**Criterion C- Development**

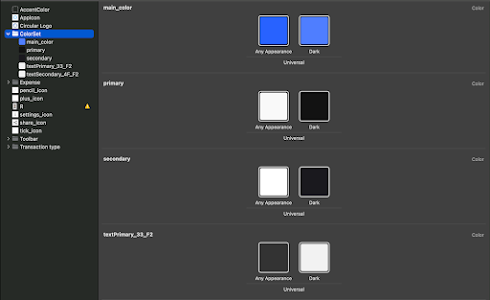
**List of techniques:**

| **Simple** | **Complex** |
| --- | --- |
| Sorting functions: different filtering criteria that could be used to sort transactions. | Searching for specific data in a file |
| Searching function: type a request to search expenses according to the title, date, and place. | Database accessibility that saves transaction information from the transaction list into the phone SSD memory |
| Simple and complex selections: if and else condition and multiple if structures.+ for and loop structures | Circle Graphs and diagrams that could help in analyzing data |
| Using additional libraries. Icon library from GitHub to bring icons to transaction + graph library to connect Transaction List to graphs to analyze it | Sharing functions that are necessary to sent transactions via emails, messages etc. to other people in pdf file |
| Input function.Users could type their expenses into the app | by Merging two or more sorted data structures such as float and int data in a transaction title |
| User-defined methods with appropriate return values (primitives or objects) | Income and expenses categories that could work together or are independent of each other. |
| Beautiful design that makes this application comfortable to use | An authentication security system that will recognize Touch ID, phone password and Face ID |

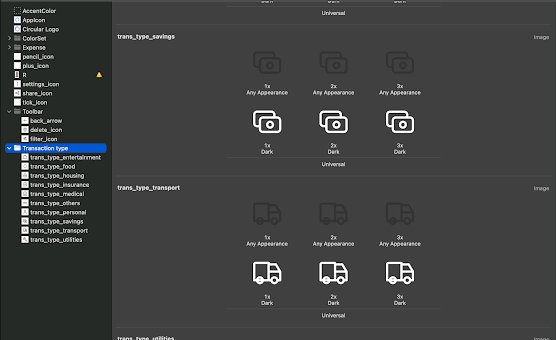
**Stages of app development:**

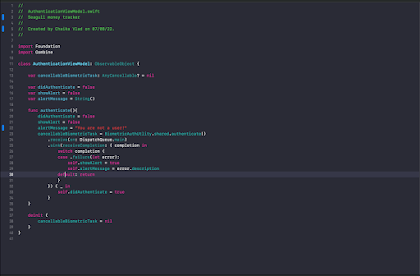
1. In the beginning, I created the App project in the Xcode software and named it “Seagull money tracker”.After creating the project, I started working with visual effects. First, I added 18 sizes of logos for my application (for different devices).

After the logos, I started working on the colour palette of the application, the main colours of which are blue shades. The primary, secondary and text colours will be black and white to maintain colour restraint.

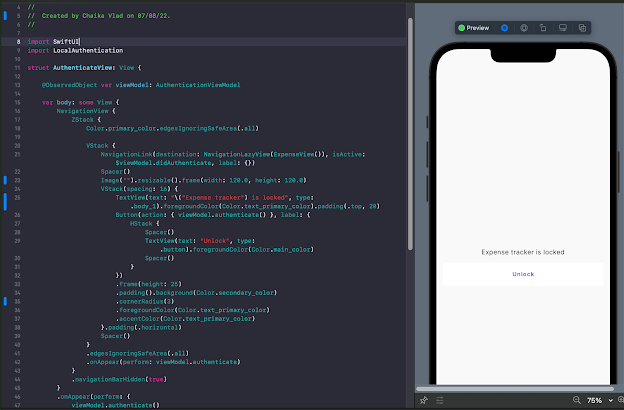


Working with the assets of my application, I decided to instantly install the library of icons from Apple, as well as icons from other authors for a more interesting design of my application.





1. My application will consist of several pages, each of which needs separate development and design. From the very beginning, the user will be greeted by a welcome page that reads the user's biometrics. If the application recognizes the client's face, it will go to the main page. If not, the application will be blocked. To recreate authentication, I watched a YouTube video and write the code in a screen view “AuthenticateView” and “AuthenticateViewModel”.



1. Biometrical code from YouTube[[1]](#footnote-0) is:

import Foundation

import Combine

import LocalAuthentication

struct BiometericAuthError: LocalizedError {

var description: String

init(description: String){

self.description = description

}

init(error: Error){

self.description = error.localizedDescription

}

var errorDescription: String?{

return description

}

}

class BiometricAuthUtlity {

static let shared = BiometricAuthUtlity()

private init(){}

/// Authenticate the user with device Authentication system.

/// If the .deviceOwnerAuthenticationWithBiometrics is not available, it will fallback to .deviceOwnerAuthentication

/// - **Returns**: future which passes `Bool` when the authentication suceeds or `BiometericAuthError` when failed to authenticate

public func authenticate() -> Future<Bool, BiometericAuthError> {

Future { promise in

func handleReply(success: Bool, error: Error?) -> Void {

if let error = error {

return promise(

.failure(BiometericAuthError(error: error))

)

}

promise(.success(success))

}

let context = LAContext()

var error: NSError?

let reason = "Please authenticate yourself to unlock \(APP\_NAME)"

if context.canEvaluatePolicy(.deviceOwnerAuthenticationWithBiometrics, error: &error) {

context.evaluatePolicy(.deviceOwnerAuthenticationWithBiometrics, localizedReason: reason, reply: handleReply)

} else if context.canEvaluatePolicy(.deviceOwnerAuthentication, error: &error) {

// fallback

context.evaluatePolicy(.deviceOwnerAuthentication, localizedReason: reason, reply: handleReply)

}else{

//cannot evaluate

let error = BiometericAuthError(description: "Something went wrong while authenticating. Please try again")

promise(.failure(error))

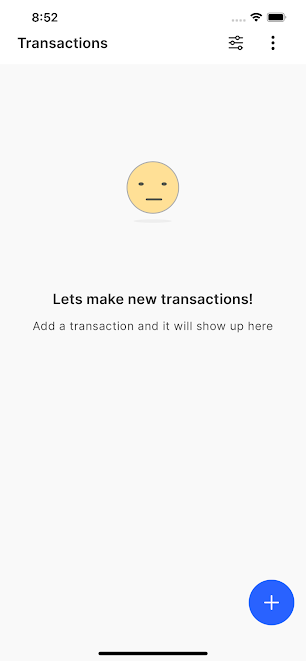
}

}

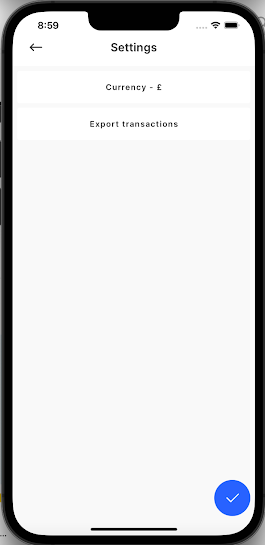
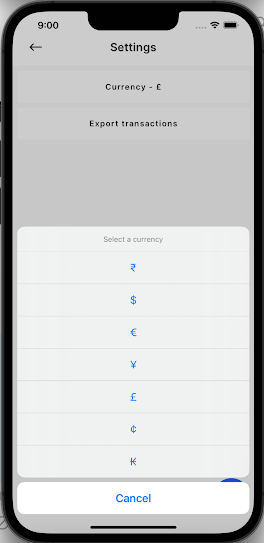
}

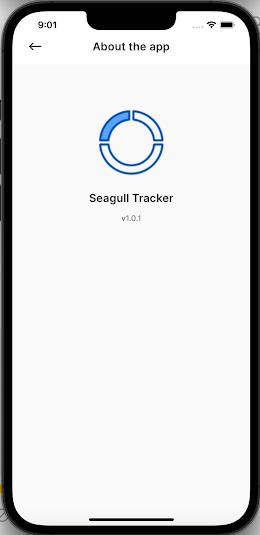
}

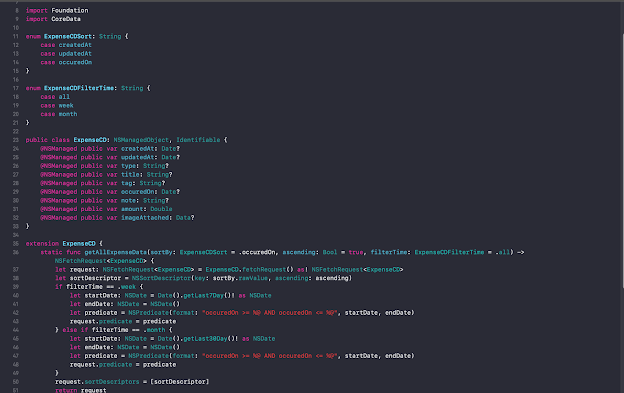
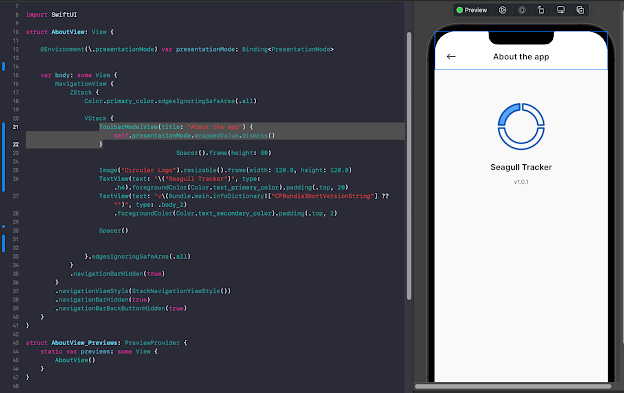
1. I'll start with the main page. An empty main page called “transactions: is a screen where the user is greeted by a model of an animated man's face and text that informs the user to add transactions.



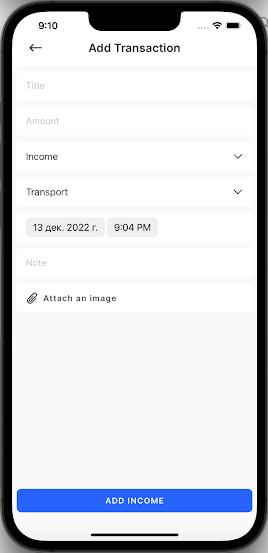
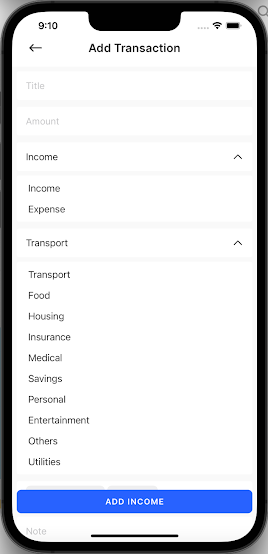
The icon with dots in the upper right corner allows you to view information and select 2 more pages: about us, which will present the name and the latest version of the application, as well as the settings tab where you can change the currency (7 in total) and the ability to share transactions as a pdf document.

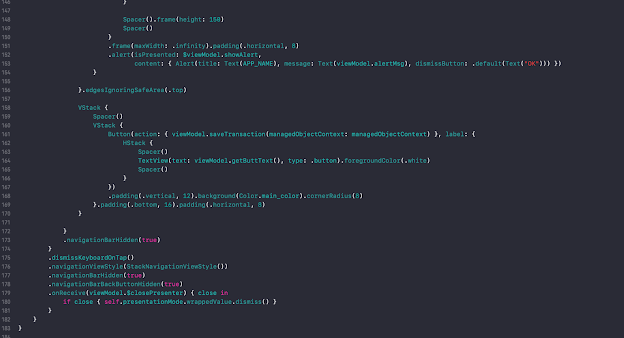
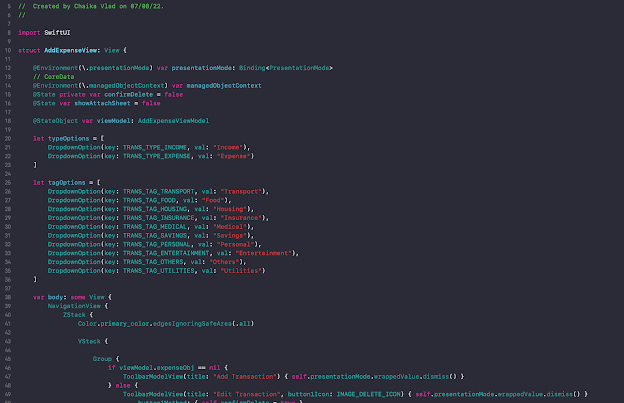


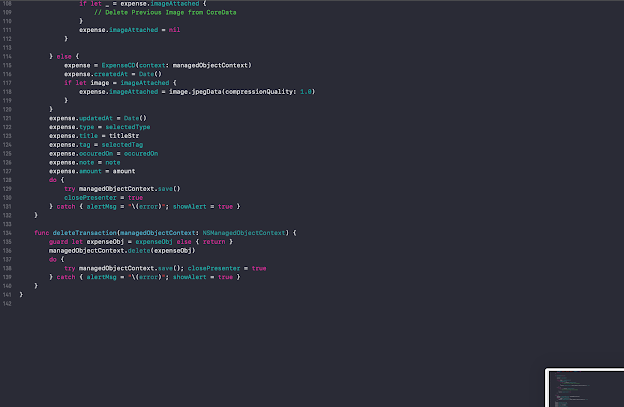
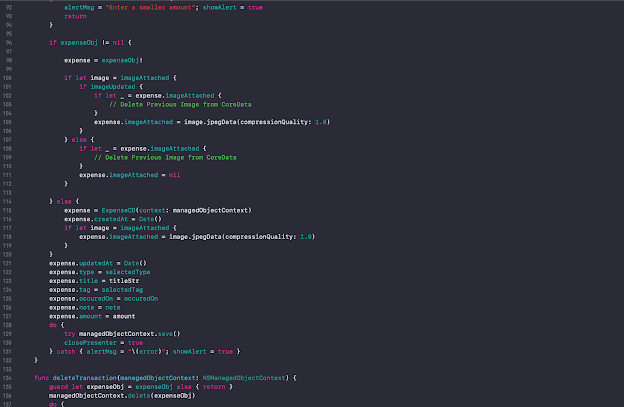
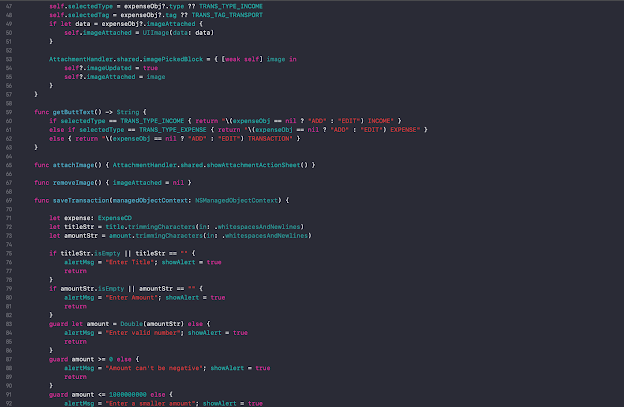
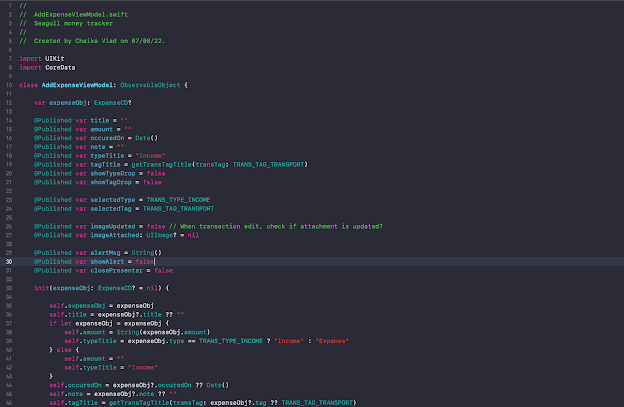


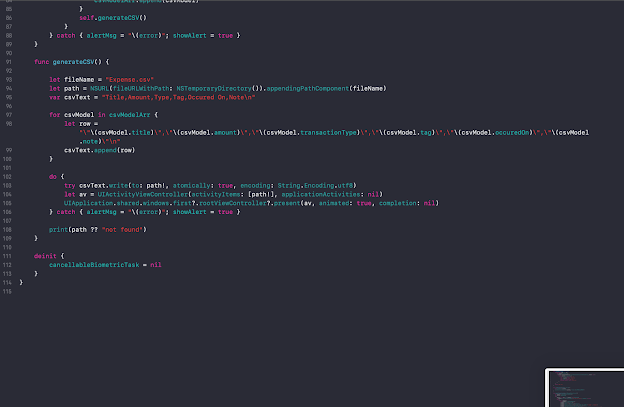
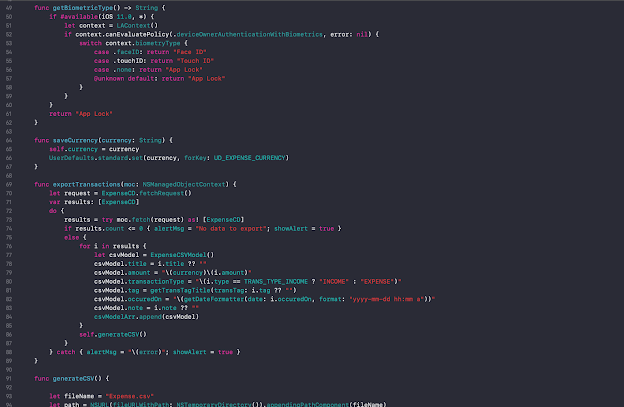
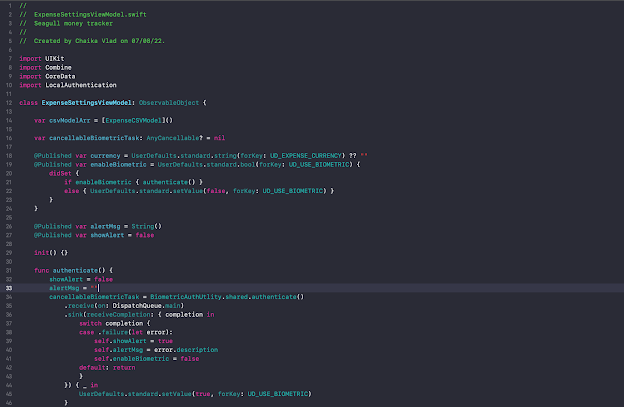


1. The blue button the plus icon allows the user to get to the page where he can add a transaction. The user must enter the necessary data, without which he will not be able to save the data (Title, Amount, Income/Expense, Date, Note and Image). My app has 2 spending characteristics-income and expenses. There are 10 categories of expenses available, each of which creates its own array and helps in filtering in the future + has its own icon that is very comfortable for the user.

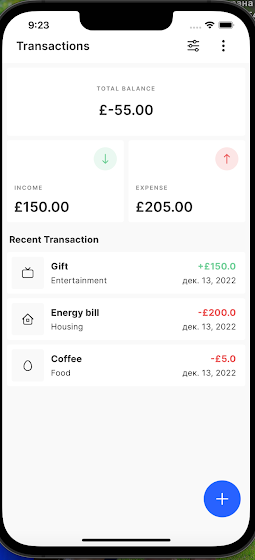




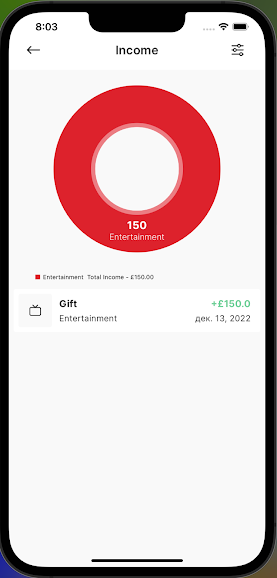
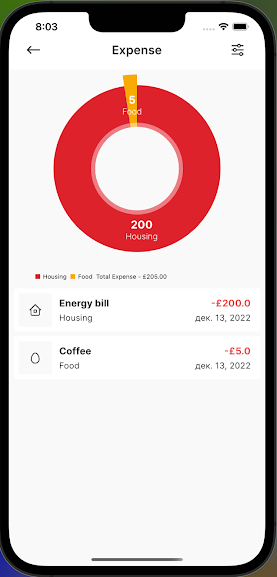
1. 
2. Transaction Settings code



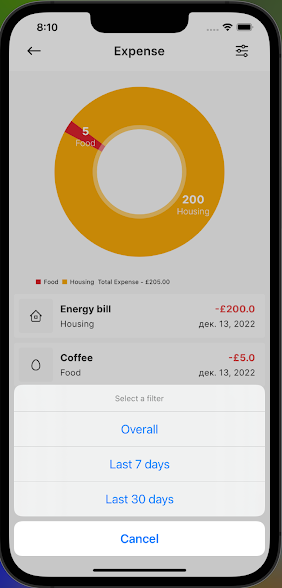
1. To better demonstrate the work of the application, I will add 2 expenses and 1 income transaction and look at their design. **Note: language is changed automatically for transaction titles and data information, according to the user’s IOS system language. In my case, I have an XCode application in the Russian language. For the client, it’s possible to have this information in various languages.**

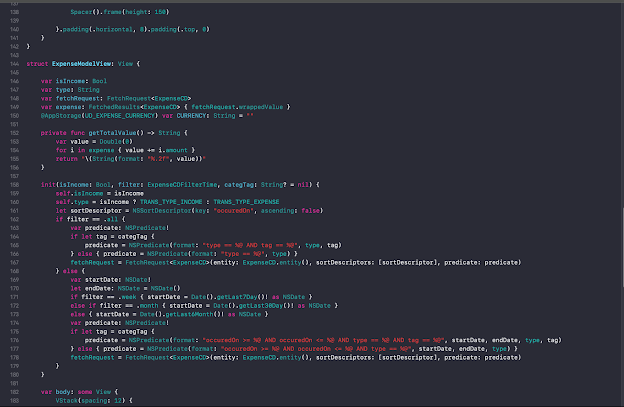
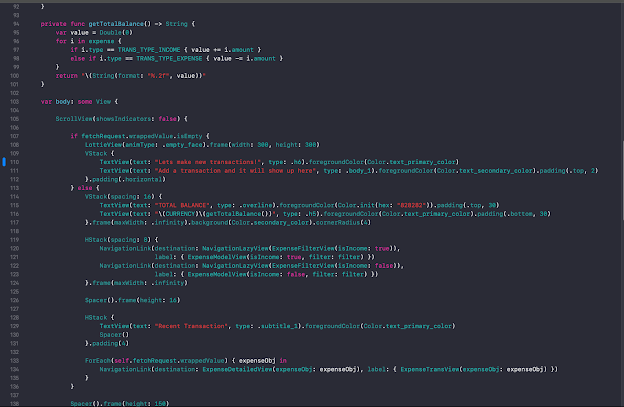
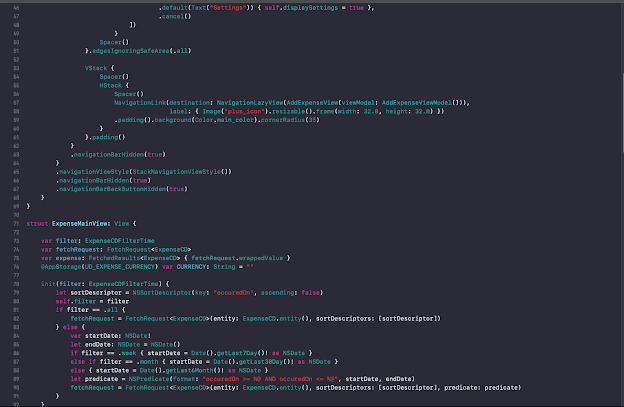
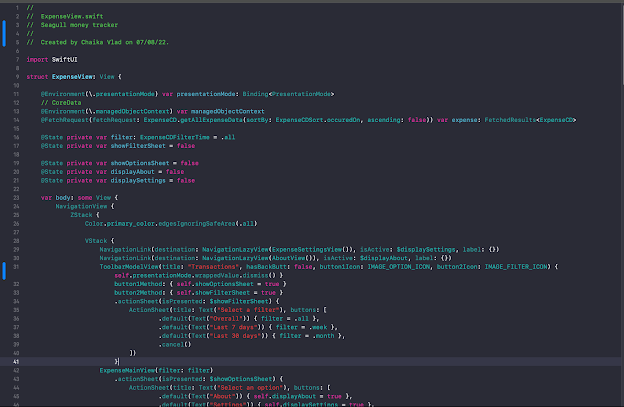
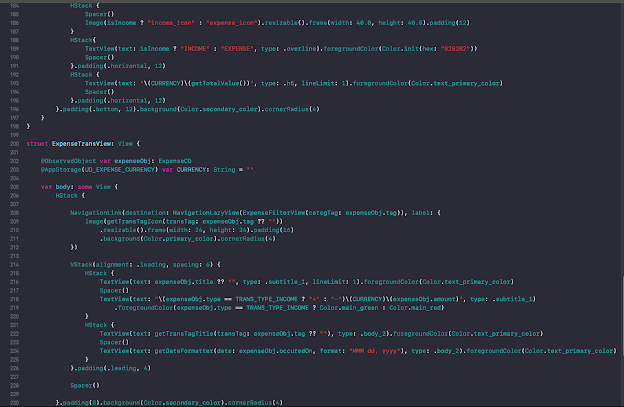


On this page, you can see 3 boxes at the top that describes each type of transaction and the total amount and at the bottom a list of transactions, each of which can be edited further. By clicking on one of the transaction categories, you can notice a pie chart that highlights the amount of each transaction

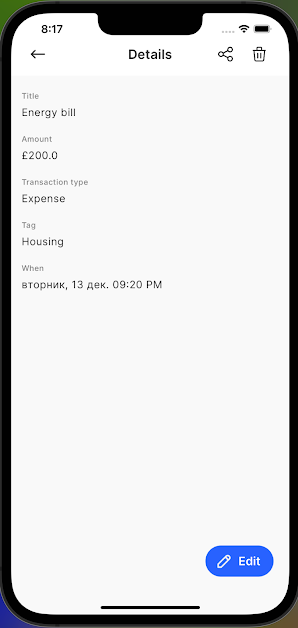


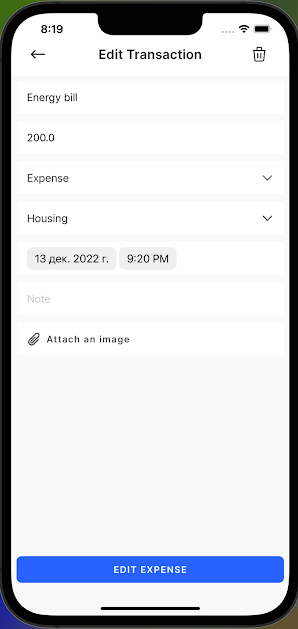
1. In these sections there is an opportunity for filtration by category using a pie diagram, and filtration by transaction’s time limit. There are 3 options: overall, last 7 days (week), last 30 days (month)



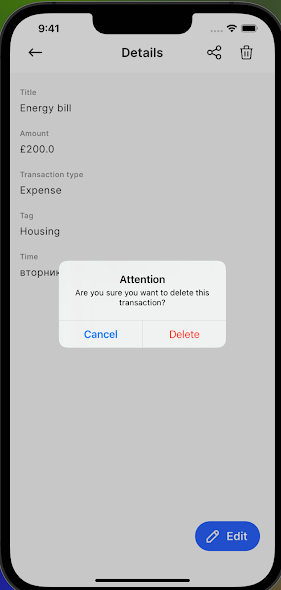
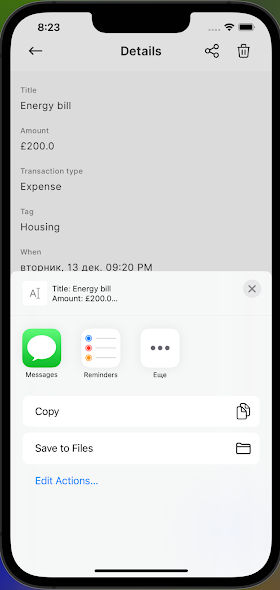


1. It’s nothing to mention that a user can look at the details of each transaction and edit the details of it. **Note: the language of time could be changed in settings.**  For example,

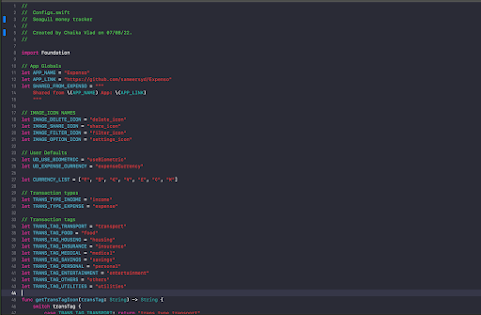




Also, users can delete transactions or share them with others through various sources like messages, WhatsApp etc.



1. Finally, I would like to share the other code (configs) that supports the functions of this application:



//

// AttachmentHandler.swift

// Seagull money tracker

// Created by Chaika Vlad on 09/09/22.

//

import Foundation

import UIKit

import MobileCoreServices

import AVFoundation

import Photos

class AttachmentHandler: NSObject {

static let shared = AttachmentHandler()

fileprivate var currentVC: UIViewController!

private override init() {

currentVC = UIApplication.shared.windows.first!.rootViewController

}

var imagePickedBlock: ((UIImage) -> Void)?

enum AttachmentType: String {

case camera, photoLibrary

}

struct Constants {

static let camera = "Camera"

static let phoneLibrary = "Phone Library"

static let alertForPhotoLibraryMessage = "App does not have access to your photos. To enable access, tap settings and turn on Photo Library Access."

static let alertForCameraAccessMessage = "App does not have access to your camera. To enable access, tap settings and turn on Camera."

static let settingsBtnTitle = "Settings"

static let cancelBtnTitle = "Cancel"

}

// This function is used to show the attachment sheet for camera, photo.

func showAttachmentActionSheet() {

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

actionSheet.addAction(UIAlertAction(title: Constants.camera, style: .default, handler: { (action) -> Void in

self.authorisationStatus(attachmentTypeEnum: .camera, vc: self.currentVC!)

}))

actionSheet.addAction(UIAlertAction(title: Constants.phoneLibrary, style: .default, handler: { (action) -> Void in

self.authorisationStatus(attachmentTypeEnum: .photoLibrary, vc: self.currentVC!)

}))

actionSheet.addAction(UIAlertAction(title: Constants.cancelBtnTitle, style: .cancel, handler: nil))

// if iPhone

if UIDevice.current.userInterfaceIdiom == .phone { currentVC.present(actionSheet, animated: true, completion: nil) }

else {

// Change Rect to position Popover

actionSheet.modalPresentationStyle = UIModalPresentationStyle.popover

actionSheet.popoverPresentationController?.sourceRect = CGRect(x: currentVC.view.frame.size.width / 2, y: currentVC.view.frame.size.height / 4, width: 0, height: 0)

actionSheet.popoverPresentationController?.sourceView = currentVC.view

actionSheet.popoverPresentationController?.permittedArrowDirections = .any

currentVC.present(actionSheet, animated: true, completion: nil)

}

}

// This is used to check the authorisation status whether user gives access to import the image, photo library.

// if the user gives access, then we can import the data safely

// if not show them alert to access from settings.

func authorisationStatus(attachmentTypeEnum: AttachmentType, vc: UIViewController) {

currentVC = vc

let cameraStatus = AVCaptureDevice.authorizationStatus(for: .video)

let photoStatus = PHPhotoLibrary.authorizationStatus()

if attachmentTypeEnum == AttachmentType.camera {

switch cameraStatus {

case .authorized:

openCamera()

case .denied:

print("permission denied")

self.addAlertForSettings(attachmentTypeEnum)

case .notDetermined:

print("Permission Not Determined")

AVCaptureDevice.requestAccess(for: .video) { success in

if success { self.openCamera() }

else {

print("restriced manually")

self.addAlertForSettings(attachmentTypeEnum)

}

}

case .restricted:

print("permission restricted")

self.addAlertForSettings(attachmentTypeEnum)

default: break

}

} else {

switch photoStatus {

case .authorized:

if attachmentTypeEnum == AttachmentType.photoLibrary { openLibrary() }

case .denied:

print("permission denied")

self.addAlertForSettings(attachmentTypeEnum)

case .notDetermined:

print("Permission Not Determined")

PHPhotoLibrary.requestAuthorization({ (status) in

if status == PHAuthorizationStatus.authorized {

// photo library access given

print("access given")

if attachmentTypeEnum == AttachmentType.photoLibrary { self.openLibrary() }

} else {

print("restriced manually")

self.addAlertForSettings(attachmentTypeEnum)

}

})

case .restricted:

print("permission restricted")

self.addAlertForSettings(attachmentTypeEnum)

default:

break

}

}

}

// This function is used to open camera from the iphone and

@objc func openCamera() {

DispatchQueue.global(qos: .background).async {

DispatchQueue.main.async {

if UIImagePickerController.isSourceTypeAvailable(.camera) {

let myPickerController = UIImagePickerController()

myPickerController.delegate = self

myPickerController.sourceType = .camera

self.currentVC?.present(myPickerController, animated: true, completion: nil)

}

}

}

}

@objc func openLibrary() {

DispatchQueue.global(qos: .background).async {

DispatchQueue.main.async {

if UIImagePickerController.isSourceTypeAvailable(.photoLibrary) {

let myPickerController = UIImagePickerController()

myPickerController.delegate = self

myPickerController.sourceType = .photoLibrary

myPickerController.mediaTypes = ["public.image"]

self.currentVC?.present(myPickerController, animated: true, completion: nil)

}

}

}

}

func addAlertForSettings(\_ attachmentTypeEnum: AttachmentType) {

DispatchQueue.global(qos: .background).async {

DispatchQueue.main.async {

var alertTitle: String = ""

if attachmentTypeEnum == AttachmentType.camera {

alertTitle = Constants.alertForCameraAccessMessage

}

if attachmentTypeEnum == AttachmentType.photoLibrary {

alertTitle = Constants.alertForPhotoLibraryMessage

}

let cameraUnavailableAlertController = UIAlertController (title: alertTitle , message: nil, preferredStyle: .alert)

let settingsAction = UIAlertAction(title: Constants.settingsBtnTitle, style: .destructive) { (\_) -> Void in

let settingsUrl = NSURL(string:UIApplication.openSettingsURLString)

if let url = settingsUrl {

UIApplication.shared.open(url as URL, options: [:], completionHandler: nil)

}

}

let cancelAction = UIAlertAction(title: Constants.cancelBtnTitle, style: .default, handler: nil)

cameraUnavailableAlertController .addAction(cancelAction)

cameraUnavailableAlertController .addAction(settingsAction)

self.currentVC?.present(cameraUnavailableAlertController , animated: true, completion: nil)

}

}

}

}

// This is responsible for image picker interface to access image and then responsible for canceling the picker

extension AttachmentHandler: UIImagePickerControllerDelegate, UINavigationControllerDelegate {

func imagePickerControllerDidCancel(\_ picker: UIImagePickerController) {

currentVC?.dismiss(animated: true, completion: nil)

}

@objc internal func imagePickerController(\_ picker: UIImagePickerController, didFinishPickingMediaWithInfo info: [UIImagePickerController.InfoKey : Any]) {

if let image = info[UIImagePickerController.InfoKey.originalImage] as? UIImage {

self.imagePickedBlock?(image)

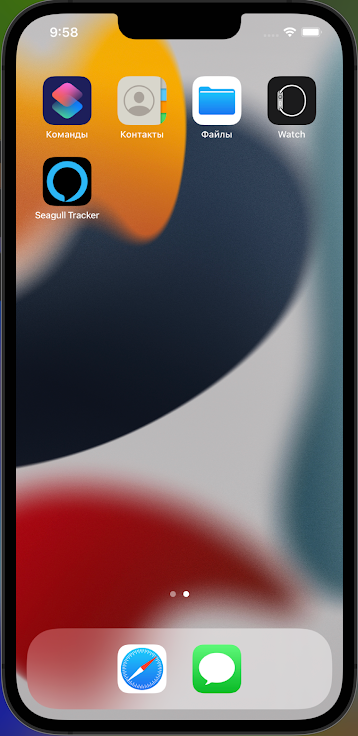
}

currentVC?.dismiss(animated: true, completion: nil)

}

}

1. My application in the desk looks like this:



1. [Face ID & Touch ID Usage in App (Swift 5, Xcode 12, Biometrics, iOS) - 2022 iOS Development](https://www.youtube.com/watch?v=SHmDljfu2lk)

   [↑](#footnote-ref-0)