messageTextView: UITextView =
    {
        let textStorage = MarklightTextStorage()
        textStorage.marklightTextProcessor.codeColor = UIColor.orange
        textStorage.marklightTextProcessor.quoteColor = UIColor.darkGray
        textStorage.marklightTextProcessor.syntaxColor = UIColor.blue
        textStorage.marklightTextProcessor.codeFontName = "Courier"
        textStorage.marklightTextProcessor.fontTextStyle = UIFont.
            TextStyle.subheadline.rawValue
        textStorage.marklightTextProcessor.hideSyntax = false

        let layoutManager = NSLayoutManager()

        // Assign the `UITextView`'s `NSLayoutManager` to the `
            NSTextStorage` subclass
        //textStorage.addLayoutManager(textView.layoutManager)
        textStorage.addLayoutManager(layoutManager)

        let textContainer = NSTextContainer()
        layoutManager.addTextContainer(textContainer)

        let textView = UITextView(frame: CGRect.zero, textContainer:
            textContainer)
        textView.isEditable = true
        textView.isScrollEnabled = true
        textView.backgroundColor = .white
        return textView
    }()
```

```
var sendButton: UIButton = {
    let button = UIButton()
    button.setTitle("Send", for: .normal)
    button.setTitleColor(.blue, for: .normal)
    button.addTarget(self, action: #selector(sendMessage), for: .
       touchUpInside)
    return button
}()

@objc func sendMessage()
{
    var destination: Model.MessageDestination?

    if let group = groupChat {
        destination = Model.MessageDestination.groupChatDestination(
           group.group.id)
    } else if let user = userChat {
        destination = Model.MessageDestination.userChatDestination(
           user.user)
    }

    if let destination = destination
    {
        Model.sendMessage(message: Model.SendMessage(body: Model.
           Attachments(markdown: messageTextView.text), destination:
           destination)) { [unowned self] in

            switch $0 {
            case .success, .success1:
                self.getMessagesNew(for: self.dialog!)
                self.messageTextView.text = ""
            default:
                self.showAlertOn(result: $0)
            }
        }

        prevLast = -1
    }
}

var lineView: UIView = {
    let view = UIView()
    view.backgroundColor = #colorLiteral(red: 0.6000000238, green:
       0.6000000238, blue: 0.6000000238, alpha: 1)
    return view
}()

var bottomConstraint: NSLayoutConstraint!

func setConstraints(){
    self.view.addSubview(tableView)
    self.view.addSubview(messageSendView)

    messageSendView.addSubview(messageTextView)
    messageSendView.addSubview(sendButton)
    messageSendView.addSubview(lineView)

    messageSendView.edgesToSuperview(excluding: [.top,.bottom])
```

```
        bottomConstraint = NSLayoutConstraint(item: messageSendView,
            attribute: .bottom, relatedBy: .equal, toItem: self.view.
            safeAreaLayoutGuide, attribute: .bottom, multiplier: 1,
            constant: 0)
        view.addConstraint(bottomConstraint)

        messageSendView.height(60)

        sendButton.edgesToSuperview(excluding: .left)
        sendButton.width(60)

        lineView.edgesToSuperview(excluding: .bottom)
        lineView.height(0.5)

        messageTextView.edgesToSuperview(excluding: .right)
        messageTextView.rightToLeft(of: sendButton)

        tableView.edgesToSuperview(excluding: .bottom)
        tableView.bottomToTop(of: messageSendView)

        self.view.layoutSubviews()
    }

    func magicForKeyboardChanges()
    {
        NotificationCenter.default.addObserver(self, selector: #selector(
            handleKeyboardNotifications), name: UIResponder.
            keyboardWillShowNotification, object: nil)

        NotificationCenter.default.addObserver(self, selector: #selector(
            handleKeyboardNotifications), name: UIResponder.
            keyboardWillHideNotification, object: nil)
    }

    @objc func handleKeyboardNotifications(notification: NSNotification){
        if let userInfo = notification.userInfo{
            // UIKeyboardFrameEndUserInfoKey

            let keyBoardFrame = (userInfo[UIResponder.
                keyboardFrameEndUserInfoKey] as! NSValue).cgRectValue

            bottomConstraint.constant = notification.name == UIResponder.
                keyboardWillShowNotification ? -(keyBoardFrame.height) +
                self.view.safeAreaInsets.bottom : 0

            UIView.animate(withDuration: 0, delay: 0, options: .
                curveEaseOut, animations: {
                 self.view.layoutIfNeeded()
            }) { (completed) in

            }
        }
    }

    func setTitleForChat(userChat: Model.UserChat){
        navigationItem.title = "\(users![userChat.user]!.fullName())"
        navigationItem.rightBarButtonItem = UIBarButtonItem(title: "_",
            style: .plain, target: self, action: #selector(showPersonPage))
```

```
    }

    @objc func showPersonPage(){
        if let id = userChat?.user{
            onUserDisplay?(id)
        }
    }

    func setTitleForGroup(groupChat: Model.GroupChat){
        navigationItem.title = "\(groupChat.group.name)"
        navigationItem.rightBarButtonItem = UIBarButtonItem(title: "Info",
            style: .plain, target: self, action: #selector(moveToInfoVC))
    }

    // onInfoShow
    @objc func moveToInfoVC(){
        let vc = UIStoryboard(name: "Main", bundle: nil).
           instantiateViewController(withIdentifier:
           chatInfoViewController) as! ChatInfoViewController

        vc.delegate = self
        vc.users = users!
        vc.onUserDiplay = onUserDisplay

        if let groupChat = groupChat {
            vc.groupChat = groupChat.group
        }

        navigationController?.pushViewController(vc, animated: true)
    }

    var currentMessagesInChat: [Model.LastMessage] = [] {
        didSet
        {
            cellData = currentMessagesInChat.map { PostCellData(
               attributedData: PostCellData.create(with: [$0.body])) }
        }
    }

    // MARK: - Table view data source

    func numberOfSections(in tableView: UITableView) -> Int {
        return 1
    }

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

    override func viewWillAppear(_ animated: Bool)
    {
        super.viewWillAppear(animated)
        tabBarController?.tabBar.isHidden = true

        if let dialog = dialog {
            getMessagesNew(for: dialog)
        }
    }
```

```
func getMessages(for dialog: Model.Dialog, starting from: Int? = nil)
{
    switch dialog {
    case .groupChat(let groupChat):
        Model.getMessagesFor(typeOfChat: Model.Dialog.groupChat(
           groupChat), chat: groupChat.group.id, exclusiveFrom: from)
        { [weak self] in
            self?.currentMessagesInChat.append(contentsOf: $0)
        }
    case .userChat(let userChat):
        Model.getMessagesFor(typeOfChat: Model.Dialog.userChat(
           userChat), chat: userChat.user, exclusiveFrom: from)
        { [weak self] in
            self?.currentMessagesInChat.append(contentsOf: $0)
        }
    }
}


func getMessagesNew(for dialog: Model.Dialog, starting from: Int? =
   nil)
{
    switch dialog {
    case .groupChat(let groupChat):
        Model.getMessagesFor(typeOfChat: Model.Dialog.groupChat(
           groupChat), chat: groupChat.group.id, exclusiveFrom: from)
        { [weak self] in
            self?.currentMessagesInChat =  $0
        }
    case .userChat(let userChat):
        Model.getMessagesFor(typeOfChat: Model.Dialog.userChat(
           userChat), chat: userChat.user, exclusiveFrom: from)
        { [weak self] in
            self?.currentMessagesInChat = $0
        }
    }
}

func tableView(_ tableView: UITableView, cellForRowAt indexPath:
   IndexPath) -> UITableViewCell
{
    let cell = tableView.dequeueReusableCell(withIdentifier: cellId,
       for: indexPath) as! DialogCell

    //In cellForRowAtIndexPath
    cell.transform = CGAffineTransform(scaleX: 1, y: -1)
    cell.selectionStyle = .none


    if let user = users?[currentMessagesInChat[indexPath.row].author]
    {
        cell.fill(with: cellData[indexPath.row].attributedData, byUser
           : user, when: currentMessagesInChat[indexPath.row].time)

        cell.onUserDisplay = onUserDisplay
    }

    return cell
```

```
    }

    var prevLast = -1
    var canBePaginated = false
    func tableView(_ tableView: UITableView, willDisplay cell:
       UITableViewCell, forRowAt indexPath: IndexPath)
    {
        if indexPath.row == cellData.count - 1 && prevLast != indexPath.
           row && canBePaginated && cellData.count >= 9
        {
            print("exclusiveFrom \(currentMessagesInChat.last?.id ?? 0)")
            if let dialog = dialog
            {
                getMessages(for: dialog, starting: currentMessagesInChat.
                   last?.id)
            }

            prevLast = indexPath.row
        }
    }
}



//
//  DialogCell.swift
//  GDproject
//
//  Created by cstore on 05/05/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit
import TinyConstraints
import MarkdownKit

class DialogCell: UITableViewCell {

    let nameLabel: UIButton = {
        let button = UIButton()
        button.setTitleColor(.black, for: .normal)
        button.titleLabel?.font = UIFont.boldSystemFont(ofSize: 16)
        button.contentHorizontalAlignment = UIControl.
           ContentHorizontalAlignment.left
        return button
    }()

    @objc func displayProfile()
    {
        if let id = self.user?.id {
            print("diplay")
            onUserDisplay?(id)
        }
    }

    var onUserDisplay: ((Int)->())?


    let timeLabel: UILabel = {
```

```
        let label = UILabel()
        label.font = UIFont.systemFont(ofSize: 9)
        label.textAlignment = NSTextAlignment.right
        label.textColor = #colorLiteral(red: 0.4352941176, green:
           0.4431372549, blue: 0.4745098039, alpha: 1)
        return label
    }()

    func createTextView(with text: NSAttributedString, _ isSelectable:
       Bool) -> UITextView
    {
        let textView = UITextView()
        textView.isScrollEnabled = false
        textView.isEditable = false
        textView.sizeToFit()

        if isSelectable {
            textView.isSelectable = true
        } else {
            textView.isUserInteractionEnabled = false
        }

        textView.attributedText = text
        return textView
    }

    override init(style: UITableViewCell.CellStyle, reuseIdentifier:
       String?)
    {
        super.init(style: style, reuseIdentifier: reuseIdentifier)
    }

    required init?(coder aDecoder: NSCoder)
    {
        fatalError("init(coder:) has not been implemented")
    }

    var user: Model.Users? {
        didSet{
            nameLabel.setTitle(user!.fullName(), for: .normal)
        }
    }

    var textView: UITextView!

    func fill(with markdownText: NSAttributedString, byUser: Model.Users,
       when: String)
    {
        let myId = DataStorage.standard.getUserId()
        // important
        contentView.subviews.forEach({ $0.removeFromSuperview() })

        nameLabel.addTarget(self, action: #selector(displayProfile), for:
           .touchUpInside)

        self.user = byUser
        textView = createTextView(with: markdownText, true)
        textView.layer.cornerRadius = 10
        textView.clipsToBounds = true
```

```
        self.contentView.addSubview(textView)
        self.contentView.addSubview(timeLabel)

        timeLabel.text = when.getDate()
        textView.width(min: 50, max: self.contentView.bounds.width-70,
           priority: .required, isActive: true)

        if myId == byUser.id
        {
            layoutMyMessage()
        } else {
            layOutOtherMessage()
        }
    }

    func layOutOtherMessage(){
        self.contentView.addSubview(nameLabel)

        nameLabel.setTitle(user!.fullName(), for: .normal)

        nameLabel.edgesToSuperview(excluding: .bottom, insets: .top(4) + .
           left(8))
        textView.edgesToSuperview(excluding: [.top, .right] , insets: .
           left(8) + .bottom(20))
        textView.topToBottom(of: nameLabel)

        textView.backgroundColor = #colorLiteral(red: 0.8039215803, green:
            0.8039215803, blue: 0.8039215803, alpha: 1)
        textView.textColor = .white
        timeLabel.edgesToSuperview(excluding: .top, insets: .right(70) + .
           left(8))
        timeLabel.topToBottom(of: textView, offset: 4)
        timeLabel.textAlignment = .left
    }

    func layoutMyMessage()
    {
        textView.edgesToSuperview(excluding: .left, insets: .right(8) + .
           bottom(20) + .top(4))
        textView.backgroundColor = UIColor(red:0.08, green:0.49, blue
           :0.98, alpha:1.0)
        textView.textColor = .white
        timeLabel.edgesToSuperview(excluding: .top, insets: .left(70) + .
           right(8))
        timeLabel.topToBottom(of: textView, offset: 4)
        timeLabel.textAlignment = .right
    }
}



//
//  MessagesViewController.swift
//  GDproject
//
//  Created by cstore on 01/05/2019.
//  Copyright   2019 drHSE. All rights reserved.
//
```

```
import UIKit

class MessagesViewController: UITableViewController {

    // curreent Active which can be displayed
    var currentActiveDialogs: [Model.Dialog] = [] {
        didSet {
            tableView.reloadData()
        }
    }

    // curreent users
    var users: [Int: Model.Users] = [:]

    var onUserDisplayList: (()->())?

    var onDialogDisplay: (((dialog: Model.Dialog, users: [Int:Model.Users
       ]))->())?

    let searchC = UISearchController(searchResultsController: nil)

    override func viewDidLoad() {
        super.viewDidLoad()

        // Uncomment the following line to display an Edit button in the
           navigation bar for this view controller.
        self.navigationItem.rightBarButtonItem = UIBarButtonItem(title: "
           Write", style: .plain, target: self, action: #selector(
           choosePerson))

        self.navigationItem.title = "Messages"
        // self.navigationItem.largeTitleDisplayMode = .always
        self.navigationItem.searchController = searchC
        navigationController?.navigationBar.prefersLargeTitles = true
        self.navigationItem.hidesSearchBarWhenScrolling = false
    }

    @objc func choosePerson(){
        onUserDisplayList?()
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        tabBarController?.tabBar.isHidden = false
        Model.getChatAll { [weak self]  in
            self?.currentActiveDialogs = $0.0
            self?.users = $0.1
        }
    }

    // MARK: - Table view data source
    override func numberOfSections(in tableView: UITableView) -> Int {
        return 1
    }

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

```
    }


    override func tableView(_ tableView: UITableView, cellForRowAt
       indexPath: IndexPath) -> UITableViewCell
    {
        let cell = tableView.dequeueReusableCell(withIdentifier: "
           MessageViewCell", for: indexPath) as! MessageViewCell

        switch currentActiveDialogs[indexPath.row].self {
        case .groupChat:
            cell.fill(with: currentActiveDialogs[indexPath.row])
        case .userChat(let userChat):
            cell.fill(with: currentActiveDialogs[indexPath.row], user:
               users[userChat.user])
        }

        return cell
    }

    override func tableView(_ tableView: UITableView, didSelectRowAt
       indexPath: IndexPath) {
        let tuple = (currentActiveDialogs[indexPath.row],users)
        onDialogDisplay?(tuple)
    }
}
//
//  ChatInfoViewController.swift
//  GDproject
//
//  Created by cstore on 02/05/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit

/// Class for displaying chat info
///
class ChatInfoViewController: UITableViewController {

    enum PersonStatus{
        case admin
        case ordinary
    }

    weak var delegate: UpdatableGroup?

    var groupChat: Model.Group? {
        didSet{
            if let groupChat = groupChat {
                usersArray = groupChat.users.map { $0.key }
            }
        }
    }

    var usersArray: [Int] = [] {
        didSet {
            tableView.reloadData()
        }
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.03-01 ТЗ 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

```
}

var users: [Int: Model.Users] = [:]

func canIEditThisChat() -> PersonStatus
{
    let myId = UserDefaults.standard.integer(forKey: UserDefaultsKeys.
       id.rawValue)

    if let groupChatUserPermission = groupChat?.users[myId]?.isAdmin {
       return groupChatUserPermission ?  .admin :  .ordinary
    }

    return .ordinary
}

var myPermissions: PersonStatus = .ordinary



override func viewDidLoad() {
    super.viewDidLoad()

    tableView.register(UITableViewCell.self, forCellReuseIdentifier: "
       cell")
    myPermissions = canIEditThisChat()
    tableView.reloadData()

    switch myPermissions {
    case .admin:
        navigationItem.rightBarButtonItem = self.editButtonItem
    default:
        return
    }
}

// MARK: - Table view data source

override func numberOfSections(in tableView: UITableView) -> Int {
    return 3
}

override func tableView(_ tableView: UITableView,
   numberOfRowsInSection section: Int) -> Int {
    switch section {
    case 0:
        return 1
    case 1:
        return 1
    default:
        return usersArray.count + (myPermissions == .ordinary ? 0 : 1)
    }
}


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

```
        let cell = tableView.dequeueReusableCell(withIdentifier: "cell",
           for: indexPath)

        if indexPath.row == 0 {
            switch indexPath.section {
            case 0:
                cell.textLabel?.text = groupChat!.name
                cell.selectionStyle = .none
                return cell
            case 1:
                cell.textLabel?.text = "Leave chat"
                cell.textLabel?.textColor = #colorLiteral(red: 1, green:
                   0.1491314173, blue: 0, alpha: 1)
                cell.selectionStyle = .none
                return cell
            default:
                switch myPermissions{
                case .admin:
                    cell.textLabel?.text = "Add participants"
                    cell.accessoryType = .disclosureIndicator
                case .ordinary:
                    cell.textLabel?.text = name(for: users[usersArray[
                       indexPath.row]])
                }
                cell.selectionStyle = .none
                return cell
            }
        }

        switch myPermissions{
        case .admin:
            cell.textLabel?.text = name(for: users[usersArray[indexPath.
               row-1]])
        case .ordinary:
            cell.textLabel?.text = name(for: users[usersArray[indexPath.
               row]])
        }
        cell.selectionStyle = .none
        return cell
    }

    private func name(for user: Model.Users?) -> String {
        if let user = user, let perm = groupChat?.users[user.id]?.isAdmin
           {
            if perm {
                return "_ \(user.fullName())"
            }
            return "_ \(user.fullName())"
        }

        return "left"
    }

    override func tableView(_ tableView: UITableView,
       titleForHeaderInSection section: Int) -> String?
    {
        switch section {
        case 0:
            return "Title"
```

```
        case 1:
            return "Opportunities"
        default:
            return "Participants"
        }
    }


    func editName(for cell: UITableViewCell){
        let alert = UIAlertController(title: "Are you sure?", message: "
           Title:", preferredStyle: .alert)

        let action1 = UIAlertAction(title: "Cancel", style: .cancel,
           handler: nil)

        alert.addTextField { [weak self] (tf) in
            tf.textColor = #colorLiteral(red: 0, green: 0, blue: 0, alpha:
                1)
            tf.text = self?.groupChat?.name
        }

        let action2 = UIAlertAction(title: "OK", style: .default) { [
           unowned self] _ in
            let name = alert.textFields?.first?.text
            cell.textLabel?.text = name
            self.groupChat?.name = name!
            Model.updateGroupChat(with: self.groupChat!)
            self.delegate?.updateGroup(with: self.groupChat!)
        }

        alert.addAction(action1)
        alert.addAction(action2)

        present(alert, animated: true, completion: nil)
    }
    override func tableView(_ tableView: UITableView, didSelectRowAt
       indexPath: IndexPath)
    {
        guard let groupChat = groupChat else {
            return
        }

        guard let cell = tableView.cellForRow(at: indexPath) else {
            return
        }

        if indexPath.row == 0 {
            switch indexPath.section{
            case 0:
                switch myPermissions{
                case .admin:
                    editName(for: cell)
                default:
                    break
                }
            case 1:
                Model.leaveGroupChat(id: groupChat.id) {
                    [weak self] in
```

```
                self?.navigationController?.popViewController(animated
                    : true)
            }
        default:
            switch myPermissions{
            case .admin:
                showUserChoiceVC()
            default:
                showParticipantPage(with: usersArray[indexPath.row])
            }
        }
    }


    if indexPath.section == 2 && indexPath.row != 0  {
        switch myPermissions{
        case .admin:
            showParticipantPage(with: usersArray[indexPath.row-1])
        default:
            showParticipantPage(with: usersArray[indexPath.row])
        }
    }
}

var onUserDiplay: ((Int)->())?

func showParticipantPage(with user: Int?) {
    if let user = user {
        onUserDiplay?(user)
    }
}

func showUserChoiceVC()
{
    let vc = storyboard?.instantiateViewController(withIdentifier:
       peopleToWriteVC) as! PeopleToWriteViewController

    vc.whatToDoWithSelection = { [weak self, weak vc] mapa in

        mapa.forEach { self?.users[$0.key] = Model.Channels.
           fullPeopleDict[$0.key] }
        mapa.forEach { self?.groupChat?.users[$0.key] = $0.value }
        vc?.navigationController?.popViewController(animated: true)
    }

    navigationController?.pushViewController(vc, animated: true)
}

override func tableView(_ tableView: UITableView, commit editingStyle:
    UITableViewCell.EditingStyle, forRowAt indexPath: IndexPath)
{
    if editingStyle == UITableViewCell.EditingStyle.delete {
        groupChat?.users[usersArray[indexPath.row-1]] = nil
    }

    tableView.reloadData()
}
```

```
override func tableView(_ tableView: UITableView, canEditRowAt
    indexPath: IndexPath) -> Bool {
    if indexPath.row == 0 {
        return false
    }

    return true
}

override func tableView(_ tableView: UITableView, editActionsForRowAt
    indexPath: IndexPath) -> [UITableViewRowAction]?
{
    let editButton = UITableViewRowAction(style: .normal, title: "
        Promote") { [unowned self] (rowAction, indexPath) in

        self.tableView.beginUpdates()
        self.groupChat?.users[self.usersArray[indexPath.row-1]]?.
            isAdmin = true
        self.tableView.reloadRows(at: [indexPath], with: .none)
        self.tableView.endUpdates()
    }

    editButton.backgroundColor = #colorLiteral(red: 0.476841867, green
        : 0.5048075914, blue: 1, alpha: 1)

    let restrictButton = UITableViewRowAction(style: .normal, title: "
        Restrict") { [unowned self] (rowAction, indexPath) in

        self.tableView.beginUpdates()
        self.groupChat?.users[self.usersArray[indexPath.row-1]]?.
            isAdmin = false
        self.tableView.reloadRows(at: [indexPath], with: .none)
        self.tableView.endUpdates()
    }

    restrictButton.backgroundColor = #colorLiteral(red: 0.8039215803,
        green: 0.8039215803, blue: 0.8039215803, alpha: 1)

    let deleteButton = UITableViewRowAction(style: .normal, title: "
        Delete") { [unowned self] (action, indexPath) in

        self.tableView.beginUpdates()
        self.groupChat?.users[self.usersArray[indexPath.row-1]] = nil
        self.tableView.deleteRows(at: [indexPath], with: .none)
        self.tableView.endUpdates()
    }

    deleteButton.backgroundColor = #colorLiteral(red: 1, green:
        0.1491314173, blue: 0, alpha: 1)

    return [editButton, restrictButton, deleteButton]
}

/// update chatInfo
override func viewWillDisappear(_ animated: Bool) {

    defer {
        super.viewWillDisappear(animated)
    }
```

```
        guard let _ = self.navigationController?.viewControllers.lastIndex
            (of: self) else {
            switch myPermissions {
            case .ordinary:
                return
            case .admin:
                Model.updateGroupChat(with: groupChat!)
                delegate?.updateGroup(with: groupChat!)
            }
            return
        }
    }
}
//
//  NewPostViewController.swift
//  GDproject
//
//  Created by cstore on 05/02/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit
import Cartography
import Marklight
import TinyConstraints

class NewPostViewController: UIViewController, UITextViewDelegate {

    @IBOutlet weak var view1: UIView!
    // Keep strong instance of the `NSTextStorage` subclass
    let textStorage = MarklightTextStorage()

    weak var parentVC: NewsController?

    static var draft: String = ""
    var textView: UITextView!

    // buttons for attaching images
    var accessoryView: UIView = {
        let view = UIView()
        view.backgroundColor = .white
        return view
    }()

    var addEquasion: UIButton = {
        var button = UIButton(type: .detailDisclosure)
        button.addTarget(self, action: #selector(addMathBrackets), for: .
            touchUpInside)
        return button
    }()

    // stack view where buttons will be places
    var stackAccessoryView: UIStackView?
    // Connect the view1's bottom layout constraint to react to keyboard
        movements

    @IBOutlet weak var bottomTextViewConstraint: NSLayoutConstraint!
```

```
override func viewDidLoad()
{
    super.viewDidLoad()
    setUpMD()
    setUpTextView()
    setUpAccessoryView()
}

override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)

    navigationController?.navigationBar.prefersLargeTitles = false
    navigationItem.title = "New post"
    textView.text = NewPostViewController.draft
}

func setUpAccessoryView(){
    let views = [UIButton(type: .contactAdd),addEquasion]
    stackAccessoryView = UIStackView(arrangedSubviews: views)

    stackAccessoryView?.alignment = .fill
    stackAccessoryView?.distribution = .equalSpacing

    view1.addSubview(accessoryView)
    accessoryView.addSubview(stackAccessoryView!)
    accessoryView.height(40)

    let separatorView = UIView()
    separatorView.backgroundColor = #colorLiteral(red: 0.8039215803,
       green: 0.8039215803, blue: 0.8039215803, alpha: 1)
    accessoryView.addSubview(separatorView)
    separatorView.height(1)
    separatorView.edgesToSuperview(excluding: .bottom)

    accessoryView.edgesToSuperview(excluding: [.top])
    stackAccessoryView!.edgesToSuperview(insets: .left(16) + .right
       (16) + .top(1))
}

func setUpTextView(){
    view.addConstraint(bottomTextViewConstraint)
    view1.addSubview(textView)
    textView.edgesToSuperview(insets: .top(8) + .left(8) + .bottom
       (40+8) + .right(8))

    if #available(iOS 11.0, *) {
        textView.smartDashesType = .no
        textView.smartQuotesType = .no
    }

    textView.isScrollEnabled = true

    guard let bottomTextViewConstraint = bottomTextViewConstraint else
        { return }
    view.addConstraint(bottomTextViewConstraint)

    // Add a beautiful padding to the `UITextView` content
```

```
        textView.textContainerInset = UIEdgeInsets(top: 4, left: 4, bottom
            : 4, right: 4)
        textView.delegate = self
        magicForKeyboardChanges()
    }


    func magicForKeyboardChanges()
    {
        ////////////////////////////
        // We do some magic to resize the `UITextView` to react the the
            keyboard size change (appearance, disappearance, ecc)
        NotificationCenter.default.addObserver(self, selector: #selector(
            handleKeyboardNotifications), name: UIResponder.
            keyboardWillShowNotification, object: nil)

        NotificationCenter.default.addObserver(self, selector: #selector(
            handleKeyboardNotifications), name: UIResponder.
            keyboardWillHideNotification, object: nil)

        // Partial fixes to a long standing bug, to keep the caret inside
            the `UITextView` always visible
        NotificationCenter.default.addObserver(forName: UITextView.
            textDidChangeNotification, object: textView, queue:
            OperationQueue.main) { (notification) -> Void in
             if self.textView.textStorage.string.hasSuffix("\n") {
                 CATransaction.setCompletionBlock({ () -> Void in
                     self.scrollToCaret(self.textView, animated: false)
                 })
             } else {
                 self.scrollToCaret(self.textView, animated: false)
             }
        }

    }

    @objc func handleKeyboardNotifications(notification: NSNotification){
        if let userInfo = notification.userInfo{
             // UIKeyboardFrameEndUserInfoKey
             let keyBoardFrame = (userInfo[UIResponder.
                keyboardFrameEndUserInfoKey] as! NSValue).cgRectValue
             bottomTextViewConstraint?.constant = notification.name ==
                UIResponder.keyboardWillShowNotification ? keyBoardFrame.
                height : 0
             print(bottomTextViewConstraint!.constant)

             UIView.animate(withDuration: 0, delay: 0, options: .
                curveEaseOut, animations: {
                 self.view.layoutIfNeeded()
             }) { (completed) in

             }
        }
    }

    @objc func attachPhoto(){

    }
```

```
@objc func addMathBrackets()
{
    if textView.text.count != 0 &&  textView.text.last != "\n" {
        textView.insertText("\n\\[\n      ")
    }
    else {
        textView.insertText("\\[\n      ")
    }

    if let selectedRange = textView.selectedTextRange
    {
        textView.insertText("\n\\]\n")

        if let newPosition = textView.position(from: selectedRange.
           start, offset: 0)
        {
            // set the new position
            textView.selectedTextRange = textView.textRange(from:
               newPosition, to: newPosition)
        }
    }
}

func setUpMD(){
    textStorage.marklightTextProcessor.codeColor = UIColor.orange
    textStorage.marklightTextProcessor.quoteColor = UIColor.darkGray
    textStorage.marklightTextProcessor.syntaxColor = UIColor.blue
    textStorage.marklightTextProcessor.codeFontName = "Courier"
    textStorage.marklightTextProcessor.fontTextStyle = UIFont.
       TextStyle.subheadline.rawValue
    textStorage.marklightTextProcessor.hideSyntax = false

    let layoutManager = NSLayoutManager()

    // Assign the `UITextView`'s `NSLayoutManager` to the `
       NSTextStorage` subclass
    textStorage.addLayoutManager(layoutManager)

    let textContainer = NSTextContainer()
    layoutManager.addTextContainer(textContainer)

    textView = UITextView(frame: view.bounds, textContainer:
       textContainer)
}

// MARK:- new post
@objc func newPost(){



    // adding row to uiTableView after adding new post
    guard let vc = parentVC else {return}
    vc.tableView.beginUpdates()
    let indexPath1: IndexPath = IndexPath(row: 0, section: 0)
    vc.dataSourse.insert(p, at: 0)
    vc.tableView.insertRows(at: [indexPath1], with: .fade)
    vc.tableView.endUpdates()
    moveBackToParentVC()
```

```
        // somewhere here i will be sending server notifications about new
            post arrival
    }

    @objc func actionSaveDraft(){
        let optionMenu = UIAlertController(title: nil, message: nil,
            preferredStyle: .actionSheet)

        let saveAction = UIAlertAction(title: "Save draft", style: .
            default)
        {
            _ in
            NewPostViewController.draft = self.textView.text
            self.moveBackToParentVC()
        }

        let deleteAction = UIAlertAction(title: "Delete draft", style: .
            destructive)
        {
            (_)
            in
            NewPostViewController.draft = ""
            self.moveBackToParentVC()
        }

        let cancelAction = UIAlertAction(title: "Cancel", style: .cancel)

        optionMenu.addAction(saveAction)
        optionMenu.addAction(deleteAction)
        optionMenu.addAction(cancelAction)

        self.present(optionMenu, animated: true, completion: nil)
    }

    @objc func closeView()
    {
        actionSaveDraft()
    }

    func moveBackToParentVC(){
        let transition = CATransition()
        transition.duration = 0.5
        transition.timingFunction = CAMediaTimingFunction(name:
            CAMediaTimingFunctionName.easeInEaseOut)
        transition.type = CATransitionType.reveal
        transition.subtype = CATransitionSubtype.fromBottom
        navigationController?.view.layer.add(transition, forKey: nil)
        navigationController?.popViewController(animated: false)
        textView!.resignFirstResponder()
    }




    func textView(_ textView: UITextView, shouldInteractWith URL: URL, in
        characterRange: NSRange) -> Bool {
        print("Should interact with: \(URL)")
        return true
    }
```

```
    func scrollToCaret(_ textView: UITextView, animated: Bool) {
        var rect = textView.caretRect(for: textView.selectedTextRange!.end
           )
        rect.size.height = rect.size.height + textView.textContainerInset.
           bottom
        textView.scrollRectToVisible(rect, animated: animated)
    }
}
//
//  MyStackView.swift
//  GDproject
//
//  Created by cstore on 14/02/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit

class MyStackView: UIStackView {

    var isFull: Bool = true


    override var bounds: CGRect {
        didSet{
            if frame.height == 300 && !isFull {
                let gradientLayer = CAGradientLayer()
                gradientLayer.colors = [UIColor(displayP3Red: 255, green:
                   255, blue: 255, alpha: 0).cgColor, UIColor.white.
                   cgColor]
                gradientLayer.startPoint = CGPoint(x: 0.5, y: 0.5)
                gradientLayer.frame = self.bounds
                self.layer.addSublayer(gradientLayer)
            }
        }
    }
}
//
//  PostTable.swift
//  GDproject
//
//  Created by cstore on 13/02/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit

class ChannelsController: UITableViewController {



}
//
//  BasicInfoCell.swift
//  NewsFeed
//
//  Created by cstore on 23/01/2019.
//  Copyright   2019 drHSE. All rights reserved.
```

```
//

import UIKit
import TinyConstraints

class InfoCell: UITableViewCell{

    var textView: UITextView = {
        let label = UITextView()
        label.font = UIFont.systemFont(ofSize: 16)
        label.sizeToFit()
        label.isScrollEnabled = false
        // label.isUserInteractionEnabled = false
        label.isEditable = false
        label.dataDetectorTypes = .all
        label.textColor =  .black
        return label
    }()

    override init(style: UITableViewCell.CellStyle, reuseIdentifier:
       String?) {
        super.init(style: style, reuseIdentifier: reuseIdentifier)
    }

    func setUpView()
    {
        addSubview(textView)
        textView.horizontalToSuperview(insets: .left(32) + .right(16))
        textView.verticalToSuperview(insets: .top(8) + .bottom(8))
    }

    required init?(coder aDecoder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    func fill(title: String){
        setUpView()
        textView.text = title
    }
}
//
//  BasicInfoCell.swift
//  NewsFeed
//
//  Created by cstore on 23/01/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit
import TinyConstraints

class BasicInfoCell: UITableViewCell{

    var titleLabel: UILabel = {
        let label = UILabel()
        label.font = UIFont.boldSystemFont(ofSize: 16)
        label.textColor =  .black
        return label
    }()
```

```
    override init(style: UITableViewCell.CellStyle, reuseIdentifier:
       String?) {
        super.init(style: style, reuseIdentifier: reuseIdentifier)
    }

    func setUpView(){
        addSubview(titleLabel)
        titleLabel.horizontalToSuperview(insets: .left(16) + .right(16))
        titleLabel.verticalToSuperview(insets: .top(8) + .bottom(8))
    }

    required init?(coder aDecoder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    func fill(title: String){
        setUpView()
        titleLabel.text = title
    }
}
//
//  LogInViewController.swift
//  NewsFeed
//
//  Created by cstore on 20/01/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit

class InviteViewController: UIViewController {

    @IBOutlet weak var mailTextField: UITextField!

    @IBOutlet weak var inviteLabel: UILabel!

    static let titleColor = UIColor(red: 0, green: 137/255, blue: 249/255,
        alpha: 0.5)

    var bottomConstraint: NSLayoutConstraint?

    let logInButton: UIButton = {
        let button = UIButton(type: .system)
        button.setTitle("Invite", for: .normal)
        button.setTitleColor(titleColor, for: .normal)
        button.titleLabel?.font  = UIFont.boldSystemFont(ofSize: 16)
        button.isEnabled = false
        button.addTarget(self, action: #selector(activateLogInProcess),
           for: .touchUpInside)
        return button
    }()

    let keyboardBar: UIView = {
        let view = UIView()
        view.backgroundColor = .white
        return view
    }()
```

```
    override func viewDidLoad() {
        super.viewDidLoad()
        setUpView()
        configureTapgesture()
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        // MARK:- uncomment when mail registration will be availiable
        //self.navigationController?.setNavigationBarHidden(true, animated
            : animated)


        navigationController?.navigationBar.prefersLargeTitles = false
    }

    override func viewWillDisappear(_ animated: Bool) {
        super.viewWillDisappear(animated)
        // MARK:- uncomment when mail registration will be availiable
        //self.navigationController?.setNavigationBarHidden(false,
            animated: animated)
    }

    private func configureTapgesture(){
        let tapGesture = UITapGestureRecognizer(target: self, action: #
            selector(handleTap))
        view.addGestureRecognizer(tapGesture)
    }

    @objc func handleTap(){
        view.endEditing(true)
    }

    @objc func activateLogInProcess(){
        if logInButton.isEnabled {
            // instead of transporting user for new vc, make him a sign
                that everything was complete
            inviteLabel.text?.append(" _")
            view.endEditing(true)
        }
    }

    func setUpView(){
        mailTextField.delegate = self
        view.addSubview(keyboardBar)

        view.addConstraintsWithFormat(format: "H:|[v0]|", views:
            keyboardBar)
        view.addConstraintsWithFormat(format: "V:[v0(50)]", views:
            keyboardBar)

        setUpBarComponents()

        bottomConstraint = NSLayoutConstraint(item: keyboardBar, attribute
            : .bottom, relatedBy: .equal, toItem: view.safeAreaLayoutGuide,
             attribute: .bottom, multiplier: 1, constant: 0)
        view.addConstraint(bottomConstraint!)

        // for keyboard notifications
```

```
        NotificationCenter.default.addObserver(self, selector: #selector(
            handleKeyboardNotifications), name: UIResponder.
            keyboardWillShowNotification, object: nil)

        NotificationCenter.default.addObserver(self, selector: #selector(
            handleKeyboardNotifications), name: UIResponder.
            keyboardWillHideNotification, object: nil)

        // for log in button notifications
        NotificationCenter.default.addObserver(self, selector: #selector(
            inputDidChanged), name: UITextField.textDidChangeNotification,
            object: mailTextField)

        NotificationCenter.default.addObserver(self, selector: #selector(
            inputDidChanged), name: UITextField.
            textDidBeginEditingNotification, object: mailTextField)
    }


    @objc func inputDidChanged(notification: NSNotification){
        if mailTextField.text?.isEmpty ?? true
        {
            logInButton.isEnabled = false
            logInButton.setTitleColor(InviteViewController.titleColor.
                withAlphaComponent(0.5), for: .normal)
        }
        else
        {
            logInButton.isEnabled = true
            logInButton.setTitleColor(InviteViewController.titleColor.
                withAlphaComponent(1), for: .normal)
        }
    }

    @objc func handleKeyboardNotifications(notification: NSNotification){
        if let userInfo = notification.userInfo{
            // UIKeyboardFrameEndUserInfoKey
            let keyBoardFrame = (userInfo[UIResponder.
                keyboardFrameEndUserInfoKey] as! NSValue).cgRectValue
            bottomConstraint?.constant = notification.name == UIResponder.
                keyboardWillShowNotification ? -keyBoardFrame.height+view.
                safeAreaInsets.bottom : 0

            UIView.animate(withDuration: 0, delay: 0, options: .
                curveEaseOut, animations: {
                 self.view.layoutIfNeeded()
            }) { (completed) in

            }
        }
    }

    func setUpBarComponents(){
        keyboardBar.addSubview(logInButton)
        keyboardBar.addConstraintsWithFormat(format: "H:[v0(60)]-16-|",
            views: logInButton)
        keyboardBar.addConstraintsWithFormat(format: "V:|[v0]|", views:
            logInButton)
    }
```

```
}


extension InviteViewController: UITextFieldDelegate{
    func textFieldShouldReturn(_ textField: UITextField) -> Bool
    {
        textField.resignFirstResponder()
        return true
    }
}
//
//  ProfileViewController.swift
//  GDproject
//
//  Created by cstore on 15/02/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit
import TinyConstraints

protocol UpdateUser: class {
    func updateUserObj(with user: Model.Users)
}

class ProfileViewController: UIViewController, UpdateUser
{
    func updateUserObj(with user: Model.Users)
    {
        self.user = user
        Model.Channels.fullPeopleDict[user.id] = user
    }

    @IBOutlet weak var segmentedControl: UISegmentedControl!

    @IBOutlet weak var profileView: UIView!

    @IBOutlet weak var tableView: UITableView!

    @IBOutlet weak var profileImageView: UIImageView!

    @IBOutlet weak var surnameLabel: UILabel!

    @IBOutlet weak var nameLabel: UILabel!

    @IBOutlet weak var facultyLabel: UILabel!

    @IBOutlet weak var placeLabel: UILabel!

    @IBOutlet weak var newMessageButton: UIButton!

    var logOut: (()->())?

    var deleteAllSessions: (()->())?

    var onSettings: (()->())?

    @IBAction func sendMessage(_ sender: UIButton)
    {
```

```
        if let userId = idProfile
        {
            let createdDialog = Model.Dialog.userChat(Model.UserChat(user:
                userId))
            let vc = DialogViewController()
            vc.users = Model.Channels.fullPeopleDict
            vc.dialog = createdDialog
            self.navigationController?.pushViewController(vc, animated:
               true)
        }
    }

    var protoDictionary: [String: UIImage] = ["135213": #imageLiteral(
       resourceName: "9"), "135288": #imageLiteral(resourceName: "5051"),
       "22723" : #imageLiteral(resourceName: "69"), "135083": #
       imageLiteral(resourceName: "42")]

    func fill(with user: Model.Users)
    {
        self.facultyLabel.text =  user.faculty.name
        self.nameLabel.text = "\(user.firstName) \(user.middleName)"
        self.surnameLabel.text = "\(user.lastName)"
        self.profileImageView.image = protoDictionary[user.faculty.
           campusCode]?.roundedImage
        self.placeLabel.text = "\(user.faculty.address)"
        if user.id == DataStorage.standard.getUserId() {
            newMessageButton.isHidden  = true
        } else {
            newMessageButton.isHidden  = false
        }
    }

    var user: Model.Users? {
        didSet {
            if let user = user {
                self.fill(with: user)
                navigationItem.title = "\(user.firstName) \(user.lastName)
                   "
            } else if let id = idProfile {
                Model.getUsers(for: [id]) { [unowned self ] in
                    self.user = $0[id]
                }
            }
        }
    }

    var channel: Model.Channels? {
        didSet {
            self.update()
        }
    }

    func update()
    {
        Model.getAnonymousChannel(by: channel!) { [unowned self] in
            self.posts.dataSourse = $0.posts
            self.posts.dictionary = $0.users
            self.user = $0.users[self.idProfile!]
        }
```

```
    }

    var basicInfo = BasicInfoController()
    var posts = NewsVC()

    override func viewDidLoad()
    {
        super.viewDidLoad()
        if idProfile == nil {
            idProfile = DataStorage.standard.getUserId()
        }

        posts.viewController = self
        posts.type = .NONE
        posts.currChannel = channel

        posts.onFullPost = {
            [weak self] (type,post) in

            let vc = self?.storyboard?.instantiateViewController(
               withIdentifier: fullPostControllerId) as!
               FullPostController
            vc.type = type
            vc.post = post
            self?.navigationController!.pushViewController(vc, animated:
               true)
        }

        tableView.register(UITableViewCell.self, forCellReuseIdentifier: "
           cell")

        tableView.register(PostViewCell.self, forCellReuseIdentifier:
           postCellId)

        tableView.register(BasicInfoCell.self, forCellReuseIdentifier:
           basicInfoCellId)

        tableView.register(InfoCell.self, forCellReuseIdentifier:
           infoCellId)

        posts.viewController = self

        tableView.delegate = posts
        tableView.dataSource = posts
        tableView.reloadData()
    }

    var idProfile: Int? {
        didSet {
            channel = Model.Channels(people: [idProfile!], name: "", id:
               0, tags: [])
        }
    }

    override func viewWillAppear(_ animated: Bool) {
        tabBarController?.tabBar.isHidden = false

        if idProfile == nil {
            idProfile = DataStorage.standard.getUserId()
```

```
        }
        user = Model.Channels.fullPeopleDict[idProfile!]

        update()

        setUpNavigarionBar()
    }

    func setUpNavigarionBar(){
        navigationController?.navigationBar.prefersLargeTitles = true
        let uibarbutton = UIBarButtonItem(title: "More", style: .plain,
           target: self, action: #selector(showInformation))
        navigationItem.rightBarButtonItems = [uibarbutton]
        navigationItem.largeTitleDisplayMode = .always
    }

    @objc func showInformation(){

        let optionMenu = UIAlertController(title: nil, message: nil,
           preferredStyle: .actionSheet)


        let cancelAction = UIAlertAction(title: "Cancel", style: .cancel)


        if idProfile == DataStorage.standard.getUserId() {
            let logoutAction = UIAlertAction(title: "Log out", style: .
               destructive)
            { [weak self] (_) in

                if let logOut = self?.logOut {
                    logOut()
                } else {
                    Model.logout() {
                        DataStorage.standard.setIsLoggedIn(value: false,
                           with: 0)
                        (UIApplication.shared.delegate as! AppDelegate).
                           relaunch()
                    }
                }
            }

            let deleetAllSessionsAction = UIAlertAction(title: "Delete all
                sessions", style: .destructive)
            { [weak self] _ in

                if let delete = self?.deleteAllSessions {
                    delete()
                } else {
                    Model.deactivateAll() {
                        DataStorage.standard.setIsLoggedIn(value: false,
                           with: 0)
                        (UIApplication.shared.delegate as! AppDelegate).
                           relaunch()
                    }
                }
            }
```

```
            let settingsAction = UIAlertAction(title: "Edit profile",
                style: .default)
            { [weak self] (_) in

                if let settings = self?.onSettings{
                    settings()
                } else {
                    let vc = self?.storyboard?.instantiateViewController(
                        withIdentifier: "RegisterTableViewController") as!
                        RegisterTableViewController
                    vc.delegate = self
                    vc.userActive = self?.user

                    self?.navigationController?.pushViewController(vc,
                        animated: true)
                }
            }

            optionMenu.addAction(settingsAction)
            optionMenu.addAction(logoutAction)
            optionMenu.addAction(deleetAllSessionsAction)
        } else {
            let channelAction = UIAlertAction(title: "Add to a channel",
                style: .default)
            {
                [weak self] (_) in

                let vc = self?.storyboard?.instantiateViewController(
                    withIdentifier: simplifiedChannelsList) as!
                    SimplifiedChannelsList
                vc.user = self?.user

                let transition = CATransition()
                transition.duration = 0.5
                transition.timingFunction = CAMediaTimingFunction(name:
                    CAMediaTimingFunctionName.easeInEaseOut)
                transition.type = CATransitionType.moveIn
                transition.subtype = CATransitionSubtype.fromTop
                self?.navigationController?.view.layer.add(transition,
                    forKey: nil)
                self?.navigationController?.pushViewController(vc,
                    animated: false)
            }

            optionMenu.addAction(channelAction)
        }

        optionMenu.addAction(cancelAction)
        self.present(optionMenu, animated: true, completion: nil)
    }

    deinit {
        print("profile clear")
    }

    @IBAction func valueChanged(_ sender: UISegmentedControl) {
        let index = sender.selectedSegmentIndex

        switch index {
```

```
        case 0:
            changeToPosts()
        case 1:
            changeToBasicInfo()
        default:
            break
        }
    }

    func changeToPosts(){
        tableView.delegate = posts
        tableView.dataSource = posts
        tableView.reloadData()
    }

    func changeToBasicInfo(){
        tableView.delegate = basicInfo
        tableView.dataSource = basicInfo
        basicInfo.userInfo = self.user
        tableView.reloadData()
    }
}
//
//  BasicInfoController.swift
//  GDproject
//
//  Created by cstore on 16/02/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit

struct CellData{
    var opened = Bool()
    var title = String()
    var sectionData = [String]()
}

class BasicInfoController: UIViewController, UITableViewDelegate,
   UITableViewDataSource {

    var userInfo: Model.Users? {
        didSet {
            if let userInfo = userInfo {
                dataSourse = [CellData(opened: false, title: "Contacts",
                    sectionData: [userInfo.email, userInfo.faculty.address
                    ]),
                CellData(opened: false, title: "Faculty", sectionData: [
                    userInfo.faculty.campusName, userInfo.faculty.name,
                    userInfo.faculty.address, userInfo.faculty.path]),
                CellData(opened: false, title: "Interests", sectionData:
                    userInfo.faculty.tags)]
            }
        }
    }

    var dataSourse: [CellData] = [ ]
```

```
    func tableView(_ tableView: UITableView, numberOfRowsInSection section
        : Int) -> Int {
        if (dataSourse[section].opened) {
            return dataSourse[section].sectionData.count + 1
        } else {
            return 1
        }
    }

    func numberOfSections(in tableView: UITableView) -> Int {
        return dataSourse.count
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath:
        IndexPath) -> UITableViewCell {
        if indexPath.row == 0 {
            let cell = tableView.dequeueReusableCell(withIdentifier: "cell
                ", for: indexPath)
            cell.textLabel?.text = dataSourse[indexPath.section].title
            cell.accessoryType = .disclosureIndicator
            cell.selectionStyle = .none
            return cell
        } else {
            let cell = tableView.dequeueReusableCell(withIdentifier:
                infoCellId, for: indexPath) as! InfoCell
            cell.fill(title: dataSourse[indexPath.section].sectionData[
                indexPath.row - 1])
            cell.accessoryType = .none
            cell.selectionStyle = .none
            return cell
        }
    }

    func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
        IndexPath) {
        if indexPath.row == 0 {
            if dataSourse[indexPath.section].opened {
                dataSourse[indexPath.section].opened = false
                let section = IndexSet.init(integer: indexPath.section)
                tableView.reloadSections(section, with: .none)
            } else {
                dataSourse[indexPath.section].opened = true
                let section = IndexSet.init(integer: indexPath.section)
                tableView.reloadSections(section, with: .none)
            }
        }
    }
    func tableView(_ tableView: UITableView, heightForRowAt indexPath:
        IndexPath) -> CGFloat {
        return UITableView.automaticDimension
    }

    func tableView(_ tableView: UITableView, estimatedHeightForRowAt
        indexPath: IndexPath) -> CGFloat {
        return 100.0
    }
}
//
//  SettingsViewController.swift
```

```
//  GDproject
//
//  Created by cstore on 16/02/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit

class SettingsViewController: UITableViewController {



    override func viewDidLoad() {
        super.viewDidLoad()
        navigationItem.title = "Settings"
    }

    // MARK: - Table view data source

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)

        navigationController?.navigationBar.prefersLargeTitles = true
    }


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

    override func tableView(_ tableView: UITableView, didSelectRowAt
       indexPath: IndexPath) {
        if dataSourse[indexPath.row]   {
            let vc = storyboard?.instantiateViewController(withIdentifier:
                inviteVC) as! InviteViewController
            navigationController?.pushViewController(vc, animated: true)
        }
    }


    override func tableView(_ tableView: UITableView, cellForRowAt
       indexPath: IndexPath) -> UITableViewCell
    {
        let cell = tableView.dequeueReusableCell(withIdentifier: "
           SettingsCell", for: indexPath)
        cell.textLabel?.text = dataSourse[indexPath.row]
        cell.selectionStyle = .none
        return cell
    }

}
//
//  FullPostViewController.swift
//  GDproject
//
//  Created by cstore on 13/02/2019.
//  Copyright   2019 drHSE. All rights reserved.
//
```

```
import UIKit
import TinyConstraints

class FullPostController: UITableViewController {

    var post: Post?

    override func viewDidLoad() {
        super.viewDidLoad()

        tableView.register(PostViewCell.self, forCellReuseIdentifier:
           postCellId)

        setUpNavigationBar()
        tableView.separatorStyle = .none
    }

    func setUpNavigationBar(){
        navigationController?.navigationBar.prefersLargeTitles = false
        navigationItem.title = "\(post?.fromUser.login ?? "")"
        navigationItem.rightBarButtonItems = [UIBarButtonItem(
           barButtonSystemItem: .action, target: self, action: #selector(
           self.options))]
    }

    @objc func options(){
        // drafts
        // saved

        let optionMenu = UIAlertController(title: nil, message: nil,
           preferredStyle: .actionSheet)
        let editAction = UIAlertAction(title: "Send via message", style: .
           default)
        let shareAction = UIAlertAction(title: "Send via project", style:
           .default)
        let settingsAction = UIAlertAction(title: "Copy link", style: .
           default)
        let cancelAction = UIAlertAction(title: "Cancel", style: .cancel)

        optionMenu.addAction(editAction)
        optionMenu.addAction(shareAction)
        optionMenu.addAction(settingsAction)
        optionMenu.addAction(cancelAction)

        self.present(optionMenu, animated: true, completion: nil)
    }
    // MARK: - Table view data source

    override func numberOfSections(in tableView: UITableView) -> Int {
        // #warning Incomplete implementation, return the number of
           sections
        return 2
    }


    override func tableView(_ tableView: UITableView,
       viewForHeaderInSection section: Int) -> UIView?
    {
```

```
        let mainView = UIView(frame: CGRect(x: 0, y: 0, width: UIScreen.
            main.bounds.width, height: 50))

        let scrollView = UIScrollView()

        scrollView.backgroundColor = .white

        mainView.addSubview(scrollView)
        scrollView.edgesToSuperview()

        scrollView.showsHorizontalScrollIndicator = false

        var buttons: [UIButton] = []
        for hash in post!.hashtags {
            let button = UIButton()
            button.setTitle("#" + hash, for: .normal)
            button.backgroundColor = #colorLiteral(red: 0.8039215803,
                green: 0.8039215803, blue: 0.8039215803, alpha: 1)
            button.layer.cornerRadius = 10
            button.titleLabel?.font = UIFont.systemFont(ofSize: 14)
            button.setTitleColor(.black, for: .normal)
            buttons.append(button)
        }

        let stackView = UIStackView(arrangedSubviews: buttons)

        scrollView.addSubview(stackView)

        stackView.axis = .horizontal
        stackView.alignment = .center
        stackView.spacing = 20

        stackView.edgesToSuperview(insets: .left(20) + .top(10) + .right
            (20))

        scrollView.contentSize = CGSize(width: 500, height: 50)

        switch section {
        case 0:
            return mainView
        default:
            return nil
        }
    }


    override func tableView(_ tableView: UITableView,
        heightForHeaderInSection section: Int) -> CGFloat {
        switch section {
        case 0:
            return 50
        default:
            return 0
        }
    }

    override func tableView(_ tableView: UITableView,
        numberOfRowsInSection section: Int) -> Int
    {
```

```
        switch section {
        case 0:
            return 1
        case 1:
            return post?.comments.count ?? 0
        default:
            return 0
        }
    }

    override func tableView(_ tableView: UITableView, cellForRowAt
       indexPath: IndexPath) -> UITableViewCell
    {
        // TODO: make different cells for sections with switch

        let cell = tableView.dequeueReusableCell(withIdentifier:
           postCellId) as! PostViewCell

        cell.fill(with: post!.dataArray, true)
        cell.selectionStyle = .none
        return cell
    }

}
//
//  HeaderNewsChannels.swift
//  GDproject
//
//  Created by cstore on 13/02/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit
import TinyConstraints

class HeaderNewsChannels: UITableViewCell {

    weak var tableView: UITableView?
    weak var vc: NewsController?

    let basicInfo = ChannelsController()

    override init(style: UITableViewCell.CellStyle, reuseIdentifier:
       String?) {
        super.init(style: style, reuseIdentifier: reuseIdentifier)
        backgroundColor = .white
        setUpCell()
    }

    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
    }

    func setUpCell(){
        addSubview(postsChannelsSegment)
        postsChannelsSegment.selectedSegmentIndex = 0
        postsChannelsSegment.addTarget(self, action: #selector(
           changedValue), for: .valueChanged)
```

```
        postsChannelsSegment.edgesToSuperview(insets: .left(16) + .right
            (16) + .top(8) + .bottom(8))
    }

    // Initialize
    let postsChannelsSegment = UISegmentedControl(items: ["Posts", "
        Channels"])

    @objc func changedValue(_ sender: UISegmentedControl) {
        let index = sender.selectedSegmentIndex
        switch index {
        case 0:
            print("posts")
            vc?.setUpNavigationItemsforPosts()
        case 1:
            print("channels")
            vc?.setUpNavigationItemsForChannels()
//            self.tableView?.delegate = basicInfo
//            self.tableView?.dataSource = basicInfo
//            tableView?.reloadData()
        default:
            break
        }
    }
}

//
//  Constants.swift
//  GDproject
//
//  Created by cstore on 13/02/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit
/// constant for cell in posts NewsController.swift
let postCellId = "PostCell"

///  constant for segmented controll in NewsController.swift and
    ChannelsController.swift
let headerNewsChannelsVC = "HeaderSegmentedControll"

///  for transitioning to full post view
let fullPostControllerId = "FullPostController"

/// constant for dequing basic cell
let basicInfoCellId = "BasicInfoCell"

/// constant for dequing info cell
let infoCellId = "InfoCell"

/// constant for switching to view controller for invitations
let inviteVC = "InviteViewController"

/// constant for dequing basic cell
let channelCellId = "ChannelCell"

/// constant for switching to channel list view controller for choosing
    channel
```

```
let channelListControllerId = "ChannelListController"

/// constant for dequing channel item cell
let itemCellId = "ItemCell"

/// constant for switching to view controller to add or edit one channel
let channelControllerId = "ChannelController"

/// constant for switching to news controller
let newsController = "NewsController"

/// constant for switching to view controller for log in app
let logInController = "LogInController"

/// constant for getting to messages
let messagesViewControllerId = "MessagesViewController"

/// constant for displaying list of people to send message to
let peopleToWriteVC = "PeopleToWriteViewController"

let dialogVC = "DialogViewController"

let profileViewController = "ProfileViewController"

let simplifiedChannelsList = "SimplifiedChannelsList"

let channelViewControllerId = "ChannelViewController"

let addToChannelVCId = "AddToChannelVC"

let taggsSelectionViewController = "TaggsSelectionViewController"

let dialogViewController = "DialogViewController"

let tagsSuggestionController = "TagsSuggestionController"

let chatInfoViewController = "ChatInfoViewController"

let blueSystemColor = UIColor(red: 0, green: 137/255, blue: 249/255, alpha
    : 0.5)

//
//  AddToChannelVC.swift
//  GDproject
//
//  Created by cstore on 02/05/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit

enum DataSourse{
    case people, tags
}

class AddToChannelVC: UITableViewController {

    var channel: Model.Channels?
    weak var update: UpdatableChannel?
```

```
// data sources for people and hashtags
var dataSourcePeople: [Model.Users] = Model.Channels.fullPeople

var reloadtable: Bool = false {
    didSet{
        tableView.reloadData()
    }
}

var filteredDataSource = [Model.Users]()

func filterContentForSearchText(_ searchText: String, scope: String =
   "All")
{
    filteredDataSource = dataSourcePeople.filter { $0.fullName().
       lowercased().contains(searchText.lowercased()) }
    tableView.reloadData()
}

let searchC = UISearchController(searchResultsController: nil)

override func viewDidLoad() {
    super.viewDidLoad()

    navigationItem.title = "People"
    tableView.register(UITableViewCell.self, forCellReuseIdentifier: "
       cell")
    self.navigationItem.searchController = searchC
    navigationItem.largeTitleDisplayMode = .never
    self.navigationItem.hidesSearchBarWhenScrolling = false
    definesPresentationContext = true
    searchC.searchResultsUpdater = self
    searchC.obscuresBackgroundDuringPresentation = false
}

override func viewWillDisappear(_ animated: Bool) {
    super.viewWillDisappear(animated)
    update?.updateChannel(with: channel!)
}
// MARK: - Table view data source

override func numberOfSections(in tableView: UITableView) -> Int {
    return 1
}

var isFiltering: Bool {
    return searchC.isActive && !searchBarIsEmpty()
}

func searchBarIsEmpty() -> Bool {
    return searchC.searchBar.text?.isEmpty ?? true
}

override func tableView(_ tableView: UITableView,
   numberOfRowsInSection section: Int) -> Int {
    if isFiltering {
        return filteredDataSource.count
    } else {
```

```
            return dataSourcePeople.count
        }
    }

    override func tableView(_ tableView: UITableView, cellForRowAt
        indexPath: IndexPath) -> UITableViewCell
    {
        let cell = tableView.dequeueReusableCell(withIdentifier: "cell",
            for: indexPath)

        if isFiltering{
            cell.textLabel?.text = filteredDataSource[indexPath.row].
                fullName()
            if channel!.people.contains(filteredDataSource[indexPath.row].
                id) {
                cell.backgroundColor = #colorLiteral(red: 0.4745098054,
                    green: 0.8392156959, blue: 0.9764705896, alpha: 1)
            } else {
                cell.backgroundColor = #colorLiteral(red: 1, green: 1,
                    blue: 1, alpha: 1)
            }
        } else {
            cell.textLabel?.text = dataSourcePeople[indexPath.row].
                fullName()
            if channel!.people.contains(dataSourcePeople[indexPath.row].id
                ) {
                cell.backgroundColor = #colorLiteral(red: 0.4745098054,
                    green: 0.8392156959, blue: 0.9764705896, alpha: 1)
            } else {
                cell.backgroundColor = #colorLiteral(red: 1, green: 1,
                    blue: 1, alpha: 1)
            }
        }

        cell.selectionStyle = .none

        return cell
    }

    override func tableView(_ tableView: UITableView, didSelectRowAt
        indexPath: IndexPath) {
        if let cell = tableView.cellForRow(at: indexPath) {
            if isFiltering{
                if cell.backgroundColor == #colorLiteral(red:
                    0.4745098054, green: 0.8392156959, blue: 0.9764705896,
                    alpha: 1) {
                    let filtered = channel!.people.filter{ $0 !=
                        filteredDataSource[indexPath.row].id }
                    channel?.people = filtered
                    cell.backgroundColor = #colorLiteral(red:
                        0.9999960065, green: 1, blue: 1, alpha: 1)
                } else {
                    channel?.people.append(filteredDataSource[indexPath.
                        row].id)
                    cell.backgroundColor = #colorLiteral(red:
                        0.4745098054, green: 0.8392156959, blue:
                        0.9764705896, alpha: 1)
                }
            } else {
```

```
                if cell.backgroundColor == #colorLiteral(red:
                   0.4745098054, green: 0.8392156959, blue: 0.9764705896,
                   alpha: 1) {
                    let filtered = channel!.people.filter{ $0 !=
                       dataSourcePeople[indexPath.row].id }
                    channel?.people = filtered
                    cell.backgroundColor = #colorLiteral(red:
                       0.9999960065, green: 1, blue: 1, alpha: 1)
                } else {
                    channel?.people.append(dataSourcePeople[indexPath.row
                       ].id)
                    cell.backgroundColor = #colorLiteral(red:
                       0.4745098054, green: 0.8392156959, blue:
                       0.9764705896, alpha: 1)
                }
            }
        }
    }
}

extension AddToChannelVC : UISearchResultsUpdating {
    func updateSearchResults(for searchController: UISearchController) {
        filterContentForSearchText(searchController.searchBar.text!)
    }
}
//
//  ChannelController.swift
//  GDproject
//
//  Created by cstore on 19/02/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit

protocol DataDelegate {
    func passData(for row: Int, channel: Model.Channels)
}

enum ActiveTable {
    case tags
    case people
}

class ChannelController: UIViewController, UITableViewDelegate,
   UITableViewDataSource, UITextFieldDelegate, UISearchBarDelegate
{
    @IBOutlet weak var tableView: UITableView!

    var index: Int?
    var channel: Model.Channels?
    var myProtocol: DataDelegate?

    var fullTags: [String] = CompletionTree.getCompletion(tree: Model.
       hashTagTree!, word: "")

    var dataSourcePeople: [Model.Users] = []

    var dataSourceTags: [String] = []
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.03-01 ТЗ 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

```
var activeDataSource: ActiveTable = .people

// func to show preview of the current channel
var onShowingPreview: ((Model.Channels)->())?

@IBOutlet weak var viewww: UIView!

@IBOutlet weak var textField: UITextField!

override func viewDidLoad() {
    super.viewDidLoad()

    dataSourceTags = channel!.tags
    setUpController()
    tableView.reloadData()
}

var reloadtable: Bool = false {
    didSet{
        tableView.reloadData()
    }
}

override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    Model.getUsers(for: channel!.people) { [weak self] (people) in
        self?.dataSourcePeople = Array(people.values)
        self?.reloadtable = true
    }
}

// TODO: update channel
override func viewWillDisappear(_ animated: Bool) {

    defer {
        super.viewWillDisappear(animated)
    }

    guard let _ = self.navigationController?.viewControllers.lastIndex
       (of: self) else {
        if let _ = channel?.id {         print("update")
            Model.updateChannel(with: channel!)
        } else {                         print("create")
           //Model.createChannel(with: channel!)
        }
        return
    }
    // nou
}

@objc func showPreview(){
    if let channel = channel {
        onShowingPreview?(channel)
    }
}

func setUpController(){
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.03-01 ТЗ 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

```
        // setting up the preview button for the editing or created
           channel
        navigationItem.rightBarButtonItem = UIBarButtonItem(title: "
           Preview", style: .done, target: self, action: #selector(
           showPreview))

        tableView.delegate = self
        tableView.dataSource = self
        tableView.register(UITableViewCell.self, forCellReuseIdentifier:
           itemCellId)
        tableView.reloadData()
        textField.delegate = self
        searchBar.delegate = self
        searchBar.selectedScopeButtonIndex = 0

        navigationItem.title = channel?.name
        textField.placeholder = channel?.name

        textField.addTarget(self, action: #selector(changedText(_:)), for:
            .editingChanged)
    }



    @objc func changedText(_ textField: UITextField){
        navigationItem.title = textField.text
        channel?.name = textField.text!
        // myProtocol?.passData(for: index!, channel: channel!)
    }

    func numberOfSections(in tableView: UITableView) -> Int {
        return 1
    }

    func tableView(_ tableView: UITableView, numberOfRowsInSection section
       : Int) -> Int {
        switch activeDataSource {
        case .people:
            return dataSourcePeople.count
        case .tags:
            return dataSourceTags.count
        }
    }

    func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
       IndexPath) {
        if searchBar.isFirstResponder, let cell = tableView.cellForRow(at:
            indexPath)
        {
                if cell.accessoryType == .none{
                    switch searchBar.selectedScopeButtonIndex{
                    case 0:
                        channel?.people.append(dataSourcePeople[indexPath.
                           row].id)
                        print("people = \(channel!.people)")
                    case 1:
                        channel?.tags.append(dataSourceTags[indexPath.row
                           ])
                        print("hashtags = \(channel!.tags)")
```

```
                default:
                    break
                }
                cell.accessoryType = .checkmark
            } else {
                cell.accessoryType = .none

                switch searchBar.selectedScopeButtonIndex{
                case 0:
                    let filtered = channel!.people.filter{ $0 != Model
                        .Channels.fullPeople[indexPath.row].id }
                    channel?.people = filtered
                case 1:
                    let filtered = channel!.tags.filter{
                        !fullTags[indexPath.row].contains($0)
                    }
                    channel?.tags = filtered
                default:
                    break
                }
            }
        }
        else
        {
            switch activeDataSource{
            case .people:
                channel?.people.remove(at: indexPath.row)
            case .tags:
                channel?.tags.remove(at: indexPath.row)
            }

            tableView.beginUpdates()
            switch activeDataSource{
            case .people:
                dataSourcePeople.remove(at: indexPath.row)
            case .tags:
                dataSourceTags.remove(at: indexPath.row)
            }
            tableView.deleteRows(at: [indexPath], with: .none)
            tableView.endUpdates()
        }
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath:
        IndexPath) -> UITableViewCell
    {
        let cell = tableView.dequeueReusableCell(withIdentifier:
            itemCellId, for: indexPath)

        cell.textLabel?.text = activeDataSource == .people ? "\(
            dataSourcePeople[indexPath.row].firstName) \(dataSourcePeople[
            indexPath.row].middleName) \(dataSourcePeople[indexPath.row].
            lastName)" : dataSourceTags[indexPath.row]
        cell.selectionStyle = .none

        if activeDataSource == .tags && channel!.tags.contains(
            dataSourceTags[indexPath.row]) ||
            activeDataSource == .people && channel!.people.contains(
                dataSourcePeople[indexPath.row].id)
```

```
        {
            cell.accessoryType = .checkmark
        } else {
            cell.accessoryType = .none
        }

        return cell
    }

    func tableView(_ tableView: UITableView, heightForHeaderInSection
       section: Int) -> CGFloat {
        return 100
    }

    let searchBar: UISearchBar = {
        let searchBar = UISearchBar(frame: CGRect(x: 0, y: 0, width:
           UIScreen.main.bounds.width, height: 70))
        searchBar.showsScopeBar = true
        searchBar.showsCancelButton = true
        searchBar.scopeButtonTitles = ["People", "Tags"]
        return searchBar
    }()

    func tableView(_ tableView: UITableView, viewForHeaderInSection
       section: Int) -> UIView?
    {
        let view = UIView(frame: CGRect(x: 0, y: 0, width: UIScreen.main.
           bounds.width, height: 70))

        if activeDataSource == .tags {
            searchBar.selectedScopeButtonIndex = 1
        } else {
            searchBar.selectedScopeButtonIndex = 0
        }

        view.addSubview(searchBar)
        return view
    }

    func textFieldShouldReturn(_ textField: UITextField) -> Bool
    {
        textField.resignFirstResponder()
        return true
    }

    func searchBar(_ searchBar: UISearchBar, textDidChange searchText:
       String) {
        if searchText.isEmpty {
            switch activeDataSource {
            case .people:
                dataSourcePeople = Model.Channels.fullPeople
            case .tags:
                dataSourceTags = fullTags
            }
            tableView.reloadData()
        } else {
            filterTableView(ind: searchBar.selectedScopeButtonIndex, text:
                searchText)
        }
```

```
    }

    func filterTableView(ind: Int, text: String){
        switch ind {
        case 0:
            dataSourcePeople = Model.Channels.fullPeople.filter {
                let fullName = "\($0.firstName) \($0.middleName) \($0.
                   lastName)"
                return fullName.lowercased().contains(text.lowercased())
            }
            tableView.reloadData()
        case 1:
            dataSourceTags = CompletionTree.getCompletion(tree: Model.
               hashTagTree!, word: text)
            tableView.reloadData()
        default:
            break
        }
    }

    func searchBar(_ searchBar: UISearchBar,
       selectedScopeButtonIndexDidChange selectedScope: Int) {
        activeDataSource = searchBar.selectedScopeButtonIndex == 0 ? .
           people : .tags
        if searchBar.isFirstResponder {
            switch activeDataSource{
            case .people :
                dataSourcePeople = Model.Channels.fullPeople
                dataSourceTags = []
            case .tags:
                dataSourceTags = fullTags
                dataSourcePeople = []
            }
            if (!(searchBar.text?.isEmpty ?? true))
            {
                filterTableView(ind: selectedScope, text: searchBar.text!)
            }
            tableView.reloadData()
        } else if !(searchBar.isFirstResponder){
            switch activeDataSource{
            case .people :
                dataSourcePeople = channel!.people.map({ (item) -> Model.
                   Users in
                    Model.Channels.fullPeopleDict[item]!
                })
                dataSourceTags = []
            case .tags:
                dataSourceTags = channel!.tags
                dataSourcePeople = []
            }
            tableView.reloadData()
        }
    }

    func searchBarTextDidEndEditing(_ searchBar: UISearchBar) {
        switch activeDataSource{
        case .people :
            dataSourcePeople = channel!.people.map({ (item) -> Model.Users
                in
```

```
                Model.Channels.fullPeopleDict[item]!
            })
            dataSourceTags = []
        case .tags:
            dataSourceTags = channel!.tags
            dataSourcePeople = []
        }
        activeDataSource = searchBar.selectedScopeButtonIndex == 0 ? .
           people : .tags
        tableView.reloadData()
    }

    func searchBarTextDidBeginEditing(_ searchBar: UISearchBar) {
        switch activeDataSource{
        case .people :
            dataSourcePeople = Model.Channels.fullPeople
            dataSourceTags = []
        case .tags:
            dataSourceTags = fullTags
            dataSourcePeople = []
        }
        activeDataSource = searchBar.selectedScopeButtonIndex == 0 ? .
           people : .tags
        tableView.reloadData()
    }

    func searchBarCancelButtonClicked(_ searchBar: UISearchBar) {
        searchBar.text = ""
        searchBar.resignFirstResponder()
    }
}
//
//  ChannelListController.swift
//  GDproject
//
//  Created by cstore on 19/02/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit

struct ChannelData{
    var title = String()
    var subtitle = String()
}

protocol ChannelListData: class {
    func reloadData (with channels: [Model.Channels])
}

// TODO: make search controller availiable
class ChannelListController: UITableViewController {

    // MARK: - Output -
    var onChannelSelected: ((Model.Channels) -> Void)?
    var onEditingModeBegins: ((Model.Channels, IndexPath)->Void)?

    // MARK: - filter search controller
    var filteredDataSource = [Model.Channels]()
```

```
var displayingChannel: Model.Channels?

var toReload: Bool = false {
    didSet {
        tableView.reloadData()
    }
}

var isFiltering: Bool {
    return searchController.isActive && !searchBarIsEmpty()
}

func searchBarIsEmpty() -> Bool {
    return searchController.searchBar.text?.isEmpty ?? true
}

func filterContentForSearchText(_ searchText: String, scope: String =
   "All")
{
    filteredDataSource = dataSource.filter({(keys : Model.Channels) ->
        Bool in
        return keys.name.lowercased().contains(searchText.lowercased()
           )
    })

    tableView.reloadData()
}

func passData(for row: Int, channel: Model.Channels) {
    // dataSource[row] = channel
}

var searchController = UISearchController(searchResultsController: nil
   )

override func viewDidLoad() {
    super.viewDidLoad()

    Model.usersAllGet()

    setUpNavigationBar()
    navigationItem.title = "Channels"
    searchController.searchResultsUpdater = self
    searchController.obscuresBackgroundDuringPresentation = false
    searchController.searchBar.placeholder  = "Search channel"
    navigationItem.searchController = searchController
    definesPresentationContext = true
}

func setUpNavigationBar(){
    navigationItem.largeTitleDisplayMode = .always
    navigationItem.rightBarButtonItems = [UIBarButtonItem(
       barButtonSystemItem: .add, target: self, action: #selector(
       addChannel))]
}

override func viewDidAppear(_ animated: Bool)
{
```

```
        super.viewDidAppear(animated)
        searchController.isActive = false
        askForUpdates()
    }

    func askForUpdates(){
        Model.channelsList { [weak self] (channels) in
            self?.dataSource = [ChannelListController.generalChannel] +
                channels
            self?.toReload = true
        }
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(true)
        //askForUpdates()
        tabBarController?.tabBar.isHidden = false
    }

    @objc func addChannel()
    {
        // editing mode is on automatically
        onEditingModeBegins?(Model.Channels(people: [], name: "Untitled",
            id: 0, tags: []),IndexPath(row: 1, section: 0))
    }

    static let generalChannel = Model.Channels(people: [], name: "General
        ", id: -1, tags: [])

    var dataSource : [Model.Channels] = [ ChannelListController.
        generalChannel ]

    override func tableView(_ tableView: UITableView,
        numberOfRowsInSection section: Int) -> Int {
        if isFiltering {
            return filteredDataSource.count
        } else {
            return dataSource.count
        }
    }

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

        let cell = tableView.dequeueReusableCell(withIdentifier:
            channelCellId, for: indexPath)


        if isFiltering{
            cell.textLabel?.text = filteredDataSource[indexPath.row].name
            cell.detailTextLabel?.text = filteredDataSource[indexPath.row
                ].tags.reduce(String(), { (r, next) -> String in "\(r) \(
                next)" })
        } else {
            cell.textLabel?.text = dataSource[indexPath.row].name
            cell.detailTextLabel?.text = dataSource[indexPath.row].tags.
                reduce(String(), { (r, next) -> String in "\(r) \(next)" })
        }
```

```
        return cell
    }


    override func tableView(_ tableView: UITableView, canEditRowAt
       indexPath: IndexPath) -> Bool {
        switch indexPath.row {
        case 0:
            return false
        default:
            return true
        }
    }

    override func tableView(_ tableView: UITableView, editActionsForRowAt
       indexPath: IndexPath) -> [UITableViewRowAction]?
    {
        let editButton = UITableViewRowAction(style: .normal, title: "Edit
           ") { [unowned self] (rowAction, indexPath) in
            self.onEditingModeBegins?(self.dataSource[indexPath.row],
               indexPath)
        }

        editButton.backgroundColor = #colorLiteral(red: 0.8039215803,
           green: 0.8039215803, blue: 0.8039215803, alpha: 1)

        let deleteButton = UITableViewRowAction(style: .normal, title: "
           Delete") { [weak self] (action, indexPath) in

            Model.channelsDelete(by: self!.dataSource[indexPath.row].id!,
               completion: {

            })

            self?.tableView.beginUpdates()
            self?.dataSource.remove(at: indexPath.row)
            self?.tableView.deleteRows(at: [indexPath], with: .none)
            self?.tableView.endUpdates()
        }

        deleteButton.backgroundColor = #colorLiteral(red: 1, green:
           0.1491314173, blue: 0, alpha: 1)

        return [editButton, deleteButton]
    }

    override func tableView(_ tableView: UITableView, didSelectRowAt
       indexPath: IndexPath)
    {
        if isFiltering {
            onChannelSelected?(filteredDataSource[indexPath.row])
        } else {
            onChannelSelected?(dataSource[indexPath.row])
        }
    }
}
```

```
extension ChannelListController : UISearchResultsUpdating {
    func updateSearchResults(for searchController: UISearchController) {
        filterContentForSearchText(searchController.searchBar.text!)
    }
}
//
//  ChannelViewController.swift
//  MessageApp
//
//  Created by cstore on 02/05/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit

protocol UpdatableName: class{
    func updateName(with name: String)
}

protocol UpdatableChannel: class{
    func updateChannel(with channel: Model.Channels)
}

class ChannelViewController: UITableViewController, UpdatableName,
   UpdatableChannel, TagsReceiver
{
    var onChoosingHashTags: (([String]?)->())?

    var onChoosingPeople: ((Model.Channels?)->())?

    func receiveTags(tags: [String]) {
        self.channel?.tags = tags
    }

    func updateName(with name: String){
        channel?.name = name
    }

    func updateChannel(with channel: Model.Channels) {
        self.channel = channel
    }

    var channel: Model.Channels?

    // func to show preview of the current channel
    var onShowingPreview: ((Model.Channels)->())?

    override func viewDidLoad()
    {
        super.viewDidLoad()
        tableView.keyboardDismissMode = .onDrag
        navigationItem.rightBarButtonItems = [ UIBarButtonItem(
           barButtonSystemItem: .play, target: self, action: #selector(
           showPreview)), self.editButtonItem]
        navigationItem.largeTitleDisplayMode = .never
        tableView.register(UITableViewCell.self, forCellReuseIdentifier: "
           cell")
        tableView.register(UITableViewCell.self, forCellReuseIdentifier: "
           searchCell")
```

```
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        tableView.reloadData()
    }

    override func viewWillDisappear(_ animated: Bool) {

        defer {
            super.viewWillDisappear(animated)
        }

        guard let _ = self.navigationController?.viewControllers.lastIndex
           (of: self) else
        {
            if let id = channel?.id, id != 0
            {
                Model.updateChannel(with: channel!)
            } else {
                Model.createChannel(with: channel!) {
                    [weak self] in
                    self?.showAlertOn(result: $0)
                    return
                }
            }
            return
        }
        // nou
    }

    @objc func showPreview()
    {
        if let channel = channel {
            onShowingPreview?(channel)
        }
    }

    // MARK: - Table view data source

    override func numberOfSections(in tableView: UITableView) -> Int {
        return 3
    }

    override func tableView(_ tableView: UITableView,
       numberOfRowsInSection section: Int) -> Int {
        switch section {
        case 1:
            return (channel?.people.count ?? 0) + 1
        case 2:
            return (channel?.tags.count ?? 0) + 1
        default:
            return 1
        }
    }

    override func tableView(_ tableView: UITableView,
       titleForHeaderInSection section: Int) -> String? {
        switch section {
```

```
        case 1:
            return "People"
        case 2:
            return "Tags"
        default:
            return "Title"
        }
    }

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

        switch indexPath.section{
        case 1 , 2:
            return whichCell(tableView, cellForRowAt: indexPath)
        default:
            let cell = tableView.dequeueReusableCell(withIdentifier: "
               titleCell") as! TitleCell

            cell.fill(title: channel?.name)
            cell.update = self

            return cell
        }
    }

    private func whichCell(_ tableView: UITableView, cellForRowAt
       indexPath: IndexPath) -> UITableViewCell
    {
        let row = indexPath.row
        let section = indexPath.section

        if row == 0
        {
            let cell = tableView.dequeueReusableCell(withIdentifier: "
               searchCell")!
            cell.textLabel?.text = "Add more"
            cell.accessoryType = .disclosureIndicator
            return cell
        }

        let cell = tableView.dequeueReusableCell(withIdentifier: "cell",
           for: indexPath)

        if section == 1 {
            cell.textLabel?.text = " \(Model.Channels.fullPeopleDict[
               channel?.people[row-1] ?? 0]!.fullName())"
        } else {
            cell.textLabel?.text = "# \(channel!.tags[row-1])"
        }

        return cell
    }

    override func tableView(_ tableView: UITableView, commit editingStyle:
        UITableViewCell.EditingStyle, forRowAt indexPath: IndexPath)
    {
        if editingStyle == UITableViewCell.EditingStyle.delete
        {
```

```
            if indexPath.section == 1 {
                channel?.people.remove(at: indexPath.row-1)
            } else {
                channel?.tags.remove(at: indexPath.row-1)
            }
        }

        tableView.reloadData()
    }

    override func tableView(_ tableView: UITableView, canEditRowAt
       indexPath: IndexPath) -> Bool {
        if indexPath.row == 0 {
            return false
        }

        return true
    }

    override func tableView(_ tableView: UITableView, didSelectRowAt
       indexPath: IndexPath)
    {
        if indexPath.row == 0 && indexPath.section == 1
        {
            onChoosingPeople?(channel)
        }

        if indexPath.row == 0 && indexPath.section == 2 {
            onChoosingHashTags?(channel?.tags)
        }
    }
}


class TitleCell: UITableViewCell, UITextFieldDelegate {

    @IBOutlet weak var titleLabel: UITextField!

    weak var update: UpdatableName?

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

    func fill(title: String?){
        titleLabel.text = title
        titleLabel.addTarget(self, action: #selector(changedText(_:)), for
           : .editingChanged)
    }

    @objc func changedText(_ textField: UITextField){
        update?.updateName(with: textField.text!)
    }
}
//
//  FullPostViewController.swift
//  GDproject
//
//  Created by cstore on 13/02/2019.
```

```
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit
import TinyConstraints

class FullPostController: UITableViewController {

    var post: Model.Posts?
    var type: HeaderType = .NONE

    override func viewDidLoad() {
        super.viewDidLoad()

        tableView.register(PostViewCell.self, forCellReuseIdentifier:
           postCellId)

        setUpNavigationBar()
        tableView.separatorStyle = .none
    }

    func setUpNavigationBar(){
        navigationItem.largeTitleDisplayMode = .never
        navigationItem.title = "Post"
    }


    // MARK: - Table view data source

    override func numberOfSections(in tableView: UITableView) -> Int {
        // #warning Incomplete implementation, return the number of
           sections
        return 2
    }


    override func tableView(_ tableView: UITableView,
       viewForHeaderInSection section: Int) -> UIView?
    {
        switch section {
        case 0:
            let v = UIView(frame: CGRect(x: 0, y: 0, width: UIScreen.main.
               bounds.width, height: 1))
            v.backgroundColor = .white
            return v
        default:
            return nil
        }
    }


    override func tableView(_ tableView: UITableView,
       heightForHeaderInSection section: Int) -> CGFloat {
        switch section {
        case 0:
            return 1
        default:
            return 0
        }
```

```
    }

    override func tableView(_ tableView: UITableView,
       numberOfRowsInSection section: Int) -> Int
    {
        switch section {
        case 0:
            return 1
        case 1:
            return 0
        default:
            return 0
        }
    }

    override func tableView(_ tableView: UITableView, cellForRowAt
       indexPath: IndexPath) -> UITableViewCell
    {
        // TODO: make different cells for sections with switch

        let cell = tableView.dequeueReusableCell(withIdentifier:
           postCellId) as! PostViewCell


        cell.fill(with: PostCellData.create(with: post!.body), true, post:
            post!)

        switch type {
        case .NEWS:
            cell.onUserDisplay = { [weak self] (id) in
                let vc = self?.storyboard!.instantiateViewController(
                   withIdentifier: profileViewController) as!
                   ProfileViewController
                vc.idProfile = id

                self?.navigationController?.pushViewController(vc,
                   animated: true)
            }
        default:
            break
        }

        cell.selectionStyle = .none
        return cell
    }

}
//
//  NewPostViewController.swift
//  GDproject
//
//  Created by cstore on 05/02/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit
import Cartography
import Marklight
import TinyConstraints
```

```
import TaggerKit
// import WSTagsField

class NewPostViewController: UIViewController, UITextViewDelegate,
   TagsReceiver
{
    func receiveTags(tags: [String]) {
        currentTags = tags
    }

    @IBOutlet weak var view1: UIView!

    var currentTags: [String] = []

    // We want the whole experience, let's create two TKCollectionViews
    let productTags = TKCollectionView()

    @IBAction func chooseTags(_ sender: UIButton) {
        let vc = storyboard?.instantiateViewController(withIdentifier:
           taggsSelectionViewController) as! TaggsSelectionViewController

        NewPostViewController.draft = textView.text
        vc.receiver = self
        vc.currentTags = currentTags

        navigationController?.pushViewController(vc, animated: true)
    }

    // fileprivate let tagsField = WSTagsField()
    // Keep strong instance of the `NSTextStorage` subclass
    let textStorage = MarklightTextStorage()

    static var draft: String = ""
    static var hashTagsDraft: [String] = [ ]

    var textView: UITextView!

    // buttons for attaching images
    var accessoryView: UIView = {
        let view = UIView()
        view.backgroundColor = .white
        return view
    }()

    var addEquasion: UIButton = {
        var button = UIButton(type: .detailDisclosure)
        button.addTarget(self, action: #selector(addMathBrackets), for: .
           touchUpInside)
        return button
    }()

    // stack view where buttons will be places
    var stackAccessoryView: UIStackView?
    // Connect the view1's bottom layout constraint to react to keyboard
       movements

    @IBOutlet weak var bottomTextViewConstraint: NSLayoutConstraint!
```

```
    override func viewDidLoad()
    {
        super.viewDidLoad()
        //createTimerForUpdates()

        setUpMD()
        setUpTextView()
        // setUpAccessoryView()
        //setUpTagsView()
    }

   /* func setUpTagsView(){
        tagsField.frame = viewForTags.bounds
        viewForTags.addSubview(tagsField)

        // tagsField.translatesAutoresizingMaskIntoConstraints = false
        // tagsField.heightAnchor.constraint(lessThanOrEqualToConstant:
           150).isActive = true

        tagsField.cornerRadius = 3.0
        tagsField.spaceBetweenLines = 10
        tagsField.spaceBetweenTags = 10

        //tagsField.numberOfLines = 3
        //tagsField.maxHeight = 100.0

        tagsField.layoutMargins = UIEdgeInsets(top: 2, left: 6, bottom: 2,
            right: 6)
        tagsField.contentInset = UIEdgeInsets(top: 10, left: 10, bottom:
           10, right: 10) //old padding

        tagsField.placeholder = "Enter a tag"
        tagsField.placeholderColor = #colorLiteral(red: 0.2392156869,
           green: 0.6745098233, blue: 0.9686274529, alpha: 1)
        tagsField.placeholderAlwaysVisible = true
        tagsField.backgroundColor = #colorLiteral(red: 0.937254902, green:
            0.937254902, blue: 0.9568627451, alpha: 1)
        tagsField.returnKeyType = .next
        tagsField.delimiter = ""

        tagsField.textDelegate = self
        tagsField.acceptTagOption = .space

        textFieldEvents()
    }*/

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)

        //navigationController?.navigationBar.prefersLargeTitles = false
        navigationItem.largeTitleDisplayMode = .never
        navigationItem.title = "New post"
        textView.text = NewPostViewController.draft
        // tagsField.addTags(NewPostViewController.hashTagsDraft)

    }

    func setUpAccessoryView(){
        let views = [UIButton(type: .contactAdd),addEquasion]
```

```
        stackAccessoryView = UIStackView(arrangedSubviews: views)

        stackAccessoryView?.alignment = .fill
        stackAccessoryView?.distribution = .equalSpacing

        view1.addSubview(accessoryView)
        accessoryView.addSubview(stackAccessoryView!)
        accessoryView.height(40)

        let separatorView = UIView()
        separatorView.backgroundColor = #colorLiteral(red: 0.8039215803,
           green: 0.8039215803, blue: 0.8039215803, alpha: 1)
        accessoryView.addSubview(separatorView)
        separatorView.height(1)
        separatorView.edgesToSuperview(excluding: .bottom)

        accessoryView.edgesToSuperview(excluding: [.top])
        stackAccessoryView!.edgesToSuperview(insets: .left(16) + .right
           (16) + .top(1))
    }

    func setUpTextView(){
        view.addConstraint(bottomTextViewConstraint)
        view1.addSubview(textView)
        textView.edgesToSuperview(insets: .top(8) + .left(8) + .bottom(8)
           + .right(8))

        if #available(iOS 11.0, *) {
            textView.smartDashesType = .no
            textView.smartQuotesType = .no
        }

        textView.isScrollEnabled = true

        guard let bottomTextViewConstraint = bottomTextViewConstraint else
            { return }
        view.addConstraint(bottomTextViewConstraint)

        // Add a beautiful padding to the `UITextView` content
        textView.textContainerInset = UIEdgeInsets(top: 4, left: 4, bottom
           : 4, right: 4)
        textView.delegate = self
        magicForKeyboardChanges()
    }


    func magicForKeyboardChanges()
    {
        ////////////////////////////
        // We do some magic to resize the `UITextView` to react the the
           keyboard size change (appearance, disappearance, ecc)
        NotificationCenter.default.addObserver(self, selector: #selector(
           handleKeyboardNotifications), name: UIResponder.
           keyboardWillShowNotification, object: nil)

        NotificationCenter.default.addObserver(self, selector: #selector(
           handleKeyboardNotifications), name: UIResponder.
           keyboardWillHideNotification, object: nil)
```

```
        // Partial fixes to a long standing bug, to keep the caret inside
           the ‘UITextView‘ always visible
        NotificationCenter.default.addObserver(forName: UITextView.
           textDidChangeNotification, object: textView, queue:
           OperationQueue.main) { [weak self] (notification) -> Void in
            if self!.textView.textStorage.string.hasSuffix("\n") {
                CATransaction.setCompletionBlock({ () -> Void in
                    self!.scrollToCaret(self!.textView, animated: false)
                })
            } else {
                self!.scrollToCaret(self!.textView, animated: false)
            }
        }

    }

    @objc func handleKeyboardNotifications(notification: NSNotification){
        if let userInfo = notification.userInfo{
            // UIKeyboardFrameEndUserInfoKey

            let keyBoardFrame = (userInfo[UIResponder.
               keyboardFrameEndUserInfoKey] as! NSValue).cgRectValue

            bottomTextViewConstraint?.constant = notification.name ==
               UIResponder.keyboardWillShowNotification ? keyBoardFrame.
               height - tabBarController!.tabBar.frame.height : 0

            print(bottomTextViewConstraint!.constant)

            UIView.animate(withDuration: 0, delay: 0, options: .
               curveEaseOut, animations: {
                self.view.layoutIfNeeded()
            }) { (completed) in

            }
        }
    }

    @objc func attachPhoto(){

    }


    @objc func addMathBrackets()
    {
       // nothing
    }

    func setUpMD(){
        textStorage.marklightTextProcessor.codeColor = UIColor.gray
        textStorage.marklightTextProcessor.quoteColor = UIColor.darkGray
        textStorage.marklightTextProcessor.syntaxColor = UIColor.blue
        textStorage.marklightTextProcessor.codeFontName = "Courier"
        textStorage.marklightTextProcessor.fontTextStyle = UIFont.
           TextStyle.subheadline.rawValue
        textStorage.marklightTextProcessor.hideSyntax = false

        let layoutManager = NSLayoutManager()
```

```
        // Assign the 'UITextView''s 'NSLayoutManager' to the '
           NSTextStorage' subclass
        //textStorage.addLayoutManager(textView.layoutManager)
        textStorage.addLayoutManager(layoutManager)

        let textContainer = NSTextContainer()
        layoutManager.addTextContainer(textContainer)

        textView = UITextView(frame: view.bounds, textContainer:
           textContainer)
    }

    var indexOfPost = 0
    // MARK:- new post
    @objc func newPost(){
        Model.createAndPublish(body: [Model.Attachments(markdown: textView
           !.text)], tags: currentTags) {
            [weak self] in

            switch $0 {
            case .success1, .success:
                NewPostViewController.draft = ""
                NewPostViewController.hashTagsDraft = []
                self?.moveBackToParentVC?()
            default:
                // self?.actionSaveDraft()
                self?.showAlertOn(result: $0)
            }
        }

        // NewPostViewController.draft = ""
        // NewPostViewController.hashTagsDraft = []
        // adding row to uiTableView after adding new post
        // myProtocol?.addPost(post: p)
        // moveBackToParentVC?()
        // somewhere here i will be sending server notifications about new
            post arrival
    }

    @objc func actionSaveDraft(){
        let optionMenu = UIAlertController(title: nil, message: nil,
           preferredStyle: .actionSheet)

        let saveAction = UIAlertAction(title: "Save draft", style: .
           default)
        {
            [weak self]
            _ in
            NewPostViewController.draft = self?.textView.text ?? ""
            NewPostViewController.hashTagsDraft = self?.currentTags ?? []
            self?.moveBackToParentVC?()
        }

        let deleteAction = UIAlertAction(title: "Delete draft", style: .
           destructive)
        {
            [weak self] (_)
            in
            NewPostViewController.draft = ""
```

```
            NewPostViewController.hashTagsDraft = []
            self?.moveBackToParentVC?()
        }

        let cancelAction = UIAlertAction(title: "Cancel", style: .cancel)

        optionMenu.addAction(saveAction)
        optionMenu.addAction(deleteAction)
        optionMenu.addAction(cancelAction)

        self.present(optionMenu, animated: true, completion: nil)
    }

    @objc func closeView()
    {
        actionSaveDraft()
    }

    var moveBackToParentVC: (()->())?


    func textView(_ textView: UITextView, shouldInteractWith URL: URL, in
       characterRange: NSRange) -> Bool {
        print("Should interact with: \(URL)")
        return true
    }

    func scrollToCaret(_ textView: UITextView, animated: Bool) {
        var rect = textView.caretRect(for: textView.selectedTextRange!.end
           )
        rect.size.height = rect.size.height + textView.textContainerInset.
           bottom
        textView.scrollRectToVisible(rect, animated: animated)
    }

    /*fileprivate func textFieldEvents() {
        tagsField.onDidAddTag = { _, _ in
            print("onDidAddTag")
        }

        tagsField.onDidRemoveTag = { _, _ in
            print("onDidRemoveTag")
        }

        tagsField.onDidChangeText = { _, text in
            print("onDidChangeText")
        }

        tagsField.onDidChangeHeightTo = { _, height in
            print("HeightTo \(height)")
        }

        tagsField.onDidSelectTagView = { _, tagView in
            print("Select \(tagView)")
        }

        tagsField.onDidUnselectTagView = { _, tagView in
            print("Unselect \(tagView)")
        }
```

```
    }*/
}


extension NewPostViewController: UITextFieldDelegate {

    func textFieldShouldReturn(_ textField: UITextField) -> Bool {
        /*if textField == tagsField {
            textView.becomeFirstResponder()
        }*/
        return true
    }

}
//
//  ViewController.swift
//  GDproject
//
//  Created by cstore on 13/02/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit
import TinyConstraints
import Cartography

protocol UpdateableWithChannel: class {
    var channel: Model.Channels? { get set }
    func updateChannel(on channel: Model.Channels)
}

protocol UpdateableWithUser: class {
    var user: Model.Users? { get set }
}
// MARK:- Controller with posts and channels availiable.
// Search is availiable within every table (posts and channels). Has
   button-functionality for boths post and chnnels
class NewsController: UIViewController, UISearchControllerDelegate,
   UpdateableWithChannel, UISearchResultsUpdating
{
    func updateChannel(on channel: Model.Channels) {
        self.channel = channel
        news.currChannel = channel
        decideWhatChannelDisplay()
    }

    var changedChannelName: ((String)->())?

    @IBOutlet weak var tableView: UITableView!

    var channel: Model.Channels?
    var anonymousChannel: (users: [Int: Model.Users], posts: [Model.Posts
       ])?

    // MARK: - Output -
    var onSelectChannel: (() -> Void)?

    var searchController: UISearchController?
```

```
var news = NewsVC()
var type: HeaderType? = .NEWS

var refreshContr =  UIRefreshControl()

override func viewDidLoad()
{
    super.viewDidLoad()

    let updater = TagsSuggestionController()
    updater.delegate = self
    searchController = UISearchController(searchResultsController:
       updater)
    navigationItem.largeTitleDisplayMode = .always

    navigationController?.navigationBar.prefersLargeTitles = true
    navigationItem.title = "Loading ..."
    tableView.refreshControl = refreshContr
    // Configure Refresh Control
    refreshContr.addTarget(self, action: #selector(refreshPostsData
       (_:)), for: .valueChanged)

    tableView.register(PostViewCell.self, forCellReuseIdentifier:
       postCellId)
    tableView.register(UITableViewCell.self, forCellReuseIdentifier: "
       cell")

    setUpSearchContr()

    // navigationItem.
    news.viewController = self
    news.type = type == .NEWS ? .NEWS : type!
    news.currChannel = channel

    setUpNavigationItemsforPosts()
}

func setUpSearchContr(){
    searchController?.delegate = self
    searchController?.obscuresBackgroundDuringPresentation = false
    navigationItem.searchController = searchController
    definesPresentationContext = true
    searchController?.searchResultsUpdater = self
    searchController?.searchBar.placeholder  = "Search tags"
}

@objc func refreshPostsData( _ ff: UIRefreshControl) {
    decideWhatChannelDisplay()
}

deinit {
    print("news clear")
}

override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    searchController?.isActive = false
    // TODO:- display something if no posts are availiable
```

```
        decideWhatChannelDisplay()

        // tableView.reloadData()
    }

    func decideWhatChannelDisplay(){

        switch type! {
        case .NEWS, .NONE:
            if let channel = channel, let id = channel.id, id != -1 {

                Model.getAnonymousChannel(by: channel) { [weak self] in
                    self?.news.dataSourse = $0.posts
                    self?.news.dictionary = $0.users
                    self?.refreshContr.endRefreshing()
                    self?.changedChannelName?(channel.name)
                }

            } else {
                Model.getLast { [weak self] in
                    self?.news.dataSourse = $0.posts
                    self?.news.dictionary = $0.users
                    self?.refreshContr.endRefreshing()
                    self?.changedChannelName?("General")
                }
            }
        default:
            refreshContr.endRefreshing()
            break
        }

    }

    func setUpNavigationItemsforPosts(){
        tableView.delegate = news
        tableView.dataSource = news
        tableView.reloadData()
    }

    func updateSearchResults(for searchController: UISearchController)
    {
        let filteredData = CompletionTree.getCompletion(tree: Model.
           hashTagTree!, word: searchController.searchBar.text?.lowercased
           () ?? "")

        // Apply the filtered results to the search results table.
        if let resultsController = searchController.
           searchResultsController as? TagsSuggestionController {
            resultsController.suggestions = filteredData
            resultsController.tableView.reloadData()
        }
    }

    func willPresentSearchController(_ searchController:
       UISearchController) {
        searchController.searchResultsController?.view.isHidden = false;
    }
```

```
    func didPresentSearchController(_ searchController: UISearchController
        ) {
        searchController.searchResultsController?.view.isHidden = false;
    }

    func didDismissSearchController(_ searchController: UISearchController
        ) {
        searchController.searchBar.text = ""
    }
}
//
//  NewsVC.swift
//  GDproject
//
//  Created by cstore on 15/02/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit
import MarkdownKit

struct PostCellData{
    var attributedData = NSAttributedString()

    static func create(with attachments: [Model.Attachments]) ->
        NSAttributedString{
        var markdown = ""

        attachments.forEach { (attachment) in markdown.append(attachment.
            markdown) }

        let markdownParser = MarkdownParser(font: UIFont.systemFont(ofSize
            : 16))
        markdownParser.enabledElements = .disabledAutomaticLink
        markdownParser.code.textBackgroundColor = #colorLiteral(red:
            0.8039215803, green: 0.8039215803, blue: 0.8039215803, alpha:
            1)

        return markdownParser.parse(markdown)
    }
}

class NewsVC: UIViewController, UITableViewDelegate, UITableViewDataSource
    {


    var onChannelDidChange: ((([Int: Model.Users],[Model.Posts]))->())?

    var onFullPost: ((HeaderType, Model.Posts)->())?

    var dictionary: [Int: Model.Users] = [:] {
        didSet {

            dataSourse = dataSourse.map {
                var copy = $0
                copy.user = dictionary[$0.authorId]
                return copy
            }
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.03-01 ТЗ 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

```
            (viewController as? NewsController)?.tableView.reloadData()
            (viewController as? ProfileViewController)?.tableView.
               reloadData()
        }
    }

    var currChannel : Model.Channels?

    var dataSourse: [Model.Posts] = [] {
        didSet {
            cellDataSourse = dataSourse.map { PostCellData(attributedData:
                PostCellData.create(with: $0.body)) }
        }
    }

    var cellDataSourse: [PostCellData] = []

    var type: HeaderType = .NONE

    weak var viewController: UIViewController?

    func numberOfSections(in tableView: UITableView) -> Int {
        return 1
    }

    func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
       IndexPath)
    {
        onFullPost?(type, dataSourse[indexPath.row])
    }

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

    func tableView(_ tableView: UITableView, cellForRowAt indexPath:
       IndexPath) -> UITableViewCell
    {
        let cell = tableView.dequeueReusableCell(withIdentifier:
           postCellId, for: indexPath) as! PostViewCell

        cell.fill(with: cellDataSourse[indexPath.row].attributedData,
           false, post: dataSourse[indexPath.row])

        switch type {
        case .NEWS:
            cell.onUserDisplay = { [weak self] (id) in
                let vc = self?.viewController!.storyboard!.
                   instantiateViewController(withIdentifier:
                   profileViewController) as! ProfileViewController
                vc.idProfile = id
                self?.viewController!.navigationController!.
                   pushViewController(vc, animated: true)
            }

            cell.onAnonymousChannelDisplay = {
                [weak self] in
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.03-01 ТЗ 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

```
                (self?.viewController as? UpdateableWithChannel)?.
                    updateChannel(on: Model.Channels(people: [], name: $0,
                    id: 0, tags: [$0]))
            }
        default:
            cell.onUserDisplay = { (id) in
                print("tapped when profile is opened already \(id)")
            }
        }

        cell.selectionStyle = .none
        return cell
    }

    func tableView(_ tableView: UITableView, heightForRowAt indexPath:
        IndexPath) -> CGFloat {
        return UITableView.automaticDimension
    }

    func tableView(_ tableView: UITableView, estimatedHeightForRowAt
        indexPath: IndexPath) -> CGFloat {
        return 100.0
    }

    var prevLast = -1

    func tableView(_ tableView: UITableView, willDisplay cell:
        UITableViewCell, forRowAt indexPath: IndexPath)
    {
        // pagination
        if indexPath.row == cellDataSourse.count - 1 && prevLast !=
            indexPath.row
        {
            if let ch = currChannel, let id = ch.id, id != -1 {
                // check this!
                Model.getAnonymousChannel(by: ch, exclusiveFrom:
                    dataSourse.last?.id) { [weak self] in
                    self?.dataSourse.append(contentsOf: $0.posts)
                    $0.users.forEach { self?.dictionary[$0.key] = $0.value
                        }
                }

            } else {
                // check this!
                Model.getLast(on: 10, from: dataSourse.last?.id )
                { [weak self] in
                    self?.dataSourse.append(contentsOf: $0.posts)
                    $0.users.forEach { self?.dictionary[$0.key] = $0.value
                        }
                }
            }

            prevLast = indexPath.row
        }
    }
}

enum HeaderType {
    case NONE
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.03-01 ТЗ 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

```
    case NEWS
    case ANONYMOUS
}
//
//  PostViewCell.swift
//  GDproject
//
//  Created by cstore on 04/02/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit
import Cartography
import TinyConstraints

class PostViewCell: UITableViewCell
{

    var onUserDisplay: ((Int)->())?

    var onAnonymousChannelDisplay: ((String)->())?

    let nameLabel: UIButton = {
        let button = UIButton()
        button.setTitleColor(.black, for: .normal)
        button.titleLabel?.font = UIFont.boldSystemFont(ofSize: 16)
        button.contentHorizontalAlignment = UIControl.
           ContentHorizontalAlignment.left
        return button
    }()

    let fullNameLabel: UILabel = {
        let label = UILabel()
        label.textColor = #colorLiteral(red: 0.3333333433, green:
           0.3333333433, blue: 0.3333333433, alpha: 1)
        label.font = UIFont.systemFont(ofSize: 12)
        return label
    }()

    let timeLabel: UILabel = {
        let label = UILabel()
        label.font = UIFont.systemFont(ofSize: 12)
        label.textColor = #colorLiteral(red: 0.6000000238, green:
           0.6000000238, blue: 0.6000000238, alpha: 1)
        return label
    }()

    let shareButton: UIButton = {
        let button = UIButton(type: UIButton.ButtonType.detailDisclosure)
        button.isHidden = true
        return button
    }()

    override init(style: UITableViewCell.CellStyle, reuseIdentifier:
       String?)
    {
        super.init(style: style, reuseIdentifier: reuseIdentifier)
    }
```

```
required init?(coder aDecoder: NSCoder)
{
    fatalError("init(coder:) has not been implemented")
}

func createTextView(with text: NSAttributedString, _ isSelectable:
   Bool) -> UITextView
{
    let textView = UITextView()
    textView.isScrollEnabled = false
    textView.isEditable = false
    if isSelectable {
        //textView.isUserInteractionEnabled = false
        textView.isSelectable = true
    } else {
        textView.isUserInteractionEnabled = false
    }

    textView.attributedText = text
    return textView
}

var views: [UIView] = []
var post: Model.Posts?

func fill(with info: NSAttributedString, _ isFullVersoin: Bool, post:
   Model.Posts)
{
    self.post = post
    // important
    contentView.subviews.forEach({ $0.removeFromSuperview() })

    views = []
    views.append(createTextView(with: info, isFullVersoin))

    nameLabel.setTitle("\(post.user?.firstName ?? "") \(post.user?.
       lastName ?? "")", for: .normal)
    nameLabel.addTarget(self, action: #selector(displayProfile), for:
       .touchUpInside)
    if let user = post.user{
        fullNameLabel.text = "\(user.faculty.name)"
    }
    else {
        fullNameLabel.text = "\(post.authorId)"
    }

    setUpInStackView(isFullVersoin)
}

var hashtags = [String]()

func setUpInStackView(_ full : Bool){

    hashtags = post!.tags

    let nameStackView = UIStackView(arrangedSubviews: [nameLabel,
       fullNameLabel])
    nameStackView.axis = .vertical
    contentView.addSubview(nameStackView)
```

```
        //nameStackView.height(46)
        nameStackView.edgesToSuperview(excluding: .bottom, insets: .left
            (20) + .right(20) + .top(8))

        let mainView = UIView(frame: CGRect(x: 0, y: 0, width: UIScreen.
            main.bounds.width, height: 30))

        let scrollView = UIScrollView()

        scrollView.backgroundColor = .white

        mainView.addSubview(scrollView)
        scrollView.edgesToSuperview()

        scrollView.showsHorizontalScrollIndicator = false

        var buttons: [UIButton] = []
        for hash in hashtags {
            let button = UIButton()
            button.setTitle("#" + hash, for: .normal)
            button.addTarget(self, action: #selector(setAnonymousChannel(
                on:)), for: .touchUpInside)
            // button.addGestureRecognizer(longPressRecognizer)
            button.titleLabel?.font = UIFont.monospacedDigitSystemFont(
                ofSize: 15, weight: .semibold)
            button.setTitleColor(#colorLiteral(red: 0, green:
                0.4784313725, blue: 1, alpha: 1), for: .normal)
            buttons.append(button)
        }

        let stackViewHashTags = UIStackView(arrangedSubviews: buttons)

        scrollView.addSubview(stackViewHashTags)

        stackViewHashTags.axis = .horizontal
        stackViewHashTags.spacing = 17

        // edgesToSuperview(insets: .left(20) + .top(10) + .right(20))
        stackViewHashTags.horizontalToSuperview(insets: .left(20) + .right
            (20))
        stackViewHashTags.verticalToSuperview()

        stackViewHashTags.distribution = .fillProportionally
        stackViewHashTags.alignment = .center

        scrollView.contentSize = CGSize(width: 500, height: 30)


        contentView.addSubview(mainView)
        mainView.edgesToSuperview(excluding: [.top, .bottom])
        mainView.topToBottom(of: nameStackView, offset: 5)

        // TODO:- change!!!!!!!!!!!
        if hashtags.isEmpty{
            mainView.height(0)
        } else {
            mainView.height(30)
        }
```

```
        let stackView = MyStackView(arrangedSubviews: views)
        stackView.isFull = full
        stackView.axis = .vertical
        contentView.addSubview(stackView)
        if !full {
            stackView.height(300, relation: .equalOrLess, isActive: true)
        }

        stackView.topToBottom(of: mainView, offset: 0, relation: .equal,
           isActive: true)

        stackView.edgesToSuperview(excluding: [.top, .bottom], insets: .
           left(16) + .right(16))

        let commentsSharesStackView = UIStackView(arrangedSubviews: [
           timeLabel, shareButton])
        contentView.addSubview(commentsSharesStackView)

        timeLabel.text = post!.convertDateFormatter()

        commentsSharesStackView.axis = .horizontal
        commentsSharesStackView.distribution = .equalSpacing
        commentsSharesStackView.edgesToSuperview(excluding: .top, insets:
           .bottom(10) + .left(16) + .right(16))
        stackView.bottomToTop(of: commentsSharesStackView)
    }

    @objc func displayProfile(){
        if let id = post?.authorId {
            onUserDisplay?(id)
        }
    }

    @objc func setAnonymousChannel(on button: UIButton){
        print("\(button.titleLabel?.text ?? "nothing")")
        onAnonymousChannelDisplay?(String(button.titleLabel!.text!.
           dropFirst()))
    }
}
//
//  SimplifiedChannelsList.swift
//  GDproject
//
//  Created by cstore on 13/03/2019.
//  Copyright   2019 drHSE. All rights reserved.
//


/// class for ADDing person to existing (or new) channel: like playlist in
    itunes
import UIKit
import TinyConstraints

class SimplifiedChannelsList: UITableViewController {

    var user: Model.Users?

    var dataSource: [Model.Channels]?{
        didSet{
```

```
            tableView.reloadData()
        }
    }

    override func viewDidLoad() {
        super.viewDidLoad()
        setUpNavigationButtons()
        navigationItem.title = "All channels"
        navigationItem.largeTitleDisplayMode = .never
        self.navigationItem.setHidesBackButton(true, animated:true)
        tableView.register(UITableViewCell.self, forCellReuseIdentifier: "
           cell1")
    }

    override func viewWillAppear(_ animated: Bool) {
        Model.channelsList { [weak self] (channels) in
            self?.dataSource = channels
        }
    }

    func setUpNavigationButtons(){
        navigationItem.rightBarButtonItem = UIBarButtonItem(
           barButtonSystemItem: .cancel, target: self, action: #selector(
           cancelAction))
    }

    func completedActions(with channel: Model.Channels){
        Model.updateChannel(with: channel)
        cancelAction()
    }

    @objc func cancelAction(){
        let transition = CATransition()
        transition.duration = 0.5
        transition.timingFunction = CAMediaTimingFunction(name:
           CAMediaTimingFunctionName.easeInEaseOut)
        transition.type = CATransitionType.reveal
        transition.subtype = CATransitionSubtype.fromBottom
        navigationController?.view.layer.add(transition, forKey: nil)
        navigationController?.popViewController(animated: false)
    }

    override func tableView(_ tableView: UITableView,
       viewForHeaderInSection section: Int) -> UIView? {
        let viewHeader = UIView(frame: CGRect(x: 0, y: 0, width: UIScreen.
           main.bounds.width, height: 30))
        viewHeader.backgroundColor = .white
        let label = UILabel()

        label.text = "All channels"
        label.font = UIFont.boldSystemFont(ofSize: 22)
        label.textColor = .black

        viewHeader.addSubview(label)
        label.edgesToSuperview(insets: .left(16))
        return viewHeader
    }
```

```
    override func tableView(_ tableView: UITableView,
       heightForHeaderInSection section: Int) -> CGFloat {
        return 30
    }

    override func tableView(_ tableView: UITableView,
       numberOfRowsInSection section: Int) -> Int {
        return dataSource?.count ?? 0
    }

    override func tableView(_ tableView: UITableView, didSelectRowAt
       indexPath: IndexPath) {
        var channel = dataSource![indexPath.row]
        channel.people.append(user!.id)
        completedActions(with: channel)
    }

    override func tableView(_ tableView: UITableView, cellForRowAt
       indexPath: IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier: "cell1",
           for: indexPath)

        cell.textLabel?.text = dataSource![indexPath.row].name
        cell.detailTextLabel?.text = "hello"

        return cell
    }
}
//
//  TaggsSelectionViewController.swift
//  GDproject
//
//  Created by cstore on 03/05/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit
import TaggerKit

protocol TagsReceiver: class {
    func receiveTags(tags: [String])
}

class TaggsSelectionViewController: UIViewController {

    @IBOutlet weak var addTagsTextField: TKTextField!

    @IBOutlet weak var searchContainerView: UIView!

    @IBOutlet weak var testContainer: UIView!

    // We want the whole experience, let's create two TKCollectionViews
    let productTags = TKCollectionView()
    let allTags     = TKCollectionView()

    var currentTags: [String]?
    weak var receiver: TagsReceiver?

    // MARK: - Lifecycle Methods
```

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

    navigationItem.title = "Hashtags"
    addTagsTextField.delegate = self
    // Customisation example
    //        testCollection.customFont = UIFont.boldSystemFont(ofSize
       : 14)        // Custom font
    //        testCollection.customCornerRadius = 14.0
                                // Corner radius of tags
    //        testCollection.customSpacing = 20.0
                                        // Spacing between cells
    //        testCollection.customBackgroundColor = UIColor.red
                        // Background of cells


    // These are the tags already added by the user, give an aray of
       strings to the collection
    if let tags = currentTags {
        productTags.tags = tags
    }
    // These are intended to be all the tags the user has added in the
        app, which are going to be filtered
    allTags.tags = CompletionTree.getCompletion(tree: Model.
       hashTagTree!, word: "")

    /*
     We set this collection's action to .removeTag,
     becasue these are supposed to be the tags the user has already
        added
     */
    productTags.action = .removeTag


    // Set the current controller as the delegate of both collections
    productTags.delegate = self
    allTags.delegate = self

    // "testCollection" takes the tags sent by "searchCollection"
    allTags.receiver = productTags

    // The tags in "searchCollection" are going to be added, so we set
        the action to addTag
    allTags.action = .addTag


    // Set the sender and receiver of the TextField
    addTagsTextField.sender     = allTags
    addTagsTextField.receiver     = productTags

    add(productTags, toView: testContainer)
    add(allTags, toView: searchContainerView)
}

override func viewWillDisappear(_ animated: Bool) {
    super.viewWillDisappear(animated)
    if let receiver = receiver{
        receiver.receiveTags(tags: productTags.tags)
    }
```

```
    }

    /*
     These methods come from UIViewController now conforming to
        TKCollectionViewDelegate,
     You use these to do whatever you want when a tag is added or removed
        (e.g. save to file, etc)
     */
    override func tagIsBeingAdded(name: String?) {
        // Example: save testCollection.tags to UserDefault
        print("added \(name!)")
    }

    override func tagIsBeingRemoved(name: String?) {
        print("removed \(name!)")
    }
}

// textField deletage to close keyboard on return
extension TaggsSelectionViewController: UITextFieldDelegate
{
    func textFieldShouldReturn(_ textField: UITextField) -> Bool
    {
        self.view.endEditing(true)
        return false
    }
}
//
//  AnonimousChannelController.swift
//  GDproject
//
//  Created by cstore on 20/04/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit

class TagsSuggestionController: UITableViewController
{
    weak var delegate: UpdateableWithChannel?

    var suggestions = [String]()
    var channel: Model.Channels?

    override func viewDidLoad()
    {
        super.viewDidLoad()
        if let tree = Model.hashTagTree {
            suggestions = CompletionTree.getCompletion(tree: tree, word:
               "")
            print(suggestions)
        }
        tableView.register(UITableViewCell.self, forCellReuseIdentifier: "
           cell")
        tableView.reloadData()
    }

    override func numberOfSections(in tableView: UITableView) -> Int {
        return 1
```

```
    }

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

    override func tableView(_ tableView: UITableView, cellForRowAt
       indexPath: IndexPath) -> UITableViewCell
    {
        let cell = tableView.dequeueReusableCell(withIdentifier: "cell",
           for: indexPath)
        cell.textLabel?.text = suggestions[indexPath.row]

        return cell
    }

    override func tableView(_ tableView: UITableView, didSelectRowAt
       indexPath: IndexPath)
    {
        channel = Model.Channels(people: [], name: "\(self.suggestions[
           indexPath.row])", id: 0, tags: [self.suggestions[indexPath.row
           ]])

        delegate?.updateChannel(on: channel!)
        self.dismiss(animated: true)
    }
}
//
//  PostTable.swift
//  GDproject
//
//  Created by cstore on 13/02/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit

class ChannelsController: UITableViewController {



}
//
//  ApplicationCoordinator.swift
//  RxSwift
//
//  Created by cstore on 04/03/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import Foundation
import UIKit



fileprivate enum LaunchInstructor {
    case main, auth
```

```
    static func configure(
        isAutorized: Bool = DataStorage.standard.isLoggedIn) ->
           LaunchInstructor {

        if isAutorized{
            return .main
        } else {
            return .auth
        }
    }
}

final class ApplicationCoordinator: BaseCoordinator{
    private var window: UIWindow!

    init(window: UIWindow) {
        self.window = window
    }

    private var instructor: LaunchInstructor {
        return LaunchInstructor.configure()
    }

    override func start() {
        switch instructor {
        case .auth: runAuthFlow()
        case .main: runMainFlow()
        }
    }

    private func runAuthFlow() {
        let coordinator = LogInCoordinator(window: window)
        coordinator.didEndFlow = { [weak self, weak coordinator] in
            self?.start()
            self?.removeDependency(coordinator)
        }
        addDependency(coordinator)
        coordinator.start()
    }

    private func runMainFlow() {
        let storyboard = UIStoryboard(name: "Main", bundle: nil)
        let tabBar = storyboard.instantiateViewController(withIdentifier:
           "TabbarController") as! TabbarController
        let coordinator = TabBarCoordinator(tabbarView: tabBar, window:
           window!)

        coordinator.didEndFlow = { [weak self, weak coordinator] in
            self?.start()
            self?.removeDependency(coordinator)
        }

        addDependency(coordinator)
        coordinator.start()
    }
}
//
//  BaseCoordinator.swift
//  RxSwift
```

```
//
//  Created by cstore on 02/03/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import Foundation
import UIKit

protocol Coordinator: class {
    func start()
}

class BaseCoordinator: Coordinator {

    var childCoordinators: [Coordinator] = []

    func start() {

    }

    // add only unique object
    func addDependency(_ coordinator: Coordinator) {
        guard !childCoordinators.contains(where: { $0 === coordinator })
           else {  return }
        childCoordinators.append(coordinator)
    }

    func removeDependency(_ coordinator: Coordinator?) {
        guard
            childCoordinators.isEmpty == false,
            let coordinator = coordinator
            else { return }

        // Clear child-coordinators recursively
        if let coordinator = coordinator as? BaseCoordinator, !coordinator
           .childCoordinators.isEmpty {
            coordinator.childCoordinators
                .filter({ $0 !== coordinator })
                .forEach({ coordinator.removeDependency($0) })
        }
        for (index, element) in childCoordinators.enumerated() where
           element === coordinator {
            childCoordinators.remove(at: index)
            break
        }
    }
}
//
//  ChannelsCoordinator.swift
//  RxSwift
//
//  Created by cstore on 03/03/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import Foundation
import UIKit
import Pulley
```

```
class ChannelsCoordinator: BaseCoordinator{
    let storyboard = UIStoryboard(name: "Main", bundle: nil)
    private weak var navigationController: UINavigationController?

    init(nc: UINavigationController) {
        self.navigationController = nc
        navigationController?.navigationItem.largeTitleDisplayMode = .
            always
    }

    override func start() {
        show()
    }

    private func show() {

        let channels = storyboard.instantiateViewController(withIdentifier
            : channelListControllerId) as! ChannelListController

        channels.onChannelSelected = { [unowned self]
            (channel) in
            self.navigationController?.pushViewController(self.
                presentNewsController(with: channel), animated: true)
        }

        channels.askForUpdates()

        channels.onEditingModeBegins = { [unowned self] (channel,
            indexPath) in
            let vc = self.presentChannelController(with: channel)
            self.navigationController?.pushViewController(vc, animated:
                true)
            vc.tabBarController?.tabBar.isHidden = true
        }

        let nc = presentNewsController()
        navigationController?.setViewControllers([channels, nc], animated:
            false)
    }

    /// Function that presents channel controller
    ///
    /// - Parameter channel: channel that needs to be displayed
    func presentChannelController(with channel: Model.Channels? = nil) ->
        ChannelViewController {
        let mainContentVC = storyboard.instantiateViewController(
            withIdentifier: channelViewControllerId) as!
            ChannelViewController

        // to show preview we need to instantiate newsController but with
            different functionality
        mainContentVC.onShowingPreview = { [weak mainContentVC, unowned
            self] ch in
            let vc = self.presentNewsController(with: ch, previewMode:
                true)
            mainContentVC?.navigationController?.pushViewController(vc,
                animated: true)
        }
```

```
        // to go for choosing hashtags
        mainContentVC.onChoosingHashTags = { [weak mainContentVC, unowned
            self] ch in
            let vc = self.storyboard.instantiateViewController(
                withIdentifier: taggsSelectionViewController) as!
                TaggsSelectionViewController

            vc.currentTags = ch
            vc.receiver = mainContentVC

            self.navigationController?.pushViewController(vc, animated:
                true)
        }

        mainContentVC.onChoosingPeople = { [weak mainContentVC, unowned
            self] channel in
            let vc = self.storyboard.instantiateViewController(
                withIdentifier: addToChannelVCId) as! AddToChannelVC

            vc.channel = channel
            vc.update = mainContentVC

            self.navigationController?.pushViewController(vc, animated:
                true)
        }

        mainContentVC.channel = channel
        return mainContentVC
    }

    func presentNewsController(with channel: Model.Channels? = nil,
        previewMode: Bool = false) -> NewsController
    {
        let mainContentVC = storyboard.instantiateViewController(
            withIdentifier: newsController) as! NewsController
        mainContentVC.channel = channel

        if !previewMode {
            mainContentVC.news.onFullPost = {
                [weak self] (type,post) in

                let vc = self?.storyboard.instantiateViewController(
                    withIdentifier: fullPostControllerId) as!
                    FullPostController
                vc.type = type
                vc.post = post
                self?.navigationController!.pushViewController(vc,
                    animated: true)
            }

            mainContentVC.news.onChannelDidChange = {
                print("anon with \($0.0.count) users")
            }

            mainContentVC.navigationItem.rightBarButtonItems = [
                UIBarButtonItem(barButtonSystemItem: .compose, target: self
                , action: #selector(writePost(_:)))
            ]
        }
```

```
        mainContentVC.changedChannelName = {
            [weak mainContentVC] (title) in mainContentVC?.navigationItem.
               title = title
        }

        return mainContentVC
    }

    @objc private func writePost(_ barItem: UIBarButtonItem)
    {
        let vc = storyboard.instantiateViewController(withIdentifier: "
           NewPostViewController") as! NewPostViewController

        vc.navigationItem.rightBarButtonItem = UIBarButtonItem(title: "
           Post", style: .plain, target: vc, action: #selector(vc.newPost)
           )
        vc.navigationItem.leftBarButtonItem = UIBarButtonItem(title: "
           Close", style: .plain, target: vc, action: #selector(vc.
           closeView))

        let transition = CATransition()
        transition.duration = 0.5
        transition.timingFunction = CAMediaTimingFunction(name:
           CAMediaTimingFunctionName.easeInEaseOut)
        transition.type = CATransitionType.moveIn
        transition.subtype = CATransitionSubtype.fromTop
        navigationController?.view.layer.add(transition, forKey: nil)
        navigationController?.pushViewController(vc, animated: false)

        vc.moveBackToParentVC = {
            [weak self] in

            let transition = CATransition()
            transition.duration = 0.5
            transition.timingFunction = CAMediaTimingFunction(name:
               CAMediaTimingFunctionName.easeInEaseOut)
            transition.type = CATransitionType.reveal
            transition.subtype = CATransitionSubtype.fromBottom
            self?.navigationController?.view.layer.add(transition, forKey:
                nil)
            self?.navigationController?.popViewController(animated: false)
        }
    }
}
//
//  LogInCoordinator.swift
//  RxSwift
//
//  Created by cstore on 01/03/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit

class LogInCoordinator: BaseCoordinator {

    let storyboard = UIStoryboard(name: "Main", bundle: nil)
```

```
    var didEndFlow: (()->())?
    var window: UIWindow!
    var navigationController: UINavigationController?

    init(window: UIWindow) {
        self.window = window
        self.window?.rootViewController = UINavigationController()
        self.navigationController = window.rootViewController as?
           UINavigationController
    }

    override func start(){
        let logInVC = LogInViewController()
        logInVC.authenticate = { [weak self, weak logInVC] (email) in

            DataStorage.standard.setEmail(email: email)
            Model.authenticate(with: email) { [weak self]
                (authStatus) in

                print(authStatus.userStatus)

                if (authStatus.userStatus == "invalid") {
                    logInVC?.presentAlertInvalidCode()
                }
                else if (authStatus.userStatus == "canRegister") { //
                   register form
                    self?.presentRegisterVC()
                } else { // validation code
                    self?.presentViewControllerForCodeInput()
                }
            }
        }

        navigationController?.pushViewController(logInVC, animated: false)
    }

    func presentViewControllerForCodeInput()
    {
        let vc = storyboard.instantiateViewController(withIdentifier: "
           CodeViewController") as! CodeViewController
        self.navigationController?.pushViewController(vc, animated: true)
        vc.onSuccessLogIn = {
            [weak self] in self?.didEndFlow?()
        }
    }

    func presentRegisterVC(){

        let vc1 = storyboard.instantiateViewController(withIdentifier: "
           RegisterTableViewController") as! RegisterTableViewController

        vc1.onRegistration = { [weak self] in
            self?.presentViewControllerForCodeInput()
        }

        self.navigationController?.pushViewController(vc1, animated: true)
    }
```

```
}
//
//  MessagesCoordinator.swift
//  GDproject
//
//  Created by cstore on 01/05/2019.
//  Copyright   2019 drHSE. All rights reserved.
//


import Foundation
import UIKit

class MessagesCoordinator: BaseCoordinator {

    var didEndSession: (()->())?

    let storyboard = UIStoryboard(name: "Main", bundle: nil)
    private weak var navigationController: UINavigationController?

    init(nc: UINavigationController) {
        self.navigationController = nc
    }

    override func start() {
        show()
    }

    private func show(){
        let vc = storyboard.instantiateViewController(withIdentifier:
           messagesViewControllerId) as! MessagesViewController

        // choose person to write message to
        vc.onUserDisplayList = { [weak vc, unowned self] in

            let newVC = self.storyboard.instantiateViewController(
               withIdentifier: peopleToWriteVC) as!
               PeopleToWriteViewController

            newVC.whatToDoWithSelection = { [weak newVC, weak self] people
                in
                newVC?.navigationController?.popViewController(animated:
                   true)

                // detect is it a user or group dialog
                let count = people.count
                if count == 1 {

                    let user = people.first!.key
                    let createdDialog = Model.Dialog.userChat(Model.
                       UserChat(user: user))

                    let vc = DialogViewController()
                    vc.users = Model.Channels.fullPeopleDict
                    vc.dialog = createdDialog
                    self?.navigationController?.pushViewController(vc,
                       animated: true)

                } else {
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.03-01 ТЗ 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

```
                    var group = Model.Group(users: people, name: "Untitled
                        ", id: 0)
                    group.users[DataStorage.standard.getUserId()] = Model.
                        UserPermission(isAdmin: true)

                    Model.createGroupChat(from: group, completion: { [weak
                         self] id in

                        let newVC1 = DialogViewController()
                        group.id = id
                        newVC1.users = Model.Channels.fullPeopleDict
                        newVC1.dialog = Model.Dialog.groupChat(Model.
                            GroupChat(group: group))
                        self?.navigationController?.pushViewController
                            (newVC1, animated: true)
                    })
                }
            }

            vc?.navigationController?.pushViewController(newVC, animated:
                true)
        }

        vc.onDialogDisplay = { [weak vc]  in

            let newVC = DialogViewController()
            newVC.dialog = $0.dialog
            newVC.users = $0.users
            newVC.onUserDisplay = { [weak self] id in

                let vc = self?.storyboard.instantiateViewController(
                    withIdentifier: profileViewController) as!
                    ProfileViewController
                vc.idProfile = id
                self?.navigationController!.pushViewController(vc,
                    animated: true)

            }

            vc?.navigationController?.pushViewController(newVC, animated:
                true)
        }

        navigationController?.viewControllers = [vc]
    }
}
//
//  ProfileCoordinator.swift
//  RxSwift
//
//  Created by cstore on 02/03/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import Foundation
import UIKit

class ProfileCoordinator: BaseCoordinator {
```

```
    var didEndSession: (()->())?

    private weak var navigationController: UINavigationController?

    init(nc: UINavigationController) {
        self.navigationController = nc
    }

    override func start() {
        show()
    }

    private func show() {
        let storyboard = UIStoryboard(name: "Main", bundle: nil)
        let profile = storyboard.instantiateViewController(withIdentifier:
            "ProfileViewController") as! ProfileViewController

        profile.logOut  = {
            Model.logout() {
                [weak self] in

                DataStorage.standard.setIsLoggedIn(value: false, with: 0)
                self?.didEndSession?()
            }
        }

        profile.deleteAllSessions = {
            Model.deactivateAll() {
                [weak self] in

                DataStorage.standard.setIsLoggedIn(value: false, with: 0)
                self?.didEndSession?()
            }
        }

        profile.onSettings = { [weak self, weak storyboard, weak profile]
           in

            let vc = storyboard?.instantiateViewController(withIdentifier:
                "RegisterTableViewController") as!
               RegisterTableViewController
            vc.delegate = profile
            vc.userActive = profile?.user

            self?.navigationController?.pushViewController(vc, animated:
               true)
        }


        navigationController?.setViewControllers([profile], animated:
           false)
    }
}
//
//  TabbarController.swift
//  RxSwift
//
//  Created by cstore on 03/03/2019.
//  Copyright   2019 drHSE. All rights reserved.
```

```
//

import Foundation
import UIKit

protocol TabbarView: class {
    var onChannelsFlowSelect: ((UINavigationController) -> ())? { get set
       }
    var onProfileFlowSelect: ((UINavigationController) -> ())? { get set }
    var onMessagesFlowSelect: ((UINavigationController) -> ())? { get set
       }
    var onViewDidLoad: ((UINavigationController) -> ())? { get set }
}

final class TabbarController: UITabBarController,
   UITabBarControllerDelegate, TabbarView {


    var onChannelsFlowSelect: ((UINavigationController) -> ())?
    var onProfileFlowSelect: ((UINavigationController) -> ())?
    var onMessagesFlowSelect: ((UINavigationController) -> ())?

    var onViewDidLoad: ((UINavigationController) -> ())?

    override func viewDidLoad() {
        super.viewDidLoad()

        delegate = self
        if let controller = customizableViewControllers?.first as?
           UINavigationController {
            onViewDidLoad?(controller)
        }
    }

    func tabBarController(_ tabBarController: UITabBarController,
       didSelect viewController: UIViewController)
    {

        guard let controller = viewControllers?[selectedIndex] as?
           UINavigationController else { return }

        if selectedIndex == 0 {
            onChannelsFlowSelect?(controller)
        } else if selectedIndex == 2 {
            onProfileFlowSelect?(controller)
        } else {
            onMessagesFlowSelect?(controller)
        }
    }
}
//
//  TabBarCoordinator.swift
//  RxSwift
//
//  Created by cstore on 02/03/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import Foundation
```

```
import UIKit


class TabBarCoordinator: BaseCoordinator {

    var didEndFlow: (()->())?

    private let tabbarView: TabbarView
    private weak var window: UIWindow?

    init(tabbarView: TabbarView, window: UIWindow) {
        self.tabbarView = tabbarView
        self.window = window
    }

    override func start() {
        tabbarView.onViewDidLoad = runChannelsFlow()
        tabbarView.onChannelsFlowSelect = runChannelsFlow()
        tabbarView.onProfileFlowSelect = runProfileFlow()
        tabbarView.onMessagesFlowSelect = runMessagesFlow()
        window?.rootViewController = tabbarView as! TabbarController
    }

    private func runChannelsFlow() -> ((UINavigationController) -> ())
    {
        return { [unowned self] navController in
            if navController.viewControllers.isEmpty == true {
                let channelCoordinator = ChannelsCoordinator(nc:
                    navController)
                self.addDependency(channelCoordinator)
                channelCoordinator.start()
            }
        }
    }

    private func runMessagesFlow() -> ((UINavigationController) -> ()){
        return { [unowned self] navController in
            if navController.viewControllers.isEmpty == true {
                let messagesCoordinator = MessagesCoordinator(nc:
                    navController)
                self.addDependency(messagesCoordinator)
                messagesCoordinator.start()
            }
        }
    }

    private func runProfileFlow() -> ((UINavigationController) -> ())
    {
        return { [unowned self] navController in
            if navController.viewControllers.isEmpty == true {
                let profileCoordinator = ProfileCoordinator(nc:
                    navController)
                profileCoordinator.didEndSession = { [weak self, weak
                    profileCoordinator] in
                    self?.removeDependency(profileCoordinator)
                    self?.didEndFlow?()
                }
                self.addDependency(profileCoordinator)
                profileCoordinator.start()
```

```
            }
        }
    }
}
//
//  FullPostViewController.swift
//  GDproject
//
//  Created by cstore on 13/02/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit
import TinyConstraints

class FullPostController: UITableViewController {

    var post: Post?

    override func viewDidLoad() {
        super.viewDidLoad()

        tableView.register(PostViewCell.self, forCellReuseIdentifier:
            postCellId)

        setUpNavigationBar()
        tableView.separatorStyle = .none
    }

    func setUpNavigationBar(){
        navigationController?.navigationBar.prefersLargeTitles = false
        navigationItem.title = "\(post?.fromUser.login ?? "")"
        navigationItem.rightBarButtonItems = [UIBarButtonItem(
            barButtonSystemItem: .action, target: self, action: #selector(
            self.options))]
    }

    @objc func options(){
        // drafts
        // saved

        let optionMenu = UIAlertController(title: nil, message: nil,
            preferredStyle: .actionSheet)
        let editAction = UIAlertAction(title: "Send via message", style: .
            default)
        let shareAction = UIAlertAction(title: "Send via project", style:
            .default)
        let settingsAction = UIAlertAction(title: "Copy link", style: .
            default)
        let cancelAction = UIAlertAction(title: "Cancel", style: .cancel)

        optionMenu.addAction(editAction)
        optionMenu.addAction(shareAction)
        optionMenu.addAction(settingsAction)
        optionMenu.addAction(cancelAction)

        self.present(optionMenu, animated: true, completion: nil)
    }
    // MARK: - Table view data source
```

```
override func numberOfSections(in tableView: UITableView) -> Int {
    // #warning Incomplete implementation, return the number of
       sections
    return 2
}


override func tableView(_ tableView: UITableView,
   viewForHeaderInSection section: Int) -> UIView?
{
    let mainView = UIView(frame: CGRect(x: 0, y: 0, width: UIScreen.
       main.bounds.width, height: 50))

    let scrollView = UIScrollView()

    scrollView.backgroundColor = .white

    mainView.addSubview(scrollView)
    scrollView.edgesToSuperview()

    scrollView.showsHorizontalScrollIndicator = false

    var buttons: [UIButton] = []
    for hash in post!.hashtags {
        let button = UIButton()
        button.setTitle("#" + hash, for: .normal)
        button.backgroundColor = #colorLiteral(red: 0.8039215803,
           green: 0.8039215803, blue: 0.8039215803, alpha: 1)
        button.layer.cornerRadius = 10
        button.titleLabel?.font = UIFont.systemFont(ofSize: 14)
        button.setTitleColor(.black, for: .normal)
        buttons.append(button)
    }

    let stackView = UIStackView(arrangedSubviews: buttons)

    scrollView.addSubview(stackView)

    stackView.axis = .horizontal
    stackView.alignment = .center
    stackView.spacing = 20

    stackView.edgesToSuperview(insets: .left(20) + .top(10) + .right
       (20))

    scrollView.contentSize = CGSize(width: 500, height: 50)

    switch section {
    case 0:
        return mainView
    default:
        return nil
    }
}


override func tableView(_ tableView: UITableView,
   heightForHeaderInSection section: Int) -> CGFloat {
```

```
        switch section {
        case 0:
            return 50
        default:
            return 0
        }
    }

    override func tableView(_ tableView: UITableView,
       numberOfRowsInSection section: Int) -> Int
    {
        switch section {
        case 0:
            return 1
        case 1:
            return post?.comments.count ?? 0
        default:
            return 0
        }
    }

    override func tableView(_ tableView: UITableView, cellForRowAt
       indexPath: IndexPath) -> UITableViewCell
    {
        // TODO: make different cells for sections with switch

        let cell = tableView.dequeueReusableCell(withIdentifier:
           postCellId) as! PostViewCell

        cell.fill(with: post!.dataArray, true)
        cell.selectionStyle = .none
        return cell
    }

}
//
//  HeaderNewsChannels.swift
//  GDproject
//
//  Created by cstore on 13/02/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit
import TinyConstraints

class HeaderNewsChannels: UITableViewCell {

    weak var tableView: UITableView?
    weak var vc: NewsController?

    let basicInfo = ChannelsController()

    override init(style: UITableViewCell.CellStyle, reuseIdentifier:
       String?) {
        super.init(style: style, reuseIdentifier: reuseIdentifier)
        backgroundColor = .white
        setUpCell()
    }
```

```
    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
    }

    func setUpCell(){
        addSubview(postsChannelsSegment)
        postsChannelsSegment.selectedSegmentIndex = 0
        postsChannelsSegment.addTarget(self, action: #selector(
           changedValue), for: .valueChanged)
        postsChannelsSegment.edgesToSuperview(insets: .left(16) + .right
           (16) + .top(8) + .bottom(8))
    }

    // Initialize
    let postsChannelsSegment = UISegmentedControl(items: ["Posts", "
       Channels"])

    @objc func changedValue(_ sender: UISegmentedControl) {
        let index = sender.selectedSegmentIndex
        switch index {
        case 0:
            print("posts")
            vc?.setUpNavigationItemsforPosts()
        case 1:
            print("channels")
            vc?.setUpNavigationItemsForChannels()
//            self.tableView?.delegate = basicInfo
//            self.tableView?.dataSource = basicInfo
//            tableView?.reloadData()
        default:
            break
        }
    }
}

//
//  CodeViewController.swift
//  GDproject
//
//  Created by cstore on 10/05/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit
import TinyConstraints

class CodeViewController: UIViewController {

    @IBOutlet weak var f1: UITextField!
    @IBOutlet weak var f2: UITextField!
    @IBOutlet weak var f3: UITextField!
    @IBOutlet weak var f4: UITextField!
    @IBOutlet weak var f5: UITextField!
    @IBOutlet weak var f6: UITextField!

    var loading = UIActivityIndicatorView(style: .gray)

    var onSuccessLogIn: (()->())?
```

```
override func viewDidLoad() {
    super.viewDidLoad()
    setUpConstraint()

    navigationItem.title = DataStorage.standard.getEmail()

    f1.addTarget(self, action: #selector(textFiledDidChange(textField
        :)), for: .editingChanged)
    f2.addTarget(self, action: #selector(textFiledDidChange(textField
        :)), for: .editingChanged)
    f3.addTarget(self, action: #selector(textFiledDidChange(textField
        :)), for: .editingChanged)
    f4.addTarget(self, action: #selector(textFiledDidChange(textField
        :)), for: .editingChanged)
    f5.addTarget(self, action: #selector(textFiledDidChange(textField
        :)), for: .editingChanged)
    f6.addTarget(self, action: #selector(textFiledDidChange(textField
        :)), for: .editingChanged)
}

override func viewWillDisappear(_ animated: Bool) {
    super.viewWillDisappear(animated)

    loading.stopAnimating()
}

override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)

    f1.becomeFirstResponder()
}

@objc func textFiledDidChange(textField: UITextField) {
    let text = textField.text

    // only 1 char
    if text?.utf16.count == 1 {
        switch textField {
        case f1:
            f2.becomeFirstResponder()
        case f2:
            f3.becomeFirstResponder()
        case f3:
            f4.becomeFirstResponder()
        case f4:
            f5.becomeFirstResponder()
        case f5:
            f6.becomeFirstResponder()
        case f6:
            f6.resignFirstResponder()
        default:
            break
        }
    }
}


func presentAlertInvalidData(with text: String)
```

```
{
    let alert = UIAlertController(title: "Invalid code", message: text
        , preferredStyle: .alert)
    alert.addAction(UIAlertAction(title: "OK", style: .default,
        handler: nil))
    self.present(alert, animated: true, completion: nil)

    loading.stopAnimating()
}

@IBOutlet weak var bottomConstraint: NSLayoutConstraint!

@IBAction func whereToGoNext(_ sender: UIButton)
{
    guard let f1t = f1.text, let f2t = f2.text, let f3t = f3.text, let
        f4t = f4.text, let f5t = f5.text, let f6t = f6.text else {
        presentAlertInvalidData(with: "Some fields are missing");
        return
    }

    if f1t.isEmpty || f2t.isEmpty || f3t.isEmpty || f4t.isEmpty || f5t
        .isEmpty || f6t.isEmpty {
        presentAlertInvalidData(with: "Some fields are missing");
        return
    }

    let code = "\(f1.text!)\(f2.text!)\(f3.text!)\(f4.text!)\(f5.text
        !)\(f6.text!)"
    if let codeToInt = Int(code)
    {
        loading.startAnimating()
        Model.verify(with: codeToInt) { [weak self] in
            // if everything is okay we can authemticicate
            if !$0 {
                self?.presentAlertInvalidData(with: "Wrong code. Try
                    again!")
                self?.loading.stopAnimating()
                return
            } else {
                // Model.authemticiateMe
                Model.authenticateMe { [weak self] (res) in
                    if res {
                        self?.onSuccessLogIn?()
                    }
                }
                return
            }
        }
    }
}

func setUpConstraint()
{
    self.view.addSubview(loading)
    loading.centerInSuperview()
    // for keyboard notifications
    NotificationCenter.default.addObserver(self, selector: #selector(
        handleKeyboardNotifications), name: UIResponder.
        keyboardWillShowNotification, object: nil)
```

```
        NotificationCenter.default.addObserver(self, selector: #selector(
            handleKeyboardNotifications), name: UIResponder.
            keyboardWillHideNotification, object: nil)
    }

    @objc func handleKeyboardNotifications(notification: NSNotification){
        if let userInfo = notification.userInfo {
            // UIKeyboardFrameEndUserInfoKey
            let keyBoardFrame = (userInfo[UIResponder.
                keyboardFrameEndUserInfoKey] as! NSValue).cgRectValue
            bottomConstraint.constant = notification.name == UIResponder.
                keyboardWillShowNotification ? keyBoardFrame.height - self.
                view.safeAreaInsets.bottom : 0

            UIView.animate(withDuration: 0, delay: 0, options: .
                curveEaseOut, animations: {
                 self.view.layoutIfNeeded()
            })
        }
    }

}
//
//  FacultyTableViewController.swift
//  GDproject
//
//  Created by cstore on 10/05/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit

protocol ChosenFactulty: class {
    func onChooseFaculty(faculty: Model.Faculty)
}

class FacultyTableViewController: UITableViewController {

    weak var delegate: ChosenFactulty?

    var currentFaculties: [Model.Faculty] = []{
        didSet {
            tableView.reloadData()
        }
    }

    var isFiltering: Bool {
        return searchController.isActive && !searchBarIsEmpty()
    }

    func searchBarIsEmpty() -> Bool {
        return searchController.searchBar.text?.isEmpty ?? true
    }

    var searchController = UISearchController(searchResultsController: nil
        )

    override func viewDidLoad() {
```

```
        super.viewDidLoad()

        self.navigationItem.title = "Faculties"
        self.navigationItem.searchController = searchController
        self.navigationItem.hidesSearchBarWhenScrolling = false
        searchController.obscuresBackgroundDuringPresentation = false

        setUpTimer()
    }

    func setUpTimer(){
        Timer.scheduledTimer(withTimeInterval: 2, repeats: true) { (timer)
            in
            DispatchQueue.main.async { [weak self] in

                if self!.isFiltering {
                    Model.searchFaculty(string: (self?.searchController.
                       searchBar.text)!) { [weak self] in
                        self?.currentFaculties = $0
                    }
                }
            }
        }
    }


    // MARK: - Table view data source

    override func numberOfSections(in tableView: UITableView) -> Int {
        return 1
    }

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


    override func tableView(_ tableView: UITableView, cellForRowAt
       indexPath: IndexPath) -> UITableViewCell
    {
        let cell = tableView.dequeueReusableCell(withIdentifier: "F", for:
            indexPath)

        cell.textLabel?.text = currentFaculties[indexPath.row].name
        cell.detailTextLabel?.text = currentFaculties[indexPath.row].
           campusName
        cell.selectionStyle = .none

        return cell
    }

    override func tableView(_ tableView: UITableView, didSelectRowAt
       indexPath: IndexPath) {
        delegate?.onChooseFaculty(faculty: currentFaculties[indexPath.row
           ])
        searchController.dismiss(animated: true, completion: nil)
        self.navigationController?.popViewController(animated: true)
    }
```

```
}
//
//  InititalsViewCell.swift
//  GDproject
//
//  Created by cstore on 10/05/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit

class InititalsViewCell: UITableViewCell {

    @IBOutlet weak var initialsTextField: UITextField!

    override func awakeFromNib() {
        super.awakeFromNib()
        // Initialization code
    }

    func fill(with name: String){
        initialsTextField.text = name
    }
}
//
//  LogInViewController.swift
//  RxSwift
//
//  Created by cstore on 01/03/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit
import TinyConstraints
import ReactiveSwift
import ReactiveCocoa
import Result

class LogInViewController: UIViewController {

    var loading = UIActivityIndicatorView(style: .gray)

    var authenticate: ((String)->())?

    func presentAlertInvalidCode()
    {
        let alert = UIAlertController(title: "Invalid email", message: "
           Try again. Use @hse.ru mail.", preferredStyle: .alert)
        alert.addAction(UIAlertAction(title: "OK", style: .default,
           handler: nil))
        self.present(alert, animated: true, completion: nil)
        mailTextField.text = ""
        loading.stopAnimating()
    }

    override func viewWillDisappear(_ animated: Bool) {
        super.viewWillDisappear(animated)
```

```
        loading.stopAnimating()
    }

    let logInLabel: UILabel = {
        let label = UILabel()
        label.text = "Log In"
        label.textColor = .black
        label.font = UIFont.boldSystemFont(ofSize: 34)
        return label
    }()

    let mailTextField: UITextField = {
        let textField = UITextField()
        textField.backgroundColor = #colorLiteral(red: 0.937254902, green:
            0.937254902, blue: 0.9568627451, alpha: 1)
        textField.placeholder = "Mail"
        textField.borderStyle = .roundedRect
        textField.textColor = .black
        textField.autocorrectionType = UITextAutocorrectionType.no
        textField.autocapitalizationType = UITextAutocapitalizationType.
           none
        textField.clearButtonMode = .always
        return textField
    }()

    let logInButton: UIButton = {
        let button = UIButton(type: .system)
        button.setTitle("Log In", for: .normal)
        button.setTitleColor(blueSystemColor, for: .normal)
        button.titleLabel?.font  = UIFont.boldSystemFont(ofSize: 16)
        button.addTarget(self, action: #selector(handleTap), for: .
           touchUpInside)
        button.isEnabled = false
        return button
    }()

    @objc func handleTap(){
        loading.startAnimating()
        authenticate?(mailTextField.text!)
    }

    private lazy var keyboardBar = UIView()
    private lazy var contentView = UIView()
    private var bottomConstraint: NSLayoutConstraint?

    func setUpView(){
        // logIn stack with textField and label
        view.addSubview(contentView)
        view.addSubview(loading)

        loading.centerInSuperview()

        contentView.edgesToSuperview(excluding: .bottom, insets: .left(16)
            + .right(16) + .top(80), usingSafeArea: true)
        let views = [logInLabel, mailTextField]
        contentView.stack(views, axis: .vertical, spacing: 10)
    }

    func configureKeyboard(){
```

```
        mailTextField.keyboardType = .emailAddress
        // configure keyboardBar setUp
        view.addSubview(keyboardBar)
        keyboardBar.height(50)
        keyboardBar.edgesToSuperview(excluding: [.top,.bottom])
        bottomConstraint = NSLayoutConstraint(item: keyboardBar, attribute
           : .bottom, relatedBy: .equal, toItem: view.safeAreaLayoutGuide,
            attribute: .bottom, multiplier: 1, constant: 0)
        view.addConstraint(bottomConstraint!)

        // configure keyboardBar components
        keyboardBar.addSubview(logInButton)
        logInButton.height(50)
        logInButton.rightToSuperview(view.rightAnchor, offset: -16,
           relation: .equal, isActive: true)

    }

    func logicOfLogInInputValidation() -> ((String?)->()) {

        let logic: ((String?)->()) = { [weak self] (someText) in
            if let text = someText, !text.isEmpty, text.contains("@") {
                self?.logInButton.isEnabled = true
            } else {
                self?.logInButton.isEnabled = false
            }
        }

        return logic
    }

    override func viewDidLoad() {
        super.viewDidLoad()
        self.view.backgroundColor = .white
        setUpView()
        configureKeyboard()

        let mailFieldValuesSignal: Signal<String, NoError> = mailTextField
           .reactive.continuousTextValues

        mailFieldValuesSignal.observeValues(logicOfLogInInputValidation())

        // for keyboard notifications
        NotificationCenter.default.addObserver(self, selector: #selector(
           handleKeyboardNotifications), name: UIResponder.
           keyboardWillShowNotification, object: nil)

        NotificationCenter.default.addObserver(self, selector: #selector(
           handleKeyboardNotifications), name: UIResponder.
           keyboardWillHideNotification, object: nil)
    }


    @objc func handleKeyboardNotifications(notification: NSNotification){
        if let userInfo = notification.userInfo{
            // UIKeyboardFrameEndUserInfoKey
            let keyBoardFrame = (userInfo[UIResponder.
               keyboardFrameEndUserInfoKey] as! NSValue).cgRectValue
```

```
            bottomConstraint?.constant = notification.name == UIResponder.
               keyboardWillShowNotification ? -keyBoardFrame.height+view.
               safeAreaInsets.bottom : 0

            UIView.animate(withDuration: 0, delay: 0, options: .
               curveEaseOut, animations: {
                self.view.layoutIfNeeded()
            })
        }
    }

}

extension UIButton {
    override open var isEnabled: Bool {
        didSet {
            let color = isEnabled ? self.titleLabel?.textColor.
               withAlphaComponent(1) : self.titleLabel?.textColor.
               withAlphaComponent(0.5)
            self.setTitleColor(color, for: .normal)
        }
    }
}
//
//  RegisterTableViewController.swift
//  GDproject
//
//  Created by cstore on 10/05/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit

class RegisterTableViewController: UITableViewController, ChosenFactulty
{

    func presentAlertInvalidData(message: String)
    {
        let alert = UIAlertController(title: "Invalid data", message:
           message, preferredStyle: .alert)
        alert.addAction(UIAlertAction(title: "OK", style: .default,
           handler: nil))
        self.present(alert, animated: true, completion: nil)
    }

    func onChooseFaculty(faculty: Model.Faculty) {
        self.faculty = faculty
        tableView.reloadData()
    }

    weak var delegate: UpdateUser?

    var onRegistration: (()->())?

    var faculty: Model.Faculty?
    var user: Model.NewRegistration = Model.NewRegistration(email:
       DataStorage.standard.getEmail()!, firstName: nil, middleName: nil,
       lastName: nil, faculty: nil)
```

```
var userActive: Model.Users? {
    didSet {
        if let userA = userActive {
            user.firstName = userA.firstName
            user.middleName = userA.middleName
            user.lastName = userA.lastName

            faculty = userA.faculty
            self.updateUserIfCan = true
            tableView.reloadData()
        }
    }
}

var updateUserIfCan: Bool = false

override func viewDidLoad()
{
    super.viewDidLoad()

    tableView.keyboardDismissMode = .interactive
    navigationItem.title = user.email
    navigationItem.rightBarButtonItem = UIBarButtonItem(
       barButtonSystemItem: .done, target: self, action: #selector(
       endRegistration))
}


@objc func endRegistration()
{
    let fieldsMissing = "Some fields are missing. Add more information
       "
    updateUser()
    guard let faculty = faculty else { presentAlertInvalidData(message
       : fieldsMissing); return }

    user.faculty = faculty.url

    guard let name = user.firstName else { presentAlertInvalidData(
       message: fieldsMissing); return  }
    guard let middle = user.middleName else { presentAlertInvalidData(
       message: fieldsMissing); return  }
    guard let last = user.lastName else { presentAlertInvalidData(
       message: fieldsMissing); return  }

    if updateUserIfCan {
        guard let delegate = delegate else { return }
        guard var userA = userActive else { return }

        // updating current
        userA.firstName = name
        userA.faculty = faculty
        userA.middleName = middle
        userA.lastName = last

        Model.userUpdate(with: user) { [weak self] in

            if $0 {
                delegate.updateUserObj(with: userA)
```

```
                self?.navigationController?.popViewController(animated
                    : true)
            } else {
                self?.presentAlertInvalidData(message: "Something went
                    wrong! Try again")
            }

        }
    } else {

        Model.register(object: user)
        { [weak self] in
            self?.onRegistration?()
        }
    }
}
// MARK: - Table view data source

override func numberOfSections(in tableView: UITableView) -> Int {
    return 2
}

override func tableView(_ tableView: UITableView,
   numberOfRowsInSection section: Int) -> Int {

    switch section {
    case 0:
        return 3
    default:
        if let _ = faculty {
            return 2
        } else {
            return 1
        }
    }
}

override func tableView(_ tableView: UITableView,
   titleForHeaderInSection section: Int) -> String?
{
    switch section {
    case 0:
        return "Initials"
    default:
        return "Faculty"
    }
}


override func tableView(_ tableView: UITableView, cellForRowAt
   indexPath: IndexPath) -> UITableViewCell
{
    switch indexPath.section {
    case 0:
        let cell = tableView.dequeueReusableCell(withIdentifier: "
           nameCell", for: indexPath) as! InititalsViewCell
        switch indexPath.row{
        case 0:
            if let name = user.firstName {
```

```
                cell.fill(with: name)
            } else {
                cell.initialsTextField.placeholder = "First name"
            }
        case 1:
            if let name = user.middleName {
                cell.fill(with: name)
            } else {
                cell.initialsTextField.placeholder = "Middle name"
            }
        default:
            if let name = user.lastName {
                cell.fill(with: name)
            } else {
                cell.initialsTextField.placeholder = "Last name"
            }
        }
        cell.selectionStyle = .none
        return cell
    default:
        switch indexPath.row {
        case 0:
            let cell = tableView.dequeueReusableCell(withIdentifier: "
               cell111", for: indexPath)
            cell.selectionStyle = .none
            return cell
        default:
            let cell = tableView.dequeueReusableCell(withIdentifier: "
               facultyCell", for: indexPath)
            cell.textLabel?.text = faculty!.name
            cell.detailTextLabel?.text = faculty!.campusName
            cell.selectionStyle = .none
            return cell
        }
    }
}


override func tableView(_ tableView: UITableView, didSelectRowAt
   indexPath: IndexPath) {
    if indexPath.section == 1 && indexPath.row == 0 {

        updateUser()

        let vc = storyboard?.instantiateViewController(withIdentifier:
            "FacultyTableViewController") as!
           FacultyTableViewController

        vc.delegate = self

        self.navigationController?.pushViewController(vc, animated:
           true)
    }
}

func updateUser(){
    guard let cell = tableView.cellForRow(at: IndexPath(row: 0,
       section: 0)) as? InititalsViewCell else {return}
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.03-01 ТЗ 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

```
        guard let cell1 = tableView.cellForRow(at: IndexPath(row: 1,
           section: 0)) as? InititalsViewCell else {return}
        guard let cell2 = tableView.cellForRow(at: IndexPath(row: 2,
           section: 0)) as? InititalsViewCell else {return}

        user.firstName = cell.initialsTextField.text!
        user.middleName = cell1.initialsTextField.text!
        user.lastName = cell2.initialsTextField.text!
    }

}
//
//  ChatInfoViewController.swift
//  GDproject
//
//  Created by cstore on 02/05/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit

/// Class for displaying chat info
///
class ChatInfoViewController: UITableViewController {

    enum PersonStatus{
        case admin
        case ordinary
    }

    weak var delegate: UpdatableGroup?

    var groupChat: Model.Group? {
        didSet{
            if let groupChat = groupChat {
                usersArray = groupChat.users.map { $0.key }
            }
        }
    }

    var usersArray: [Int] = [] {
        didSet {
            tableView.reloadData()
        }
    }

    var users: [Int: Model.Users] = [:]

    func canIEditThisChat() -> PersonStatus
    {
        let myId = UserDefaults.standard.integer(forKey: UserDefaultsKeys.
           id.rawValue)

        if let groupChatUserPermission = groupChat?.users[myId]?.isAdmin {
           return groupChatUserPermission ?  .admin :  .ordinary
        }

        return .ordinary
    }
```

```
var myPermissions: PersonStatus = .ordinary



override func viewDidLoad() {
    super.viewDidLoad()

    tableView.register(UITableViewCell.self, forCellReuseIdentifier: "
       cell")
    myPermissions = canIEditThisChat()
    tableView.reloadData()

    switch myPermissions {
    case .admin:
        navigationItem.rightBarButtonItem = self.editButtonItem
    default:
        return
    }
}

// MARK: - Table view data source

override func numberOfSections(in tableView: UITableView) -> Int {
    return 3
}

override func tableView(_ tableView: UITableView,
   numberOfRowsInSection section: Int) -> Int {
    switch section {
    case 0:
        return 1
    case 1:
        return 1
    default:
        return usersArray.count + (myPermissions == .ordinary ? 0 : 1)
    }
}


override func tableView(_ tableView: UITableView, cellForRowAt
   indexPath: IndexPath) -> UITableViewCell
{
    let cell = tableView.dequeueReusableCell(withIdentifier: "cell",
       for: indexPath)

    if indexPath.row == 0 {
        switch indexPath.section {
        case 0:
            cell.textLabel?.text = groupChat!.name
            cell.selectionStyle = .none
            return cell
        case 1:
            cell.textLabel?.text = "Leave chat"
            cell.textLabel?.textColor = #colorLiteral(red: 1, green:
               0.1491314173, blue: 0, alpha: 1)
            cell.selectionStyle = .none
            return cell
        default:
```

```
                switch myPermissions{
                case .admin:
                    cell.textLabel?.text = "Add participants"
                    cell.accessoryType = .disclosureIndicator
                case .ordinary:
                    cell.textLabel?.text = name(for: users[usersArray[
                        indexPath.row]])
                }
                cell.selectionStyle = .none
                return cell
            }
        }

        switch myPermissions{
        case .admin:
            cell.textLabel?.text = name(for: users[usersArray[indexPath.
                row-1]])
        case .ordinary:
            cell.textLabel?.text = name(for: users[usersArray[indexPath.
                row]])
        }
        cell.selectionStyle = .none
        return cell
    }

    private func name(for user: Model.Users?) -> String {
        if let user = user, let perm = groupChat?.users[user.id]?.isAdmin
            {
            if perm {
                return "\(user.fullName())"
            }
            return " \(user.fullName())"
        }

        return "left"
    }

    override func tableView(_ tableView: UITableView,
        titleForHeaderInSection section: Int) -> String?
    {
        switch section {
        case 0:
            return "Title"
        case 1:
            return "Opportunities"
        default:
            return "Participants"
        }
    }


    func editName(for cell: UITableViewCell){
        let alert = UIAlertController(title: "Are you sure?", message: "
            Title:", preferredStyle: .alert)

        let action1 = UIAlertAction(title: "Cancel", style: .cancel,
            handler: nil)

        alert.addTextField { [weak self] (tf) in
```

```
            tf.textColor = #colorLiteral(red: 0, green: 0, blue: 0, alpha:
                1)
            tf.text = self?.groupChat?.name
        }

        let action2 = UIAlertAction(title: "OK", style: .default) { [
            unowned self] _ in
            let name = alert.textFields?.first?.text
            cell.textLabel?.text = name
            self.groupChat?.name = name!
            Model.updateGroupChat(with: self.groupChat!)
            self.delegate?.updateGroup(with: self.groupChat!)
        }

        alert.addAction(action1)
        alert.addAction(action2)

        present(alert, animated: true, completion: nil)
    }
    override func tableView(_ tableView: UITableView, didSelectRowAt
        indexPath: IndexPath)
    {
        guard let groupChat = groupChat else {
            return
        }

        guard let cell = tableView.cellForRow(at: indexPath) else {
            return
        }

        if indexPath.row == 0 {
            switch indexPath.section{
            case 0:
                switch myPermissions{
                case .admin:
                    editName(for: cell)
                default:
                    break
                }
            case 1:
                Model.leaveGroupChat(id: groupChat.id) {
                    [weak self] in
                    self?.navigationController?.popViewController(animated
                        : true)
                }
            default:
                switch myPermissions{
                case .admin:
                    showUserChoiceVC()
                default:
                    showParticipantPage(with: usersArray[indexPath.row])
                }
            }
        }


        if indexPath.section == 2 && indexPath.row != 0  {
            switch myPermissions{
            case .admin:
```

```
                showParticipantPage(with: usersArray[indexPath.row-1])
            default:
                showParticipantPage(with: usersArray[indexPath.row])
            }
        }
    }

    var onUserDiplay: ((Int)->())?

    func showParticipantPage(with user: Int?) {
        if let user = user {
            onUserDiplay?(user)
        }
    }

    func showUserChoiceVC()
    {
        let vc = storyboard?.instantiateViewController(withIdentifier:
           peopleToWriteVC) as! PeopleToWriteViewController

        vc.whatToDoWithSelection = { [weak self, weak vc] mapa in

            mapa.forEach { self?.users[$0.key] = Model.Channels.
               fullPeopleDict[$0.key] }
            mapa.forEach { self?.groupChat?.users[$0.key] = $0.value }
            vc?.navigationController?.popViewController(animated: true)
        }

        navigationController?.pushViewController(vc, animated: true)
    }

    override func tableView(_ tableView: UITableView, commit editingStyle:
        UITableViewCell.EditingStyle, forRowAt indexPath: IndexPath)
    {
        if editingStyle == UITableViewCell.EditingStyle.delete {
            groupChat?.users[usersArray[indexPath.row-1]] = nil
        }

        tableView.reloadData()
    }

    override func tableView(_ tableView: UITableView, canEditRowAt
       indexPath: IndexPath) -> Bool {
        if indexPath.row == 0 {
            return false
        }

        return true
    }

    override func tableView(_ tableView: UITableView, editActionsForRowAt
       indexPath: IndexPath) -> [UITableViewRowAction]?
    {
        let editButton = UITableViewRowAction(style: .normal, title: "
           Promote") { [unowned self] (rowAction, indexPath) in

            self.tableView.beginUpdates()
            self.groupChat?.users[self.usersArray[indexPath.row-1]]?.
               isAdmin = true
```

```
            self.tableView.reloadRows(at: [indexPath], with: .none)
            self.tableView.endUpdates()
        }

        editButton.backgroundColor = #colorLiteral(red: 0.476841867, green
            : 0.5048075914, blue: 1, alpha: 1)

        let restrictButton = UITableViewRowAction(style: .normal, title: "
            Restrict") { [unowned self] (rowAction, indexPath) in

            self.tableView.beginUpdates()
            self.groupChat?.users[self.usersArray[indexPath.row-1]]?.
                isAdmin = false
            self.tableView.reloadRows(at: [indexPath], with: .none)
            self.tableView.endUpdates()
        }

        restrictButton.backgroundColor = #colorLiteral(red: 0.8039215803,
            green: 0.8039215803, blue: 0.8039215803, alpha: 1)

        let deleteButton = UITableViewRowAction(style: .normal, title: "
            Delete") { [unowned self] (action, indexPath) in

            self.tableView.beginUpdates()
            self.groupChat?.users[self.usersArray[indexPath.row-1]] = nil
            self.tableView.deleteRows(at: [indexPath], with: .none)
            self.tableView.endUpdates()
        }

        deleteButton.backgroundColor = #colorLiteral(red: 1, green:
            0.1491314173, blue: 0, alpha: 1)

        return [editButton, restrictButton, deleteButton]
    }

    /// update chatInfo
    override func viewWillDisappear(_ animated: Bool) {

        defer {
            super.viewWillDisappear(animated)
        }

        guard let _ = self.navigationController?.viewControllers.lastIndex
            (of: self) else {
            switch myPermissions {
            case .ordinary:
                return
            case .admin:
                Model.updateGroupChat(with: groupChat!)
                delegate?.updateGroup(with: groupChat!)
            }
            return
        }
    }
}
//
//  DialogCell.swift
//  GDproject
//
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.03-01 ТЗ 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

```
//  Created by cstore on 05/05/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit
import TinyConstraints
import MarkdownKit

class DialogCell: UITableViewCell {

    let nameLabel: UIButton = {
        let button = UIButton()
        button.setTitleColor(.black, for: .normal)
        button.titleLabel?.font = UIFont.boldSystemFont(ofSize: 16)
        button.contentHorizontalAlignment = UIControl.
           ContentHorizontalAlignment.left
        return button
    }()

    @objc func displayProfile()
    {
        if let id = self.user?.id {
            print("diplay")
            onUserDisplay?(id)
        }
    }

    var onUserDisplay: ((Int)->())?


    let timeLabel: UILabel = {
        let label = UILabel()
        label.font = UIFont.systemFont(ofSize: 9)
        label.textAlignment = NSTextAlignment.right
        label.textColor = #colorLiteral(red: 0.4352941176, green:
           0.4431372549, blue: 0.4745098039, alpha: 1)
        return label
    }()

    func createTextView(with text: NSAttributedString, _ isSelectable:
       Bool) -> UITextView
    {
        let textView = UITextView()
        textView.isScrollEnabled = false
        textView.isEditable = false
        textView.sizeToFit()

        if isSelectable {
            textView.isSelectable = true
        } else {
            textView.isUserInteractionEnabled = false
        }

        textView.attributedText = text
        return textView
    }

    override init(style: UITableViewCell.CellStyle, reuseIdentifier:
       String?)
```

```
{
    super.init(style: style, reuseIdentifier: reuseIdentifier)
}

required init?(coder aDecoder: NSCoder)
{
    fatalError("init(coder:) has not been implemented")
}

var user: Model.Users? {
    didSet{
        nameLabel.setTitle(user!.fullName(), for: .normal)
    }
}

var textView: UITextView!

func fill(with markdownText: NSAttributedString, byUser: Model.Users,
   when: String)
{
    let myId = DataStorage.standard.getUserId()
    // important
    contentView.subviews.forEach({ $0.removeFromSuperview() })

    nameLabel.addTarget(self, action: #selector(displayProfile), for:
       .touchUpInside)

    self.user = byUser
    textView = createTextView(with: markdownText, true)
    textView.layer.cornerRadius = 10
    textView.clipsToBounds = true

    self.contentView.addSubview(textView)
    self.contentView.addSubview(timeLabel)

    timeLabel.text = when.getDate()
    textView.width(min: 50, max: self.contentView.bounds.width-70,
       priority: .required, isActive: true)

    if myId == byUser.id
    {
        layoutMyMessage()
    } else {
        layOutOtherMessage()
    }
}

func layOutOtherMessage(){
    self.contentView.addSubview(nameLabel)

    nameLabel.setTitle(user!.fullName(), for: .normal)

    nameLabel.edgesToSuperview(excluding: .bottom, insets: .top(4) + .
       left(8))
    textView.edgesToSuperview(excluding: [.top, .right] , insets: .
       left(8) + .bottom(20))
    textView.topToBottom(of: nameLabel)
```

```
        textView.backgroundColor = #colorLiteral(red: 0.8039215803, green:
            0.8039215803, blue: 0.8039215803, alpha: 1)
        textView.textColor = .white
        timeLabel.edgesToSuperview(excluding: .top, insets: .right(70) + .
           left(8))
        timeLabel.topToBottom(of: textView, offset: 4)
        timeLabel.textAlignment = .left
    }

    func layoutMyMessage()
    {
        textView.edgesToSuperview(excluding: .left, insets: .right(8) + .
           bottom(20) + .top(4))
        textView.backgroundColor = UIColor(red:0.08, green:0.49, blue
           :0.98, alpha:1.0)
        textView.textColor = .white
        timeLabel.edgesToSuperview(excluding: .top, insets: .left(70) + .
           right(8))
        timeLabel.topToBottom(of: textView, offset: 4)
        timeLabel.textAlignment = .right
    }
}



//
//  DialogViewController.swift
//  GDproject
//
//  Created by cstore on 01/05/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit
import TinyConstraints
import Marklight

protocol UpdatableGroup:class  {
    func updateGroup(with group: Model.Group)
}

class DialogViewController: UIViewController, UpdatableGroup,
   UITableViewDelegate, UITableViewDataSource
{
    var onUserDisplay: ((Int)->())?

    var tableView: UITableView = UITableView()

    func updateGroup(with group: Model.Group){
        self.groupChat?.group = group
        setTitleForGroup(groupChat: groupChat!)
    }

    let cellId = "cell3"
    var onInfoShow: (()->())?

    var dialog: Model.Dialog? {
        didSet{
            switch dialog! {
```

```
            case .groupChat(let chat):
                self.groupChat = chat
            case .userChat(let chat):
                self.userChat = chat
            }
        }
    }

    var groupChat: Model.GroupChat?
    var userChat: Model.UserChat?

    var users: [Int: Model.Users]?

    var groupId: Int?

    var cellData: [PostCellData] = [] {
        didSet {
            tableView.reloadData()
            canBePaginated = true
        }
    }

    override func viewDidLoad() {
        super.viewDidLoad()

        setConstraints()
        magicForKeyboardChanges()

        self.tableView.delegate = self
        self.tableView.dataSource = self

        navigationItem.largeTitleDisplayMode = .never

        tableView.keyboardDismissMode = .onDrag
        tableView.separatorStyle = .none
        tableView.register(DialogCell.self, forCellReuseIdentifier: cellId
            )
        tableView.transform = CGAffineTransform(scaleX: 1, y: -1)
        tableView.contentInsetAdjustmentBehavior = .never
        tableView.contentOffset = CGPoint(x: 0, y: 30)

        if let groupChat = groupChat {
            setTitleForGroup(groupChat: groupChat)
        } else if let userChat = userChat{
            setTitleForChat(userChat: userChat)
        }
    }

    var messageSendView: UIView = {
        let view = UIView()
        view.backgroundColor = #colorLiteral(red: 1.0, green: 1.0, blue:
            1.0, alpha: 1.0)
        return view
    }()

    var messageTextView: UITextView =
    {
        let textStorage = MarklightTextStorage()
        textStorage.marklightTextProcessor.codeColor = UIColor.orange
```

```
        textStorage.marklightTextProcessor.quoteColor = UIColor.darkGray
        textStorage.marklightTextProcessor.syntaxColor = UIColor.blue
        textStorage.marklightTextProcessor.codeFontName = "Courier"
        textStorage.marklightTextProcessor.fontTextStyle = UIFont.
           TextStyle.subheadline.rawValue
        textStorage.marklightTextProcessor.hideSyntax = false

        let layoutManager = NSLayoutManager()

        // Assign the `UITextView``s `NSLayoutManager` to the `
           NSTextStorage` subclass
        //textStorage.addLayoutManager(textView.layoutManager)
        textStorage.addLayoutManager(layoutManager)

        let textContainer = NSTextContainer()
        layoutManager.addTextContainer(textContainer)

        let textView = UITextView(frame: CGRect.zero, textContainer:
           textContainer)
        textView.isEditable = true
        textView.isScrollEnabled = true
        textView.backgroundColor = .white
        return textView
    }()

    var sendButton: UIButton = {
        let button = UIButton()
        button.setTitle("Send", for: .normal)
        button.setTitleColor(.blue, for: .normal)
        button.addTarget(self, action: #selector(sendMessage), for: .
           touchUpInside)
        return button
    }()

    @objc func sendMessage()
    {
        var destination: Model.MessageDestination?

        if let group = groupChat {
            destination = Model.MessageDestination.groupChatDestination(
               group.group.id)
        } else if let user = userChat {
            destination = Model.MessageDestination.userChatDestination(
               user.user)
        }

        if let destination = destination
        {
            Model.sendMessage(message: Model.SendMessage(body: Model.
               Attachments(markdown: messageTextView.text), destination:
               destination)) { [unowned self] in

                switch $0 {
                case .success, .success1:
                    self.getMessagesNew(for: self.dialog!)
                    self.messageTextView.text = ""
                default:
                    self.showAlertOn(result: $0)
                }
```

```
            }

            prevLast = -1
        }
    }

    var lineView: UIView = {
        let view = UIView()
        view.backgroundColor = #colorLiteral(red: 0.6000000238, green:
           0.6000000238, blue: 0.6000000238, alpha: 1)
        return view
    }()

    var bottomConstraint: NSLayoutConstraint!

    func setConstraints(){
        self.view.addSubview(tableView)
        self.view.addSubview(messageSendView)

        messageSendView.addSubview(messageTextView)
        messageSendView.addSubview(sendButton)
        messageSendView.addSubview(lineView)

        messageSendView.edgesToSuperview(excluding: [.top,.bottom])

        bottomConstraint = NSLayoutConstraint(item: messageSendView,
           attribute: .bottom, relatedBy: .equal, toItem: self.view.
           safeAreaLayoutGuide, attribute: .bottom, multiplier: 1,
           constant: 0)
        view.addConstraint(bottomConstraint)

        messageSendView.height(60)

        sendButton.edgesToSuperview(excluding: .left)
        sendButton.width(60)

        lineView.edgesToSuperview(excluding: .bottom)
        lineView.height(0.5)

        messageTextView.edgesToSuperview(excluding: .right)
        messageTextView.rightToLeft(of: sendButton)

        tableView.edgesToSuperview(excluding: .bottom)
        tableView.bottomToTop(of: messageSendView)

        self.view.layoutSubviews()
    }

    func magicForKeyboardChanges()
    {
        NotificationCenter.default.addObserver(self, selector: #selector(
           handleKeyboardNotifications), name: UIResponder.
           keyboardWillShowNotification, object: nil)

        NotificationCenter.default.addObserver(self, selector: #selector(
           handleKeyboardNotifications), name: UIResponder.
           keyboardWillHideNotification, object: nil)
    }
```

```
    @objc func handleKeyboardNotifications(notification: NSNotification){
        if let userInfo = notification.userInfo{
            // UIKeyboardFrameEndUserInfoKey

            let keyBoardFrame = (userInfo[UIResponder.
               keyboardFrameEndUserInfoKey] as! NSValue).cgRectValue

            bottomConstraint.constant = notification.name == UIResponder.
               keyboardWillShowNotification ? -(keyBoardFrame.height) +
               self.view.safeAreaInsets.bottom : 0

            UIView.animate(withDuration: 0, delay: 0, options: .
               curveEaseOut, animations: {
                self.view.layoutIfNeeded()
            }) { (completed) in

            }
        }
    }

    func setTitleForChat(userChat: Model.UserChat){
        navigationItem.title = "\(users![userChat.user]!.fullName())"
        navigationItem.rightBarButtonItem = UIBarButtonItem(title: "_",
           style: .plain, target: self, action: #selector(showPersonPage))
    }

    @objc func showPersonPage(){
        if let id = userChat?.user{
            onUserDisplay?(id)
        }
    }

    func setTitleForGroup(groupChat: Model.GroupChat){
        navigationItem.title = "\(groupChat.group.name)"
        navigationItem.rightBarButtonItem = UIBarButtonItem(title: "Info",
            style: .plain, target: self, action: #selector(moveToInfoVC))
    }

    // onInfoShow
    @objc func moveToInfoVC(){
        let vc = UIStoryboard(name: "Main", bundle: nil).
           instantiateViewController(withIdentifier:
           chatInfoViewController) as! ChatInfoViewController

        vc.delegate = self
        vc.users = users!
        vc.onUserDiplay = onUserDisplay

        if let groupChat = groupChat {
            vc.groupChat = groupChat.group
        }

        navigationController?.pushViewController(vc, animated: true)
    }

    var currentMessagesInChat: [Model.LastMessage] = [] {
        didSet
        {
```

```
            cellData = currentMessagesInChat.map { PostCellData(
                attributedData: PostCellData.create(with: [$0.body])) }
        }
    }

    // MARK: - Table view data source

    func numberOfSections(in tableView: UITableView) -> Int {
        return 1
    }

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

    override func viewWillAppear(_ animated: Bool)
    {
        super.viewWillAppear(animated)
        tabBarController?.tabBar.isHidden = true

        if let dialog = dialog {
            getMessagesNew(for: dialog)
        }
    }

    func getMessages(for dialog: Model.Dialog, starting from: Int? = nil)
    {
        switch dialog {
        case .groupChat(let groupChat):
            Model.getMessagesFor(typeOfChat: Model.Dialog.groupChat(
                groupChat), chat: groupChat.group.id, exclusiveFrom: from)
            { [weak self] in
                self?.currentMessagesInChat.append(contentsOf: $0)
            }
        case .userChat(let userChat):
            Model.getMessagesFor(typeOfChat: Model.Dialog.userChat(
                userChat), chat: userChat.user, exclusiveFrom: from)
            { [weak self] in
                self?.currentMessagesInChat.append(contentsOf: $0)
            }
        }
    }


    func getMessagesNew(for dialog: Model.Dialog, starting from: Int? =
        nil)
    {
        switch dialog {
        case .groupChat(let groupChat):
            Model.getMessagesFor(typeOfChat: Model.Dialog.groupChat(
                groupChat), chat: groupChat.group.id, exclusiveFrom: from)
            { [weak self] in
                self?.currentMessagesInChat =  $0
            }
        case .userChat(let userChat):
            Model.getMessagesFor(typeOfChat: Model.Dialog.userChat(
                userChat), chat: userChat.user, exclusiveFrom: from)
            { [weak self] in
```

```
                self?.currentMessagesInChat = $0
            }
        }
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath:
       IndexPath) -> UITableViewCell
    {
        let cell = tableView.dequeueReusableCell(withIdentifier: cellId,
           for: indexPath) as! DialogCell

        //In cellForRowAtIndexPath
        cell.transform = CGAffineTransform(scaleX: 1, y: -1)
        cell.selectionStyle = .none


        if let user = users?[currentMessagesInChat[indexPath.row].author]
        {
            cell.fill(with: cellData[indexPath.row].attributedData, byUser
               : user, when: currentMessagesInChat[indexPath.row].time)

            cell.onUserDisplay = onUserDisplay
        }

        return cell
    }

    var prevLast = -1
    var canBePaginated = false
    func tableView(_ tableView: UITableView, willDisplay cell:
       UITableViewCell, forRowAt indexPath: IndexPath)
    {
        if indexPath.row == cellData.count - 1 && prevLast != indexPath.
           row && canBePaginated && cellData.count >= 9
        {
            print("exclusiveFrom \(currentMessagesInChat.last?.id ?? 0)")
            if let dialog = dialog
            {
                getMessages(for: dialog, starting: currentMessagesInChat.
                   last?.id)
            }

            prevLast = indexPath.row
        }
    }
}



//
//  MessagesViewController.swift
//  GDproject
//
//  Created by cstore on 01/05/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit
```

```
class MessagesViewController: UITableViewController {

    // curreent Active which can be displayed
    var currentActiveDialogs: [Model.Dialog] = [] {
        didSet {
            tableView.reloadData()
        }
    }

    // curreent users
    var users: [Int: Model.Users] = [:]

    var onUserDisplayList: (()->())?

    var onDialogDisplay: (((dialog: Model.Dialog, users: [Int:Model.Users
       ]))->())?

    let searchC = UISearchController(searchResultsController: nil)

    override func viewDidLoad() {
        super.viewDidLoad()

        // Uncomment the following line to display an Edit button in the
           navigation bar for this view controller.
        self.navigationItem.rightBarButtonItem = UIBarButtonItem(title: "
           Write", style: .plain, target: self, action: #selector(
           choosePerson))

        self.navigationItem.title = "Messages"
        // self.navigationItem.largeTitleDisplayMode = .always
        self.navigationItem.searchController = searchC
        navigationController?.navigationBar.prefersLargeTitles = true
        self.navigationItem.hidesSearchBarWhenScrolling = false
    }

    @objc func choosePerson(){
        onUserDisplayList?()
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        tabBarController?.tabBar.isHidden = false
        Model.getChatAll { [weak self]  in
            self?.currentActiveDialogs = $0.0
            self?.users = $0.1
        }
    }

    // MARK: - Table view data source
    override func numberOfSections(in tableView: UITableView) -> Int {
        return 1
    }

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

```
    override func tableView(_ tableView: UITableView, cellForRowAt
       indexPath: IndexPath) -> UITableViewCell
    {
        let cell = tableView.dequeueReusableCell(withIdentifier: "
           MessageViewCell", for: indexPath) as! MessageViewCell

        switch currentActiveDialogs[indexPath.row].self {
        case .groupChat:
            cell.fill(with: currentActiveDialogs[indexPath.row])
        case .userChat(let userChat):
            cell.fill(with: currentActiveDialogs[indexPath.row], user:
               users[userChat.user])
        }

        return cell
    }

    override func tableView(_ tableView: UITableView, didSelectRowAt
       indexPath: IndexPath) {
        let tuple = (currentActiveDialogs[indexPath.row],users)
        onDialogDisplay?(tuple)
    }
}
//
//  MessageViewCell.swift
//  GDproject
//
//  Created by cstore on 13/05/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit

class MessageViewCell: UITableViewCell {

    @IBOutlet weak var dialogName: UILabel!

    @IBOutlet weak var lastMessagePreview: UILabel!

    @IBOutlet weak var dateLabel: UILabel!

    override func awakeFromNib() {
        super.awakeFromNib()
        // Initialization code
    }

    func fill(with dialog: Model.Dialog, user: Model.Users? = nil)
    {
        switch dialog {

        case .groupChat(let group):
            dialogName.text = group.group.name
            lastMessagePreview.text = group.lastMessage?.body.markdown
            dateLabel.text = group.lastMessage?.time.getDate()
        case .userChat(let userChat):
            dialogName.text = "\(user!.fullName())"
            lastMessagePreview.text = userChat.lastMessage?.body.markdown
            dateLabel.text = userChat.lastMessage?.time.getDate()
        }
```

```
    }
}
//
//  PeopleToWriteViewController.swift
//  GDproject
//
//  Created by cstore on 01/05/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit

class PeopleToWriteViewController: UITableViewController {

    // TODO: - edit button when it's used for selection
    var whatToDoWithSelection: (([Int: Model.UserPermission])->())?

    let searchC = UISearchController(searchResultsController: nil)

    let users = Model.Channels.fullPeople

    var chosenUsers: [Int: Model.UserPermission] = [:]

    override func viewDidLoad() {
        super.viewDidLoad()

        tableView.isEditing = true
        self.navigationItem.title = "People"

        self.navigationItem.rightBarButtonItem = UIBarButtonItem(
           barButtonSystemItem: .done, target: self, action: #selector(
           newMessage))

        self.navigationItem.largeTitleDisplayMode = .never
        self.navigationItem.searchController = searchC
        self.navigationItem.hidesSearchBarWhenScrolling = false
    }

    override func setEditing(_ editing: Bool, animated: Bool) {
        super.setEditing(true, animated: animated)
    }

    override func numberOfSections(in tableView: UITableView) -> Int {
        return 1
    }

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

    override func tableView(_ tableView: UITableView, didSelectRowAt
       indexPath: IndexPath) {
        chosenUsers[users[indexPath.row].id] = Model.UserPermission(
           isAdmin: false)
    }

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

```
    {
        let cell = tableView.dequeueReusableCell(withIdentifier: "
           peopleToWriteCell", for: indexPath)

        cell.textLabel?.text = users[indexPath.row].fullName()

        return cell
    }

    @objc func newMessage()
    {
        if chosenUsers.count != 0
        {
            whatToDoWithSelection?(chosenUsers)
        }
    }
}
//
//  MyStackView.swift
//  GDproject
//
//  Created by cstore on 14/02/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit

class MyStackView: UIStackView {

    var isFull: Bool = true


    override var bounds: CGRect {
        didSet{
            if frame.height == 300 && !isFull {
                let gradientLayer = CAGradientLayer()
                gradientLayer.colors = [UIColor(displayP3Red: 255, green:
                   255, blue: 255, alpha: 0).cgColor, UIColor.white.
                   cgColor]
                gradientLayer.startPoint = CGPoint(x: 0.5, y: 0.5)
                gradientLayer.frame = self.bounds
                self.layer.addSublayer(gradientLayer)
            }
        }
    }
}
//
//  NewPostViewController.swift
//  GDproject
//
//  Created by cstore on 05/02/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit
import Cartography
import Marklight
import TinyConstraints
```

```
class NewPostViewController: UIViewController, UITextViewDelegate {

    @IBOutlet weak var view1: UIView!
    // Keep strong instance of the ‘NSTextStorage‘ subclass
    let textStorage = MarklightTextStorage()

    weak var parentVC: NewsController?

    static var draft: String = ""
    var textView: UITextView!

    // buttons for attaching images
    var accessoryView: UIView = {
        let view = UIView()
        view.backgroundColor = .white
        return view
    }()

    var addEquasion: UIButton = {
        var button = UIButton(type: .detailDisclosure)
        button.addTarget(self, action: #selector(addMathBrackets), for: .
           touchUpInside)
        return button
    }()

    // stack view where buttons will be places
    var stackAccessoryView: UIStackView?
    // Connect the view1's bottom layout constraint to react to keyboard
       movements

    @IBOutlet weak var bottomTextViewConstraint: NSLayoutConstraint!


    override func viewDidLoad()
    {
        super.viewDidLoad()
        setUpMD()
        setUpTextView()
        setUpAccessoryView()
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)

        navigationController?.navigationBar.prefersLargeTitles = false
        navigationItem.title = "New post"
        textView.text = NewPostViewController.draft
    }

    func setUpAccessoryView(){
        let views = [UIButton(type: .contactAdd),addEquasion]
        stackAccessoryView = UIStackView(arrangedSubviews: views)

        stackAccessoryView?.alignment = .fill
        stackAccessoryView?.distribution = .equalSpacing

        view1.addSubview(accessoryView)
        accessoryView.addSubview(stackAccessoryView!)
        accessoryView.height(40)
```

```
        let separatorView = UIView()
        separatorView.backgroundColor = #colorLiteral(red: 0.8039215803,
            green: 0.8039215803, blue: 0.8039215803, alpha: 1)
        accessoryView.addSubview(separatorView)
        separatorView.height(1)
        separatorView.edgesToSuperview(excluding: .bottom)

        accessoryView.edgesToSuperview(excluding: [.top])
        stackAccessoryView!.edgesToSuperview(insets: .left(16) + .right
            (16) + .top(1))
    }

    func setUpTextView(){
        view.addConstraint(bottomTextViewConstraint)
        view1.addSubview(textView)
        textView.edgesToSuperview(insets: .top(8) + .left(8) + .bottom
            (40+8) + .right(8))

        if #available(iOS 11.0, *) {
            textView.smartDashesType = .no
            textView.smartQuotesType = .no
        }

        textView.isScrollEnabled = true

        guard let bottomTextViewConstraint = bottomTextViewConstraint else
            { return }
        view.addConstraint(bottomTextViewConstraint)

        // Add a beautiful padding to the `UITextView` content
        textView.textContainerInset = UIEdgeInsets(top: 4, left: 4, bottom
            : 4, right: 4)
        textView.delegate = self
        magicForKeyboardChanges()
    }


    func magicForKeyboardChanges()
    {
        ////////////////////////////
        // We do some magic to resize the `UITextView` to react the the
            keyboard size change (appearance, disappearance, ecc)
        NotificationCenter.default.addObserver(self, selector: #selector(
            handleKeyboardNotifications), name: UIResponder.
            keyboardWillShowNotification, object: nil)

        NotificationCenter.default.addObserver(self, selector: #selector(
            handleKeyboardNotifications), name: UIResponder.
            keyboardWillHideNotification, object: nil)

        // Partial fixes to a long standing bug, to keep the caret inside
            the `UITextView` always visible
        NotificationCenter.default.addObserver(forName: UITextView.
            textDidChangeNotification, object: textView, queue:
            OperationQueue.main) { (notification) -> Void in
            if self.textView.textStorage.string.hasSuffix("\n") {
                CATransaction.setCompletionBlock({ () -> Void in
                    self.scrollToCaret(self.textView, animated: false)
```

```
            })
        } else {
            self.scrollToCaret(self.textView, animated: false)
        }
    }

}

@objc func handleKeyboardNotifications(notification: NSNotification){
    if let userInfo = notification.userInfo{
        // UIKeyboardFrameEndUserInfoKey
        let keyBoardFrame = (userInfo[UIResponder.
            keyboardFrameEndUserInfoKey] as! NSValue).cgRectValue
        bottomTextViewConstraint?.constant = notification.name ==
            UIResponder.keyboardWillShowNotification ? keyBoardFrame.
            height : 0
        print(bottomTextViewConstraint!.constant)

        UIView.animate(withDuration: 0, delay: 0, options: .
            curveEaseOut, animations: {
            self.view.layoutIfNeeded()
        }) { (completed) in

        }
    }
}

@objc func attachPhoto(){

}


@objc func addMathBrackets()
{
    if textView.text.count != 0 &&  textView.text.last != "\n" {
        textView.insertText("\n\\[\n      ")
    }
    else {
        textView.insertText("\\[\n      ")
    }

    if let selectedRange = textView.selectedTextRange
    {
        textView.insertText("\n\\]\n")

        if let newPosition = textView.position(from: selectedRange.
            start, offset: 0)
        {
            // set the new position
            textView.selectedTextRange = textView.textRange(from:
                newPosition, to: newPosition)
        }
    }
}

func setUpMD(){
    textStorage.marklightTextProcessor.codeColor = UIColor.orange
    textStorage.marklightTextProcessor.quoteColor = UIColor.darkGray
    textStorage.marklightTextProcessor.syntaxColor = UIColor.blue
```

```
        textStorage.marklightTextProcessor.codeFontName = "Courier"
        textStorage.marklightTextProcessor.fontTextStyle = UIFont.
            TextStyle.subheadline.rawValue
        textStorage.marklightTextProcessor.hideSyntax = false

        let layoutManager = NSLayoutManager()

        // Assign the `UITextView`'s `NSLayoutManager` to the `
            NSTextStorage` subclass
        textStorage.addLayoutManager(layoutManager)

        let textContainer = NSTextContainer()
        layoutManager.addTextContainer(textContainer)

        textView = UITextView(frame: view.bounds, textContainer:
            textContainer)
    }

    // MARK:- new post
    @objc func newPost(){


        // adding row to uiTableView after adding new post
        guard let vc = parentVC else {return}
        vc.tableView.beginUpdates()
        let indexPath1: IndexPath = IndexPath(row: 0, section: 0)
        vc.dataSourse.insert(p, at: 0)
        vc.tableView.insertRows(at: [indexPath1], with: .fade)
        vc.tableView.endUpdates()
        moveBackToParentVC()
        // somewhere here i will be sending server notifications about new
            post arrival
    }

    @objc func actionSaveDraft(){
        let optionMenu = UIAlertController(title: nil, message: nil,
            preferredStyle: .actionSheet)

        let saveAction = UIAlertAction(title: "Save draft", style: .
            default)
        {
            _ in
            NewPostViewController.draft = self.textView.text
            self.moveBackToParentVC()
        }

        let deleteAction = UIAlertAction(title: "Delete draft", style: .
            destructive)
        {
            (_)
            in
            NewPostViewController.draft = ""
            self.moveBackToParentVC()
        }

        let cancelAction = UIAlertAction(title: "Cancel", style: .cancel)

        optionMenu.addAction(saveAction)
        optionMenu.addAction(deleteAction)
```

```
        optionMenu.addAction(cancelAction)

        self.present(optionMenu, animated: true, completion: nil)
    }

    @objc func closeView()
    {
        actionSaveDraft()
    }

    func moveBackToParentVC(){
        let transition = CATransition()
        transition.duration = 0.5
        transition.timingFunction = CAMediaTimingFunction(name:
           CAMediaTimingFunctionName.easeInEaseOut)
        transition.type = CATransitionType.reveal
        transition.subtype = CATransitionSubtype.fromBottom
        navigationController?.view.layer.add(transition, forKey: nil)
        navigationController?.popViewController(animated: false)
        textView!.resignFirstResponder()
    }



    func textView(_ textView: UITextView, shouldInteractWith URL: URL, in
       characterRange: NSRange) -> Bool {
        print("Should interact with: \(URL)")
        return true
    }

    func scrollToCaret(_ textView: UITextView, animated: Bool) {
        var rect = textView.caretRect(for: textView.selectedTextRange!.end
           )
        rect.size.height = rect.size.height + textView.textContainerInset.
           bottom
        textView.scrollRectToVisible(rect, animated: animated)
    }
}
//
//  ViewController.swift
//  GDproject
//
//  Created by cstore on 13/02/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit
import TinyConstraints

// MARK:- Controller with posts and channels availiable.
// Search is availiable within every table (posts and channels). Has
   button-functionality for boths post and chnnels
class NewsController: UIViewController
{

    @IBOutlet weak var tableView: UITableView!
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.03-01 ТЗ 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

```
var searchController = UISearchController(searchResultsController: nil
    )



override func viewDidLoad()
{
    super.viewDidLoad()
    tableView.register(HeaderNewsChannels.self, forCellReuseIdentifier
        : headerNewsChannelsVC)
    tableView.register(PostViewCell.self, forCellReuseIdentifier:
        postCellId)

    tableView.delegate = self
    tableView.dataSource = self
    tableView.reloadData()

    setUpNavigationItemsforPosts()
    searchController.searchBar.placeholder  = "Search anything"

    navigationItem.searchController = searchController
    definesPresentationContext = true

    setUpBanner()
}

override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    navigationController?.navigationBar.prefersLargeTitles = true
}

func setUpNavigationItemsforPosts(){
    navigationItem.title = "Posts"
    navigationItem.rightBarButtonItems = [UIBarButtonItem(
        barButtonSystemItem: .compose, target: self, action: #selector(
        self.writePost(_:)))]
}

func setUpNavigationItemsForChannels(){
    navigationItem.title = "Channels"
    navigationItem.rightBarButtonItems = [UIBarButtonItem(
        barButtonSystemItem: .add, target: self, action: #selector(
        addChannel))]
}

@objc func writePost(_ barItem: UIBarButtonItem)
{
    let vc = storyboard?.instantiateViewController(withIdentifier: "
        NewPostViewController") as! NewPostViewController

    vc.navigationItem.rightBarButtonItem = UIBarButtonItem(title: "
        Post", style: .plain, target: vc, action: #selector(vc.newPost)
        )
    vc.navigationItem.leftBarButtonItem = UIBarButtonItem(title: "
        Close", style: .plain, target: vc, action: #selector(vc.
        closeView))

    vc.parentVC = self
```

```
        let transition = CATransition()
        transition.duration = 0.5
        transition.timingFunction = CAMediaTimingFunction(name:
            CAMediaTimingFunctionName.easeInEaseOut)
        transition.type = CATransitionType.moveIn
        transition.subtype = CATransitionSubtype.fromTop
        navigationController?.view.layer.add(transition, forKey: nil)
        navigationController?.pushViewController(vc, animated: false)
    }

    @objc func addChannel(){
        print("Add channel")
    }

    // for animating the banner
    var topConstraint: NSLayoutConstraint?

    let bannerView: UIView = {
        let view = UIView()
        view.backgroundColor = .blue
        view.layer.cornerRadius = 25.0
        view.clipsToBounds = true
        return view
    }()

    let statusLabel: UILabel = {
        let label = UILabel()
        label.text = "-"
        label.textColor = .white
        label.textAlignment = .center
        return label
    }()

    private func setUpBanner()
    {
        view.addSubview(bannerView)
        bannerView.addSubview(statusLabel)
        statusLabel.edgesToSuperview()

        let tap = UITapGestureRecognizer(target: self, action: #selector(
            handleTap(sender:)))

        bannerView.addGestureRecognizer(tap)
        topConstraint = NSLayoutConstraint(item: bannerView, attribute: .
            top, relatedBy: .equal, toItem: view.safeAreaLayoutGuide,
            attribute: .top, multiplier: 1, constant: -300)
        view.addConstraint(topConstraint!)

        bannerView.edgesToSuperview(excluding: [.bottom, .top, .left],
            insets: .right(20), usingSafeArea: true)
        bannerView.height(50)
        bannerView.width(50)
    }

    // when table is scrolling no deletion is availiable
    @objc func handleTap(sender: UITapGestureRecognizer? = nil) {
        let indexPath = IndexPath(row: 0, section: 0)
        self.tableView.scrollToRow(at: indexPath, at: .top, animated: true
            )
```

```
//        isBannerVisible = false
//        changeConstraint(isVisible: false)
    }

    // animation for banner
    func changeConstraint(isVisible: Bool){
        topConstraint?.constant =  isVisible ? 50 : -300

        UIView.animate(withDuration: 0.5, delay: 0, options: .curveEaseOut
           , animations: {
            self.view.layoutIfNeeded()
        }) { (completed) in

        }
    }

    var isBannerVisible: Bool = false

     func scrollViewDidScroll(_ scrollView: UIScrollView) {
        print(scrollView.contentOffset.y)
        if scrollView.contentOffset.y >= 50 && !isBannerVisible{
            isBannerVisible = true
            changeConstraint(isVisible: true)
        }

        if isBannerVisible && scrollView.contentOffset.y == 0{
            isBannerVisible = false
            changeConstraint(isVisible: false)
        }
    }
}

// MARK:- tableView delegate
extension NewsController: UITableViewDelegate{
    func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
       IndexPath)
    {
        tableView.deselectRow(at: indexPath, animated: true)

        let vc = storyboard!.instantiateViewController(withIdentifier:
           fullPostControllerId) as! FullPostController

        vc.post = dataSourse[indexPath.row]
        navigationController!.pushViewController(vc, animated: true)
    }
}

// MARK:- tableView dataSourse
extension NewsController: UITableViewDataSource
{
    func tableView(_ tableView: UITableView, numberOfRowsInSection section
       : Int) -> Int {
        return dataSourse.count
    }

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

```
        let cell = tableView.dequeueReusableCell(withIdentifier:
            postCellId, for: indexPath) as! PostViewCell

        cell.fill(with: dataSourse[indexPath.row].dataArray, false)
        cell.selectionStyle = .none
        return cell
    }

    func tableView(_ tableView: UITableView, heightForRowAt indexPath:
        IndexPath) -> CGFloat {
        return UITableView.automaticDimension
    }

    func tableView(_ tableView: UITableView, estimatedHeightForRowAt
        indexPath: IndexPath) -> CGFloat {
        return 100.0
    }

    func tableView(_ tableView: UITableView, heightForHeaderInSection
        section: Int) -> CGFloat {
        return 46.0
    }

    func tableView(_ tableView: UITableView, viewForHeaderInSection
        section: Int) -> UIView?
    {
        let cell = tableView.dequeueReusableCell(withIdentifier:
            headerNewsChannelsVC) as! HeaderNewsChannels
        cell.vc = self

        let view = UIView(frame: CGRect(x: 0, y: 0, width: UIScreen.main.
            bounds.width, height: 46.0))
        view.addSubview(cell)
        cell.edgesToSuperview()

        return view
    }


}
//
//  BasicInfoCell.swift
//  NewsFeed
//
//  Created by cstore on 23/01/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit
import TinyConstraints

class BasicInfoCell: UITableViewCell{

    var titleLabel: UILabel = {
        let label = UILabel()
        label.font = UIFont.boldSystemFont(ofSize: 16)
        label.textColor =  .black
        return label
    }()
```

```
    override init(style: UITableViewCell.CellStyle, reuseIdentifier:
       String?) {
        super.init(style: style, reuseIdentifier: reuseIdentifier)
    }

    func setUpView(){
        addSubview(titleLabel)
        titleLabel.horizontalToSuperview(insets: .left(16) + .right(16))
        titleLabel.verticalToSuperview(insets: .top(8) + .bottom(8))
    }

    required init?(coder aDecoder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    func fill(title: String){
        setUpView()
        titleLabel.text = title
    }
}
//
//  BasicInfoController.swift
//  GDproject
//
//  Created by cstore on 16/02/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit

struct CellData{
    var opened = Bool()
    var title = String()
    var sectionData = [String]()
}

class BasicInfoController: UIViewController, UITableViewDelegate,
   UITableViewDataSource {

    var userInfo: Model.Users? {
        didSet {
            if let userInfo = userInfo {
                dataSourse = [CellData(opened: false, title: "Contacts",
                   sectionData: [userInfo.email, userInfo.faculty.address
                   ]),
                CellData(opened: false, title: "Faculty", sectionData: [
                   userInfo.faculty.campusName, userInfo.faculty.name,
                   userInfo.faculty.address, userInfo.faculty.path]),
                CellData(opened: false, title: "Interests", sectionData:
                   userInfo.faculty.tags)]
            }
        }
    }

    var dataSourse: [CellData] = [ ]

    func tableView(_ tableView: UITableView, numberOfRowsInSection section
       : Int) -> Int {
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.03-01 ТЗ 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

```
        if (dataSourse[section].opened) {
            return dataSourse[section].sectionData.count + 1
        } else {
            return 1
        }
    }

    func numberOfSections(in tableView: UITableView) -> Int {
        return dataSourse.count
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath:
       IndexPath) -> UITableViewCell {
        if indexPath.row == 0 {
            let cell = tableView.dequeueReusableCell(withIdentifier: "cell
               ", for: indexPath)
            cell.textLabel?.text = dataSourse[indexPath.section].title
            cell.accessoryType = .disclosureIndicator
            cell.selectionStyle = .none
            return cell
        } else {
            let cell = tableView.dequeueReusableCell(withIdentifier:
               infoCellId, for: indexPath) as! InfoCell
            cell.fill(title: dataSourse[indexPath.section].sectionData[
               indexPath.row - 1])
            cell.accessoryType = .none
            cell.selectionStyle = .none
            return cell
        }
    }

    func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
       IndexPath) {
        if indexPath.row == 0 {
            if dataSourse[indexPath.section].opened {
                dataSourse[indexPath.section].opened = false
                let section = IndexSet.init(integer: indexPath.section)
                tableView.reloadSections(section, with: .none)
            } else {
                dataSourse[indexPath.section].opened = true
                let section = IndexSet.init(integer: indexPath.section)
                tableView.reloadSections(section, with: .none)
            }
        }
    }
    func tableView(_ tableView: UITableView, heightForRowAt indexPath:
       IndexPath) -> CGFloat {
        return UITableView.automaticDimension
    }

    func tableView(_ tableView: UITableView, estimatedHeightForRowAt
       indexPath: IndexPath) -> CGFloat {
        return 100.0
    }
}
//
//  BasicInfoCell.swift
//  NewsFeed
//
```

```
//  Created by cstore on 23/01/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit
import TinyConstraints

class InfoCell: UITableViewCell{

    var textView: UITextView = {
        let label = UITextView()
        label.font = UIFont.systemFont(ofSize: 16)
        label.sizeToFit()
        label.isScrollEnabled = false
        // label.isUserInteractionEnabled = false
        label.isEditable = false
        label.dataDetectorTypes = .all
        label.textColor =  .black
        return label
    }()

    override init(style: UITableViewCell.CellStyle, reuseIdentifier:
       String?) {
        super.init(style: style, reuseIdentifier: reuseIdentifier)
    }

    func setUpView()
    {
        addSubview(textView)
        textView.horizontalToSuperview(insets: .left(32) + .right(16))
        textView.verticalToSuperview(insets: .top(8) + .bottom(8))
    }

    required init?(coder aDecoder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    func fill(title: String){
        setUpView()
        textView.text = title
    }
}
//
//  LogInViewController.swift
//  NewsFeed
//
//  Created by cstore on 20/01/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit

class InviteViewController: UIViewController {

    @IBOutlet weak var mailTextField: UITextField!

    @IBOutlet weak var inviteLabel: UILabel!
```

```
    static let titleColor = UIColor(red: 0, green: 137/255, blue: 249/255,
        alpha: 0.5)

    var bottomConstraint: NSLayoutConstraint?

    let logInButton: UIButton = {
        let button = UIButton(type: .system)
        button.setTitle("Invite", for: .normal)
        button.setTitleColor(titleColor, for: .normal)
        button.titleLabel?.font  = UIFont.boldSystemFont(ofSize: 16)
        button.isEnabled = false
        button.addTarget(self, action: #selector(activateLogInProcess),
           for: .touchUpInside)
        return button
    }()

    let keyboardBar: UIView = {
        let view = UIView()
        view.backgroundColor = .white
        return view
    }()

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

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        // MARK:- uncomment when mail registration will be availiable
        //self.navigationController?.setNavigationBarHidden(true, animated
           : animated)


        navigationController?.navigationBar.prefersLargeTitles = false
    }

    override func viewWillDisappear(_ animated: Bool) {
        super.viewWillDisappear(animated)
        // MARK:- uncomment when mail registration will be availiable
        //self.navigationController?.setNavigationBarHidden(false,
           animated: animated)
    }

    private func configureTapgesture(){
        let tapGesture = UITapGestureRecognizer(target: self, action: #
           selector(handleTap))
        view.addGestureRecognizer(tapGesture)
    }

    @objc func handleTap(){
        view.endEditing(true)
    }

    @objc func activateLogInProcess(){
        if logInButton.isEnabled {
            // instead of transporting user for new vc, make him a sign
               that everything was complete
```

```
            inviteLabel.text?.append(" _")
            view.endEditing(true)
        }
    }

    func setUpView(){
        mailTextField.delegate = self
        view.addSubview(keyboardBar)

        view.addConstraintsWithFormat(format: "H:|[v0]|", views:
            keyboardBar)
        view.addConstraintsWithFormat(format: "V:[v0(50)]", views:
            keyboardBar)

        setUpBarComponents()

        bottomConstraint = NSLayoutConstraint(item: keyboardBar, attribute
            : .bottom, relatedBy: .equal, toItem: view.safeAreaLayoutGuide,
             attribute: .bottom, multiplier: 1, constant: 0)
        view.addConstraint(bottomConstraint!)

        // for keyboard notifications
        NotificationCenter.default.addObserver(self, selector: #selector(
            handleKeyboardNotifications), name: UIResponder.
            keyboardWillShowNotification, object: nil)

        NotificationCenter.default.addObserver(self, selector: #selector(
            handleKeyboardNotifications), name: UIResponder.
            keyboardWillHideNotification, object: nil)

        // for log in button notifications
        NotificationCenter.default.addObserver(self, selector: #selector(
            inputDidChanged), name: UITextField.textDidChangeNotification,
            object: mailTextField)

        NotificationCenter.default.addObserver(self, selector: #selector(
            inputDidChanged), name: UITextField.
            textDidBeginEditingNotification, object: mailTextField)
    }


    @objc func inputDidChanged(notification: NSNotification){
        if mailTextField.text?.isEmpty ?? true
        {
            logInButton.isEnabled = false
            logInButton.setTitleColor(InviteViewController.titleColor.
                withAlphaComponent(0.5), for: .normal)
        }
        else
        {
            logInButton.isEnabled = true
            logInButton.setTitleColor(InviteViewController.titleColor.
                withAlphaComponent(1), for: .normal)
        }
    }

    @objc func handleKeyboardNotifications(notification: NSNotification){
        if let userInfo = notification.userInfo{
            // UIKeyboardFrameEndUserInfoKey
```

```
            let keyBoardFrame = (userInfo[UIResponder.
                keyboardFrameEndUserInfoKey] as! NSValue).cgRectValue
            bottomConstraint?.constant = notification.name == UIResponder.
                keyboardWillShowNotification ? -keyBoardFrame.height+view.
                safeAreaInsets.bottom : 0

            UIView.animate(withDuration: 0, delay: 0, options: .
                curveEaseOut, animations: {
                 self.view.layoutIfNeeded()
            }) { (completed) in

            }
        }
    }

    func setUpBarComponents(){
        keyboardBar.addSubview(logInButton)
        keyboardBar.addConstraintsWithFormat(format: "H:[v0(60)]-16-|",
            views: logInButton)
        keyboardBar.addConstraintsWithFormat(format: "V:|[v0]|", views:
            logInButton)
    }
}


extension InviteViewController: UITextFieldDelegate{
    func textFieldShouldReturn(_ textField: UITextField) -> Bool
    {
        textField.resignFirstResponder()
        return true
    }
}
//
//  ProfileViewController.swift
//  GDproject
//
//  Created by cstore on 15/02/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit
import TinyConstraints

protocol UpdateUser: class {
    func updateUserObj(with user: Model.Users)
}

class ProfileViewController: UIViewController, UpdateUser
{
    func updateUserObj(with user: Model.Users)
    {
        self.user = user
        Model.Channels.fullPeopleDict[user.id] = user
    }

    @IBOutlet weak var segmentedControl: UISegmentedControl!

    @IBOutlet weak var profileView: UIView!
```

```
    @IBOutlet weak var tableView: UITableView!

    @IBOutlet weak var profileImageView: UIImageView!

    @IBOutlet weak var surnameLabel: UILabel!

    @IBOutlet weak var nameLabel: UILabel!

    @IBOutlet weak var facultyLabel: UILabel!

    @IBOutlet weak var placeLabel: UILabel!

    @IBOutlet weak var newMessageButton: UIButton!

    var logOut: (()->())?

    var deleteAllSessions: (()->())?

    var onSettings: (()->())?

    @IBAction func sendMessage(_ sender: UIButton)
    {
        if let userId = idProfile
        {
            let createdDialog = Model.Dialog.userChat(Model.UserChat(user:
                userId))
            let vc = DialogViewController()
            vc.users = Model.Channels.fullPeopleDict
            vc.dialog = createdDialog
            self.navigationController?.pushViewController(vc, animated:
               true)
        }
    }

    var protoDictionary: [String: UIImage] = ["135213": #imageLiteral(
       resourceName: "9"), "135288": #imageLiteral(resourceName: "5051"),
       "22723" : #imageLiteral(resourceName: "69"), "135083": #
       imageLiteral(resourceName: "42")]

    func fill(with user: Model.Users)
    {
        self.facultyLabel.text =  user.faculty.name
        self.nameLabel.text = "\(user.firstName) \(user.middleName)"
        self.surnameLabel.text = "\(user.lastName)"
        self.profileImageView.image = protoDictionary[user.faculty.
           campusCode]?.roundedImage
        self.placeLabel.text = "\(user.faculty.address)"
        if user.id == DataStorage.standard.getUserId() {
            newMessageButton.isHidden  = true
        } else {
            newMessageButton.isHidden  = false
        }
    }

    var user: Model.Users? {
        didSet {
            if let user = user {
                self.fill(with: user)
```

```
                navigationItem.title = "\(user.firstName) \(user.lastName)
                    "
            } else if let id = idProfile {
                Model.getUsers(for: [id]) { [unowned self ] in
                    self.user = $0[id]
                }
            }
        }
    }

    var channel: Model.Channels? {
        didSet {
            self.update()
        }
    }

    func update()
    {
        Model.getAnonymousChannel(by: channel!) { [unowned self] in
            self.posts.dataSourse = $0.posts
            self.posts.dictionary = $0.users
            self.user = $0.users[self.idProfile!]
        }
    }

    var basicInfo = BasicInfoController()
    var posts = NewsVC()

    override func viewDidLoad()
    {
        super.viewDidLoad()
        if idProfile == nil {
            idProfile = DataStorage.standard.getUserId()
        }

        posts.viewController = self
        posts.type = .NONE
        posts.currChannel = channel

        posts.onFullPost = {
            [weak self] (type,post) in

            let vc = self?.storyboard?.instantiateViewController(
                withIdentifier: fullPostControllerId) as!
                FullPostController
            vc.type = type
            vc.post = post
            self?.navigationController!.pushViewController(vc, animated:
                true)
        }

        tableView.register(UITableViewCell.self, forCellReuseIdentifier: "
            cell")

        tableView.register(PostViewCell.self, forCellReuseIdentifier:
            postCellId)

        tableView.register(BasicInfoCell.self, forCellReuseIdentifier:
            basicInfoCellId)
```

```
        tableView.register(InfoCell.self, forCellReuseIdentifier:
            infoCellId)

        posts.viewController = self

        tableView.delegate = posts
        tableView.dataSource = posts
        tableView.reloadData()
    }

    var idProfile: Int? {
        didSet {
            channel = Model.Channels(people: [idProfile!], name: "", id:
                0, tags: [])
        }
    }

    override func viewWillAppear(_ animated: Bool) {
        tabBarController?.tabBar.isHidden = false

        if idProfile == nil {
            idProfile = DataStorage.standard.getUserId()
        }
        user = Model.Channels.fullPeopleDict[idProfile!]

        update()

        setUpNavigarionBar()
    }

    func setUpNavigarionBar(){
        navigationController?.navigationBar.prefersLargeTitles = true
        let uibarbutton = UIBarButtonItem(title: "More", style: .plain,
            target: self, action: #selector(showInformation))
        navigationItem.rightBarButtonItems = [uibarbutton]
        navigationItem.largeTitleDisplayMode = .always
    }

    @objc func showInformation(){

        let optionMenu = UIAlertController(title: nil, message: nil,
            preferredStyle: .actionSheet)


        let cancelAction = UIAlertAction(title: "Cancel", style: .cancel)


        if idProfile == DataStorage.standard.getUserId() {
            let logoutAction = UIAlertAction(title: "Log out", style: .
                destructive)
            { [weak self] (_) in

                if let logOut = self?.logOut {
                    logOut()
                } else {
                    Model.logout() {
                        DataStorage.standard.setIsLoggedIn(value: false,
                            with: 0)
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.03-01 ТЗ 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

```
                    (UIApplication.shared.delegate as! AppDelegate).
                        relaunch()
                }
            }
        }

        let deleetAllSessionsAction = UIAlertAction(title: "Delete all
            sessions", style: .destructive)
        { [weak self] _ in

            if let delete = self?.deleteAllSessions {
                delete()
            } else {
                Model.deactivateAll() {
                    DataStorage.standard.setIsLoggedIn(value: false,
                        with: 0)
                    (UIApplication.shared.delegate as! AppDelegate).
                        relaunch()
                }
            }
        }

        let settingsAction = UIAlertAction(title: "Edit profile",
            style: .default)
        { [weak self] (_) in

            if let settings = self?.onSettings{
                settings()
            } else {
                let vc = self?.storyboard?.instantiateViewController(
                    withIdentifier: "RegisterTableViewController") as!
                    RegisterTableViewController
                vc.delegate = self
                vc.userActive = self?.user

                self?.navigationController?.pushViewController(vc,
                    animated: true)
            }
        }

        optionMenu.addAction(settingsAction)
        optionMenu.addAction(logoutAction)
        optionMenu.addAction(deleetAllSessionsAction)
    } else {
        let channelAction = UIAlertAction(title: "Add to a channel",
            style: .default)
        {
            [weak self] (_) in

            let vc = self?.storyboard?.instantiateViewController(
                withIdentifier: simplifiedChannelsList) as!
                SimplifiedChannelsList
            vc.user = self?.user

            let transition = CATransition()
            transition.duration = 0.5
            transition.timingFunction = CAMediaTimingFunction(name:
                CAMediaTimingFunctionName.easeInEaseOut)
            transition.type = CATransitionType.moveIn
```

```
                transition.subtype = CATransitionSubtype.fromTop
                self?.navigationController?.view.layer.add(transition,
                    forKey: nil)
                self?.navigationController?.pushViewController(vc,
                    animated: false)
            }

            optionMenu.addAction(channelAction)
        }

        optionMenu.addAction(cancelAction)
        self.present(optionMenu, animated: true, completion: nil)
    }

    deinit {
        print("profile clear")
    }

    @IBAction func valueChanged(_ sender: UISegmentedControl) {
        let index = sender.selectedSegmentIndex

        switch index {
        case 0:
            changeToPosts()
        case 1:
            changeToBasicInfo()
        default:
            break
        }
    }

    func changeToPosts(){
        tableView.delegate = posts
        tableView.dataSource = posts
        tableView.reloadData()
    }

    func changeToBasicInfo(){
        tableView.delegate = basicInfo
        tableView.dataSource = basicInfo
        basicInfo.userInfo = self.user
        tableView.reloadData()
    }
}

//
//  MyStackView.swift
//  GDproject
//
//  Created by cstore on 14/02/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit

class MyStackView: UIStackView {

    var isFull: Bool = true
```

```
    override var bounds: CGRect {
        didSet{
            if frame.height == 300 && !isFull {
                let gradientLayer = CAGradientLayer()
                gradientLayer.colors = [UIColor(displayP3Red: 255, green:
                    255, blue: 255, alpha: 0).cgColor, UIColor.white.
                    cgColor]
                gradientLayer.startPoint = CGPoint(x: 0.5, y: 0.5)
                gradientLayer.frame = self.bounds
                self.layer.addSublayer(gradientLayer)
            }
        }
    }
}
//
//  Channel.swift
//  GDproject
//
//  Created by cstore on 20/02/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import Foundation

class Channel: Equatable {
    static func == (lhs: Channel, rhs: Channel) -> Bool {
        return (lhs.title == rhs.title &&
                lhs.subtitle == rhs.subtitle &&
                lhs.people == rhs.people &&
                lhs.hashtags == rhs.hashtags)
    }

    var title = String()
    var subtitle = String()

    var hashtags: [String] = []
    var people: [String] = []

    var posts: [Model.Posts] = []

    init(title: String, subtitle: String = "none", hashtags: [String] =
        [], people: [String] = [], posts: [Model.Posts])
    {
        self.title = title
        self.subtitle = subtitle
        self.people = people
        self.hashtags = hashtags
        self.posts = posts
    }
}
//
//  CompletionTree.swift
//  GDproject
//
//  Created by cstore on 01/05/2019.
//  Copyright   2019 drHSE. All rights reserved.
//
```

```
import Foundation


struct CompletionTree: Codable {
    var value: String?
    var subtree: [String : CompletionTree]

    static func getCompletion(tree: CompletionTree, word: String) -> [
       String] {
        if word == "" {
            return getValues(tree: tree)
        }

        let character = String(word.first!)

        if tree.subtree[character] == nil {
            return []
        } else {
            return getCompletion(tree: tree.subtree[character]!, word:
               String(word.dropFirst()))
        }
    }

    static func getValues(tree: CompletionTree) -> [String]
    {
        var out = [String]()

        if let treeVal = tree.value {
            out.append(treeVal)
        }

        for (_, subtree) in tree.subtree {
            out += getValues(tree: subtree)
        }

        return out
    }
}
//
//  Model.swift
//  GDproject
//
//  Created by cstore on 23/02/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import Foundation
import UIKit
import Alamofire

enum ResultR: Int {
    case internalServerError = 500
    case exceededContent = 470
    case longContent = 471
    case incorrectContent = 472
    case impossibleContent = 473
    case invalidTocken = 498
    case alreadyRegistered = 409
    case invalidCode = 403
```

```
    case badAccess = 400
    case tooMuchToAdd = 406
    case notFound = 404
    case success1 = 204
    case success = 200
}

class Model{

    static let invalidTocken = 498

    static var hashTagTree: CompletionTree?

    private static var isValidTocken: ((Int)->())? = { responce in
        print(responce)
        if responce == invalidTocken {

            DataStorage.standard.setIsLoggedIn(value: false, with: 0)
            (UIApplication.shared.delegate as! AppDelegate).relaunch()

            Model.logout {
                print("Logout Is Successful")
            }
        }
    }

    private static let baseUrl = "https://valera-denis.herokuapp.com"
    static let decoder = JSONDecoder()

    static let authMeURL = URL(string: "\(baseUrl)/authentication/me")!
    static let deactivateURL = URL(string: "\(baseUrl)/deactivateAll")!
    static let registerMeURL = URL(string: "\(baseUrl)/authentication/
       register")!
    static let authVerifyURL = URL(string: "\(baseUrl)/authentication/
       verify")!
    static let authenticationURL = URL(string:"\(baseUrl)/authentication/
       login")!
    static let logOutURL = URL(string:"\(baseUrl)/authentication/logout")!
    static let postsLastURL = URL(string:"\(baseUrl)/posts/last")!
    static let postsForUserURL = URL(string:"\(baseUrl)/posts/forUser")!
    static let postsPublishURL = URL(string:"\(baseUrl)/posts/publish")!
    static let usersURL = URL(string:"\(baseUrl)/users")!
    static let usersAllURL = URL(string:"\(baseUrl)/users/all")!
    static let usersUpdateURL = URL(string:"\(baseUrl)/users/update")!
    static let channelsGetURL = URL(string: "\(baseUrl)/channels/get")!
    static let channelsUpdateURL = URL(string: "\(baseUrl)/channels/update
       ")!
    static let channelsListURL = URL(string: "\(baseUrl)/channels")!
    static let channelsCreateURL = URL(string: "\(baseUrl)/channels/create
       ")!
    static let channelsDeleteURL = URL(string: "\(baseUrl)/channels/delete
       ")!
    static let channelsGetAnonURL = URL(string: "\(baseUrl)/channels/
       getAnonymous")!
    static let complexURL = URL(string: "\(baseUrl)/complex")!
    static let hashTagTreeURL = URL(string: "\(baseUrl)/tags/completions")
       !
    static let createGroupChatURL = URL(string: "\(baseUrl)/chats/
       createGroupChat")!
```

```
static let chatsGetAllURL = URL(string: "\(baseUrl)/chats/getAll")!
static let getGroupChatURL = URL(string: "\(baseUrl)/chats/
   getGroupChat")!
static let leaveGroupChatURL = URL(string: "\(baseUrl)/chats/
   leaveGroupChat")!
static let updateGroupChatURL = URL(string: "\(baseUrl)/chats/
   updateGroupChat")!
static let messagesGetGroupChatURL = URL(string: "\(baseUrl)/messages/
   get/groupChat")! //r
static let messagesSendURL = URL(string: "\(baseUrl)/messages/send")!
static let messagesGetUserChatURL = URL(string: "\(baseUrl)/messages/
   get/userChat")!
static let facultySearchURL = URL(string: "\(baseUrl)/faculty/search")
   !


struct QueryPosts<T: Codable>: Codable {
    var users: [Int: Users]
    var response: [T]
}

struct Posts: Codable {

    var body: [Attachments]
    var authorId: Int
    var id: Int
    var user: Model.Users?
    var updated: String
    var tags: [String]

    init(body: [Attachments], authorId: Int, id: Int, date: String,
       tags: [String]) {
        self.body = body
        self.authorId = authorId
        self.id = id
        self.updated = date
        self.tags = tags
    }

    init(body: [Attachments], authorId: Int, id: Int, user: Model.
       Users,  date: String, tags: [String]) {
        self.body = body
        self.authorId = authorId
        self.id = id
        self.user = user
        self.updated = date
        self.tags = tags
    }

    func convertDateFormatter() -> String
    {

        let dateFormatter = DateFormatter()
        dateFormatter.dateFormat = "yyyy-MM-dd'T'HH:mm:ss.SSSZ"//this
           your string date format
        dateFormatter.timeZone = TimeZone(abbreviation: "UTC")
        let date = dateFormatter.date(from: updated)
```

```
            dateFormatter.dateFormat = "MMM d, yyyy HH:mm" ///this is what
                you want to convert format
            dateFormatter.timeZone = NSTimeZone.local
            let timeStamp = dateFormatter.string(from: date!)

            return timeStamp
        }
    }

    struct CreatedPost: Codable {
        var body = [Attachments]()
        var tags = [String]()

        enum CodingKeys: String, CodingKey {
            case body
            case tags
        }

        func encode(to encoder: Encoder) throws {
            var container = encoder.container(keyedBy: CodingKeys.self)
            try container.encode(body, forKey: .body)
            try container.encode(tags, forKey: .tags)
        }
    }

    struct Users: Codable{

        var middleName: String
        var lastName: String
        var firstName: String
        var id: Int
        var faculty: Faculty
        var email: String

        func fullName() -> String {
            return "\(firstName) \(lastName)"
        }
    }

    struct Attachments: Codable {
        var markdown: String

        init(markdown: String) {
            self.markdown = markdown
        }

        enum CodingKeys: String, CodingKey {
            case markdown
        }

        func encode(to encoder: Encoder) throws {
            var container = encoder.container(keyedBy: CodingKeys.self)
            try container.encode(markdown, forKey: .markdown)
        }
    }

    struct Channels: Codable {

        // static var fullTags = Set<String>()
```

```
    static var fullPeople = [Users]()
    static var fullPeopleDict = [Int:Users]()

    var people: [Int]
    var name: String
    var id: Int?
    var tags: [String]

    init(people: [Int], name: String, id: Int, tags: [String]) {
        self.id = id
        self.people = people
        self.tags = tags
        self.name = name
    }

    init(people: [Int], name: String, tags: [String]) {
        self.people = people
        self.tags = tags
        self.name = name
    }

    enum CodingKeys: String, CodingKey {
        case people
        case name
        case id
        case tags
    }

    func encode(to encoder: Encoder) throws {
        var container = encoder.container(keyedBy: CodingKeys.self)
        try container.encode(people, forKey: .people)
        try container.encode(name, forKey: .name)
        try container.encode(id, forKey: .id)
        try container.encode(tags, forKey: .tags)
    }
}

struct AuthStatus: Codable {
    var userStatus: String

    init(){
        userStatus = "invalid"
    }
}

static func logout(completion: @escaping (()->())){
    var request = URLRequest(url: logOutURL)
    request.httpMethod = "POST"

    AF.request(request).response { (responce) in
        isValidTocken?(responce.response?.statusCode ?? 498)

        if let code = responce.response?.statusCode, code == 498 {
            return
        }

        completion()
    }
}
```

```
/*
 "email": "vchernyshev@hse.ru",
 "middleName": "Algebrovich",
 "lastName": "Leonidov",
 "faculty": "cs.hse.ru/dse",
 "firstName": "Seva"
 */

struct NewRegistration: Codable{
    var email: String
    var firstName: String?
    var middleName: String?
    var lastName: String?
    var faculty: String?
}

static func register(object: NewRegistration, completion: @escaping
   (()->())) {
    var request = URLRequest(url: registerMeURL)
    request.httpBody = try? JSONEncoder().encode(object)

    request.httpMethod = "POST"
    request.setValue("application/json; charset=utf-8",
       forHTTPHeaderField: "Content-Type")

    AF.request(request).response { (responce) in

        guard let code = responce.response?.statusCode else { return }
        isValidTocken?(code)

        if let code = responce.response?.statusCode, code == 498 {
            return
        }

        if code == 204 {

            let fields = responce.response?.allHeaderFields as? [
               String :String]
            let cookies = HTTPCookie.cookies(withResponseHeaderFields:
                fields!, for: responce.response!.url!)
            HTTPCookieStorage.shared.setCookie(cookies[0])

            completion()
        }
    }
}

// true only if everything is good
static func verify(with code: Int, completion: @escaping ((Bool)->()))
   {
    var request = URLRequest(url: authVerifyURL)
    request.httpMethod = "POST"
    request.httpBody = "\(code)".data(using: .utf8)
    request.setValue("application/json; charset=utf-8",
       forHTTPHeaderField: "Content-Type")

    AF.request(request).response { (responce) in
```

```
            guard let statusCode = responce.response?.statusCode else {
               completion(false); return }
            isValidTocken?(statusCode)

            if let code = responce.response?.statusCode, code == 498 {
                return
            }

            if  statusCode == 204 {
                // at this point cookies are set
                completion(true)
            } else {
                completion(false)
            }
        }
    }

    // yes or no
    static func authenticateMe(completion: @escaping ((Bool)->())){
        var  request = URLRequest(url: authMeURL)
        request.httpMethod = "POST"

        AF.request(request).response { (responce) in

            guard let statusCode = responce.response?.statusCode else {
               completion(false); return }
            isValidTocken?(statusCode)

            if let code = responce.response?.statusCode, code == 498 {
                return
            }

            guard let json = responce.data else { return }
            guard let personId = Int(String(data: json, encoding: String.
               Encoding.utf8)!) else {
                completion(false)
                return
            }

            DataStorage.standard.setIsLoggedIn(value: true, with: personId
               )
            completion(true)
        }
    }

    // register, ok or invalid
    static func authenticate(with email: String, completion: @escaping ((
       AuthStatus)->())) {

        let json: [String:Any] = ["authenticationEmail" : email]
        let jsonData = try? JSONSerialization.data(withJSONObject: json)

        var request = URLRequest(url: authenticationURL)
        request.httpMethod = "POST"

        // insert json data to the request
        request.httpBody = jsonData
        request.setValue("application/json; charset=utf-8",
           forHTTPHeaderField: "Content-Type")
```

```
        AF.request(request).response { (response) in

            guard let realResponse = response.response, realResponse.
               statusCode == 200 else
            {
                print("Not a 200 response")
                completion(AuthStatus())
                return
            }

            guard let json = response.data else { return }
            guard let res = try? decoder.decode(AuthStatus.self, from:
               json) else { return }

            if  realResponse.statusCode == 200 && res.userStatus == "
               registered"
            {
                // at this point cookies are set
                let fields = realResponse.allHeaderFields as? [String :
                   String]
                let cookies = HTTPCookie.cookies(withResponseHeaderFields:
                    fields!, for: realResponse.url!)
                HTTPCookieStorage.shared.setCookie(cookies[0])
            }

            completion(res)
        }
    }

    struct GeneralRequest<T: Codable>: Codable
    {
        var direction: String
        var limit: Int
        var exclusiveFrom: Int?
        var request: T

        init(direction: String = "backward", limit: Int, exclusiveFrom:
           Int?, request: T) {
            self.direction = direction
            self.limit = limit
            self.exclusiveFrom = exclusiveFrom
            self.request = request
        }

        enum CodingKeys: String, CodingKey {
            case limit
            case exclusiveFrom
            case request
            case direction
        }

        func encode(to encoder: Encoder) throws {
            var container = encoder.container(keyedBy: CodingKeys.self)
            try container.encode(limit, forKey: .limit)
            try container.encode(exclusiveFrom, forKey: .exclusiveFrom)
            try container.encode(request, forKey: .request)
            try container.encode(direction, forKey: .direction)
        }
```

```
    }

    static func getLast(on limit: Int = 10, from pointInTime: Int? = nil,
       completion: @escaping (((users:[Int: Users], posts:[Posts]))->()))
    {
        let postRequest = GeneralRequest<[Int]>(limit: limit,
           exclusiveFrom: pointInTime, request: [])

        var request = URLRequest(url: postsLastURL)
        request.httpBody = try? JSONEncoder().encode(postRequest)
        request.httpMethod = "POST"

        // insert json data to the request
        request.setValue("application/json; charset=utf-8",
           forHTTPHeaderField: "Content-Type")

        AF.request(request).responseJSON {
            (response) in

            isValidTocken?(response.response?.statusCode ?? 498)

            if let code = response.response?.statusCode, code == 498 {
                return
            }

            guard let json = response.data else { return }

            guard let newQueery = try? decoder.decode(QueryPosts<Posts>.
               self, from: json) else { print("no")
                return }

            idUser = newQueery.users
            completion((newQueery.users, newQueery.response))
        }
    }


    static func createAndPublish(body: [Attachments], tags: [String],
       completion: @escaping ((ResultR)->())){
        let jsonUpd = CreatedPost(body: body, tags: tags)
        var request = URLRequest(url: postsPublishURL)

        request.httpMethod = "POST"

        // insert json data to the request
        request.httpBody = try? JSONEncoder().encode(jsonUpd)

        request.setValue("application/json; charset=utf-8",
           forHTTPHeaderField: "Content-Type")
        AF.request(request).response {
            (response) in

            isValidTocken?(response.response?.statusCode ?? 498)

            if let code = response.response?.statusCode, code == 498 {
                return
            }

            if let code = response.response?.statusCode,
```

```
                let result =  ResultR(rawValue: code)
            {
                completion(result)
            } else {
                completion(.internalServerError)
            }
        }
    }

//    static func requestForPosts(limit: Int = 10, ){
//
//    }

    static func getPostsForUser(for limit: Int = 10, from pointInTime: Int
       ? = nil, with id: Int, completion: @escaping (([Posts])->()))
    {
        let postsRequest = GeneralRequest<Int>(limit: limit, exclusiveFrom
           : pointInTime, request: id)

        let jsonData = try? JSONEncoder().encode(postsRequest)

        var request = URLRequest(url: postsForUserURL)
        request.httpMethod = "POST"

        // insert json data to the request
        request.httpBody = jsonData
        request.setValue("application/json; charset=utf-8",
           forHTTPHeaderField: "Content-Type")

        AF.request(request).responseJSON {
            (response) in

            isValidTocken?(response.response?.statusCode ?? 498)

            if let code = response.response?.statusCode, code == 498 {
                return
            }

            guard let json = response.data else { return }

            guard let newPost = try? decoder.decode(QueryPosts<Posts>.self
               , from: json) else {  return }

            completion(newPost.response)
        }
    }

    static var idUser: [Int:Users] = [:]

    static func getUsers(for ids: [Int], completion: @escaping (([Int:
       Users])->())){
        let json = "\(Set(ids))"

        var request = URLRequest(url: usersURL)

        request.httpMethod = "POST"

        // insert json data to the request
        request.httpBody = json.data(using: .utf8)
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.03-01 ТЗ 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

```
        request.setValue("application/json; charset=utf-8",
           forHTTPHeaderField: "Content-Type")

        AF.request(request).responseJSON {
            (response) in

            isValidTocken?(response.response?.statusCode ?? 498)

            if let code = response.response?.statusCode, code == 498 {
                return
            }

            guard let json = response.data else { return }

            guard let users = try? decoder.decode([Users].self, from: json
               ) else {  return }

            var dict: [Int:Users] = [:]

            users.forEach({ (user) in
                dict[user.id] = user
            })

            completion(dict)
        }
    }

    // get channel (with id): in responce -- PostQuery
    static func getChannel(with channelId: Int, on limit: Int = 10, from
       pointInTime: Int? = nil, completion: @escaping (((users:[Int: Users
       ], posts:[Posts]))->()))
    {
        let postRequest = GeneralRequest<Int>(limit: limit, exclusiveFrom:
            pointInTime, request: channelId)
        let jsonData = try? JSONEncoder().encode(postRequest)

        var request = URLRequest(url: channelsGetURL)
        request.httpMethod = "POST"

        // insert json data to the request
        request.httpBody = jsonData
        request.setValue("application/json; charset=utf-8",
           forHTTPHeaderField: "Content-Type")

        AF.request(request).responseJSON {
            (response) in

            isValidTocken?(response.response?.statusCode ?? 498)

            if let code = response.response?.statusCode, code == 498 {
                return
            }

            guard let json = response.data else { return }

            guard let newQueery = try? decoder.decode(QueryPosts<Posts>.
               self, from: json) else {  return }

            idUser = newQueery.users
```

```
            completion((newQueery.users, newQueery.response))
        }
    }


    static func createChannel(with channel: Channels, completion:
       @escaping ((ResultR)->())) {

        var request = URLRequest(url: channelsCreateURL)
        request.httpMethod = "POST"
        request.httpBody = try? JSONEncoder().encode(channel)

        request.setValue("application/json; charset=utf-8",
           forHTTPHeaderField: "Content-Type")
        AF.request(request).response {
            (response) in

            isValidTocken?(response.response?.statusCode ?? 498)

            if let code = response.response?.statusCode, let result =
               ResultR(rawValue: code) {
                completion(result)
                return
            }

            completion(ResultR.internalServerError)
        }

    }

    static func channelsList(completion: @escaping (([Channels])->())){
        var request = URLRequest(url: channelsListURL)
        request.httpMethod = "POST"
        request.setValue("application/json; charset=utf-8",
           forHTTPHeaderField: "Content-Type")

        AF.request(request).responseJSON { (response) in

            isValidTocken?(response.response?.statusCode ?? 498)
            if let code = response.response?.statusCode, code == 498 {
                return
            }

            guard let json = response.data else { return }

            guard let channelsList = try? decoder.decode([Channels].self,
               from: json) else { return }

            completion(channelsList)
        }
    }

    static func channelsDelete(by id: Int, completion: @escaping(()->())){
        var request = URLRequest(url: channelsDeleteURL)
        request.httpMethod = "POST"
        request.httpBody = "\(id)".data(using: .utf8)
        print("\(id)")
        request.setValue("application/json; charset=utf-8",
           forHTTPHeaderField: "Content-Type")
```

```
        AF.request(request).response { (response) in
            isValidTocken?(response.response?.statusCode ?? 498)
            if let code = response.response?.statusCode, code == 498 {
                return
            }
        }
        completion()
    }

    static func updateChannel(with channel: Channels) {

        var request = URLRequest(url: channelsUpdateURL)
        request.httpMethod = "POST"
        request.httpBody = try? JSONEncoder().encode(channel)

        request.setValue("application/json; charset=utf-8",
           forHTTPHeaderField: "Content-Type")
        AF.request(request).response {
            (response) in

            isValidTocken?(response.response?.statusCode ?? 498)
            if let code = response.response?.statusCode, code == 498 {
                return
            }
        }
    }

    static func usersAllGet() {
        var request = URLRequest(url: usersAllURL)
        request.httpMethod = "POST"

        request.setValue("application/json; charset=utf-8",
           forHTTPHeaderField: "Content-Type")
        AF.request(request).response {
            (response) in
            guard let json = response.data else { return }

            guard let users = try? decoder.decode([Users].self, from: json
               ) else { return }

            Channels.fullPeople = users
            Channels.fullPeopleDict = users.reduce([Int: Users]()) { (dict
               , person) -> [Int: Users] in
                var dict = dict
                dict[person.id] = person
                return dict
            }

            isValidTocken?(response.response?.statusCode ?? 498)
            if let code = response.response?.statusCode, code == 498 {
                return
            }
        }
    }

    struct AnonymousChannel: Codable {

        var limit = 10
```

```
        var request: RequestPeopleTags

        enum CodingKeys: String, CodingKey {
            case limit
            case request
        }

        init(people: [Int], tags: [String]) {
            self.request = RequestPeopleTags(people: people, tags: tags)
        }

        func encode(to encoder: Encoder) throws {
            var container = encoder.container(keyedBy: CodingKeys.self)
            try container.encode(limit, forKey: .limit)
            try container.encode(request, forKey: .request)
        }

        struct RequestPeopleTags: Codable {

            var people: [Int]
            var tags: [String]

            init(people: [Int], tags: [String]) {
                self.people = people
                self.tags = tags
            }

            func encode(to encoder: Encoder) throws {
                var container = encoder.container(keyedBy: CodingKeys.self
                    )
                try container.encode(people, forKey: .people)
                try container.encode(tags, forKey: .tags)
            }

            enum CodingKeys: String, CodingKey {
                case people
                case tags
            }

        }
    }
    /*
     {
     "limit": 11,
     "request": {
         "people": [
                2,
                7,
                8
            ],
         "tags": [
                "thisIsHashTag",
                "thisIsAlsoHashTag"
            ]
         }
     }
     */
    static func getAnonymousChannel(by anonymousChannel: Model.Channels,
       exclusiveFrom: Int? = nil, completion: @escaping (((users:[Int:
```

```
    Users], posts:[Posts]))->()){

        let req = GeneralRequest<Model.Channels>(limit: 10, exclusiveFrom:
            exclusiveFrom, request: anonymousChannel)

        var request = URLRequest(url: channelsGetAnonURL)
        request.httpMethod = "POST"
        request.httpBody = try? JSONEncoder().encode(req)

        request.setValue("application/json; charset=utf-8",
           forHTTPHeaderField: "Content-Type")
        AF.request(request).response {
            (response) in

            isValidTocken?(response.response?.statusCode ?? 498)
            if let code = response.response?.statusCode, code == 498 {
                return
            }

            guard let json = response.data else { return }

            guard let newQueery = try? decoder.decode(QueryPosts<Posts>.
               self, from: json) else {  return }
            completion((newQueery.users, newQueery.response))
        }
    }

    static func getCompl(completion: @escaping ((CompletionTree)->())) {

        AF.request(URLRequest(url: hashTagTreeURL)).responseJSON {
            (response) in
            isValidTocken?(response.response?.statusCode ?? 498)
            if let code = response.response?.statusCode, code == 498 {
                return
            }
            guard let json = response.data else { return }
            guard let tree = try? decoder.decode(CompletionTree.self, from
               : json) else {  return }
            completion(tree)
        }
    }

    static func getChatAll(limit: Int = 10, exclusiveFrom: Int? = nil,
       request: [Int] = [], completion: @escaping ((([Dialog],[Int:Model.
       Users]))->()))
    {
        let req = GeneralRequest<[Int]>(limit: limit, exclusiveFrom:
           exclusiveFrom, request: request)
        var request = URLRequest(url: chatsGetAllURL)
        request.httpMethod = "POST"
        request.httpBody = try? JSONEncoder().encode(req)
        request.setValue("application/json; charset=utf-8",
           forHTTPHeaderField: "Content-Type")

        AF.request(request).responseJSON { response in
            isValidTocken?(response.response?.statusCode ?? 498)

            if let code = response.response?.statusCode, code == 498 {
                return
```

```
        }

        guard let json = response.data else { return }
        let dialogs = try! decoder.decode(QueryPosts<Dialog>.self,
            from: json)

        completion((dialogs.response,dialogs.users))
    }
}

enum Dialog: Codable
{
    case groupChat(GroupChat)
    case userChat(UserChat)

    private enum DialogKeys: CodingKey{
        case groupChat
        case userChat
    }

    init(from decoder: Decoder) throws {
        let container = try decoder.container(keyedBy: DialogKeys.self
            )
        if container.contains(.groupChat){
            self = .groupChat(try GroupChat(from: container.
                superDecoder(forKey: .groupChat)))
        } else if container.contains(.userChat){
            self = .userChat(try UserChat(from: container.superDecoder
                (forKey: .userChat)))
        } else {
            throw DecodingError.keyNotFound(DialogKeys.groupChat,
                DecodingError.Context(codingPath: container.codingPath,
                 debugDescription: "hz"))
        }
    }

    func encode(to encoder: Encoder) throws {
        var container = encoder.container(keyedBy: DialogKeys.self)
        switch self{
        case .groupChat(let chat):
            try chat.encode(to: container.superEncoder(forKey: .
                groupChat))
        case .userChat(let chat):
            try chat.encode(to: container.superEncoder(forKey: .
                userChat))
        }
    }
}

struct GroupChat: Codable {
    var group: Group
    var lastMessage: LastMessage?


    init(group: Group, lastMessage: LastMessage? = nil) {
        self.group = group
        self.lastMessage = lastMessage
    }
}
```

```
struct UserChat: Codable {
    var user: Int
    var lastMessage: LastMessage?

    init(user: Int, lastMessage: LastMessage? = nil) {
        self.user = user
        self.lastMessage = lastMessage
    }
}

struct Group: Codable {
    var users: [Int: UserPermission]
    var name: String
    var id: Int

    init(users: [Int: UserPermission] = [:], name: String = "", id:
       Int) {
        self.id = id
        self.users = users
        self.name = name
    }
}

struct UserPermission: Codable {
    var isAdmin: Bool
}

struct LastMessage: Codable {
    var body: Attachments
    var destination: MessageDestination
    var time: String
    var author: Int
    var id: Int

    enum MessageCodingKeys: CodingKey{
        case user
        case group
        case body
        case time
        case author
        case id
    }

    init(from decoder: Decoder) throws
    {
        let container = try decoder.container(keyedBy:
           MessageCodingKeys.self)
        time = try container.decode(String.self, forKey: .time)
        body = try container.decode(Attachments.self, forKey: .body)
        author = try container.decode(Int.self, forKey: .author)
        id = try container.decode(Int.self, forKey: .id)

        if container.contains(.user){
            destination = MessageDestination.userChatDestination(try
               Int(from: container.superDecoder(forKey: .user)))
        } else if container.contains(.group){
            destination = MessageDestination.groupChatDestination(try
               Int(from: container.superDecoder(forKey: .group)))
```

```
            } else {
                throw DecodingError.keyNotFound(MessageCodingKeys.group,
                    DecodingError.Context(codingPath: container.codingPath,
                     debugDescription: "hz gr"))
            }
        }

        func encode(to encoder: Encoder) throws {
            var container = encoder.container(keyedBy: MessageCodingKeys.
                self)

            try container.encode(time, forKey: .time)
            try container.encode(id, forKey: .id)
            try container.encode(author, forKey: .author)
            try container.encode(body, forKey: .body)

            switch destination {
            case .userChatDestination(let uId):
                try uId.encode(to: container.superEncoder(forKey: .user))
            case .groupChatDestination(let gId):
                try gId.encode(to: container.superEncoder(forKey: .group))
            }
        }
    }

    enum MessageDestination: Codable
    {
        func encode(to encoder: Encoder) throws
        {
            var container = encoder.container(keyedBy: MessCodingKeys.self
                )
            switch self {
            case .userChatDestination(let uId):
                try uId.encode(to: container.superEncoder(forKey: .user))
            case .groupChatDestination(let gId):
                try gId.encode(to: container.superEncoder(forKey: .group))
            }
        }

        init(from decoder: Decoder) throws
        {
            let container = try decoder.container(keyedBy: MessCodingKeys.
                self)
            if container.contains(.user){
                self = .userChatDestination(try Int(from: container.
                    superDecoder(forKey: .user)))
            } else if container.contains(.group){
                self = .groupChatDestination(try Int(from: container.
                    superDecoder(forKey: .group)))
            } else {
                throw DecodingError.keyNotFound(MessCodingKeys.group,
                    DecodingError.Context(codingPath: container.codingPath,
                     debugDescription: "hz gr"))
            }
        }

        enum MessCodingKeys: CodingKey{
            case user
            case group
```

```
        }

        case userChatDestination(Int)
        case groupChatDestination(Int)

    }

    static func createGroupChat(from group: Group, completion: @escaping
       ((Int)->())) {
        let req = group
        var request = URLRequest(url: createGroupChatURL)
        request.httpMethod = "POST"
        request.httpBody = try? JSONEncoder().encode(req)
        request.setValue("application/json; charset=utf-8",
           forHTTPHeaderField: "Content-Type")

        AF.request(request).response { (response) in

            isValidTocken?(response.response?.statusCode ?? 498)
            if let code = response.response?.statusCode, code == 498 {
                return
            }

            guard let json = response.data else { return }
            let stringInt = String.init(data: json, encoding: String.
               Encoding.utf8)
            let intId = Int.init(stringInt!)

            completion(intId!)
        }
    }

    static func getMessagesFor(typeOfChat: Model.Dialog, chat id: Int,
       exclusiveFrom: Int? = nil, limit l: Int = 10, direction: String = "
       backward", completion: @escaping (([LastMessage])->()))
    {
        let req = GeneralRequest<Int>(direction: direction, limit: l,
           exclusiveFrom: exclusiveFrom, request: id)
        var request: URLRequest?

        switch typeOfChat {
        case .groupChat:
            request = URLRequest(url: messagesGetGroupChatURL)
        case .userChat:
            request = URLRequest(url: messagesGetUserChatURL)
        }

        request!.httpMethod = "POST"
        request!.httpBody = try? JSONEncoder().encode(req)
        request!.setValue("application/json; charset=utf-8",
           forHTTPHeaderField: "Content-Type")

        AF.request(request!).response { (response) in

            isValidTocken?(response.response?.statusCode ?? 498)
            if let code = response.response?.statusCode, code == 498 {
                return
            }
```

```
            guard let json = response.data else { return }
            guard let messages = try? decoder.decode([LastMessage].self,
               from: json) else {  return }

            completion(messages)
        }
    }

    static func leaveGroupChat(id: Int, completion: @escaping (()->()))
    {
        var request = URLRequest(url: leaveGroupChatURL)
        request.httpMethod = "POST"
        request.httpBody = "\(id)".data(using: .utf8)
        request.setValue("application/json; charset=utf-8",
           forHTTPHeaderField: "Content-Type")

        AF.request(request).response { (response) in
            isValidTocken?(response.response?.statusCode ?? 498)

            if let code = response.response?.statusCode, code == 498 {
                return
            }
        }

        completion()
    }

    static func updateGroupChat(with group: Model.Group)
    {
        var request = URLRequest(url: updateGroupChatURL)
        request.httpMethod = "POST"
        request.httpBody = try? JSONEncoder().encode(group)
        request.setValue("application/json; charset=utf-8",
           forHTTPHeaderField: "Content-Type")

        AF.request(request).response { (response) in
            isValidTocken?(response.response?.statusCode ?? 498)

            if let code = response.response?.statusCode, code == 498 {
                return
            }
        }

        // completion()
    }

    struct SendMessage: Codable {
        var body: Attachments
        var destination: MessageDestination

        init(from decoder: Decoder) throws
        {
            let container = try decoder.container(keyedBy: LastMessage.
               MessageCodingKeys.self)
            body = try container.decode(Attachments.self, forKey: .body)

            if container.contains(.user){
                destination = MessageDestination.userChatDestination(try
                   Int(from: container.superDecoder(forKey: .user)))
```

```
            } else if container.contains(.group){
                destination = MessageDestination.groupChatDestination(try
                    Int(from: container.superDecoder(forKey: .group)))
            } else {
                throw DecodingError.keyNotFound(LastMessage.
                    MessageCodingKeys.group, DecodingError.Context(
                    codingPath: container.codingPath, debugDescription: "hz
                     gr1"))
            }
        }

        func encode(to encoder: Encoder) throws {
            var container = encoder.container(keyedBy: LastMessage.
                MessageCodingKeys.self)
            try container.encode(body, forKey: .body)

            switch destination {
            case .userChatDestination(let uId):
                try uId.encode(to: container.superEncoder(forKey: .user))
            case .groupChatDestination(let gId):
                try gId.encode(to: container.superEncoder(forKey: .group))
            }
        }

        init(body: Attachments, destination: MessageDestination)
        {
            self.body = body
            self.destination = destination
        }
    }

    static func sendMessage(message: SendMessage, completion: @escaping ((
        ResultR)->())){

        var request = URLRequest(url: messagesSendURL)
        request.setValue("application/json; charset=utf-8",
            forHTTPHeaderField: "Content-Type")
        request.httpMethod = "POST"
        request.httpBody = try? JSONEncoder().encode(message)

        AF.request(request).response { (response) in
            isValidTocken?(response.response?.statusCode ?? 498)

            if let code = response.response?.statusCode, code == 498 {
                return
            }

            if let result = ResultR(rawValue: response.response!.
                statusCode){
                completion(result)
            } else {
                completion(.internalServerError)
            }
        }
    }

    struct Faculty: Codable
    {
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.03-01 ТЗ 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

```
        var path: String
        var url: String
        var campusName: String
        var campusCode: String
        var name: String
        var tags: [String]
        var address: String

    }

    static func searchFaculty(string: String, completion: @escaping (([
       Model.Faculty])->()))
    {
        var request = URLRequest(url: facultySearchURL)
        request.httpMethod = "POST"
        request.httpBody = string.data(using: .utf8)
        request.setValue("text/plain;charset=utf-8", forHTTPHeaderField: "
           Content-Type")

        AF.request(request).response { (response) in
            isValidTocken?(response.response?.statusCode ?? 498)

            if let code = response.response?.statusCode, code == 498 {
                return
            }

            guard let json = response.data else { return }
            guard let faculties = try? decoder.decode([Faculty].self, from
               : json) else {  return }

            completion(faculties)
        }
    }

    static func userUpdate(with newUser: NewRegistration, completion:
       @escaping ((Bool)->())) {
        var request = URLRequest(url: usersUpdateURL)
        request.setValue("application/json; charset=utf-8",
           forHTTPHeaderField: "Content-Type")
        request.httpMethod = "POST"
        request.httpBody = try? JSONEncoder().encode(newUser)

        AF.request(request).response { (response) in
            isValidTocken?(response.response?.statusCode ?? 498)

            if let code = response.response?.statusCode, code == 498 {
                return
            }

            if let code = response.response?.statusCode, code != 204{
                completion (false)
            } else {
                completion(true)
            }
        }
    }

    static func deactivateAll(completion: @escaping (()->())){
        var request = URLRequest(url: deactivateURL)
```

```
        request.httpMethod = "DELETE"

        AF.request(request).response { (responce) in
            isValidTocken?(responce.response?.statusCode ?? 498)

            if let code = responce.response?.statusCode, code == 498 {
                return
            }

            completion()
        }
    }
}
//
//  Post.swift
//  NewsFeed
//
//  Created by cstore on 12/01/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import Foundation
import UIKit

enum Media{
    case text(String)
    case image([UIImage])
    case quote
}

struct Comment {
    var atTime: String
    var withData: String
}
//
//  AppDelegate.swift
//  GDproject
//
//  Created by cstore on 13/02/2019.
//  Copyright   2019 drHSE. All rights reserved.
//

import UIKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {


    var window: UIWindow?
    var appCoordinator:  ApplicationCoordinator!


    func application(_ application: UIApplication,
       didFinishLaunchingWithOptions launchOptions: [UIApplication.
       LaunchOptionsKey: Any]?) -> Bool
    {
        window = UIWindow(frame: UIScreen.main.bounds)
        window?.makeKeyAndVisible()
```

```
        // window?.rootViewController = UINavigationController()
        Model.getCompl { (complTree) in
            Model.hashTagTree = complTree
        }

        appCoordinator = ApplicationCoordinator(window: window!)
        appCoordinator.start()

        return true
    }

    func relaunch(){
        appCoordinator.start()
    }
}
```

Листинг 1 — Текст программы

Лист регистрации изменений									
Изм.	Номера листов				Всего листов в документе	№ документа	Входящий № сопроводит. докум. и дата	Подпись	Дата
	измененных	замененных	новых	аннулированных					