# 기술 문서: 가계부 앱

**2021111835 가파로브 자혼기르**

## 1. 개요

지금까지 여러가지 가계부 앱들을 사용해봤는데 저에게 완벽히 맞는 것을 찾지 못했습니다. 기능이 한두 가지 부족하거나, 반면에 복잡하고 느린 경우도 있었습니다. 또한 비용이 부담스럽거나 원하는 디자인이 아니라서 만족스러운 앱을 찾지 못했습니다. 그래서 제가 개발자로서 제가 원하는 기능과 사용자들의 니즈를 고려한 가계부 앱을 만들었습니다.

## 2. 주요 기능

### 홈 페이지

- 모든 비용들을 확인할 수 있는 페이지
- 비용 추가 버튼
- 비용 목록(비용 날짜, 카테고리, 제목)
- 비용 제목으로 검색 기능

### 카테고리 페이지

- 카테고리 관리
- 카테고리 목록
- 카테고리 추가

### 비용 보고서 페이지

- 카테고리 별로 비용 보고 제공
- Weekly, Monthly, Yearly 보고서 제공
- 차트로 시각화 기능

## 3. 기술 스택

**iOS 버전**: 17.5

**Xcode 버전**: 15.4

**프로그래밍 언어**: Swift

**UI 프레임워크**: SwiftUI

**이 프로젝트에서는 CoreData 대신 새로운 기술인 SwiftData를 사용했습니다.**

### SwiftData 설명

SwiftData는 Apple이 Xcode 15와 함께 발표한 새로운 데이터 관리 프레임워크입니다. 이 프레임워크는 SwiftUI와 밀접하게 통합되어 있으며, 데이터 모델링, 저장, 검색 등을 손쉽게 처리할 수 있도록 돕습니다.

### 왜 SwiftData를 사용했는가?

**프로젝트에서 SwiftData를 선택한 이유는 다음과 같습니다:**

- **간편한 데이터 모델링:** SwiftData는 Xcode 15와 함께 소개된 새로운 데이터 관리 프레임워크로, 데이터 모델링을 간편하게 처리할 수 있도록 설계되었습니다. @Model 어노테이션을 사용해 데이터를 쉽게 정의하고 관리할 수 있습니다.
- **SwiftUI와의 통합:** SwiftData는 SwiftUI와 원활하게 통합됩니다. 이는 UI와 데이터 간의 상호 작용을 단순화하고, SwiftUI의 강력한 상태 관리 기능을 활용할 수 있게 합니다.
- **코드의 간결함:** SwiftData는 Boilerplate 코드를 줄이고, 데이터 관련 작업을 간결하게 만듭니다. 이는 개발자의 생산성을 높이고, 유지보수성을 향상시킵니다.
- **기능의 확장성**: SwiftData는 CoreData의 기능을 확장하여 더 많은 기능을 제공하며, 기존 CoreData 모델을 쉽게 전환할 수 있습니다.

## 4. 소스 코드 및 프로젝트 관리

**Github 리포지토리**: https://github.com/jaxon93/ExpenseTrackerIOS

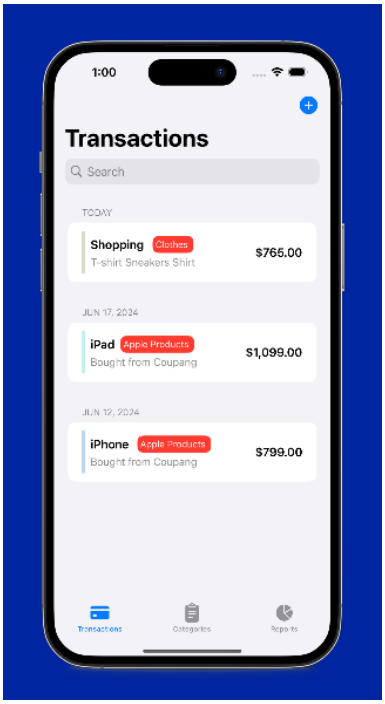
**프로젝트 일정 관리 노션**:
https://www.notion.so/78fa707ef10d416da26a93595f795528?v=ca4dbf152e444dbcbb1c89e4d

aaedea0&pvs=4

## 5. 역할

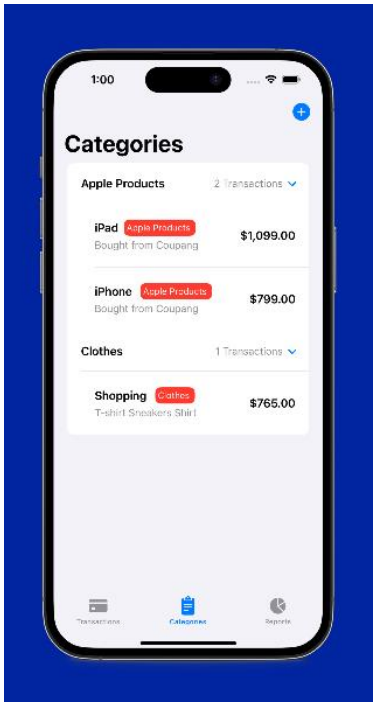
**참여 및 개발 역할:** 가파로브 자혼기르 - 기획, 디자인, 개발 및 테스팅

## 6. 화면 구성 및 구현 내용




### Transactions 화면

- 모든 거래 내역을 확인
- 새로운 거래 내역 추가 버튼
- 거래 내역을 제목으로 검색할 수 있는 기능

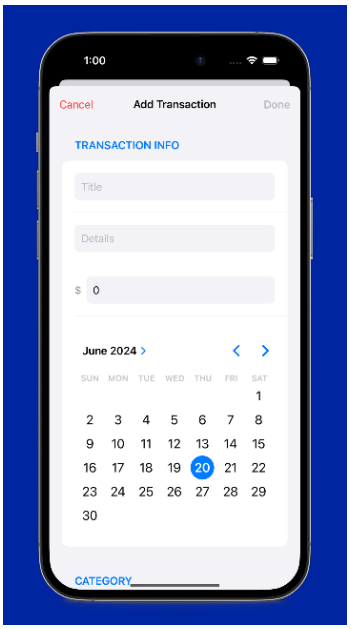



### Categories 화면

- 카테고리별로 거래 내역 확인
- 새로운 카테고리 추가 버튼

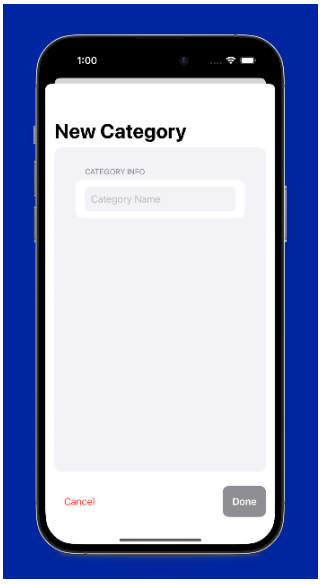
## 새 거래 내역 추가 화면

- 제목 입력 필드
- 세부 정보 입력 필드
- 금액 입력 필드
- 날짜 선택 필드
- 카테고리 선택 필드



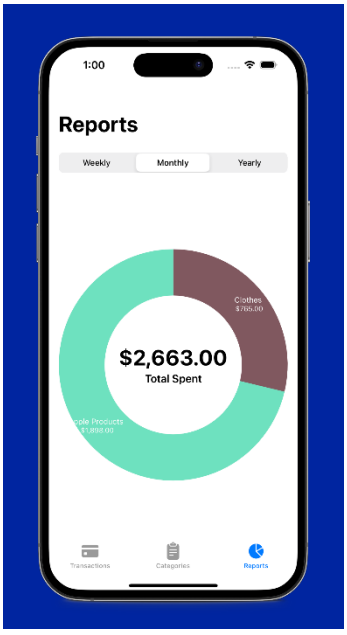

## 새 카테고리 추개 화면

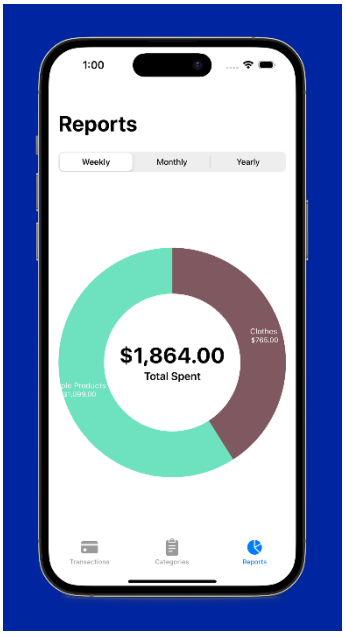
- 새 카테고리 이름 입력 필드




## 보고서 화면

- Weekly, Monthly, Yearly 보고서 제공
- 카테고리별 비용 시각화 차트 제공



## 7. 주요 클래스 설명

### Transaction.swift

Transaction 클래스는 트랜잭션 데이터를 모델링합니다.

- **속성**:
  - title: 트랜잭션 제목 (String)
  - details: 트랜잭션 세부사항 (String)
  - amount: 트랜잭션 금액 (Double)
  - date: 트랜잭션 날짜 (Date)
  - category: 트랜잭션 카테고리 (Category?)
- **메서드**:
  - init(title:details:amount:date:category:): 초기화 메서드
  - currencyFormatted: 트랜잭션 금액을 USD 형식으로 반환하는 속성

```
//
//  Transaction.swift
//  ExpenseTrackerIOS
//
//  Created by jaxon on 6/4/24.
//

import SwiftUI
import SwiftData

@Model
class Transaction {
    var title: String
    var details: String
    var amount: Double
    var date: Date
    var category: Category?

    init(title: String, details: String, amount: Double, date: Date,
category: Category? = nil) {
        self.title = title
        self.details = details
        self.amount = amount
        self.date = date
        self.category = category
```

```
    }

    @Transient
    var currencyFormatted: String {
        if amount.isNaN {
            print("Error: Transaction amount is NaN for transaction:
\(title)")
            return "$0.00"
        }

        let formatter = NumberFormatter()
        formatter.numberStyle = .currency
        formatter.currencyCode = "USD"  // Ensure the currency is formatted
as USD
        return formatter.string(for: amount) ?? "$0.00"
    }
}
```

### Category.swift

Category 클래스는 트랜잭션 카테고리를 모델링합니다.

- **속성**:
  - name: 카테고리 이름 (String)
  - transactions: 카테고리에 속한 트랜잭션 목록 ([Transaction]?)
- **메서드**:
  - init(name:): 초기화 메서드

```
//
//  Category.swift
//  ExpenseTrackerIOS
//
//  Created by jaxon on 6/4/24.
//

import SwiftUI
import SwiftData

@Model
class Category {
    var name: String
    @Relationship(deleteRule: .cascade, inverse: \Transaction.category)
    var transactions: [Transaction]?
```

```
    init(name: String) {
        self.name = name
    }
}
```

**GroupedExpenses.swift**

GroupedTransactions 구조체는 날짜별로 그룹화된 트랜잭션 목록을 나타냅니다.

- **속성**:
    - id: 그룹 ID (UUID)
    - date: 그룹 날짜 (Date)
    - transactions: 트랜잭션 목록 ([Transaction])
- **메서드**:
    - displayTitle: 날짜를 읽기 쉬운 형식으로 반환하는 속성

```
//
//  GroupedExpenses.swift
//  ExpenseTrackerIOS
//
//  Created by jaxon on 6/5/24.
//

import SwiftUI

struct GroupedTransactions: Identifiable {
    var id: UUID = .init()
    var date: Date
    var transactions: [Transaction]

    var displayTitle: String {
        let calendar = Calendar.current
        if calendar.isDateInToday(date) {
            return "Today"
        } else if calendar.isDateInYesterday(date) {
            return "Yesterday"
        } else {
            return date.formatted(date: .abbreviated, time: .omitted)
        }
    }
}
```

**AddTransactionView.swift**

AddTransactionView 구조체는 새로운 트랜잭션을 추가하기 위한 뷰입니다.

- **상태 변수**:
  - title, details, amount, date, category: 트랜잭션 정보 입력을 위한 변수들
  - allCategories: 모든 카테고리 목록
- **UI 구성 요소**:
  - Form: 트랜잭션 정보를 입력하는 폼
  - Picker: 카테고리 선택을 위한 피커
  - Button: 트랜잭션 추가를 위한 버튼

```
//
//  AddTransactionView.swift
//  ExpenseTrackerIOS
//
//  Created by jaxon on 6/5/24.
//

import SwiftUI
import SwiftData

struct AddTransactionView: View {
    @Environment(\.modelContext) private var context
    @Environment(\.dismiss) private var dismiss

    @Query(animation: .snappy) private var allCategories: [Category]

    @State private var title: String = ""
    @State private var details: String = ""
    @State private var amount: Double = 0.0
    @State private var date: Date = .init()
    @State private var category: Category?

    var body: some View {
        NavigationStack {
            Form {
                Section(header: Text("Transaction
Info").font(.headline).foregroundColor(.blue)) {
                    TextField("Title", text: $title)
                        .padding(10)
                        .background(Color(.secondarySystemBackground))
                        .cornerRadius(8)

                    TextField("Details", text: $details)
                        .padding(10)
```

```
                        .background(Color(.secondarySystemBackground))
                        .cornerRadius(8)

                    HStack {
                        Text("$")
                            .foregroundColor(.gray)
                        TextField("Amount", value: $amount, format: .number)
                            .keyboardType(.decimalPad)
                            .padding(10)
                            .background(Color(.secondarySystemBackground))
                            .cornerRadius(8)
                    }

                    DatePicker("Date", selection: $date, displayedComponents:
[.date])
                        .datePickerStyle(.graphical)
                }
                .padding(.vertical, 8)

                Section(header:
Text("Category").font(.headline).foregroundColor(.blue)) {
                    Picker("Category", selection: $category) {
                        Text("Uncategorized")
                            .tag(nil as Category?)
                        ForEach(allCategories) { category in
                            Text(category.name)
                                .tag(Optional(category))
                        }
                    }
                    .pickerStyle(MenuPickerStyle())
                    .padding(10)
                    .background(Color(.secondarySystemBackground))
                    .cornerRadius(8)
                }
                .padding(.vertical, 8)
            }
            .navigationTitle("Add Transaction")
            .navigationBarTitleDisplayMode(.inline)
            .toolbar {
                ToolbarItem(placement: .navigationBarLeading) {
                    Button("Cancel") {
                        dismiss()
                    }
                    .foregroundColor(.red)
                }
                ToolbarItem(placement: .navigationBarTrailing) {
                    Button("Done") {
                        let newTransaction = Transaction(
                            title: title,
                            details: details,
```

```
                            amount: amount,
                            date: date,
                            category: category
                        )
                        context.insert(newTransaction)
                        dismiss()
                    }
                    .bold()
                    .disabled(title.isEmpty || amount <= 0)
                }
            }
        }
    }
}

#Preview {
    AddTransactionView()
}
```

### CategoriesView.swift

CategoriesView 구조체는 트랜잭션 카테고리를 관리하기 위한 뷰입니다.

- **상태 변수**:
  - addCategory, categoryName, deleteAlert, categoryToDelete: 카테고리 추가 및 삭제 관련 상태 변수들
  - allCategories: 모든 카테고리 목록
- **UI 구성 요소**:
  - List: 카테고리 목록
  - Button: 카테고리 추가 버튼
  - alert: 카테고리 삭제 확인 알림

```
            //

//  CategoriesView.swift
//  ExpenseTrackerIOS
//
//  Created by jaxon on 6/4/24.
//

import SwiftUI
import SwiftData

struct CategoriesView: View {
```

```
    @Query(animation: .snappy) private var allCategories: [Category]
    @Environment(\.modelContext) private var context

    @State private var addCategory: Bool = false
    @State private var categoryName: String = ""
    @State private var deleteAlert: Bool = false
    @State private var categoryToDelete: Category?

    var body: some View {
        NavigationStack {
            List {
                ForEach(allCategories.sorted(by: {
                    ($0.transactions?.count ?? 0) >
($1.transactions?.count ?? 0)
                })) { category in
                    DisclosureGroup {
                        if let transactions =
category.transactions, !transactions.isEmpty {
                            ForEach(transactions) { transaction in
                                TransactionCardView(transaction: transaction,
showTag: false)
                                    .padding(.vertical, 4)
                            }
                        } else {
                            HStack {
                                Spacer()
                                Text("No Transactions")
                                    .foregroundColor(.gray)
                                Spacer()
                            }
                            .padding(.vertical, 8)
                        }
                    } label: {
                        HStack {
                            Text(category.name)
                                .font(.headline)
                            Spacer()
                            Text("\(category.transactions?.count ?? 0)
Transactions")
                                .font(.subheadline)
                                .foregroundColor(.secondary)
                        }
                        .padding(.vertical, 8)
                    }
                    .swipeActions(edge: .trailing, allowsFullSwipe: false) {
                        Button {
                            deleteAlert.toggle()
                            categoryToDelete = category
                        } label: {
                            Image(systemName: "trash")
```

```
                        }
                        .tint(.red)
                    }
                }
            }
            .listStyle(InsetGroupedListStyle())
            .navigationTitle("Categories")
            .overlay {
                if allCategories.isEmpty {
                    VStack {
                        Label("No Categories", systemImage: "tray.fill")
                            .font(.title)
                            .foregroundColor(.gray)
                        Text("Add a new category to get started.")
                            .foregroundColor(.gray)
                    }
                }
            }
            .toolbar {
                ToolbarItem(placement: .navigationBarTrailing) {
                    Button {
                        addCategory.toggle()
                    } label: {
                        Image(systemName: "plus.circle.fill")
                            .font(.title3)
                    }
                }
            }
            .sheet(isPresented: $addCategory) {
                categoryName = ""
            } content: {
                addCategoryView
                    .interactiveDismissDisabled()
            }
            .alert("Are you sure you want to delete this category?",
isPresented: $deleteAlert) {
                Button("Delete", role: .destructive) {
                    withAnimation {
                        if let categoryToDelete = categoryToDelete {
                            context.delete(categoryToDelete)
                        }
                    }
                }
                Button("Cancel", role: .cancel) {}
            }
        }
    }

    @ViewBuilder
    var addCategoryView: some View {
```

```
        NavigationStack {
            VStack {
                Form {
                    Section(header: Text("Category Info")) {
                        TextField("Category Name", text: $categoryName)
                            .padding(10)
                            .background(Color(.secondarySystemBackground))
                            .cornerRadius(8)
                    }
                }
                .padding()
                .background(Color(.systemGroupedBackground))
                .cornerRadius(12)
                .padding(.horizontal)

                Spacer()

                HStack {
                    Button("Cancel") {
                        addCategory.toggle()
                    }
                    .padding()
                    .foregroundColor(.red)

                    Spacer()

                    Button("Done") {
                        let newCategory = Category(name: categoryName)
                        context.insert(newCategory)
                        addCategory.toggle()
                    }
                    .padding()
                    .bold()
                    .disabled(categoryName.isEmpty)
                    .background(categoryName.isEmpty ? Color.gray :
Color.blue)
                    .foregroundColor(.white)
                    .cornerRadius(10)
                }
                .padding()
            }
            .navigationTitle("New Category")
        }
    }
}

#Preview {
    CategoriesView()
}
```

**EditExpenseView.swift**

EditExpenseView 구조체는 트랜잭션을 편집하기 위한 뷰입니다.

- **상태 변수**:
  - title, subTitle, amount, date: 트랜잭션 정보 편집을 위한 변수들
  - showErrorAlert, errorMessage: 에러 알림 관련 변수들
- **UI 구성 요소**:
  - List: 트랜잭션 정보를 편집하기 위한 리스트
  - Button: 저장 및 취소 버튼
  - alert: 에러 알림

```
            //
//  EditExpenseView.swift
//  ExpenseTrackerIOS
//
//  Created by jaxon on 6/7/24.
//

import SwiftUI

struct EditExpenseView: View {
    @Environment(\.modelContext) private var context
    @State private var title: String
    @State private var subTitle: String
    @State private var amount: Double
    @State private var date: Date
    var expense: Expense

    @State private var showErrorAlert: Bool = false
    @State private var errorMessage: String = ""

    init(expense: Expense) {
        self.expense = expense
        _title = State(initialValue: expense.title)
        _subTitle = State(initialValue: expense.subTitle)
        _amount = State(initialValue: expense.amount)
        _date = State(initialValue: expense.date)
    }

    var body: some View {
        NavigationStack {
            List {
```

```
                Section("Details") {
                    TextField("Title", text: $title)
                    TextField("Subtitle", text: $subTitle)
                    TextField("Amount", value: $amount,
format: .currency(code: "USD"))
                    DatePicker("Date", selection: $date,
displayedComponents: .date)
                }
            }
            .navigationTitle("Edit Expense")
            .toolbar {
                ToolbarItem(placement: .cancellationAction) {
                    Button("Cancel") {
                        context.rollback()
                    }
                }
                ToolbarItem(placement: .confirmationAction) {
                    Button("Save") {
                        do {
                            expense.title = title
                            expense.subTitle = subTitle
                            expense.amount = amount
                            expense.date = date
                            try context.save()
                        } catch {
                            errorMessage = error.localizedDescription
                            showErrorAlert = true
                        }
                    }
                }
            }
            .alert(isPresented: $showErrorAlert) {
                Alert(title: Text("Error"), message: Text(errorMessage),
dismissButton: .default(Text("OK")))
            }
        }
    }
}
```

**ReportsView.swift**

ReportsView 구조체는 트랜잭션 보고서를 생성하기 위한 뷰입니다.

- **상태 변수**:
    - selectedReportType, categoryColors: 보고서 유형 및 카테고리 색상 관련 변수들

    - allTransactions: 모든 트랜잭션 목록
- **UI 구성 요소**:
    - Picker: 보고서 유형 선택기
    - Chart: 차트 데이터를 시각화하는 차트
    - generateChartData: 차트 데이터를 생성하는 메서드
    - assignColor: 카테고리에 색상을 할당하는 메서드

```
            //

// ReportsView.swift
// ExpenseTrackerIOS
//
// Created by jaxon on 6/7/24.
//

import SwiftUI
import Charts
import SwiftData

struct ReportsView: View {
    @Query(sort: [
        SortDescriptor(\Transaction.date, order: .reverse)
    ]) private var allTransactions: [Transaction]

    @State private var selectedReportType: ReportType = .monthly
    @State private var categoryColors: [String: Color] = [:]

    var body: some View {
        NavigationStack {
            VStack {
                Picker("Select Report", selection: $selectedReportType) {
                    ForEach(ReportType.allCases, id: \.self) { reportType in
                        Text(reportType.rawValue.capitalized).tag(reportType)
                    }
                }
                .pickerStyle(SegmentedPickerStyle())
                .padding()

                if let chartData = generateChartData() {
                    ZStack {
                        Chart {
                            ForEach(chartData) { data in
                                SectorMark(
                                    angle: .value("Value", data.value),
                                    innerRadius: .ratio(0.6),
                                    outerRadius: .ratio(1.0)
```

```
                                )
                                .foregroundStyle(categoryColors[data.category
, default: assignColor(to: data.category)])
                                .annotation(position: .overlay) {
                                    VStack {
                                        Text(data.category)
                                            .font(.caption)
                                            .foregroundColor(.white)
                                        Text(data.value.formatted(.currency(c
ode: "USD")))
                                            .font(.caption2)
                                            .foregroundColor(.white)
                                    }
                                }
                            }
                        }
                        .chartLegend(.visible)
                        .padding()

                        VStack {
                            Text(totalSpent(chartData).formatted(.currency(co
de: "USD")))
                                .font(.largeTitle.bold())
                            Text("Total Spent")
                                .font(.headline)
                        }
                    }
                } else {
                    ContentUnavailableView {
                        Label("No Data", systemImage: "tray.fill")
                    }
                }
            }
            .navigationTitle("Reports")
        }
    }

    private func generateChartData() -> [ChartData]? {
        let calendar = Calendar.current
        var filteredTransactions: [Transaction] = []

        switch selectedReportType {
        case .weekly:
            let weekAgo = calendar.date(byAdding: .day, value: -7, to:
Date())!
            filteredTransactions = allTransactions.filter { $0.date >=
weekAgo }
        case .monthly:
            let monthAgo = calendar.date(byAdding: .month, value: -1, to:
Date())!
```

```
            filteredTransactions = allTransactions.filter { $0.date >=
monthAgo }
        case .yearly:
            let yearAgo = calendar.date(byAdding: .year, value: -1, to:
Date())!
            filteredTransactions = allTransactions.filter { $0.date >=
yearAgo }
        }

        guard !filteredTransactions.isEmpty else { return nil }

        let categoryTotals = Dictionary(grouping: filteredTransactions, by:
{ $0.category?.name ?? "Uncategorized" })
            .mapValues { expenses in
                expenses.reduce(0) { $0 + $1.amount }
            }

        return categoryTotals.map { ChartData(category: $0.key, value:
$0.value) }
    }

    private func totalSpent(_ chartData: [ChartData]) -> Double {
        return chartData.reduce(0) { $0 + $1.value }
    }

    private func assignColor(to category: String) -> Color {
        if let existingColor = categoryColors[category] {
            return existingColor
        } else {
            let newColor = Color.random
            categoryColors[category] = newColor
            return newColor
        }
    }
}

extension Color {
    static var random: Color {
        return Color(
            red: Double.random(in: 0...1),
            green: Double.random(in: 0...1),
            blue: Double.random(in: 0...1)
        )
    }
}

enum ReportType: String, CaseIterable {
    case weekly, monthly, yearly
}
```

```
struct ChartData: Identifiable {
    var id: String { category }
    let category: String
    let value: Double
}

#Preview {
    ReportsView()
}
```

### TransactionCardView.swift

TransactionCardView 구조체는 개별 트랜잭션을 표시하기 위한 뷰입니다.

- **속성**:
  - transaction: 트랜잭션 데이터
  - showTag: 태그 표시 여부
- **UI 구성 요소**:
  - HStack: 트랜잭션 정보를 표시하는 수평 스택
  - Text: 트랜잭션 제목 및 세부사항 텍스트

```
           //

//  TransactionCardView.swift
//  ExpenseTrackerIOS
//
//  Created by jaxon on 6/6/24.
//

import SwiftUI

struct TransactionCardView: View {
    @Bindable var transaction: Transaction
    var showTag: Bool = true

    var body: some View {
        HStack {
            if showTag {
                Rectangle()
                    .fill(Color.random.opacity(0.3))
                    .frame(width: 4)
            }
            VStack(alignment: .leading, spacing: 4) {
                HStack {
                    Text(transaction.title)
```

```
                        .font(.headline)

                    // Display category if available
                    if let category = transaction.category {
                        Text(category.name)
                            .font(.footnote)
                            .foregroundColor(.white)
                            .padding(4)
                            .background(Color.red)
                            .cornerRadius(8)
                    }
                }
                Text(transaction.details)
                    .font(.subheadline)
                    .foregroundColor(.gray)
            }
            Spacer()
            if !transaction.amount.isNaN {
                Text(transaction.currencyFormatted)
                    .font(.headline)
            } else {
                Text("$0.00")
                    .font(.headline)
                    .foregroundColor(.red)
            }
        }
        .padding(.vertical, 8)
    }
}
```

**TransactionView.swift**

TransactionsView 구조체는 모든 트랜잭션을 관리하고 표시하기 위한 뷰입니다.

- **상태 변수**:
  - groupedTransactions, originalGroupedTransactions: 그룹화된 트랜잭션 목록
  - addTransaction, searchQuery: 트랜잭션 추가 및 검색 관련 변수들
  - allTransactions: 모든 트랜잭션 목록
- **UI 구성 요소**:
  - List: 트랜잭션 목록
  - Button: 트랜잭션 추가 버튼
  - searchable: 트랜잭션 검색 필드
- **메서드**:

- filterTransactions: 트랜잭션을 검색어에 따라 필터링
- groupTransactions: 트랜잭션을 날짜별로 그룹화

```
        //

//  TransactionView.swift
//  ExpenseTrackerIOS
//
//  Created by jaxon on 6/4/24.
//
import SwiftUI
import SwiftData

struct TransactionsView: View {
    @Binding var selectedTab: String
    @Query(sort: [
        SortDescriptor(\Transaction.date, order: .reverse)
    ], animation: .snappy) private var allTransactions: [Transaction]
    @Environment(\.modelContext) private var context

    @State private var groupedTransactions: [GroupedTransactions] = []
    @State private var originalGroupedTransactions: [GroupedTransactions] =
[]
    @State private var addTransaction: Bool = false
    @State private var searchQuery: String = ""

    var body: some View {
        NavigationStack {
            List {
                ForEach($groupedTransactions) { $group in
                    Section(group.displayTitle) {
                        ForEach(group.transactions) { transaction in
                            TransactionCardView(transaction: transaction)
                                .swipeActions(edge: .trailing,
allowsFullSwipe: false) {
                                    Button {
                                        context.delete(transaction)
                                        withAnimation {
                                            group.transactions.removeAll(wher
e: { $0.id == transaction.id })

                                            if group.transactions.isEmpty {
                                                groupedTransactions.removeAll
(where: { $0.id == group.id })

                                            }
                                        }
                                    } label: {
                                        Image(systemName: "trash")
                                    }
                                    .tint(.red)
```

```
                                }
                            }
                        }
                    }
                }
                .navigationTitle("Transactions")
                .searchable(text: $searchQuery, placement: .navigationBarDrawer,
prompt: Text("Search"))
                .overlay {
                    if allTransactions.isEmpty || groupedTransactions.isEmpty {
                        ContentUnavailableView {
                            Label("No Transactions", systemImage: "tray.fill")
                        }
                    }
                }
                .toolbar {
                    ToolbarItem(placement: .topBarTrailing) {
                        Button {
                            addTransaction.toggle()
                        } label: {
                            Image(systemName: "plus.circle.fill")
                                .font(.title3)
                        }
                    }
                }
            }
            .onChange(of: searchQuery, initial: false) { oldValue, newValue in
                if !newValue.isEmpty {
                    filterTransactions(newValue)
                } else {
                    groupedTransactions = originalGroupedTransactions
                }
            }
            .onChange(of: allTransactions, initial: true) { oldValue, newValue in
                if newValue.count > oldValue.count || groupedTransactions.isEmpty
|| selectedTab == "Categories" {
                    groupTransactions(newValue)
                }
            }
            .sheet(isPresented: $addTransaction) {
                AddTransactionView()
                    .interactiveDismissDisabled()
            }
        }

        func filterTransactions(_ text: String) {
            Task.detached(priority: .high) {
                let query = text.lowercased()
                let filteredTransactions = originalGroupedTransactions.compactMap
{ group -> GroupedTransactions? in
```

```
                let filtered = group.transactions.filter { transaction in
                    return transaction.title.lowercased().contains(query)
                }
                if filtered.isEmpty {
                    return nil
                }
                return GroupedTransactions(date: group.date, transactions:
filtered)
            }

            await MainActor.run {
                groupedTransactions = filteredTransactions
            }
        }
    }

    func groupTransactions(_ transactions: [Transaction]) {
        Task.detached(priority: .high) {
            let groupedDict = Dictionary(grouping: transactions)
{ transaction in
                Calendar.current.dateComponents([.day, .month, .year], from:
transaction.date)
            }

            let sortedDict = groupedDict.sorted { lhs, rhs in
                let date1 = Calendar.current.date(from: lhs.key) ?? Date()
                let date2 = Calendar.current.date(from: rhs.key) ?? Date()
                return date1 > date2
            }

            let newGroupedTransactions = sortedDict.compactMap { dict ->
GroupedTransactions? in
                let date = Calendar.current.date(from: dict.key) ?? Date()
                return GroupedTransactions(date: date, transactions:
dict.value)
            }

            await MainActor.run {
                groupedTransactions = newGroupedTransactions
                originalGroupedTransactions = newGroupedTransactions
            }
        }
    }
}

#Preview {
    MainView()
}
```

### ExpenseTrackerIOSApp.swift

ExpenseTrackerIOSApp 구조체는 애플리케이션의 메인 엔트리 포인트입니다.

- **메서드**:
    - body: 애플리케이션의 메인 뷰를 설정

```
     //
//  ExpenseTrackerIOSApp.swift
//  ExpenseTrackerIOS
//
//  Created by jaxon on 6/4/24.
//

import SwiftUI
import SwiftData

@main
struct ExpenseTrackerIOSApp: App {
    var body: some Scene {
        WindowGroup {
            MainView()
        }
        .modelContainer(for: [Transaction.self, Category.self])
    }
}
```

### MainView.swift

MainView 구조체는 애플리케이션의 메인 탭 뷰입니다.

- **상태 변수**:
    - selectedTab: 선택된 탭
- **UI 구성 요소**:
    - TabView: 트랜잭션, 카테고리, 보고서 탭을 포함한 탭 뷰

```
   //
//  MainView.swift
//  ExpenseTrackerIOS
//
//  Created by jaxon on 6/4/24.
//
```

```swift
import SwiftUI
import SwiftData

struct MainView: View {
    @State private var selectedTab: String = "Transactions"
    var body: some View {
        TabView(selection: $selectedTab) {
            TransactionsView(selectedTab: $selectedTab)
                .tag("Transactions")
                .tabItem {
                    Image(systemName: "creditcard.fill")
                    Text("Transactions")
                }
            CategoriesView()
                .tag("Categories")
                .tabItem {
                    Image(systemName: "list.clipboard.fill")
                    Text("Categories")
                }
            ReportsView()
                .tag("Reports")
                .tabItem {
                    Image(systemName: "chart.pie.fill")
                    Text("Reports")
                }
        }
    }
}

#Preview {
    MainView()
}
```

## 8. 프로젝트 결과물

- 동영상 시연: https://www.youtube.com/watch?v=BhQAHCpfbd8

## 9. 향후 계획

- 새로운 기능 추가
- 사용자 피드백을 반영한 지속적인 개선
- 개인적으로 사용할 계획