Criterion C- Development

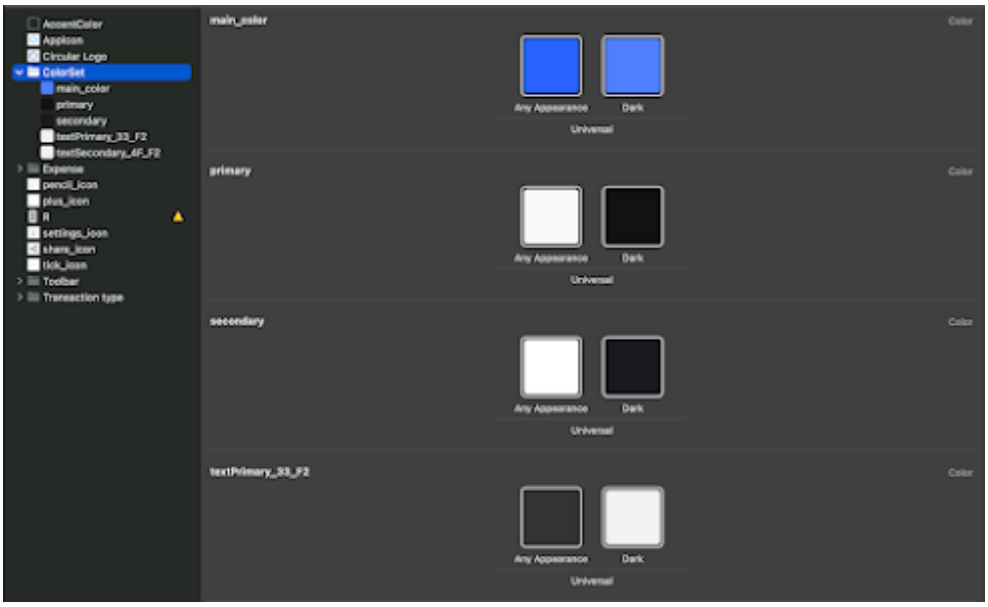
List of techniques:

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

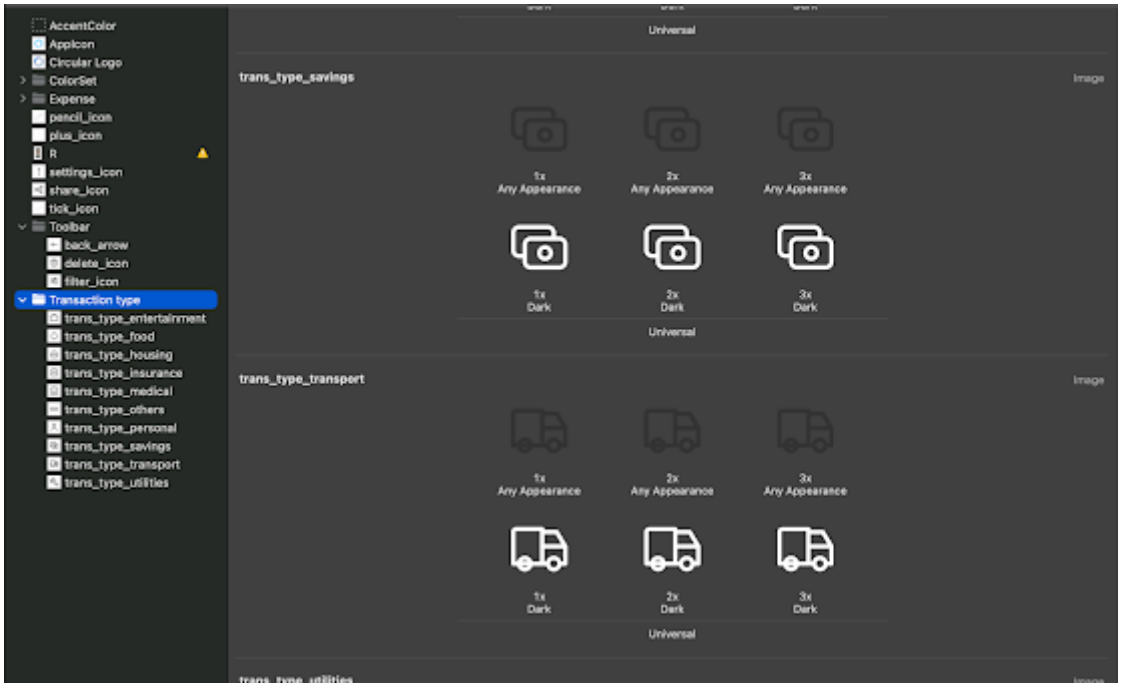
Stages of app development:

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.



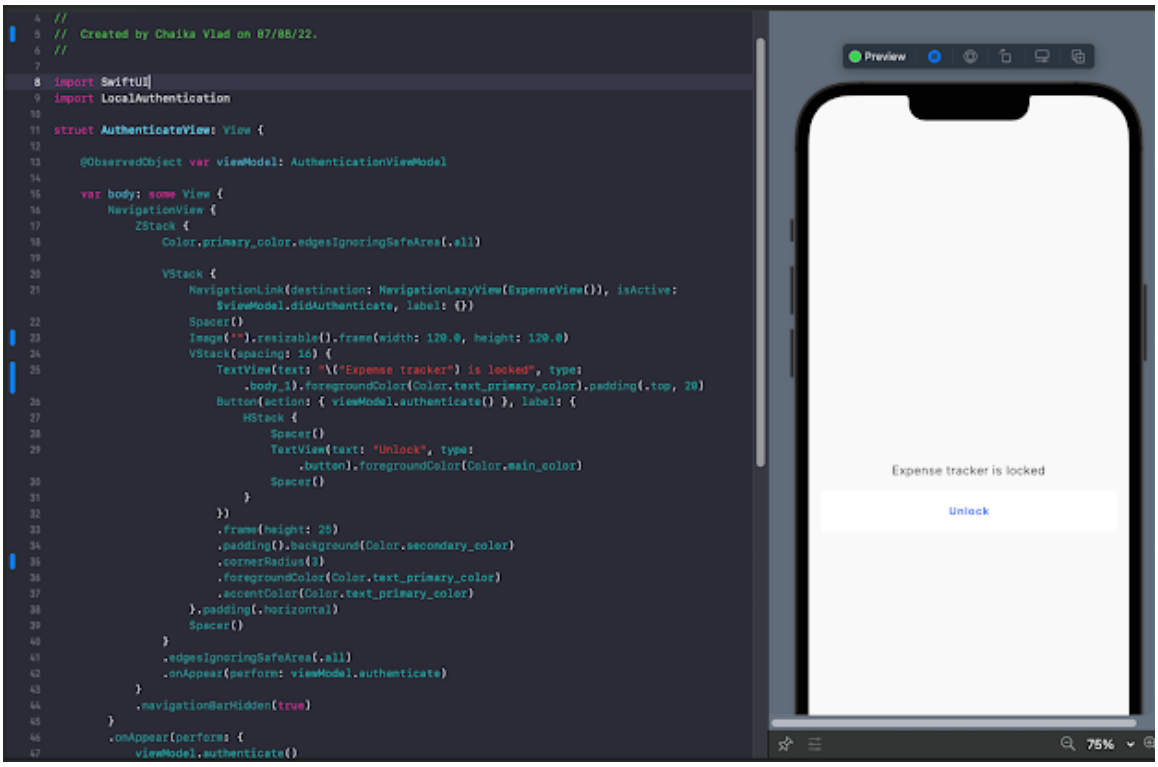
Working with the assets of my application, I decided to instantly install the library of icons from Apple, as well as icons from other authors for a more interesting design of my application.



2) 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”.

```swift
//
//  AuthenticationViewModel.swift
//  Seagull money tracker
//
//  Created by Chaika Vlad on 07/08/22.
//

import Foundation
import Combine

class AuthenticationViewModel: ObservableObject {

    var cancellableBiometricTask: AnyCancellable? = nil

    var didAuthenticate = false
    var showAlert = false
    var alertMessage = String()

    func authenticate() {
        didAuthenticate = false
        showAlert = false
        alertMessage = "You are not a user!"
        cancellableBiometricTask = BiometricAuthUtility.shared.authenticate()
            .receive(on: DispatchQueue.main)
            .sink(receiveCompletion: { completion in
                switch completion {
                case .failure(let error):
                    self.showAlert = true
                    self.alertMessage = error.description
                default: return
                }
            }) { _ in
                self.didAuthenticate = true
            }
    }

    deinit {
        cancellableBiometricTask = nil
    }
}
```



```swift
//
//  Created by Chaika Vlad on 07/08/22.
//

import SwiftUI
import LocalAuthentication

struct AuthenticateView: View {

    @ObservedObject var viewModel: AuthenticationViewModel

    var body: some View {
        NavigationView {
            ZStack {
                Color.primary_color.edgesIgnoringSafeArea(.all)

                VStack {
                    NavigationLink(destination: NavigationLazyView(ExpenseView()), isActive:
                        $viewModel.didAuthenticate, label: {})
                    Spacer()
                    Image("").resizable().frame(width: 120.0, height: 120.0)
                    VStack(spacing: 16) {
                        TextView(text: "\("Expense tracker") is locked", type:
                            .body_1).foregroundColor(Color.text_primary_color).padding(.top, 20)
                        Button(action: { viewModel.authenticate() }, label: {
                            HStack {
                                Spacer()
                                TextView(text: "Unlock", type:
                                    .button).foregroundColor(Color.main_color)
                                Spacer()
                            }
                        })
                        .frame(height: 25)
                        .padding().background(Color.secondary_color)
                        .cornerRadius(3)
                        .foregroundColor(Color.text_primary_color)
                        .accentColor(Color.text_primary_color)
                    }.padding(.horizontal)
                    Spacer()
                }
                .edgesIgnoringSafeArea(.all)
                .onAppear(perform: viewModel.authenticate)
            }
            .navigationBarHidden(true)
        }
        .onAppear(perform: {
            viewModel.authenticate()
```

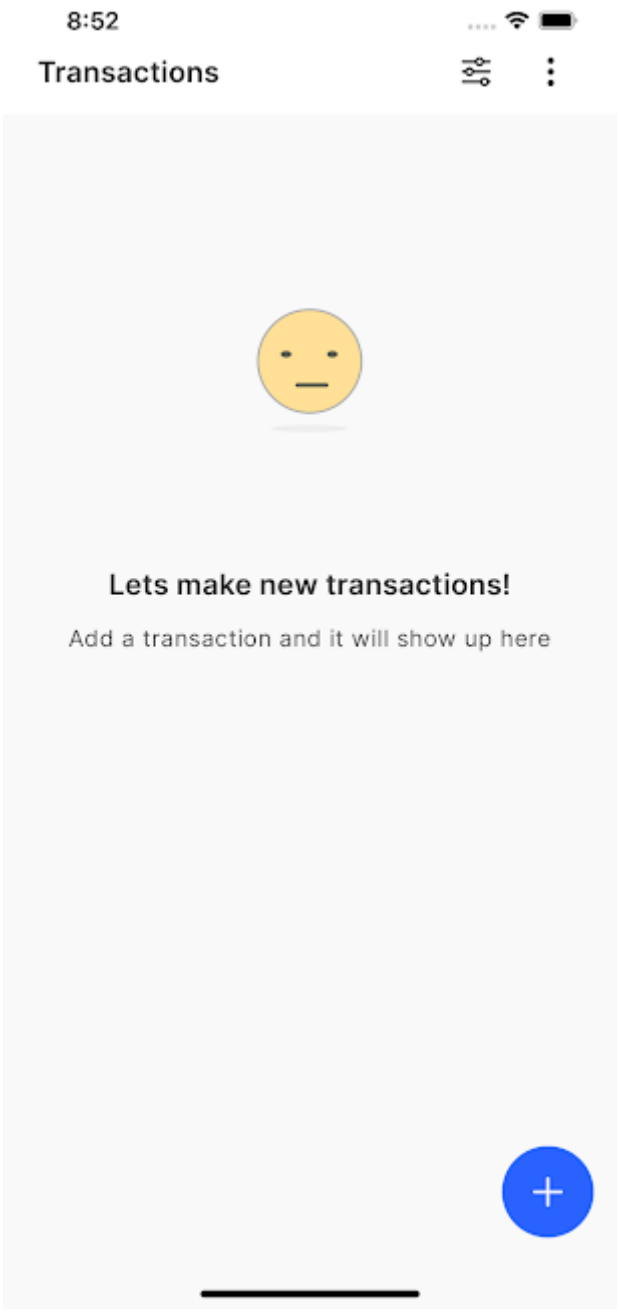
3) Biometrical code from YouTube[1] is:

```
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) {
```

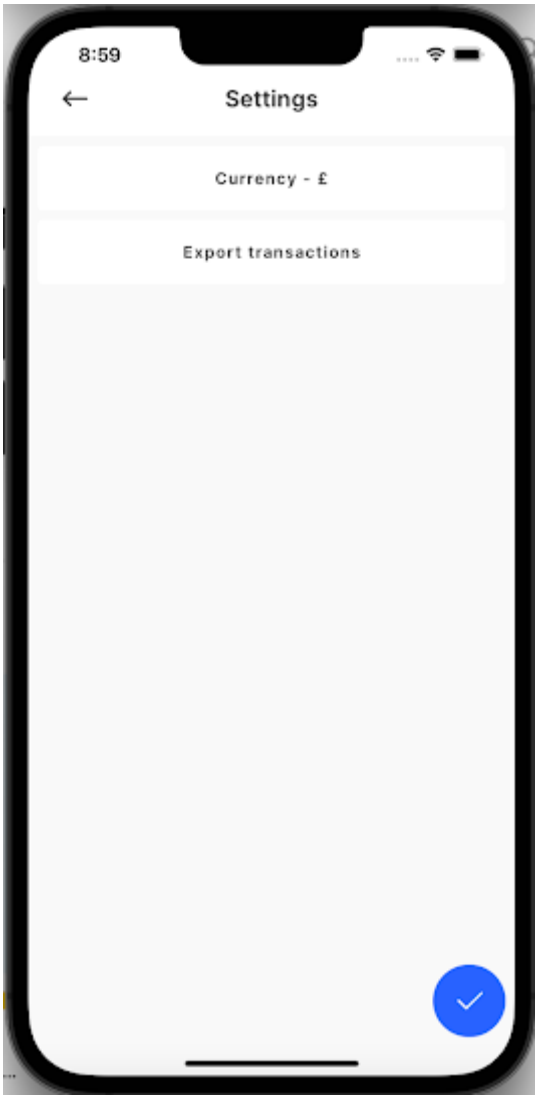
[1] Face ID & Touch ID Usage in App (Swift 5, Xcode 12, Biometrics, iOS) - 2022 iOS Development

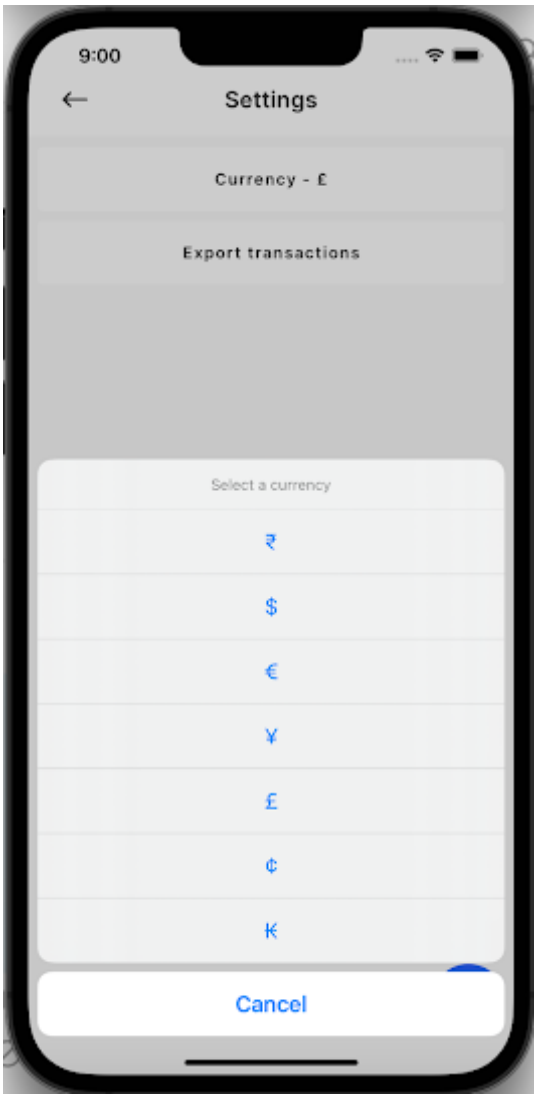
```
            // fallback
            context.evaluatePolicy(.deviceOwnerAuthentication, localizedReason: reason, reply: handleReply)
        }else{
            //cannot evaluate
            let error = BiometericAuthError(description: "Something went wrong while authenticating. Please try again")
            promise(.failure(error))
        }
    }
  }


}
```

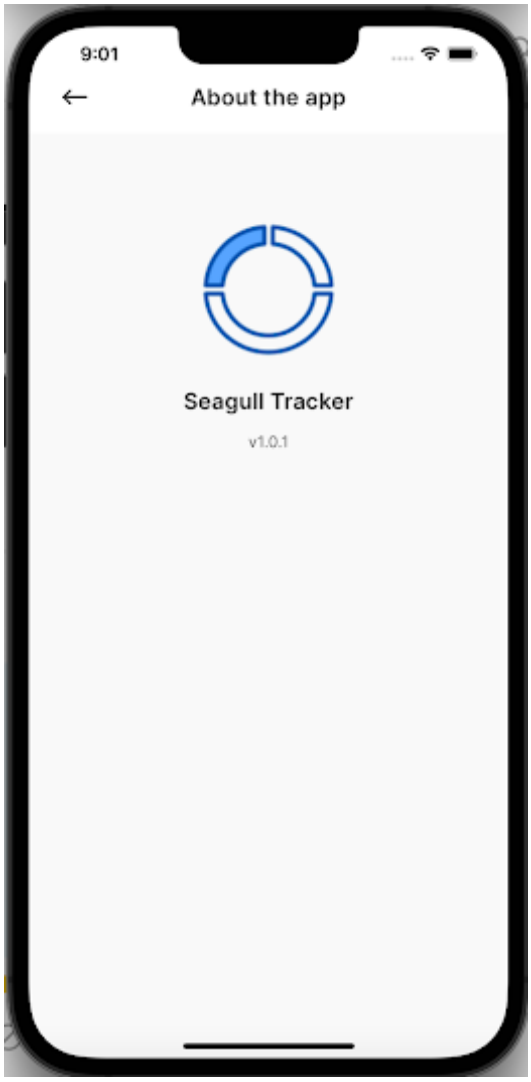
4) I'll start with the main page. An empty main page called “transactions: is a screen where the user is greeted by a model of an animated man's face and text that informs the user to add transactions.

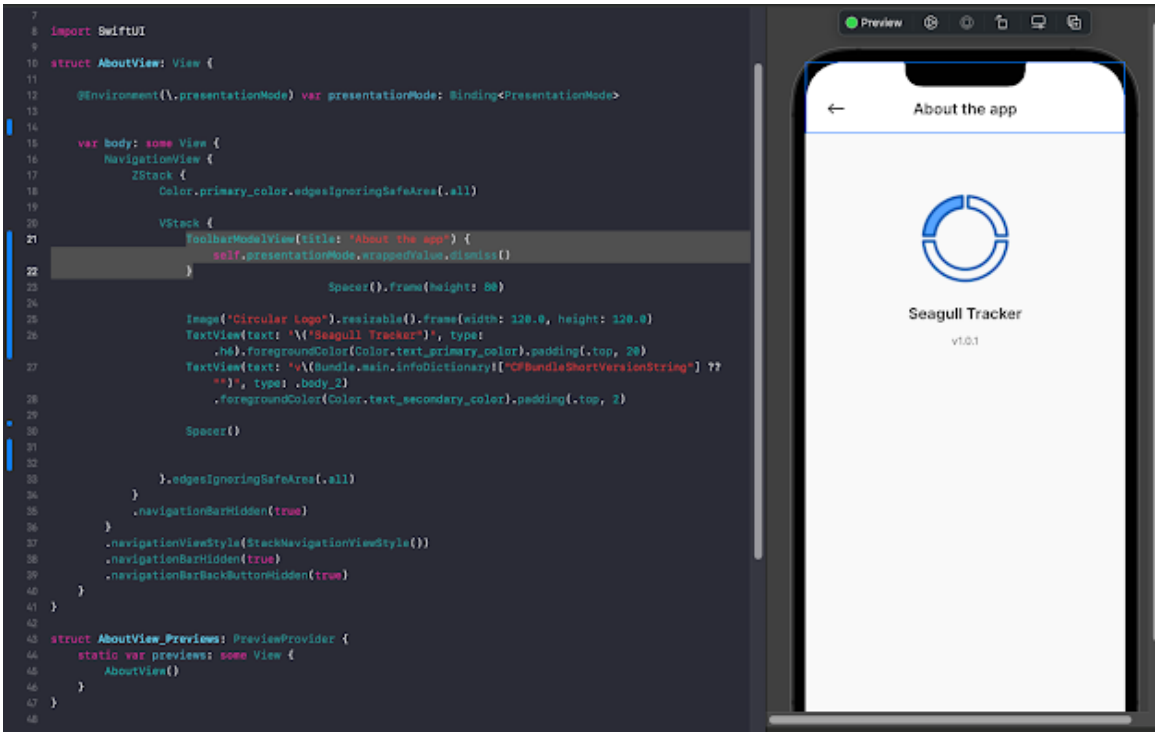


The icon with dots in the upper right corner allows you to view information and select 2 more pages: about us, which will present the name and the latest version of the application, as well as the settings tab where you can change the currency (7 in total) and the ability to share transactions as a pdf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


8:59
Settings
Currency - £
Export transactions


9:00
Settings
Currency - £
Export transactions
Select a currency
₹
$
€
¥
£
¢
₭
Cancel


9:01
About the app
Seagull Tracker
v1.0.1

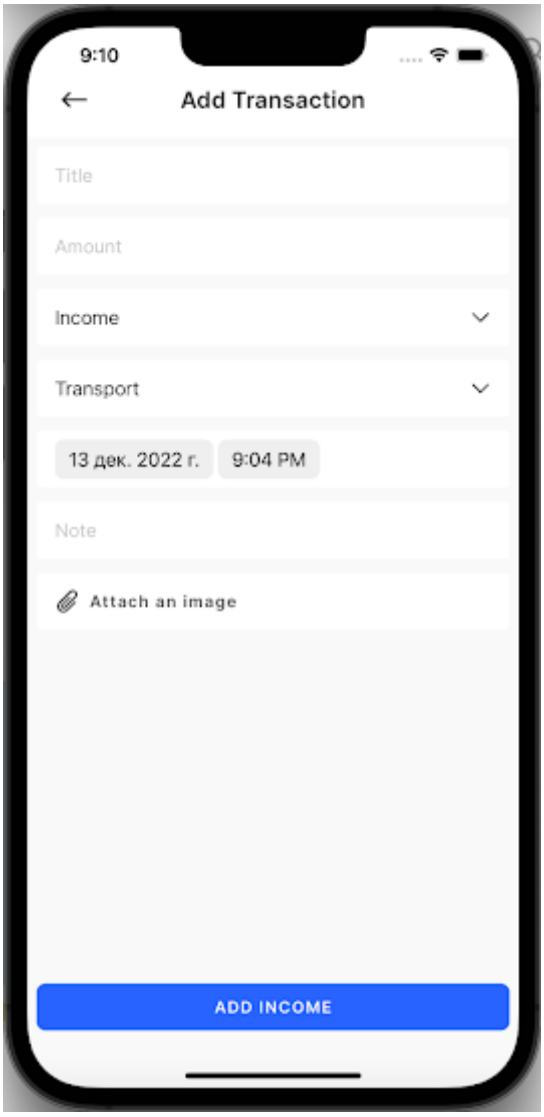


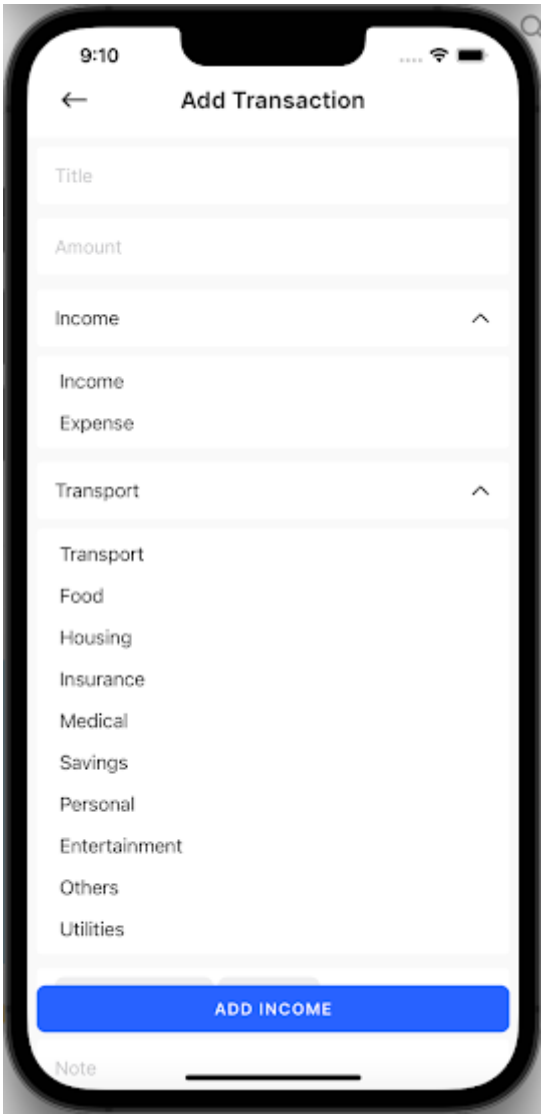

```swift
import SwiftUI

struct AboutView: View {

    @Environment(\.presentationMode) var presentationMode: Binding<PresentationMode>


    var body: some View {
        NavigationView {
            ZStack {
                Color.primary_color.edgesIgnoringSafeArea(.all)

                VStack {
                    ToolbarModelView(title: "About the app") {
                        self.presentationMode.wrappedValue.dismiss()
                    }
                                        Spacer().frame(height: 80)

                    Image("Circular Logo").resizable().frame(width: 120.0, height: 120.0)
                    TextView(text: "\("Seagull Tracker")", type:
                        .h6).foregroundColor(Color.text_primary_color).padding(.top, 20)
                    TextView(text: "v\(Bundle.main.infoDictionary![("CFBundleShortVersionString")] ??
                        "")", type: .body_2)
                        .foregroundColor(Color.text_secondary_color).padding(.top, 2)

                    Spacer()


                }.edgesIgnoringSafeArea(.all)
            }
            .navigationBarHidden(true)
        }
        .navigationViewStyle(StackNavigationViewStyle())
        .navigationBarHidden(true)
        .navigationBarBackButtonHidden(true)
    }
}

struct AboutView_Previews: PreviewProvider {
    static var previews: some View {
        AboutView()
    }
}
```

```swift
import Foundation
import CoreData

enum ExpenseCDSort: String {
    case createdAt
    case updatedAt
    case occuredOn
}

enum ExpenseCDFilterTime: String {
    case all
    case week
    case month
}

public class ExpenseCD: NSManagedObject, Identifiable {
    @NSManaged public var createdAt: Date?
    @NSManaged public var updatedAt: Date?
    @NSManaged public var type: String?
    @NSManaged public var title: String?
    @NSManaged public var tag: String?
    @NSManaged public var occuredOn: Date?
    @NSManaged public var note: String?
    @NSManaged public var amount: Double
    @NSManaged public var imageAttached: Data?
}

extension ExpenseCD {
    static func getAllExpenseData(sortBy: ExpenseCDSort = .occuredOn, ascending: Bool = true, filterTime: ExpenseCDFilterTime = .all) ->
        NSFetchRequest<ExpenseCD> {
        let request: NSFetchRequest<ExpenseCD> = ExpenseCD.fetchRequest() as! NSFetchRequest<ExpenseCD>
        let sortDescriptor = NSSortDescriptor(key: sortBy.rawValue, ascending: ascending)
        if filterTime == .week {
            let startDate: NSDate = Date().getLast7Day()! as NSDate
            let endDate: NSDate = NSDate()
            let predicate = NSPredicate(format: "occuredOn >= %@ AND occuredOn <= %@", startDate, endDate)
            request.predicate = predicate
        } else if filterTime == .month {
            let startDate: NSDate = Date().getLast30Day()! as NSDate
            let endDate: NSDate = NSDate()
            let predicate = NSPredicate(format: "occuredOn >= %@ AND occuredOn <= %@", startDate, endDate)
            request.predicate = predicate
        }
        request.sortDescriptors = [sortDescriptor]
        return request
```

5) 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.





```swift
//  Created by Chaika Vlad on 07/08/22.
//

import SwiftUI

struct AddExpenseView: View {

    @Environment(\.presentationMode) var presentationMode: Binding<PresentationMode>
    // CoreData
    @Environment(\.managedObjectContext) var managedObjectContext
    @State private var confirmDelete = false
    @State var showAttachSheet = false

    @StateObject var viewModel: AddExpenseViewModel

    let typeOptions = [
        DropdownOption(key: TRANS_TYPE_INCOME, val: "Income"),
        DropdownOption(key: TRANS_TYPE_EXPENSE, val: "Expense")
    ]

    let tagOptions = [
        DropdownOption(key: TRANS_TAG_TRANSPORT, val: "Transport"),
        DropdownOption(key: TRANS_TAG_FOOD, val: "Food"),
        DropdownOption(key: TRANS_TAG_HOUSING, val: "Housing"),
        DropdownOption(key: TRANS_TAG_INSURANCE, val: "Insurance"),
        DropdownOption(key: TRANS_TAG_MEDICAL, val: "Medical"),
        DropdownOption(key: TRANS_TAG_SAVINGS, val: "Savings"),
        DropdownOption(key: TRANS_TAG_PERSONAL, val: "Personal"),
        DropdownOption(key: TRANS_TAG_ENTERTAINMENT, val: "Entertainment"),
        DropdownOption(key: TRANS_TAG_OTHERS, val: "Others"),
        DropdownOption(key: TRANS_TAG_UTILITIES, val: "Utilities")
    ]

    var body: some View {
        NavigationView {
            ZStack {
                Color.primary_color.edgesIgnoringSafeArea(.all)

                VStack {

                    Group {
                        if viewModel.expenseObj == nil {
                            ToolbarModelView(title: "Add Transaction") { self.presentationMode.wrappedValue.dismiss() }
                        } else {
                            ToolbarModelView(title: "Edit Transaction", button1Icon: IMAGE_DELETE_ICON) { self.presentationMode.wrappedValue.dismiss() }
                                button1Method: { self.confirmDelete = true }
                        }
                    }.alert(isPresented: $confirmDelete,
                            content: {
                                Alert(title: Text(APP_NAME), message: Text("Are you sure you want to delete this transaction?"),
                                    primaryButton: .destructive(Text("Delete")) {
                                        viewModel.deleteTransaction(managedObjectContext: self.managedObjectContext)
                                    }, secondaryButton: Alert.Button.cancel(Text("Cancel"), action: { confirmDelete = false })
                                )
                            })

                    ScrollView(showsIndicators: false) {

                        VStack(spacing: 12) {

                            TextField("Title", text: $viewModel.title)
                                .modifier(InterFont(.regular, size: 16))
                                .accentColor(Color.text_primary_color)
                                .frame(height: 50).padding(.leading, 16)
                                .background(Color.secondary_color)
                                .cornerRadius(4)

                            TextField("Amount", text: $viewModel.amount)
                                .modifier(InterFont(.regular, size: 16))
                                .accentColor(Color.text_primary_color)
                                .frame(height: 50).padding(.leading, 16)
                                .background(Color.secondary_color)
                                .cornerRadius(4).keyboardType(.decimalPad)

                            DropdownButton(shouldShowDropdown: $viewModel.showTypeDrop, displayText: $viewModel.typeTitle,
                                           options: typeOptions, mainColor: Color.text_primary_color,
                                           backgroundColor: Color.secondary_color, cornerRadius: 4, buttonHeight: 50) { key in
                                let selectedObj = typeOptions.filter({ $0.key == key }).first
                                if let object = selectedObj {
                                    viewModel.typeTitle = object.val
                                    viewModel.selectedType = key
                                }
                                viewModel.showTypeDrop = false
                            }

                            DropdownButton(shouldShowDropdown: $viewModel.showTagDrop, displayText: $viewModel.tagTitle,
                                           options: tagOptions, mainColor: Color.text_primary_color,
                                           backgroundColor: Color.secondary_color, cornerRadius: 4, buttonHeight: 50) { key in
                                let selectedObj = tagOptions.filter({ $0.key == key }).first
                                if let object = selectedObj {
```

```
                        }

                        Spacer().frame(height: 150)
                        Spacer()
                    }
                    .frame(maxWidth: .infinity).padding(.horizontal, 8)
                    .alert(isPresented: $viewModel.showAlert,
                           content: { Alert(title: Text(APP_NAME), message: Text(viewModel.alertMsg), dismissButton: .default(Text("OK"))) })
                }

            }.edgesIgnoringSafeArea(.top)

            VStack {
                Spacer()
                VStack {
                    Button(action: { viewModel.saveTransaction(managedObjectContext: managedObjectContext) }, label: {
                        HStack {
                            Spacer()
                            TextView(text: viewModel.getButtText(), type: .button).foregroundColor(.white)
                            Spacer()
                        }
                    })
                    .padding(.vertical, 12).background(Color.main_color).cornerRadius(8)
                }.padding(.bottom, 16).padding(.horizontal, 8)
            }

        }
        .navigationBarHidden(true)
    }
    .dismissKeyboardOnTap()
    .navigationViewStyle(StackNavigationViewStyle())
    .navigationBarHidden(true)
    .navigationBarBackButtonHidden(true)
    .onReceive(viewModel.$closePresenter) { close in
        if close { self.presentationMode.wrappedValue.dismiss() }
    }
  }
}
```

```
                        }

                        HStack {
                            DatePicker("PickerView", selection: $viewModel.occuredOn,
                                       displayedComponents: [.date, .hourAndMinute]).labelsHidden().padding(.leading, 16)
                            Spacer()
                        }
                        .frame(height: 50).frame(maxWidth: .infinity)
                        .accentColor(Color.text_primary_color)
                        .background(Color.secondary_color).cornerRadius(4)

                        TextField("Note", text: $viewModel.note)
                            .modifier(InterFont(.regular, size: 16))
                            .accentColor(Color.text_primary_color)
                            .frame(height: 50).padding(.leading, 16)
                            .background(Color.secondary_color)
                            .cornerRadius(4)

                        Button(action: { viewModel.attachImage() }, label: {
                            HStack {
                                Image(systemName: "paperclip")
                                    .font(.system(size: 18.0, weight: .bold))
                                    .foregroundColor(Color.text_secondary_color)
                                    .padding(.leading, 16)
                                TextView(text: "Attach an image", type: .button).foregroundColor(Color.text_secondary_color)
                                Spacer()
                            }
                        })
                        .frame(height: 50).frame(maxWidth: .infinity)
                        .background(Color.secondary_color)
                        .cornerRadius(4)
                        .actionSheet(isPresented: $showAttachSheet) {
                            ActionSheet(title: Text("Do you want to remove the attachment?"), buttons: [
                                .default(Text("Remove")) { viewModel.removeImage() },
                                .cancel()
                            ])
                        }

                        if let image = viewModel.imageAttached {
                            Button(action: { showAttachSheet = true }, label: {
                                Image(uiImage: image)
                                    .resizable()
                                    .scaledToFill()
                                    .frame(height: 250).frame(maxWidth: .infinity)
                                    .background(Color.secondary_color)
                                    .cornerRadius(4)
```

6)

```swift
//
//  AddExpenseViewModel.swift
//  Seagull money tracker
//
//  Created by Chaika Vlad on 07/08/22.

import UIKit
import CoreData

class AddExpenseViewModel: ObservableObject {

    var expenseObj: ExpenseCD?

    @Published var title = ""
    @Published var amount = ""
    @Published var occuredOn = Date()
    @Published var note = ""
    @Published var typeTitle = "Income"
    @Published var tagTitle = getTransTagTitle(transTag: TRANS_TAG_TRANSPORT)
    @Published var showTypeDrop = false
    @Published var showTagDrop = false

    @Published var selectedType = TRANS_TYPE_INCOME
    @Published var selectedTag = TRANS_TAG_TRANSPORT

    @Published var imageUpdated = false // When transaction edit, check if attachment is updated?
    @Published var imageAttached: UIImage? = nil

    @Published var alertMsg = String()
    @Published var showAlert = false
    @Published var closePresenter = false

    init(expenseObj: ExpenseCD? = nil) {

        self.expenseObj = expenseObj
        self.title = expenseObj?.title ?? ""
        if let expenseObj = expenseObj {
            self.amount = String(expenseObj.amount)
            self.typeTitle = expenseObj.type == TRANS_TYPE_INCOME ? "Income" : "Expense"
        } else {
            self.amount = ""
            self.typeTitle = "Income"
        }
        self.occuredOn = expenseObj?.occuredOn ?? Date()
        self.note = expenseObj?.note ?? ""
        self.tagTitle = getTransTagTitle(transTag: expenseObj?.tag ?? TRANS_TAG_TRANSPORT)
        self.selectedType = expenseObj?.type ?? TRANS_TYPE_INCOME
        self.selectedTag = expenseObj?.tag ?? TRANS_TAG_TRANSPORT
        if let data = expenseObj?.imageAttached {
            self.imageAttached = UIImage(data: data)
        }

        AttachementHandler.shared.imagePickedBlock = { [weak self] image in
            self?.imageUpdated = true
            self?.imageAttached = image
        }
    }

    func getButtText() -> String {
        if selectedType == TRANS_TYPE_INCOME { return "\(expenseObj == nil ? "ADD" : "EDIT") INCOME" }
        else if selectedType == TRANS_TYPE_EXPENSE { return "\(expenseObj == nil ? "ADD" : "EDIT") EXPENSE" }
        else { return "\(expenseObj == nil ? "ADD" : "EDIT") TRANSACTION" }
    }

    func attachImage() { AttachementHandler.shared.showAttachementActionSheet() }

    func removeImage() { imageAttached = nil }

    func saveTransaction(managedObjectContext: NSManagedObjectContext) {

        let expense: ExpenseCD
        let titleStr = title.trimmingCharacters(in: .whitespacesAndNewlines)
        let amountStr = amount.trimmingCharacters(in: .whitespacesAndNewlines)

        if titleStr.isEmpty || titleStr == "" {
            alertMsg = "Enter Title"; showAlert = true
            return
        }
        if amountStr.isEmpty || amountStr == "" {
            alertMsg = "Enter Amount"; showAlert = true
            return
        }
        guard let amount = Double(amountStr) else {
            alertMsg = "Enter valid number"; showAlert = true
            return
        }
        guard amount >= 0 else {
            alertMsg = "Amount can't be negative"; showAlert = true
            return
        }
        guard amount <= 1000000000 else {
            alertMsg = "Enter a smaller amount"; showAlert = true
```

```
            alertMsg = "Enter a smaller amount"; showAlert = true
            return
        }

        if expenseObj != nil {

            expense = expenseObj!

            if let image = imageAttached {
                if imageUpdated {
                    if let _ = expense.imageAttached {
                        // Delete Previous Image from CoreData
                    }
                    expense.imageAttached = image.jpegData(compressionQuality: 1.0)
                }
            } else {
                if let _ = expense.imageAttached {
                    // Delete Previous Image from CoreData
                }
                expense.imageAttached = nil
            }

        } else {
            expense = ExpenseCD(context: managedObjectContext)
            expense.createdAt = Date()
            if let image = imageAttached {
                expense.imageAttached = image.jpegData(compressionQuality: 1.0)
            }
        }
        expense.updatedAt = Date()
        expense.type = selectedType
        expense.title = titleStr
        expense.tag = selectedTag
        expense.occuredOn = occuredOn
        expense.note = note
        expense.amount = amount
        do {
            try managedObjectContext.save()
            closePresenter = true
        } catch { alertMsg = "\(error)"; showAlert = true }
    }

    func deleteTransaction(managedObjectContext: NSManagedObjectContext) {
        guard let expenseObj = expenseObj else { return }
        managedObjectContext.delete(expenseObj)
        do {
```

```
                if let _ = expense.imageAttached {
                    // Delete Previous Image from CoreData
                }
                expense.imageAttached = nil
            }

        } else {
            expense = ExpenseCD(context: managedObjectContext)
            expense.createdAt = Date()
            if let image = imageAttached {
                expense.imageAttached = image.jpegData(compressionQuality: 1.0)
            }
        }
        expense.updatedAt = Date()
        expense.type = selectedType
        expense.title = titleStr
        expense.tag = selectedTag
        expense.occuredOn = occuredOn
        expense.note = note
        expense.amount = amount
        do {
            try managedObjectContext.save()
            closePresenter = true
        } catch { alertMsg = "\(error)"; showAlert = true }
    }

    func deleteTransaction(managedObjectContext: NSManagedObjectContext) {
        guard let expenseObj = expenseObj else { return }
        managedObjectContext.delete(expenseObj)
        do {
            try managedObjectContext.save(); closePresenter = true
        } catch { alertMsg = "\(error)"; showAlert = true }
    }
}
```

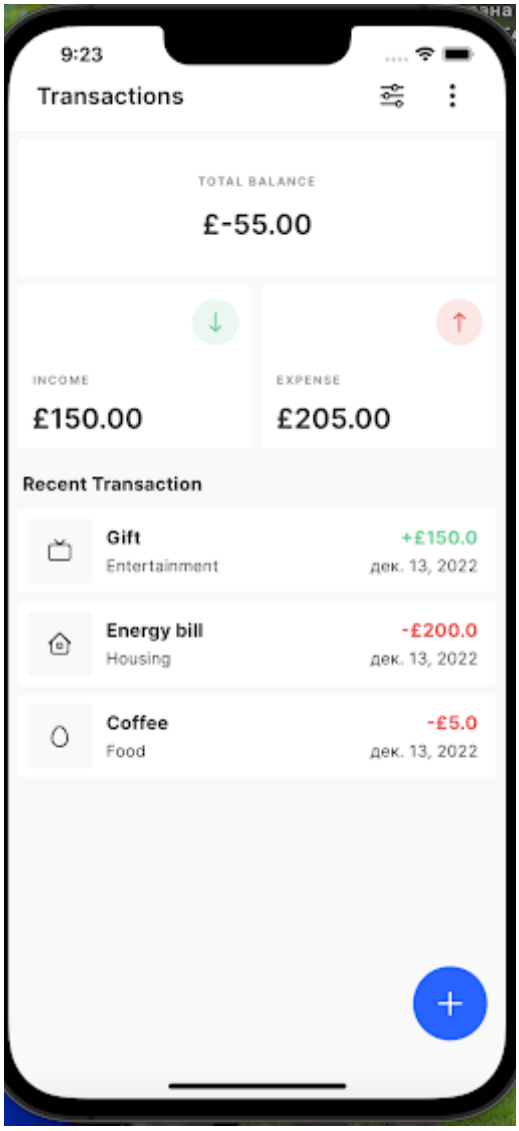
7) Transaction Settings code

```swift
//
//  ExpenseSettingsViewModel.swift
//  Seagull money tracker
//
//  Created by Chaika Vlad on 07/08/22.

import UIKit
import Combine
import CoreData
import LocalAuthentication

class ExpenseSettingsViewModel: ObservableObject {

    var csvModelArr = [ExpenseCSVModel]()

    var cancellableBiometricTask: AnyCancellable? = nil

    @Published var currency = UserDefaults.standard.string(forKey: UD_EXPENSE_CURRENCY) ?? ""
    @Published var enableBiometric = UserDefaults.standard.bool(forKey: UD_USE_BIOMETRIC) {
        didSet {
            if enableBiometric { authenticate() }
            else { UserDefaults.standard.setValue(false, forKey: UD_USE_BIOMETRIC) }
        }
    }

    @Published var alertMsg = String()
    @Published var showAlert = false

    init() {}

    func authenticate() {
        showAlert = false
        alertMsg = ""
        cancellableBiometricTask = BiometricAuthUtlity.shared.authenticate()
            .receive(on: DispatchQueue.main)
            .sink(receiveCompletion: { completion in
                switch completion {
                case .failure(let error):
                    self.showAlert = true
                    self.alertMsg = error.description
                    self.enableBiometric = false
                default: return
                }
            }) { _ in
                UserDefaults.standard.setValue(true, forKey: UD_USE_BIOMETRIC)
            }
```

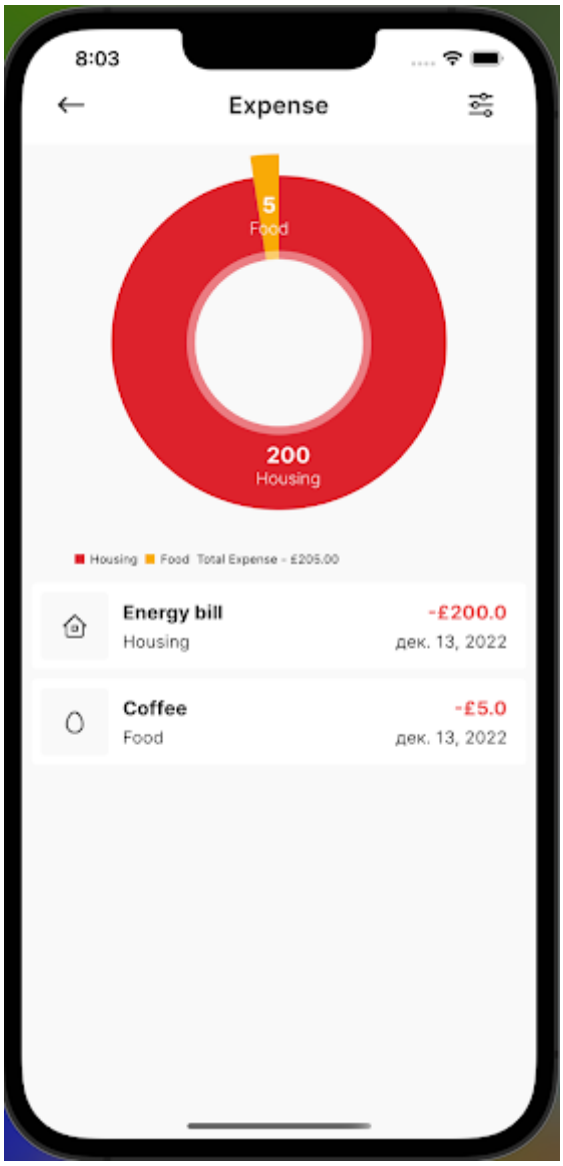
```swift
    func getBiometricType() -> String {
        if #available(iOS 11.0, *) {
            let context = LAContext()
            if context.canEvaluatePolicy(.deviceOwnerAuthenticationWithBiometrics, error: nil) {
                switch context.biometryType {
                    case .faceID: return "Face ID"
                    case .touchID: return "Touch ID"
                    case .none: return "App Lock"
                    @unknown default: return "App Lock"
                }
            }
        }
        return "App Lock"
    }

    func saveCurrency(currency: String) {
        self.currency = currency
        UserDefaults.standard.set(currency, forKey: UD_EXPENSE_CURRENCY)
    }

    func exportTransactions(moc: NSManagedObjectContext) {
        let request = ExpenseCD.fetchRequest()
        var results: [ExpenseCD]
        do {
            results = try moc.fetch(request) as! [ExpenseCD]
            if results.count <= 0 { alertMsg = "No data to export"; showAlert = true }
            else {
                for i in results {
                    let csvModel = ExpenseCSVModel()
                    csvModel.title = i.title ?? ""
                    csvModel.amount = "\(currency)\(i.amount)"
                    csvModel.transactionType = "\(i.type == TRANS_TYPE_INCOME ? "INCOME" : "EXPENSE")"
                    csvModel.tag = getTransTagTitle(transTag: i.tag ?? "")
                    csvModel.occuredOn = "\(getDateFormatter(date: i.occuredOn, format: "yyyy-mm-dd hh:mm a"))"
                    csvModel.note = i.note ?? ""
                    csvModelArr.append(csvModel)
                }
                self.generateCSV()
            }
        } catch { alertMsg = "\(error)"; showAlert = true }
    }

    func generateCSV() {

        let fileName = "Expense.csv"
        let path = NSURL(fileURLWithPath: NSTemporaryDirectory()).appendingPathComponent(fileName)
```

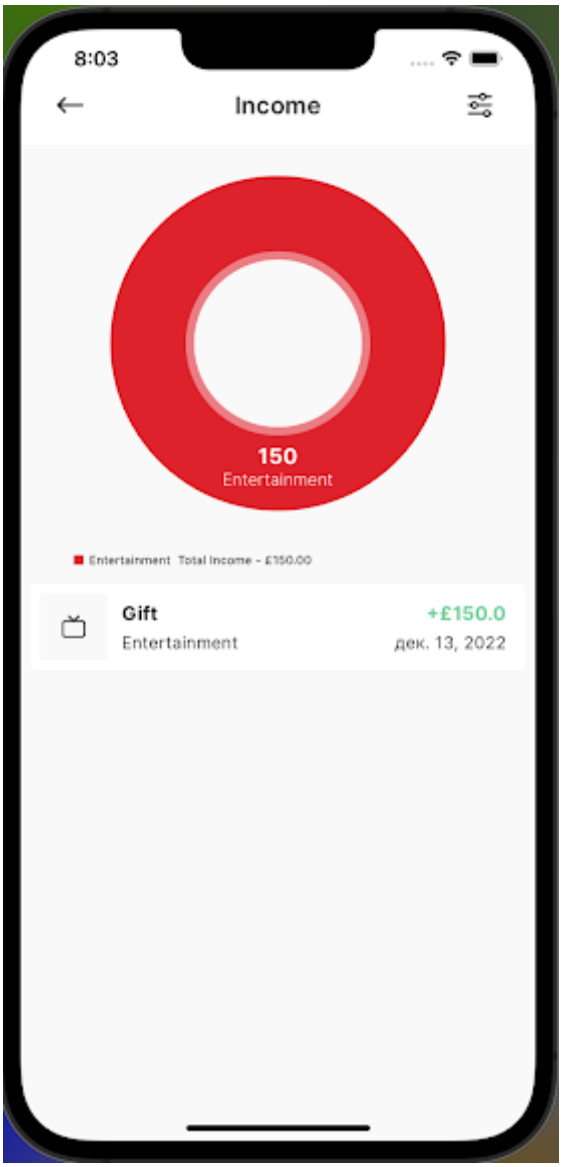
```
                }
                self.generateCSV()
            }
        } catch { alertMsg = "\(error)"; showAlert = true }
    }

    func generateCSV() {

        let fileName = "Expense.csv"
        let path = NSURL(fileURLWithPath: NSTemporaryDirectory()).appendingPathComponent(fileName)
        var csvText = "Title,Amount,Type,Tag,Occured On,Note\n"

        for csvModel in csvModelArr {
            let row =
                "\"\(csvModel.title)\",\"\(csvModel.amount)\",\"\(csvModel.transactionType)\",\"\(csvModel.tag)\",\"\(csvModel.occuredOn)\",\"\(csvModel
                .note)\"\n"
            csvText.append(row)
        }

        do {
            try csvText.write(to: path!, atomically: true, encoding: String.Encoding.utf8)
            let av = UIActivityViewController(activityItems: [path!], applicationActivities: nil)
            UIApplication.shared.windows.first?.rootViewController?.present(av, animated: true, completion: nil)
        } catch { alertMsg = "\(error)"; showAlert = true }

        print(path ?? "not found")
    }

    deinit {
        cancellableBiometricTask = nil
    }
}
```

8) 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.**

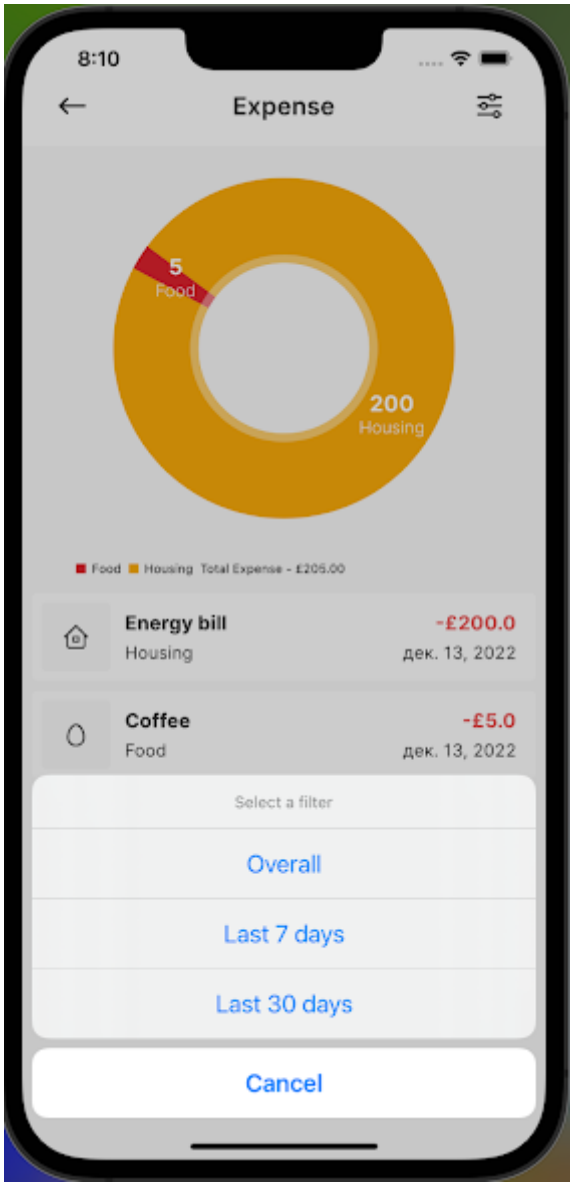


On this page, you can see 3 boxes at the top that describes each type of transaction and the total amount and at the bottom a list of transactions, each of which can be edited further. By clicking on one of the transaction categories, you can notice a pie chart that highlights the amount of each transaction


8:03
Expense
5
Food
200
Housing
Housing Food Total Expense - £205.00
Energy bill
Housing
-£200.0
дек. 13, 2022
Coffee
Food
-£5.0
дек. 13, 2022


8:03
Income
150
Entertainment
Entertainment Total Income - £150.00
Gift
Entertainment
+£150.0
дек. 13, 2022

9) 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)



```
            HStack {
                Spacer()
                Image(isIncome ? "income_icon" : "expense_icon").resizable().frame(width: 40.0, height: 40.0).padding(12)
            }
            HStack{
                TextView(text: isIncome ? "INCOME" : "EXPENSE", type: .overline).foregroundColor(Color.init(hex: "828282"))
                Spacer()
            }.padding(.horizontal, 12)
            HStack {
                TextView(text: "\(CURRENCY)\(getTotalValue())", type: .h5, lineLimit: 1).foregroundColor(Color.text_primary_color)
                Spacer()
            }.padding(.horizontal, 12)
        }.padding(.bottom, 12).background(Color.secondary_color).cornerRadius(4)
    }
}

struct ExpenseTransView: View {

    @ObservedObject var expenseObj: ExpenseCD
    @AppStorage(UD_EXPENSE_CURRENCY) var CURRENCY: String = ""

    var body: some View {
        HStack {

            NavigationLink(destination: NavigationLazyView(ExpenseFilterView(categTag: expenseObj.tag)), label: {
                Image(getTransTagIcon(transTag: expenseObj.tag ?? ""))
                    .resizable().frame(width: 24, height: 24).padding(16)
                    .background(Color.primary_color).cornerRadius(4)
            })

            VStack(alignment: .leading, spacing: 6) {
                HStack {
                    TextView(text: expenseObj.title ?? "", type: .subtitle_1, lineLimit: 1).foregroundColor(Color.text_primary_color)
                    Spacer()
                    TextView(text: "\(expenseObj.type == TRANS_TYPE_INCOME ? "+" : "-")\(CURRENCY)\(expenseObj.amount)", type: .subtitle_1)
                        .foregroundColor(expenseObj.type == TRANS_TYPE_INCOME ? Color.main_green : Color.main_red)
                }
                HStack {
                    TextView(text: getTransTagTitle(transTag: expenseObj.tag ?? ""), type: .body_2).foregroundColor(Color.text_primary_color)
                    Spacer()
                    TextView(text: getDateFormatter(date: expenseObj.occuredOn, format: "MMM dd, yyyy"), type: .body_2).foregroundColor(Color.text_primary_color)
                }
            }.padding(.leading, 4)

            Spacer()

        }.padding(8).background(Color.secondary_color).cornerRadius(4)
```

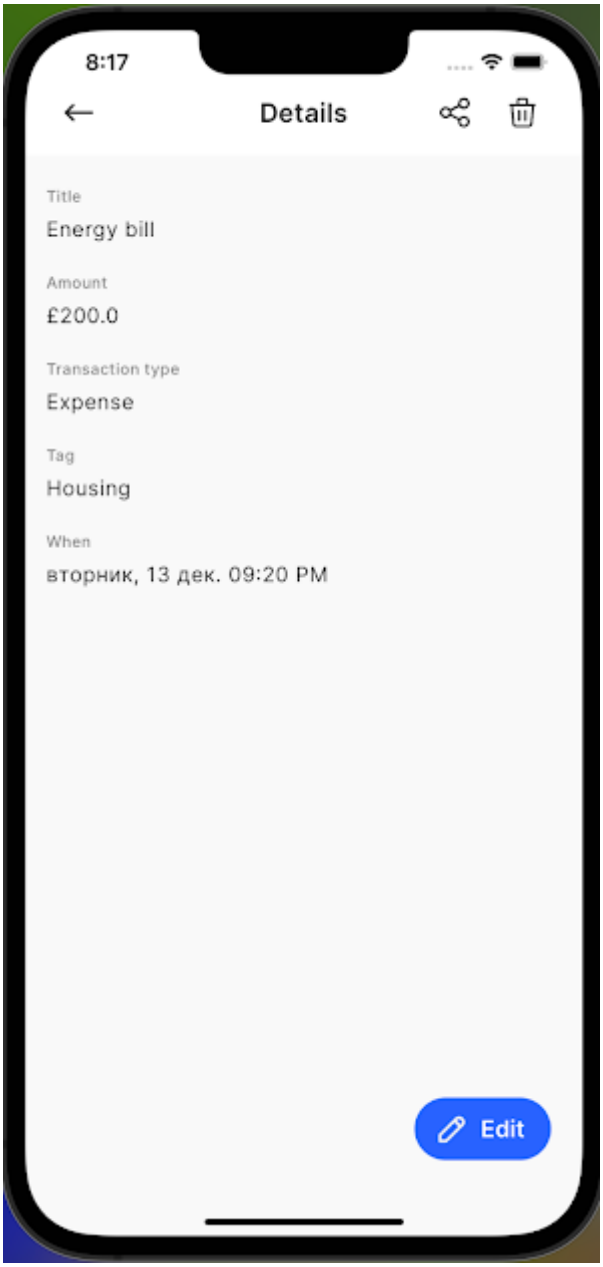
```
//
//  ExpenseView.swift
//  Seagull money tracker
//
//  Created by Chaika Vlad on 07/08/22.

import SwiftUI

struct ExpenseView: View {

    @Environment(\.presentationMode) var presentationMode: Binding<PresentationMode>
    // CoreData
    @Environment(\.managedObjectContext) var managedObjectContext
    @FetchRequest(fetchRequest: ExpenseCD.getAllExpenseData(sortBy: ExpenseCDSort.occuredOn, ascending: false)) var expense: FetchedResults<ExpenseCD>

    @State private var filter: ExpenseCDFilterTime = .all
    @State private var showFilterSheet = false

    @State private var showOptionsSheet = false
    @State private var displayAbout = false
    @State private var displaySettings = false

    var body: some View {
        NavigationView {
            ZStack {
                Color.primary_color.edgesIgnoringSafeArea(.all)

                VStack {
                    NavigationLink(destination: NavigationLazyView(ExpenseSettingsView()), isActive: $displaySettings, label: {})
                    NavigationLink(destination: NavigationLazyView(AboutView()), isActive: $displayAbout, label: {})
                    ToolbarModelView(title: "Transactions", hasBackButt: false, button1Icon: IMAGE_OPTION_ICON, button2Icon: IMAGE_FILTER_ICON) {
                        self.presentationMode.wrappedValue.dismiss() }
                        button1Method: { self.showOptionsSheet = true }
                        button2Method: { self.showFilterSheet = true }
                        .actionSheet(isPresented: $showFilterSheet) {
                            ActionSheet(title: Text("Select a filter"), buttons: [
                                    .default(Text("Overall")) { filter = .all },
                                    .default(Text("Last 7 days")) { filter = .week },
                                    .default(Text("Last 30 days")) { filter = .month },
                                    .cancel()
                            ])
                        }
                    ExpenseMainView(filter: filter)
                        .actionSheet(isPresented: $showOptionsSheet) {
                            ActionSheet(title: Text("Select an option"), buttons: [
                                    .default(Text("About")) { self.displayAbout = true },
                                    .default(Text("Settings")) { self.displaySettings = true },
```

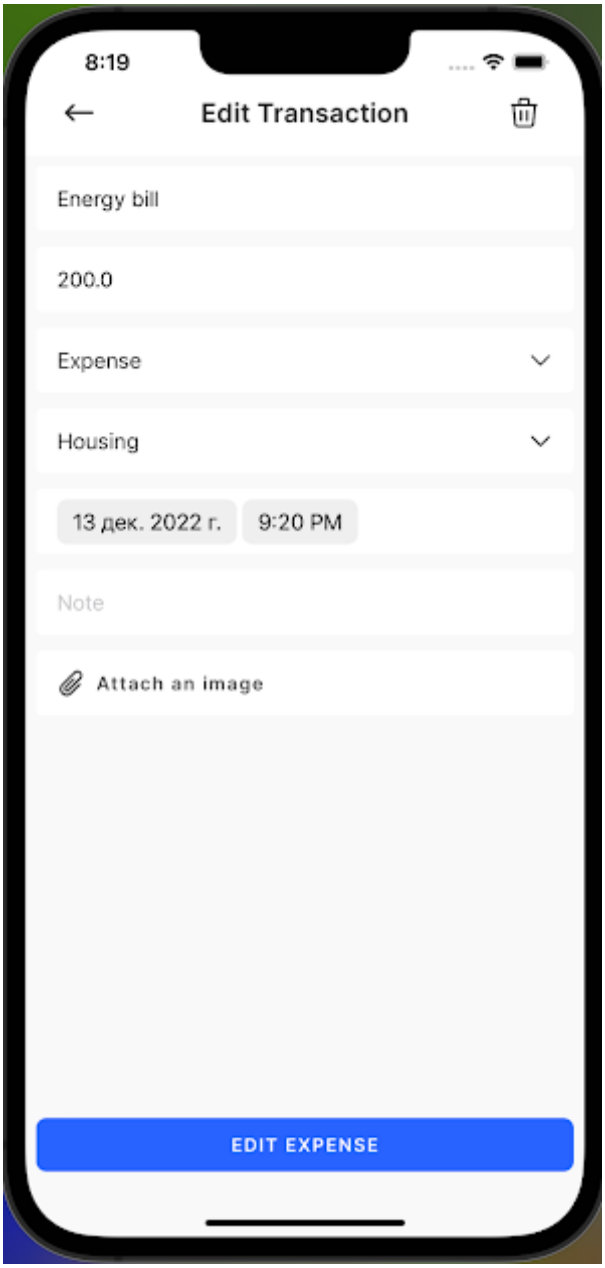
```
                                .default(Text("Settings")) { self.displaySettings = true },
                                .cancel()
                            ])
                        }
                    Spacer()
                }.edgesIgnoringSafeArea(.all)

                VStack {
                    Spacer()
                    HStack {
                        Spacer()
                        NavigationLink(destination: NavigationLazyView(AddExpenseView(viewModel: AddExpenseViewModel())),
                                       label: { Image("plus_icon").resizable().frame(width: 32.0, height: 32.0) })
                        .padding().background(Color.main_color).cornerRadius(35)
                    }
                }.padding()
            }
            .navigationBarHidden(true)
        }
        .navigationViewStyle(StackNavigationViewStyle())
        .navigationBarHidden(true)
        .navigationBarBackButtonHidden(true)
    }
}

struct ExpenseMainView: View {

    var filter: ExpenseCDFilterTime
    var fetchRequest: FetchRequest<ExpenseCD>
    var expense: FetchedResults<ExpenseCD> { fetchRequest.wrappedValue }
    @AppStorage(UD_EXPENSE_CURRENCY) var CURRENCY: String = ""

    init(filter: ExpenseCDFilterTime) {
        let sortDescriptor = NSSortDescriptor(key: "occuredOn", ascending: false)
        self.filter = filter
        if filter == .all {
            fetchRequest = FetchRequest<ExpenseCD>(entity: ExpenseCD.entity(), sortDescriptors: [sortDescriptor])
        } else {
            var startDate: NSDate!
            let endDate: NSDate = NSDate()
            if filter == .week { startDate = Date().getLast7Day()! as NSDate }
            else if filter == .month { startDate = Date().getLast30Day()! as NSDate }
            else { startDate = Date().getLast4Month()! as NSDate }
            let predicate = NSPredicate(format: "occuredOn >= %@ AND occuredOn <= %@", startDate, endDate)
            fetchRequest = FetchRequest<ExpenseCD>(entity: ExpenseCD.entity(), sortDescriptors: [sortDescriptor], predicate: predicate)
        }
    }
```

```
    }

    private func getTotalBalance() -> String {
        var value = Double(0)
        for i in expense {
            if i.type == TRANS_TYPE_INCOME { value += i.amount }
            else if i.type == TRANS_TYPE_EXPENSE { value -= i.amount }
        }
        return "\(String(format: "%.2f", value))"
    }

    var body: some View {

        ScrollView(showsIndicators: false) {

            if fetchRequest.wrappedValue.isEmpty {
                LottieView(animType: .empty_face).frame(width: 300, height: 300)
                VStack {
                    TextView(text: "Lets make new transactions!", type: .h6).foregroundColor(Color.text_primary_color)
                    TextView(text: "Add a transaction and it will show up here", type: .body_1).foregroundColor(Color.text_secondary_color).padding(.top, 2)
                }.padding(.horizontal)
            } else {
                VStack(spacing: 16) {
                    TextView(text: "TOTAL BALANCE", type: .overline).foregroundColor(Color.init(hex: "828282")).padding(.top, 30)
                    TextView(text: "\(CURRENCY)\(getTotalBalance())", type: .h5).foregroundColor(Color.text_primary_color).padding(.bottom, 30)
                }.frame(maxWidth: .infinity).background(Color.secondary_color).cornerRadius(4)

                HStack(spacing: 8) {
                    NavigationLink(destination: NavigationLazyView(ExpenseFilterView(isIncome: true)),
                                   label: { ExpenseModelView(isIncome: true, filter: filter) })
                    NavigationLink(destination: NavigationLazyView(ExpenseFilterView(isIncome: false)),
                                   label: { ExpenseModelView(isIncome: false, filter: filter) })
                }.frame(maxWidth: .infinity)

                Spacer().frame(height: 16)

                HStack {
                    TextView(text: "Recent Transaction", type: .subtitle_1).foregroundColor(Color.text_primary_color)
                    Spacer()
                }.padding(4)

                ForEach(self.fetchRequest.wrappedValue) { expenseObj in
                    NavigationLink(destination: ExpenseDetailedView(expenseObj: expenseObj), label: { ExpenseTransView(expenseObj: expenseObj) })
                }
            }

            Spacer().frame(height: 150)
```

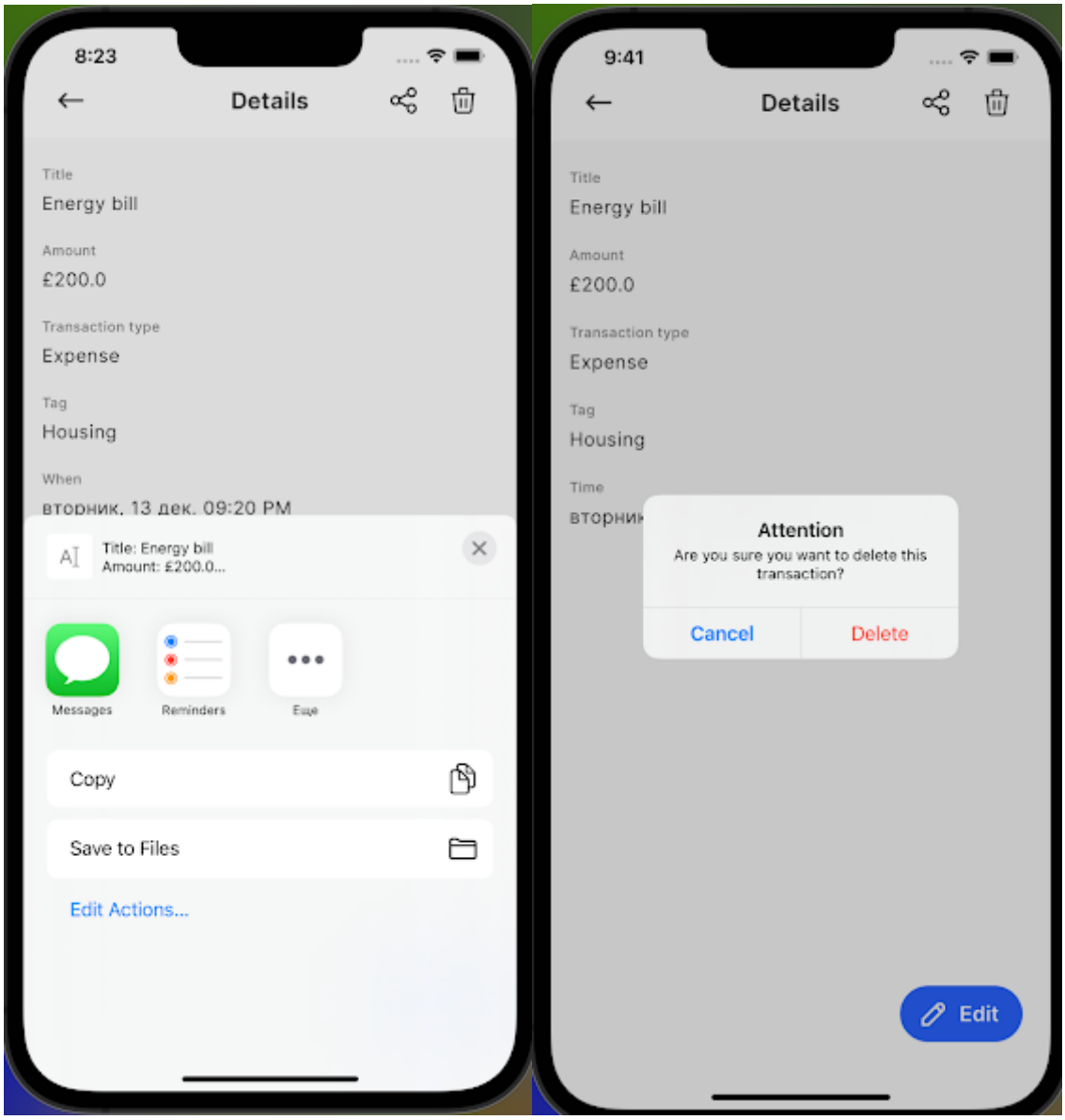
```
            Spacer().frame(height: 150)

        }.padding(.horizontal, 8).padding(.top, 8)
    }
}

struct ExpenseModalView: View {

    var isIncome: Bool
    var type: String
    var fetchRequest: FetchRequest<ExpenseCD>
    var expense: FetchedResults<ExpenseCD> { fetchRequest.wrappedValue }
    @AppStorage(UD_EXPENSE_CURRENCY) var CURRENCY: String = ""

    private func getTotalValue() -> String {
        var value = Double(0)
        for i in expense { value += i.amount }
        return "\(String(format: "%.2f", value))"
    }

    init(isIncome: Bool, filter: ExpenseCDFilterTime, categTag: String? = nil) {
        self.isIncome = isIncome
        self.type = isIncome ? TRANS_TYPE_INCOME : TRANS_TYPE_EXPENSE
        let sortDescriptor = NSSortDescriptor(key: "occuredOn", ascending: false)
        if filter == .all {
            var predicate: NSPredicate!
            if let tag = categTag {
                predicate = NSPredicate(format: "type == %@ AND tag == %@", type, tag)
            } else { predicate = NSPredicate(format: "type == %@", type) }
            fetchRequest = FetchRequest<ExpenseCD>(entity: ExpenseCD.entity(), sortDescriptors: [sortDescriptor], predicate: predicate)
        } else {
            var startDate: NSDate!
            let endDate: NSDate = NSDate()
            if filter == .week { startDate = Date().getLast7Day()! as NSDate }
            else if filter == .month { startDate = Date().getLast30Day()! as NSDate }
            else { startDate = Date().getLast4Month()! as NSDate }
            var predicate: NSPredicate!
            if let tag = categTag {
                predicate = NSPredicate(format: "occuredOn >= %@ AND occuredOn <= %@ AND type == %@ AND tag == %@", startDate, endDate, type, tag)
            } else { predicate = NSPredicate(format: "occuredOn >= %@ AND occuredOn <= %@ AND type == %@", startDate, endDate, type) }
            fetchRequest = FetchRequest<ExpenseCD>(entity: ExpenseCD.entity(), sortDescriptors: [sortDescriptor], predicate: predicate)
        }
    }

    var body: some View {
        VStack(spacing: 12) {
```

10) 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.



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

```swift
//
//  Configs.swift
//  Seagull money tracker
//
//  Created by Chaika Vlad on 07/08/22.
//

import Foundation

// App Globals
let APP_NAME = "Expenso"
let APP_LINK = "https://github.com/sameersyd/Expenso"
let SHARED_FROM_EXPENSO = """
    Shared from \(APP_NAME) App: \(APP_LINK)
    """

// IMAGE_ICON NAMES
let IMAGE_DELETE_ICON = "delete_icon"
let IMAGE_SHARE_ICON = "share_icon"
let IMAGE_FILTER_ICON = "filter_icon"
let IMAGE_OPTION_ICON = "settings_icon"

// User Defaults
let UD_USE_BIOMETRIC = "useBiometric"
let UD_EXPENSE_CURRENCY = "expenseCurrency"

let CURRENCY_LIST = ["₹", "$", "€", "¥", "£", "¢", "₩"]

// Transaction types
let TRANS_TYPE_INCOME = "income"
let TRANS_TYPE_EXPENSE = "expense"

// Transaction tags
let TRANS_TAG_TRANSPORT = "transport"
let TRANS_TAG_FOOD = "food"
let TRANS_TAG_HOUSING = "housing"
let TRANS_TAG_INSURANCE = "insurance"
let TRANS_TAG_MEDICAL = "medical"
let TRANS_TAG_SAVINGS = "savings"
let TRANS_TAG_PERSONAL = "personal"
let TRANS_TAG_ENTERTAINMENT = "entertainment"
let TRANS_TAG_OTHERS = "others"
let TRANS_TAG_UTILITIES = "utilities"

func getTransTagIcon(transTag: String) -> String {
    switch transTag {
        case TRANS_TAG_TRANSPORT: return "trans_type_transport"
```

```swift
        case TRANS_TAG_FOOD: return "trans_type_food"
        case TRANS_TAG_HOUSING: return "trans_type_housing"
        case TRANS_TAG_INSURANCE: return "trans_type_insurance"
        case TRANS_TAG_MEDICAL: return "trans_type_medical"
        case TRANS_TAG_SAVINGS: return "trans_type_savings"
        case TRANS_TAG_PERSONAL: return "trans_type_personal"
        case TRANS_TAG_ENTERTAINMENT: return "trans_type_entertainment"
        case TRANS_TAG_OTHERS: return "trans_type_others"
        case TRANS_TAG_UTILITIES: return "trans_type_utilities"
        default: return "trans_type_others"
    }
}

func getTransTagTitle(transTag: String) -> String {
    switch transTag {
        case TRANS_TAG_TRANSPORT: return "Transport"
        case TRANS_TAG_FOOD: return "Food"
        case TRANS_TAG_HOUSING: return "Housing"
        case TRANS_TAG_INSURANCE: return "Insurance"
        case TRANS_TAG_MEDICAL: return "Medical"
        case TRANS_TAG_SAVINGS: return "Savings"
        case TRANS_TAG_PERSONAL: return "Personal"
        case TRANS_TAG_ENTERTAINMENT: return "Entertainment"
        case TRANS_TAG_OTHERS: return "Others"
        case TRANS_TAG_UTILITIES: return "Utilities"
        default: return "Unknown"
    }
}

func getDateFormatter(date: Date?, format: String = "yyyy-MM-dd") -> String {
    guard let date = date else { return "" }
    let dateFormatter = DateFormatter()
    dateFormatter.dateFormat = format
    return dateFormatter.string(from: date)
}
```

```swift
//
//  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)
  }
}
```

12) My application in the desk looks like this:


9:58
Команды
Контакты
Файлы
Watch
Seagull Tracker