QuickBudget Report

Team Members: Lucia Jayne and Arthur May

State of QuickBudget

The iOS application focuses on displaying, adding, editing and deleting transactions. Due to switching platforms from Java to SwiftUI, the team needed to reduce the complexity of budget limit functionality and show a summary by category of the entire budget rather than monthly/yearly. Additionally, the register and login use cases were removed from the project.

In the proposal, the team suggested the use of .csv files to store data locally. However, to enhance the project, the final application uses Core Data instead of a .csv, which creates a SQLite database to store the data permanently on the device. Lastly, storing the transactions, summaries, and budget categories in a database adds complexity to the display, add, edit and remove transaction functions. At the beginning, the information was only stored temporarily on arrays.

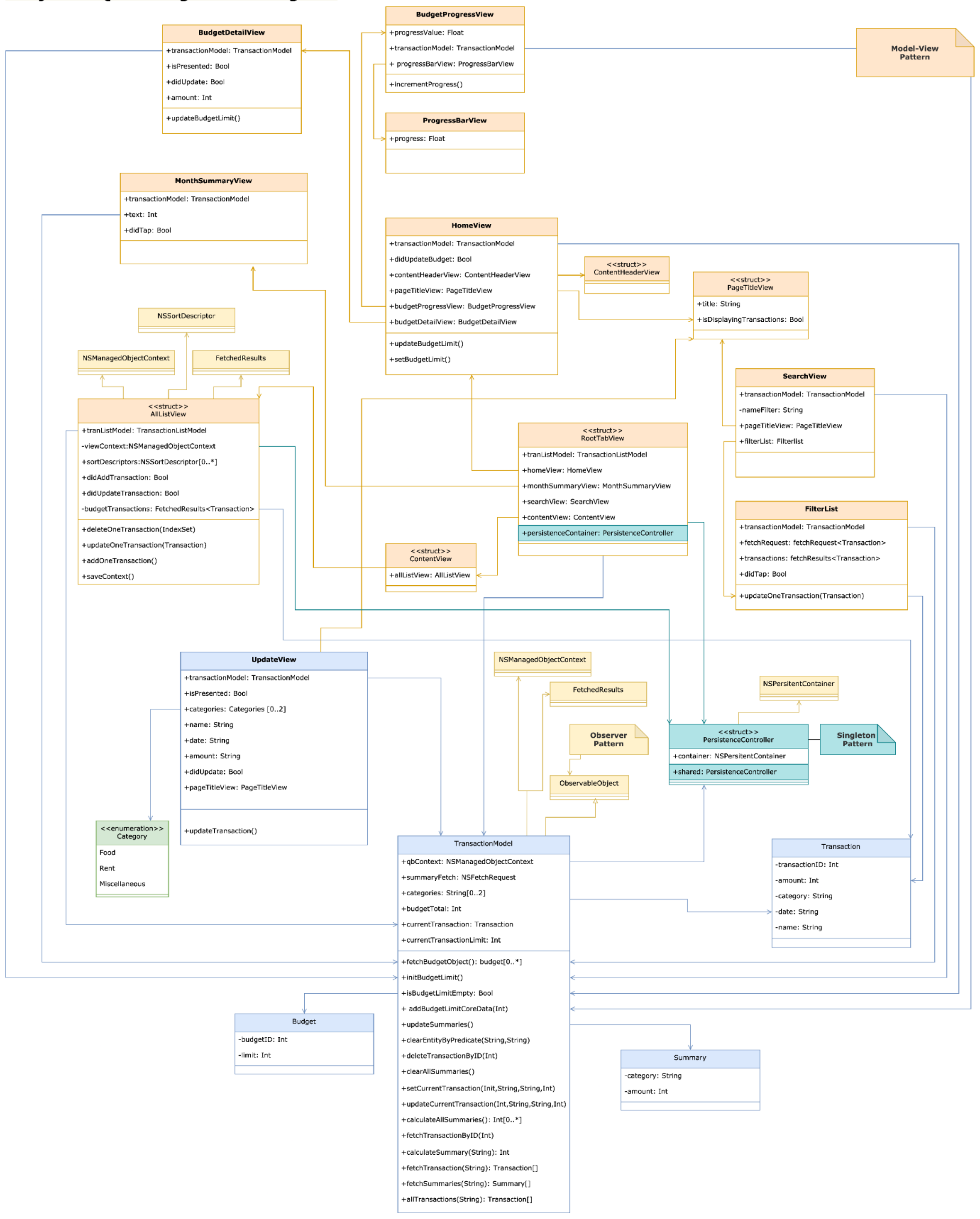
Use Cases	**Project 5** (Proposal)	**Project 6**	**Project 7** (Final)
Add Transaction	Save transaction details: date, category, name, and amount.	Saves transactions in CoreData.	✓
Search Transactions	Display a list of transactions based on specific keywords.	Searches for specific transactions based on their name attribute.	✓

Delete Transactions	N/A	Deletes the selected transaction from CoreData.	✓
View Monthly Expenses Summary per Category	Display monthly and yearly expenses by category.	User Interface design of monthly and yearly tabs.	Display a single budget summary by category stored in Core Data.
Edit Financial Profile	Edit budget limits for specific categories.	N/A	Set and update the limit of the entire budget in Core Data.
Register and Login users	Verify username and password. Created a new account if hasn't been previously registered	×	×

Class Diagram

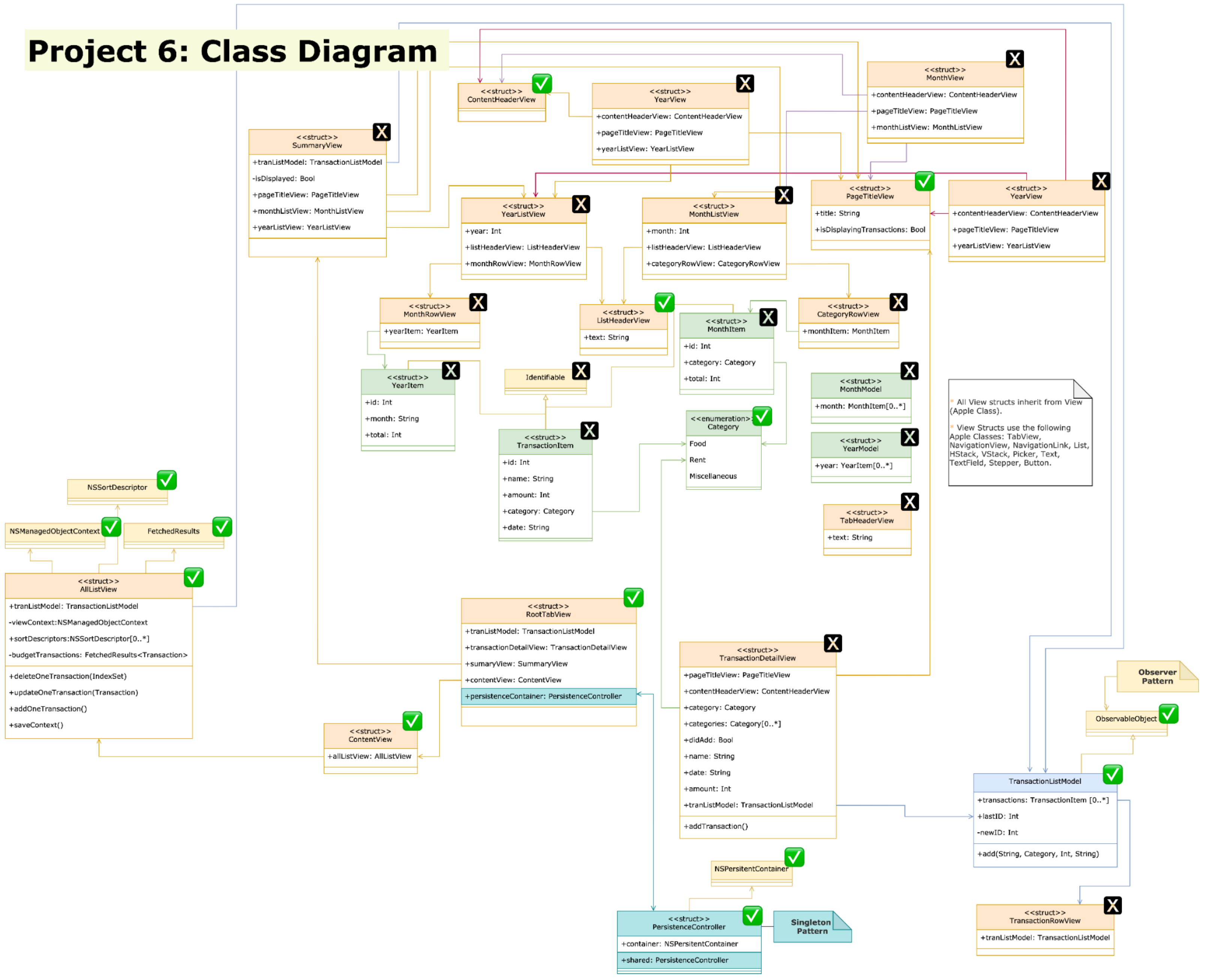
For the last iteration, *Project 7*, several views from *Project 6* were deleted because the summary was no longer divided into months and years, instead it was consolidated in a single summary. Also, most of the functionality has been implemented in the Transaction Model class, which expanded considerably from *Project 6*. For future iterations, it would be best to create a Summary Model class to divide the responsibilities. Lastly, the command pattern was not used in the final iteration because the tabs were managed using TabView provided by SwiftUI.

Project 7: QuickBudget Class Diagram


BudgetDetailView
+transactionModel: TransactionModel
+isPresented: Bool
+didUpdate: Bool
+amount: Int
+updateBudgetLimit()
BudgetProgressView
+progressValue: Float
+transactionModel: TransactionModel
+ progressBarView: ProgressBarView
+incrementProgress()
Model-View Pattern
ProgressBarView
+progress: Float
MonthSummaryView
+transactionModel: TransactionModel
+text: Int
+didTap: Bool
HomeView
+transactionModel: TransactionModel
+didUpdateBudget: Bool
+contentHeaderView: ContentHeaderView
+pageTitleView: PageTitleView
+budgetProgressView: BudgetProgressView
+budgetDetailView: BudgetDetailView
+updateBudgetLimit()
+setBudgetLimit()
<<struct>> ContentHeaderView
<<struct>> PageTitleView
+title: String
+isDisplayingTransactions: Bool
NSSortDescriptor
NSManagedObjectContext
FetchedResults
<<struct>> AllListView
+tranListModel: TransactionListModel
-viewContext:NSManagedObjectContext
+sortDescriptors:NSSortDescriptor[0..*]
+didAddTransaction: Bool
+didUpdateTransaction: Bool
-budgetTransactions: FetchedResults<Transaction>
+deleteOneTransaction(IndexSet)
+updateOneTransaction(Transaction)
+addOneTransaction()
+saveContext()
SearchView
+transactionModel: TransactionModel
-nameFilter: String
+pageTitleView: PageTitleView
+filterList: Filterlist
<<struct>> RootTabView
+tranListModel: TransactionListModel
+homeView: HomeView
+monthSummaryView: MonthSummaryView
+searchView: SearchView
+contentView: ContentView
+persistenceContainer: PersistenceController
FilterList
+transactionModel: TransactionModel
+fetchRequest: fetchRequest<Transaction>
+transactions: fetchResults<Transaction>
+didTap: Bool
+updateOneTransaction(Transaction)
<<struct>> ContentView
+allListView: AllListView
UpdateView
+transactionModel: TransactionModel
+isPresented: Bool
+categories: Categories [0..2]
+name: String
+date: String
+amount: String
+didUpdate: Bool
+pageTitleView: PageTitleView
+updateTransaction()
NSManagedObjectContext
FetchedResults
NSPersitentContainer
Observer Pattern
<<struct>> PersistenceController
+container: NSPersitentContainer
+shared: PersistenceController
Singleton Pattern
ObservableObject
<<enumeration>> Category
Food
Rent
Miscellaneous
TransactionModel
+qbContext: NSManagedObjectContext
+summaryFetch: NSFetchRequest
+categories: String[0..2]
+budgetTotal: Int
+currentTransaction: Transaction
+currentTransactionLimit: Int
+fetchBudgetObject(): budget[0..*]
+initBudgetLimit()
+isBudgetLimitEmpty: Bool
+ addBudgetLimitCoreData(Int)
+updateSummaries()
+clearEntityByPredicate(String,String)
+deleteTransactionByID(Int)
+clearAllSummaries()
+setCurrentTransaction(Init,String,String,Int)
+updateCurrentTransaction(Int,String,String,Int)
+calculateAllSummaries(): Int[0..*]
+fetchTransactionByID(Int)
+calculateSummary(String): Int
+fetchTransaction(String): Transaction[]
+fetchSummaries(String): Summary[]
+allTransactions(String): Transaction[]
Transaction
-transactionID: Int
-amount: Int
-category: String
-date: String
-name: String
Budget
-budgetID: Int
-limit: Int
Summary
-category: String
-amount: Int

5

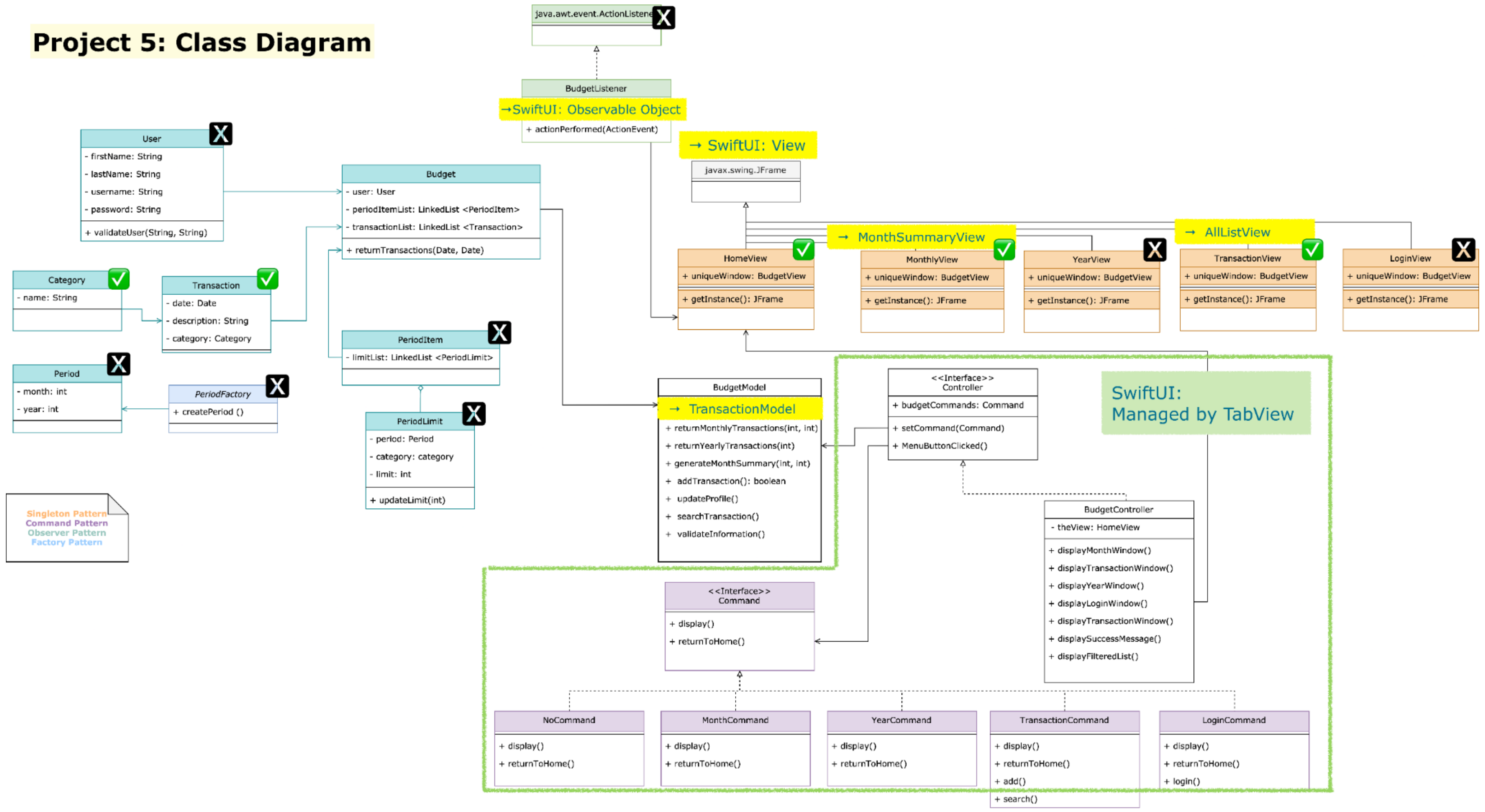
Project 6: Class Diagram



<<struct>> ContentHeaderView
<<struct>> YearView
+contentHeaderView: ContentHeaderView
+pageTitleView: PageTitleView
+yearListView: YearListView
<<struct>> MonthView
+contentHeaderView: ContentHeaderView
+pageTitleView: PageTitleView
+monthListView: MonthListView
<<struct>> SummaryView
+tranListModel: TransactionListModel
-isDisplayed: Bool
+pageTitleView: PageTitleView
+monthListView: MonthListView
+yearListView: YearListView
<<struct>> YearListView
+year: Int
+listHeaderView: ListHeaderView
+monthRowView: MonthRowView
<<struct>> MonthListView
+month: Int
+listHeaderView: ListHeaderView
+categoryRowView: CategoryRowView
<<struct>> PageTitleView
+title: String
+isDisplayingTransactions: Bool
<<struct>> YearView
+contentHeaderView: ContentHeaderView
+pageTitleView: PageTitleView
+yearListView: YearListView
<<struct>> MonthRowView
+yearItem: YearItem
<<struct>> ListHeaderView
+text: String
<<struct>> MonthItem
+id: Int
+category: Category
+total: Int
<<struct>> CategoryRowView
+monthItem: MonthItem
<<struct>> YearItem
+id: Int
+month: String
+total: Int
Identifiable
<<struct>> MonthModel
+month: MonthItem[0..*]
<<enumeration>> Category
Food
Rent
Miscellaneous
<<struct>> TransactionItem
+id: Int
+name: String
+amount: Int
+category: Category
+date: String
<<struct>> YearModel
+year: YearItem[0..*]
<<struct>> TabHeaderView
+text: String
* All View structs inherit from View (Apple Class).
* View Structs use the following Apple Classes: TabView, NavigationView, NavigationLink, List, HStack, VStack, Picker, Text, TextField, Stepper, Button.
NSSortDescriptor
NSManagedObjectContext
FetchedResults
<<struct>> AllListView
+tranListModel: TransactionListModel
-viewContext:NSManagedObjectContext
+sortDescriptors:NSSortDescriptor[0..*]
-budgetTransactions: FetchedResults<Transaction>
+deleteOneTransaction(IndexSet)
+updateOneTransaction(Transaction)
+addOneTransaction()
+saveContext()
<<struct>> RootTabView
+tranListModel: TransactionListModel
+transactionDetailView: TransactionDetailView
+sumaryView: SummaryView
+contentView: ContentView
+persistenceContainer: PersistenceController
<<struct>> ContentView
+allListView: AllListView
<<struct>> TransactionDetailView
+pageTitleView: PageTitleView
+contentHeaderView: ContentHeaderView
+category: Category
+categories: Category[0..*]
+didAdd: Bool
+name: String
+date: String
+amount: Int
+tranListModel: TransactionListModel
+addTransaction()
Observer Pattern
ObservableObject
TransactionListModel
+transactions: TransactionItem [0..*]
+lastID: Int
-newID: Int
+add(String, Category, Int, String)
NSPersitentContainer
<<struct>> PersistenceController
+container: NSPersitentContainer
+shared: PersistenceController
Singleton Pattern
<<struct>> TransactionRowView
+tranListModel: TransactionListModel

Project 7

Project 5: Class Diagram



Third-Party Code

QuickBudget's Core Design:

- Based on chapters 1 to 5 of the LinkedIn Learning course SwiftUI Essential Training.

Features:

- The "Add" and "Delete" features are based on the Youtube video SwiftUI 2.0: Core Data - How To Use Core Data From Scratch (2020)
- The "Search" feature is based on the Youtube video Dynamically filtering @FetchRequest with SwiftUI – Core Data SwiftUI Tutorial

Core Data:

- The code to fully delete the content of an entity in core data was adapted from the article Batch Delete And Delete Everything In Core Data

Progress Bar:

- The code to display a circular bar, showing the percentage remaining on the budget was edited from the article How to build a circular progress bar in SwiftUI

OOAD Process

1. Refactoring Code: SwiftUI makes it easy for developers to manage lists and Core Data entities within the View. However, from OOAD, we understand that it is best practice to maintain those functions in the Model classes. The team needed to gain more understanding of the persistent store functionality to implement the functions such as “update” manually in a separate class, rather than view.
2. Pattern: Observable pattern was useful to automatically update the budget limit and the budget total application wide. Both attributes are published within an observable object.
3. Multi-Threading: The team encountered a multi-threading issue when trying to calculate the budget total after a transaction was deleted. The application automatically updates the budget total after a transaction has been added. However, it encounters multi-threading issues when a transaction is deleted. As a result, the team decided to add a “refresh” button to update the summaries manually.