Appendices

Table of Contents

Appendix A: Discussion with Client before Creating Solution **3**

Appendix B: Response to Questionnaire **5**

Appendix C: Review of Final Product **6**

Appendix D: Code Index **8**

SimplyBudget *57*

Extensions 57

AppDelegate.swift 58

SceneDelegate.swift 60

SimplyBudget / Model / CoreDataObjects *8*

Expenditure+CoreDataClass.swift 8

User+CoreDataClass.swift 8

SimplyBudget / Views / TableScreen *8*

ExpendituresTableTitleView.swift 8

ExpendituresTableCell.swift 10

SimplyBudget / Views / AnalyticsScreen *12*

AnalyticsTitleView.swift 12

AnalyticsTableViewCell.swift 14

SimplyBudget / Views / HomeScreen *16*

HomeTimeRangeView.swift 16

HomeBudgetCounterView.swift 19

SimplyBudget / Views / HomeScreen / HomeCategories *22*

CategoryButton.swift 22

HomeCategoriesView.swift 27

HomeBudgetMenuView.swift 31

SimplyBudget / Views / Controllers *36*

HomeViewController.swift 36

AnalyticsViewController.swift 40

ExpendituresTableViewController.swift 54

Appendix A: Discussion with Client before Creating Solution

Developer: Hello Daniel, I heard you were spending all your allowance, even though the extra $5 per day are for emergencies only. Did you run into any emergencies?

Daniel: No, it’s just that I have a bad habit of misspending. My parents asked me where the money went, and I couldn’t recall exactly how I spent it.

Developer: I see. Do you take your phone with you throughout the day? Maybe you could try using a budgeting app to keep track of all your expenses.

Daniel: I do carry my phone with me almost all the time, and I have tried using a budgeting app. However, all the budgeting apps I could find on the app store, I use an iPhone, are just needlessly complicated. They ask me to connect my credit cards, income sources, and paychecks and then also ask me provide information a lot of expenditures that don’t really apply to me. They ask me to put in information about my rent, utilities, and taxes, and sometimes cannot show me how much I can spend in a day as they usually focus on monthly budgets. Because of this complexity I usually end up not using the apps at all. I couldn’t find an app that only considers my daily budget and lets me track my spending simply, without this extra information which is unnecessary for me.

Developer: Then in that case, I could develop a mobile app for you that only considers your budget and lets you simply track your expenditures. The app could even show you on the home screen how much more you can spend each day, month, and year, to stay within your budget of $10 per day.

Daniel: That would be perfect! Just make sure that it is an app that I can use on my iPhone, as that’s the most convenient way for me to track my spending since I just have to open an app. Also, when I log my expenditures into the app, make sure that the app lets me input the amount I spent, what category of spending that expenditure is, and the app should record what time I logged that expenditure as I will log them right after spending the money. Please make this input process as easy and quick as possible.

Developer: Certainly! Could you tell me more about the category of each spending? Which categories are you referring to?

Daniel: By that I am referring to spending categories such as: Food, Transportation, Healthcare and Miscellaneous expenses. You could also include some categories such as Savings or Housing, as I am going to graduate soon and those will be important for college, but please remember to make it is simple and does not ask for a lot of unnecessary information.

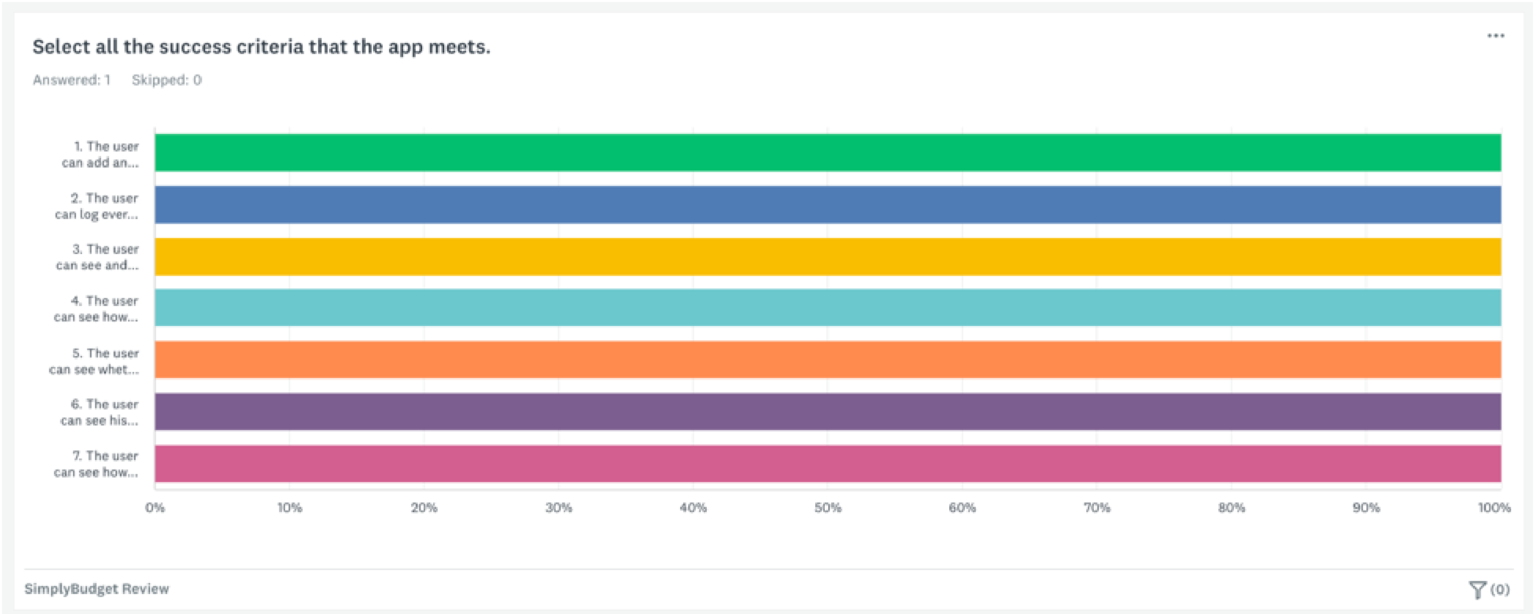
Developer: I understand. Now besides showing you how much you can spend each day, month, or year, is there anything else you'd like the application to do?

Daniel: Yes, make sure that I can also see all the expenditures I've logged onto the app with the information I mentioned previously besides them and sorted chronologically. That way, I can look through them to see how I spent my money at a certain date. And if possible, could you also show me some sort of analytics of my spending in the form of graphs? Like whether I've been spending over or under my budget recently, my total spending each day over time, and my total spending in each category in this month or in this year. That's so that I can understand my spending overall.

Developer: I will definitely show all your expenditures sorted chronologically, and I am sure showing those analytics of your spending in the form of graphs is possible. I will take notes on all the requests you made, are there anymore?

Daniel: With that I think the application will be perfect.

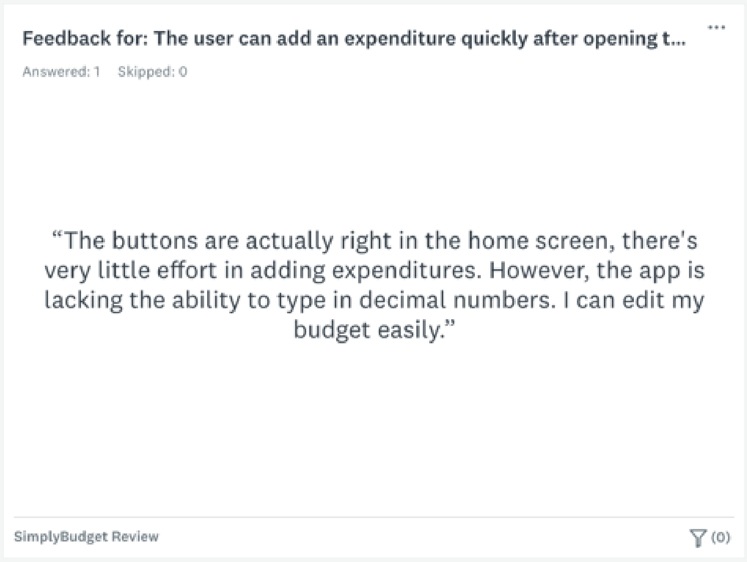
Appendix B: Response to Questionnaire



Select all the success criteria that the app meets.

Answered: 1 Skipped: 0

SimplyBudget Review

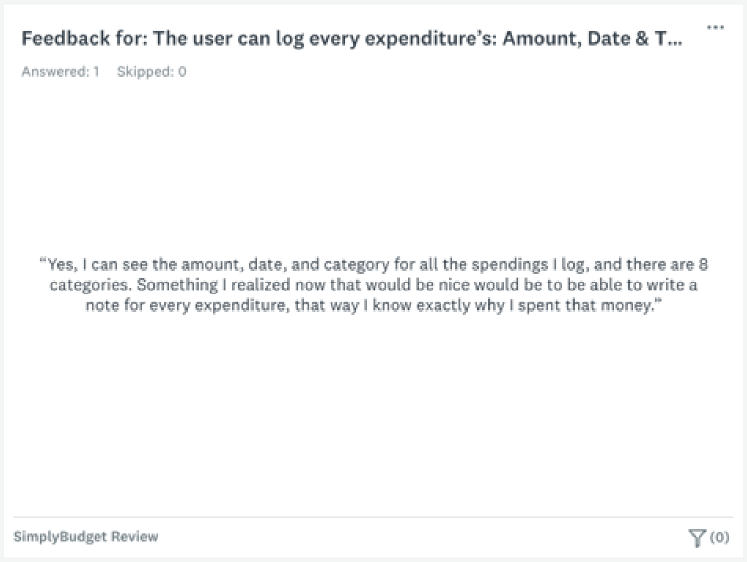


Feedback for: The user can add an expenditure quickly after opening t...

Answered: 1 Skipped: 0

“The buttons are actually right in the home screen, there's very little effort in adding expenditures. However, the app is lacking the ability to type in decimal numbers. I can edit my budget easily.”

SimplyBudget Review

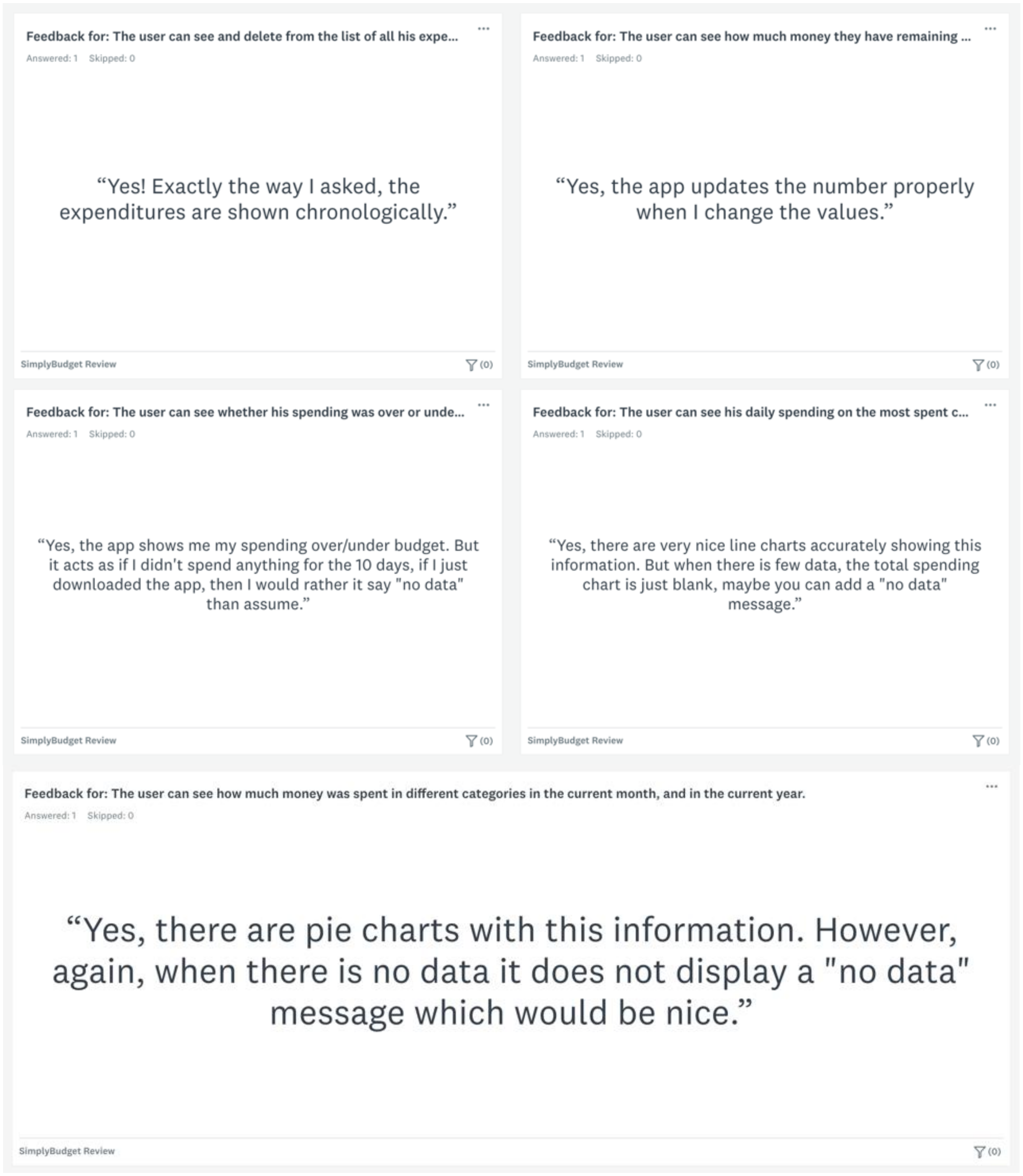


Feedback for: The user can log every expenditure’s: Amount, Date & T...

Answered: 1 Skipped: 0

“Yes, I can see the amount, date, and category for all the spendings I log, and there are 8 categories. Something I realized now that would be nice would be to be able to write a note for every expenditure, that way I know exactly why I spent that money.”

SimplyBudget Review



Feedback for: The user can see and delete from the list of all his expe...

Answered: 1 Skipped: 0

“Yes! Exactly the way I asked, the expenditures are shown chronologically.”

SimplyBudget Review (0)

Feedback for: The user can see how much money they have remaining ...

Answered: 1 Skipped: 0

“Yes, the app updates the number properly when I change the values.”

SimplyBudget Review (0)

Feedback for: The user can see whether his spending was over or unde...

Answered: 1 Skipped: 0

“Yes, the app shows me my spending over/under budget. But it acts as if I didn't spend anything for the 10 days, if I just downloaded the app, then I would rather it say "no data" than assume.”

SimplyBudget Review (0)

Feedback for: The user can see his daily spending on the most spent c...

Answered: 1 Skipped: 0

“Yes, there are very nice line charts accurately showing this information. But when there is few data, the total spending chart is just blank, maybe you can add a "no data" message.”

SimplyBudget Review (0)

Feedback for: The user can see how much money was spent in different categories in the current month, and in the current year.

Answered: 1 Skipped: 0

“Yes, there are pie charts with this information. However, again, when there is no data it does not display a "no data" message which would be nice.”

SimplyBudget Review (0)

Appendix C: Review of Final Product

The following is a transcript of a recorded conversation between me (the developer) and the client, where I ask the client to give me feedback and rate my product out of 10.

Developer: Hello Daniel! I have given you some weeks to try out the app, how are you liking it?

Client: This app is seriously exactly what I was looking for. It's simple, logging spending is easy, I can see analytics about my spending, and, most importantly, I can see how much of my budget I have remaining to spend. During the weeks of me using the app, I actually have rarely spent over my budget, and for the days that I did I was able to look back and remember where I spent my money. Having said that, I want to tell you that there are a few areas that could be improved to make the app even better, the first one being having decimals in expenditures. This isn't a big problem for me, as I wouldn't specify the cents and I usually round up or down depending on the thing I'm buying to make the logging process easier, but it would be nice to have. Also, the app doesn't show any "no data" sign for charts when I first started using the app, and there was just an empty chart, so I thought there was a malfunction. Adding this would be nice. Also, it would be nice to be able to add specific notes to every spending I log so I can really understand why I spent the money. Another possible functionality would be being able to edit the order of the charts in the analytics view, so that I can pick what I get to see first. And yeah, that's about it.

Developer: Thank you for the feedback. Now could you rate the app out of 10?

Client: Yes, out of 10 I would give it a solid 9.

Appendix D: Code Index

Below is all the code for this project. The headings represent the file structure, with a large heading representing folders and a sub-heading representing file names.

SimplyBudget / Model / CoreDataObjects

Expenditure+CoreDataClass.swift

```
//
//  Expenditure+CoreDataClass.swift
//

import Foundation
import CoreData

@objc(Expenditure)
public class Expenditure: NSManagedObject {

}
```

User+CoreDataClass.swift

```
//
//  User+CoreDataClass.swift
//

import Foundation
import CoreData

@objc(User)
public class User: NSManagedObject {

}
```

SimplyBudget / Views / TableScreen

ExpendituresTableTitleView.swift

```
//
//  ExpendituresTableTitleView.swift
//
```

```
import UIKit

class ExpendituresTableTitleView: UIView {
    // Title view for expenditures table.

    //MARK: - Variables and Init
    var parentViewController: ExpendituresTableViewController?

    init(frame: CGRect, parentVC: ExpendituresTableViewController) {
        super.init(frame: frame)
        self.translatesAutoresizingMaskIntoConstraints = false
        self.backgroundColor = .clear

        self.parentViewController = parentVC

        setupUI()
    }

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

    //MARK: - UI Elements and Setup
    var expendituresTableBackBtn: UIButton = {
        let button = UIButton()
        button.translatesAutoresizingMaskIntoConstraints = false
        button.setImage(UIImage(systemName: "chevron.right"), for: .normal)
        button.contentVerticalAlignment = .fill
        button.contentHorizontalAlignment = .fill
        button.tintColor = .black
        button.backgroundColor = .clear
        button.clipsToBounds = true
        button.addTarget(self, action: #selector(expendituresTableBackBtnClicked), for: .touchUpInside)
        return button
    }()

    var expendituresTableTitle: UILabel = {
        let label = UILabel()
        label.translatesAutoresizingMaskIntoConstraints = false
        label.text = "Expenses"
        label.font = .systemFont(ofSize: 40, weight: .medium)
        return label
    }()
    @objc func expendituresTableBackBtnClicked() {
        self.parentViewController?.popExpendituresTableView()
    }

    func setupUI() {
        self.addSubview(expendituresTableBackBtn)
        expendituresTableBackBtn.heightAnchor.constraint(equalToConstant: 25).isActive = true
        expendituresTableBackBtn.widthAnchor.constraint(equalToConstant: 20).isActive = true
        expendituresTableBackBtn.bottomAnchor.constraint(equalTo: self.bottomAnchor, constant: -20).isActive = true
        expendituresTableBackBtn.rightAnchor.constraint(equalTo: self.rightAnchor, constant: -30).isActive = true

        self.addSubview(expendituresTableTitle)
        expendituresTableTitle.bottomAnchor.constraint(equalTo: self.bottomAnchor, constant: -7).isActive = true
//        expendituresTableTitle.leftAnchor.constraint(equalTo: self.leftAnchor, constant: 30).isActive = true
        expendituresTableTitle.centerXAnchor.constraint(equalTo: self.centerXAnchor).isActive = true
    }

}
```

ExpendituresTableCell.swift

```
//
//  ExpendituresTableCell.swift
//

import UIKit

class ExpendituresTableCell: UITableViewCell {

    //MARK: - Variables, Init and Functions
    static let identifier = "ExpendituresTableCell"

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

    }

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

    func configure(category: String, amount: Int64, timeStamp: Date) {
        self.categoryImage.image = UIImage(named: "\(category)-img")
//        self.categoryImage.image = UIImage(named: "speed-img")
        self.amountLabel.text = "\(amount)"

        let usersCalendar = Calendar.current

        let day = usersCalendar.component(.day, from: timeStamp)
        let month = usersCalendar.component(.month, from: timeStamp)
        let monthName = DateFormatter().monthSymbols[month - 1].prefix(3)
        let year = usersCalendar.component(.year, from: timeStamp)
        let hour = usersCalendar.component(.hour, from: timeStamp)
        let minute = usersCalendar.component(.minute, from: timeStamp)

        self.dateLabel.text = "\(day) \(monthName). \(year) at \(hour):\(minute)"
    }

    //MARK: - UI elements and setup.
    var categoryImage: UIImageView = {
        let imageView = UIImageView()
        imageView.translatesAutoresizingMaskIntoConstraints = false
        imageView.contentMode = .scaleAspectFit
        imageView.image = UIImage(systemName: "chevron.right")
        imageView.tintColor = .label
        return imageView
    }()

    var amountLabel: UILabel = {
        let label = UILabel()
        label.translatesAutoresizingMaskIntoConstraints = false
        label.textColor = .label
        label.textAlignment = .left
        label.font = .systemFont(ofSize: 20, weight: .medium)
        label.text = "1_"
        return label
    }()

    var dateLabel: UILabel = {
        let label = UILabel()
        label.translatesAutoresizingMaskIntoConstraints = false
        label.textColor = .label
        label.textAlignment = .left
        label.font = .systemFont(ofSize: 17, weight: .light)
        label.text = "2_"
        return label
    }()
    func setupUI() {

        let margins = self.contentView.layoutMarginsGuide
        self.contentView.addSubview(categoryImage)
        categoryImage.heightAnchor.constraint(equalToConstant: 60).isActive = true
        categoryImage.widthAnchor.constraint(equalToConstant: 60).isActive = true
        categoryImage.centerYAnchor.constraint(equalTo: margins.centerYAnchor).isActive = true
        categoryImage.leftAnchor.constraint(equalTo: margins.leftAnchor).isActive = true

        self.contentView.addSubview(amountLabel)
        amountLabel.leftAnchor.constraint(equalTo: categoryImage.rightAnchor, constant: 30).isActive = true
        amountLabel.topAnchor.constraint(equalTo: margins.topAnchor, constant: 12).isActive = true
        amountLabel.bottomAnchor.constraint(equalTo: margins.bottomAnchor, constant: -8).isActive = true

        self.contentView.addSubview(dateLabel)
        dateLabel.rightAnchor.constraint(equalTo: margins.rightAnchor, constant: -20).isActive = true
        dateLabel.centerYAnchor.constraint(equalTo: amountLabel.centerYAnchor).isActive = true
    }

}
```

SimplyBudget / Views / AnalyticsScreen

AnalyticsTitleView.swift

```
//
//  AnalyticsTitleView.swift
//

import UIKit

class AnalyticsTitleView: UIView {
    //MARK: - Variables and Init

    var parentViewController: AnalyticsViewController?

    init(frame: CGRect, parentVC: AnalyticsViewController) {
        super.init(frame: frame)
        self.translatesAutoresizingMaskIntoConstraints = false

        self.parentViewController = parentVC

        setupUI()
```

```
}

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

//MARK: - UI Elements and Setup
var analyticsBackBtn: UIButton = {
    let button = UIButton()
    button.translatesAutoresizingMaskIntoConstraints = false
    button.setImage(UIImage(systemName: "chevron.left"), for: .normal)
    button.contentVerticalAlignment = .fill
    button.contentHorizontalAlignment = .fill
    button.tintColor = .black
    button.backgroundColor = .clear
    button.clipsToBounds = true
    button.addTarget(self, action: #selector(analyticsBackBtnClicked), for: .touchUpInside)
    return button
}()

var expendituresTableTitle: UILabel = {
    let label = UILabel()
    label.translatesAutoresizingMaskIntoConstraints = false
    label.text = "Analytics"
    label.font = .systemFont(ofSize: 40, weight: .medium)
    return label
}()

@objc func analyticsBackBtnClicked() {
    self.parentViewController?.popAnalyticsView()
}

func setupUI() {
    self.backgroundColor = .clear

    self.addSubview(analyticsBackBtn)
    analyticsBackBtn.heightAnchor.constraint(equalToConstant: 25).isActive = true
    analyticsBackBtn.widthAnchor.constraint(equalToConstant: 20).isActive = true
    analyticsBackBtn.topAnchor.constraint(equalTo: self.topAnchor, constant: 75).isActive = true
    analyticsBackBtn.leftAnchor.constraint(equalTo: self.leftAnchor, constant: 30).isActive = true

    self.addSubview(expendituresTableTitle)
    expendituresTableTitle.bottomAnchor.constraint(equalTo: self.bottomAnchor, constant: 0).isActive = true
    expendituresTableTitle.centerXAnchor.constraint(equalTo: self.centerXAnchor).isActive = true
    }

}
```

AnalyticsTableViewCell.swift

```
//
//  AnalyticsTableViewCell.swift
//

import UIKit
import DGCharts

class AnalyticsTableViewCell: UITableViewCell {

    //MARK: - Variables, Init and Functions
    static let identifier = "AnalyticsTableCell"



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

    }

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

    func configurePieChart(title: String, pieChart: PieChartView) {
        self.chartTitle.text = "\(title)"
        let cellView = self.contentView

        self.contentView.addSubview(pieChart)
        pieChart.translatesAutoresizingMaskIntoConstraints = false
        pieChart.centerXAnchor.constraint(equalTo: cellView.centerXAnchor).isActive = true
        pieChart.centerYAnchor.constraint(equalTo: cellView.centerYAnchor, constant: 20).isActive = true
        pieChart.heightAnchor.constraint(equalToConstant: 300).isActive = true
        pieChart.widthAnchor.constraint(equalToConstant: 300).isActive = true
    }

    func configureLineChart(title: String, lineChart: LineChartView) {
        self.chartTitle.text = "\(title)"
        let cellView = self.contentView
        self.contentView.addSubview(lineChart)
        lineChart.translatesAutoresizingMaskIntoConstraints = false
        lineChart.centerXAnchor.constraint(equalTo: cellView.centerXAnchor).isActive = true
        lineChart.centerYAnchor.constraint(equalTo: cellView.centerYAnchor, constant: 20).isActive = true
        lineChart.heightAnchor.constraint(equalToConstant: 300).isActive = true
        lineChart.widthAnchor.constraint(equalTo: cellView.widthAnchor, multiplier: 0.9).isActive = true
    }

    func configureBarChart(title: String, barChart: BarChartView) {
        self.chartTitle.text = "\(title)"
        let cellView = self.contentView

        self.contentView.addSubview(barChart)
        barChart.translatesAutoresizingMaskIntoConstraints = false
        barChart.centerXAnchor.constraint(equalTo: cellView.centerXAnchor).isActive = true
        barChart.centerYAnchor.constraint(equalTo: cellView.centerYAnchor, constant: 20).isActive = true
        barChart.heightAnchor.constraint(equalToConstant: 300).isActive = true
        barChart.widthAnchor.constraint(equalTo: cellView.widthAnchor, multiplier: 0.9).isActive = true
    }

    //MARK: - UI elements and setup.

    var chartTitle: UILabel = {
        let label = UILabel()
        label.translatesAutoresizingMaskIntoConstraints = false
        label.textColor = .label
        label.textAlignment = .left
        label.font = .systemFont(ofSize: 20, weight: .medium)
        label.text = "_"
        return label
    }()


    func setupUI() {
        let cellView = self.contentView

        self.contentView.addSubview(chartTitle)
        chartTitle.leftAnchor.constraint(equalTo: cellView.leftAnchor, constant: 20).isActive = true
        chartTitle.topAnchor.constraint(equalTo: cellView.topAnchor, constant: 15).isActive = true
    }

}
```

SimplyBudget / Views / HomeScreen

HomeTimeRangeView.swift

```swift
//
//  HomeTitleView.swift
//

import UIKit
import DropDown

class HomeTimeRangeView: UIView {
    // View for all category options in home screen.

    //MARK: - Variables and Init
    var budgetCounter: HomeBudgetCounterView?

    var timeRange = "Daily" as NSString
    let context = (UIApplication.shared.delegate as! AppDelegate).persistentContainer.viewContext



    // Init requires budgetCount as this view will call functions defined in budgetCounter as its time-range changes.
    init(frame: CGRect, budgetCount: HomeBudgetCounterView) {
        super.init(frame: frame)
        self.translatesAutoresizingMaskIntoConstraints = false
        self.backgroundColor = .clear

        budgetCounter = budgetCount
        setupUI()
    }

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

    //MARK: - Dropdown Variables and Functions
    lazy var titleDropdownButton: UIButton = {
        let button = UIButton()
        button.translatesAutoresizingMaskIntoConstraints = false
        button.backgroundColor = UIColor(red: 1.00, green: 0.69, blue: 0.50, alpha: 0)
        button.addTarget(self, action: #selector(titleTextIsClicked), for: .touchUpInside)
        button.backgroundColor = .clear
        return button
    }()
```

```
@objc func titleTextIsClicked() {
    // Executes when the title is clicked.
    titleDropdown.show()
}

lazy var titleDropdownImage: UIImageView = {
    let image = UIImageView(image: UIImage(systemName: "chevron.up")!)
    image.translatesAutoresizingMaskIntoConstraints = false
    image.tintColor = .black
    return image
}()

lazy var titleDropdownView: UIView = {
    let view = UIView()
    view.translatesAutoresizingMaskIntoConstraints = false
    view.backgroundColor = .clear
    return view
}()

let titleDropdown: DropDown = DropDown()
let titleDropdownOptions = ["Daily", "Monthly", "Yearly"]

func titleDropdownImageAnimation(_ multiplier: CGFloat) {
    UIView.animate(withDuration: 0) {
        self.titleDropdownImage.transform = CGAffineTransform(scaleX: 1, y: 1 * multiplier)
    }
}


//MARK: - UI Elements and Setup
lazy var titleTimeRangeLabel: UILabel = {
    let textLabel = UILabel()
    textLabel.translatesAutoresizingMaskIntoConstraints = false
    textLabel.font = UIFont(name: "Arial", size: 40)
    textLabel.textColor = .black

    // Underline Capability
    let mutableStr = NSMutableAttributedString(string: self.timeRange as String)
    let underlineAtr = [NSAttributedString.Key.underlineStyle: NSUnderlineStyle.single.rawValue]
    mutableStr.addAttributes(underlineAtr, range: self.timeRange.range(of: self.timeRange as String))
    textLabel.attributedText = mutableStr
    return textLabel
}()
```

```
lazy var titleStaticText: UILabel = {
    let textLabel = UILabel()
    textLabel.translatesAutoresizingMaskIntoConstraints = false
    textLabel.font = UIFont(name: "Arial", size: 40)
    textLabel.textColor = .black
    textLabel.text = "Budget"
    return textLabel
}()


private func setupUI() {
    // Title Text related views.
    self.addSubview(titleTimeRangeLabel)
    titleTimeRangeLabel.leftAnchor.constraint(equalTo: self.leftAnchor, constant: 50).isActive = true
    titleTimeRangeLabel.topAnchor.constraint(equalTo: self.topAnchor, constant: 90).isActive = true

    self.addSubview(titleStaticText)
    titleStaticText.topAnchor.constraint(equalTo: titleTimeRangeLabel.topAnchor, constant: 2).isActive = true
    titleStaticText.leftAnchor.constraint(equalTo: titleTimeRangeLabel.rightAnchor, constant: 10).isActive = true


    // Dropdown views and actions setup.
    self.addSubview(titleDropdownImage)
    titleDropdownImage.topAnchor.constraint(equalTo: titleTimeRangeLabel.bottomAnchor, constant: -33).isActive = true
    titleDropdownImage.leftAnchor.constraint(equalTo: self.leftAnchor, constant: 28).isActive = true

    self.addSubview(titleDropdownView)
    titleDropdownView.widthAnchor.constraint(equalTo: titleTimeRangeLabel.widthAnchor, multiplier: 1).isActive = true
    titleDropdownView.heightAnchor.constraint(equalTo: titleTimeRangeLabel.heightAnchor, multiplier: 1).isActive = true
    titleDropdownView.topAnchor.constraint(equalTo: titleTimeRangeLabel.topAnchor).isActive = true
    titleDropdownView.centerXAnchor.constraint(equalTo: titleTimeRangeLabel.centerXAnchor).isActive = true

    titleDropdown.anchorView = titleDropdownView
    titleDropdown.dataSource = titleDropdownOptions

    titleDropdown.bottomOffset = CGPoint(x: 0, y: (titleDropdown.anchorView?.plainView.bounds.height)!)
    titleDropdown.topOffset = CGPoint(x: 0, y: -(titleDropdown.anchorView?.plainView.bounds.height)!)
    titleDropdown.direction = .bottom

    // Setup of actions of different dropdown events.
    titleDropdown.selectionAction = { (index: Int, item: String) in
        self.titleTimeRangeLabel.text = self.titleDropdownOptions[index]
        self.titleTimeRangeLabel.textColor = .black
```

```
            self.titleDropdownImageAnimation(1)

            self.budgetCounter!.progressBarMetricChanged(to: self.titleTimeRangeLabel.text!)
        }

        titleDropdown.cancelAction = { [] in
            self.titleDropdownImageAnimation(1)
        }

        titleDropdown.willShowAction = { [] in
            self.titleDropdownImageAnimation(-1)
        }

        self.addSubview(titleDropdownButton)
        titleDropdownButton.widthAnchor.constraint(equalTo: titleTimeRangeLabel.widthAnchor, multiplier: 1.2).isActive = true
        titleDropdownButton.heightAnchor.constraint(equalTo: titleTimeRangeLabel.heightAnchor, multiplier: 1).isActive = true
        titleDropdownButton.topAnchor.constraint(equalTo: titleTimeRangeLabel.topAnchor).isActive = true
        titleDropdownButton.centerXAnchor.constraint(equalTo: titleTimeRangeLabel.centerXAnchor).isActive = true
    }
}
```

HomeBudgetCounterView.swift

```
//
//  HomeProgressBarView.swift
//

import UIKit

class HomeBudgetCounterView: UIView {
    // Responsible for displaying the correct budget according to different time-ranges.

    //MARK: - Variables and Init
    let context = (UIApplication.shared.delegate as! AppDelegate).persistentContainer.viewContext
    var appUser: User?

    init(frame: CGRect, parentVC: HomeViewController) {
        super.init(frame: frame)
        self.translatesAutoresizingMaskIntoConstraints = false
        self.backgroundColor = .clear

        setupUI()
    }
```

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

//MARK: - (Budget Changed) Functions
func progressBarMetricChanged(to text: String) {
    // Changes budgetCounterText in accordance to time-range selected.

    var spendingToBeCalculated: [Expenditure] = []
    var userDailyBudget: Int
    var userBudgetForTime: Int
    let usersCalendar = Calendar.current

    do {
        self.appUser = try context.fetch(User.fetchRequest())[0]
        userDailyBudget = Int(appUser!.dailyBudget)
    }
    catch {
        print("Error fetching budget data")
        return
    }



    if text == "Daily" {
        userBudgetForTime = userDailyBudget

    } else if text == "Monthly" {
        guard let daysInMonth = usersCalendar.range(of: .day, in: .month, for: Date())?.count else {
            print("Error counting days in month ")
            return
        }

        userBudgetForTime = userDailyBudget * daysInMonth

    } else {

        guard let daysInYear = usersCalendar.range(of: .day, in: .year, for: Date())?.count else {
            print("Error counting days in year. ")
            return
        }

        userBudgetForTime = userDailyBudget * daysInYear

    }
    for expenditure in appUser!.totalSpending! {
        // Go through all the expenditures, and depending on selected time-range, include these in the spendingToBeCalculated,
        // Also, change userBudgetForTime depending on selected time-range.

        guard let safeExpenditure = expenditure as? Expenditure else {
            print("Expenditure was not of type Expenditure")
            return
        }

        // Making sure timeStamp has a date. NOT NEEDED

        let dateOfExpense = safeExpenditure.timeStamp

        if text == "Daily" {
            if usersCalendar.isDateInToday(dateOfExpense!) {
                spendingToBeCalculated.append(safeExpenditure)
            }

        } else if text == "Monthly" {
            if usersCalendar.isDateInThisMonth(dateOfExpense!) {
                spendingToBeCalculated.append(safeExpenditure)
            }

        } else {
            if usersCalendar.isDateInThisYear(dateOfExpense!) {
                spendingToBeCalculated.append(safeExpenditure)
            }
        }
    }

    // For selected time-range: Remaining Budget = budget for time-range MINUS total spending in time-range.
    var totalAmountSpent = 0

    for expenditure in spendingToBeCalculated {
        totalAmountSpent += Int(expenditure.amount)
    }

    let remainingBudget = userBudgetForTime - totalAmountSpent

    // Update the counter text with the final budget.
    self.budgetCounterText.text? = remainingBudget.addCommas()
}
```

```
    //MARK: - UI Elements and Setup
    lazy var budgetCounterText: UITextView = {
        let textView = UITextView()
        textView.translatesAutoresizingMaskIntoConstraints = false
        textView.isEditable = false
        textView.textAlignment = .center
        textView.text = "1080"
        textView.font = UIFont(name: "Arial", size: 70)
        return textView
    }()

    private func setupUI() {
        self.addSubview(budgetCounterText)
        budgetCounterText.centerYAnchor.constraint(equalTo: self.centerYAnchor).isActive = true
        budgetCounterText.centerXAnchor.constraint(equalTo: self.centerXAnchor).isActive = true
        budgetCounterText.widthAnchor.constraint(equalTo: self.widthAnchor).isActive = true
        budgetCounterText.heightAnchor.constraint(equalToConstant: 100).isActive = true
    }

}
```

SimplyBudget / Views / HomeScreen / HomeCategories

CategoryButton.swift

```
//
//  CategoryButton.swift
//

import Foundation
import UIKit

class CategoryButton: UIButton {
    var categoryName: String?
    var imageName: String?
}
```

CategoryButton.swift

```
//
//  HomeAddMenuView.swift
//

import UIKit
```

```
class HomeAddExpenditureView: UIView {
    // View of a single category (small squares in bottom half of the home screen).

    //MARK: - Variables and Init
    var parentView: HomeCategoriesView?
    var currentCategoryName: String?

    let context = (UIApplication.shared.delegate as! AppDelegate).persistentContainer.viewContext

    // Needs parentVC because class calls refreshBudget2 from HomeCategoriesView, as it is nested within 2 parentVCs.
    init(frame: CGRect, parentVC: HomeCategoriesView) {
        super.init(frame: frame)
        self.translatesAutoresizingMaskIntoConstraints = false

        parentView = parentVC
        setupUI()
    }

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

    //MARK: - (Add Expenditure) Functions
    @objc func addExpenditureButtonClicked() {
        // When plus button is clicked on any one of the categories. Add expenditure to database.

        guard let textFieldText = addExpenditureTextField.text else { return }

        if textFieldText != "" {
            // Add new expenditure to database
            do {
                let appUser = try context.fetch(User.fetchRequest())[0]
                let newExpenditure = Expenditure(context: context)

                newExpenditure.timeStamp = Date()
                newExpenditure.amount = Int64(textFieldText)! // Only numbers allowed in textfield.
                newExpenditure.category = self.currentCategoryName! // Category name is set as soon as button is clicked.
                appUser.addToTotalSpending(newExpenditure)
                self.addExpenditureTextField.text = ""
                self.endEditing(true)

                do {
                    // Try to add expenditure to the database.
                    try context.save()
                }
                catch {
                    // If adding expenditure fails abort operations.
                    fatalError("Error: failed to save data.")
                }

            }
            catch {
                // If you cannot fetch data from the database abort operations.
                fatalError("Error fetching budget data")
            }

            self.parentView!.refreshBudget2()

            // Hide addExpenditure view after button is clicked.
            self.showHideAddExpenditure(0, category: nil, imageName: nil)
        }
    }

    func showHideAddExpenditure(_ value: Int, category: String?, imageName: String?) {
        // Shows and hides add menu with appropriate category when a categoryIsClicked.
        // Value = 1 to show, 0 to hide.
        if value == 1 {
            addMenuCategoryImage.image = UIImage(named: imageName!)
            addMenuCategoryTitle.text = category?.capitalized
            self.currentCategoryName = category

            self.isHidden = false
        }
        else {
            self.isHidden = true
            addExpenditureTextField.text = ""
        }
    }

    @objc func didTapDarkBackground(sender : UITapGestureRecognizer) {
        // AddMenu is hidden when background is tapped.
        self.showHideAddExpenditure(0, category: nil, imageName: nil)
        self.endEditing(true)
    }

    //MARK: - UI Elements and Setup
    var darkBackgroundLayer: UIView = {
        let view = UIView()
        view.translatesAutoresizingMaskIntoConstraints = false
        view.backgroundColor = UIColor(red: 0, green: 0, blue: 0, alpha: 0.4)
        return view
    }()

    var addMenuBackground: UIView = {
        let view = UIView()
        view.translatesAutoresizingMaskIntoConstraints = false
        view.layer.cornerRadius = 5
        view.backgroundColor = UIColor(red: 0.95, green: 0.99, blue: 0.96, alpha: 1.00)
        return view
    }()

    var addExpenditureTextField: UITextField = {
        let textField = UITextField()
        textField.translatesAutoresizingMaskIntoConstraints = false
        textField.layer.cornerRadius = 10
        textField.clipsToBounds = true
        textField.backgroundColor = .white
        textField.placeholder = "Enter Quantity..."
        textField.setLeftPaddingPoints(20)
        textField.keyboardType = .numberPad
        return textField
    }()

    var addMenuCategoryImage: UIImageView = {
        let imageView = UIImageView()
        imageView.translatesAutoresizingMaskIntoConstraints = false
        return imageView
    }()

    var addMenuCategoryTitle: UITextView = {
        let textView = UITextView()
        textView.translatesAutoresizingMaskIntoConstraints = false
        textView.textColor = .black
        textView.backgroundColor = .clear
        textView.isEditable = false
        textView.text = ""
        textView.isScrollEnabled = false
        textView.font = UIFont(name: "Arial", size: 30)
        return textView
    }()

    var addMenuAddButton: UIButton = {
        let button = UIButton()
        button.translatesAutoresizingMaskIntoConstraints = false
        button.setImage(UIImage(systemName: "plus"), for: .normal)
        button.tintColor = .white
        button.backgroundColor = UIColor(red: 0.21, green: 0.66, blue: 0.44, alpha: 1.00)
        button.clipsToBounds = true
        button.layer.cornerRadius = 25
        button.addTarget(self, action: #selector(addExpenditureButtonClicked), for: .touchUpInside)
        return button
    }()

    private func setupUI() {
        self.showHideAddExpenditure(0, category: nil, imageName: nil)

        self.addSubview(darkBackgroundLayer)
        darkBackgroundLayer.heightAnchor.constraint(equalTo: self.heightAnchor, multiplier: 1).isActive = true
        darkBackgroundLayer.widthAnchor.constraint(equalTo: self.widthAnchor, multiplier: 1).isActive = true
        darkBackgroundLayer.centerYAnchor.constraint(equalTo: self.centerYAnchor).isActive = true
        darkBackgroundLayer.centerXAnchor.constraint(equalTo: self.centerXAnchor).isActive = true

        self.addSubview(addMenuBackground)
        addMenuBackground.widthAnchor.constraint(equalToConstant: 300).isActive = true
        addMenuBackground.heightAnchor.constraint(equalToConstant: 150).isActive = true
        addMenuBackground.centerYAnchor.constraint(equalTo: self.centerYAnchor).isActive = true
        addMenuBackground.centerXAnchor.constraint(equalTo: self.centerXAnchor).isActive = true

        self.addSubview(addExpenditureTextField)
        addExpenditureTextField.delegate = self
        addExpenditureTextField.widthAnchor.constraint(equalTo: addMenuBackground.widthAnchor, multiplier: 0.7).isActive = true
        addExpenditureTextField.heightAnchor.constraint(equalToConstant: 50).isActive = true
        addExpenditureTextField.leftAnchor.constraint(equalTo: addMenuBackground.leftAnchor, constant: 15).isActive = true
        addExpenditureTextField.bottomAnchor.constraint(equalTo: addMenuBackground.bottomAnchor, constant: -15).isActive = true

        self.addSubview(addMenuCategoryImage)
        addMenuCategoryImage.widthAnchor.constraint(equalToConstant: 70).isActive = true
        addMenuCategoryImage.heightAnchor.constraint(equalToConstant: 70).isActive = true
        addMenuCategoryImage.leftAnchor.constraint(equalTo: addMenuBackground.leftAnchor, constant: 10).isActive = true
        addMenuCategoryImage.topAnchor.constraint(equalTo: addMenuBackground.topAnchor, constant: 5).isActive = true

        self.addSubview(addMenuCategoryTitle)
        addMenuCategoryTitle.leftAnchor.constraint(equalTo: addMenuCategoryImage.rightAnchor, constant: 10).isActive = true
        addMenuCategoryTitle.centerYAnchor.constraint(equalTo: addMenuCategoryImage.centerYAnchor).isActive = true

        self.addSubview(addMenuAddButton)
```

```
        addMenuAddButton.heightAnchor.constraint(equalToConstant: 50).isActive = true
        addMenuAddButton.widthAnchor.constraint(equalToConstant: 50).isActive = true
        addMenuAddButton.leftAnchor.constraint(equalTo: addExpenditureTextField.rightAnchor, constant: 10).isActive = true
        addMenuAddButton.centerYAnchor.constraint(equalTo: addExpenditureTextField.centerYAnchor).isActive = true

        let gesture = UITapGestureRecognizer(target: self, action:  #selector(self.didTapDarkBackground))
        self.darkBackgroundLayer.addGestureRecognizer(gesture)
    }
}

//MARK: - Extensions: Text Field
extension HomeAddExpenditureView: UITextFieldDelegate {
    func textFieldShouldReturn(_ textField: UITextField) -> Bool {
        return true
    }

    func textField(_ textField: UITextField, shouldChangeCharactersIn range: NSRange, replacementString string: String) -> Bool {
        let maxLength = 10
        let currentString: NSString = textField.text! as NSString
        let newString: NSString =  currentString.replacingCharacters(in: range, with: string) as NSString

        return newString.length <= maxLength
    }

}
```

HomeCategoriesView.swift

```
//
//  HomeCategoriesView.swift
//

import UIKit
import CoreData

class HomeCategoriesView: UIView {
    // Class responsible for creating the whole of the categories view (bottom half of the home screen).

    var addMenu: HomeAddExpenditureView?
    var parentViewController: HomeViewController?
    var budgetMenu: HomeBudgetMenuView?

    let fillerFrame = CGRect(x: 0, y: 0, width: 0, height: 0)
    init(frame: CGRect, parentVC: HomeViewController) {
        super.init(frame: frame)

        self.translatesAutoresizingMaskIntoConstraints = false
        self.backgroundColor = .clear



        self.budgetMenu = HomeBudgetMenuView(frame: fillerFrame, parentVC: self)
        self.addMenu = HomeAddExpenditureView(frame: fillerFrame, parentVC: self)
        self.parentViewController = parentVC

        setupUI()
    }

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

    //MARK: - Functions
    func refreshBudget2() {
        self.parentViewController?.refreshBudget()
//        print("refresh budget 2 called.")
    }

    func deleteAllData() {
        // Deletes data in case database needs to be changed.
        let context = (UIApplication.shared.delegate as! AppDelegate).persistentContainer.viewContext

        do {
            let appUser = try context.fetch(User.fetchRequest())[0]
            let allExpenditures = appUser.totalSpending!
            for expenditure in allExpenditures {
                context.delete(expenditure as! NSManagedObject)
            }
        }
        catch {
            fatalError("Error deleting budget data for table view.")
        }

        do {
            try context.save()
        }
        catch {
            print("Error: failed to save data.")
        }
    }

    //MARK: - UI Elements and Setup
    lazy var categoriesMainStackView: UIStackView = {
        let stackView = UIStackView()
        stackView.translatesAutoresizingMaskIntoConstraints = false
        stackView.axis = .vertical
        stackView.distribution = .fillEqually
        return stackView
    }()

    private func categoriesSubViewFunc(item1: UIView, item2: UIView, item3: UIView) -> UIStackView {
        let stackView = UIStackView()
        stackView.translatesAutoresizingMaskIntoConstraints = false
        stackView.axis = .horizontal
        stackView.distribution = .fillEqually
        stackView.addArrangedSubview(item1)
        stackView.addArrangedSubview(item2)
        stackView.addArrangedSubview(item3)
        return stackView
    }

    func createCategoriesButon(for category: String, with image: String) -> UIButton {
        let button = CategoryButton()
        button.translatesAutoresizingMaskIntoConstraints = false
        button.setImage(UIImage(named: image), for: .normal)
        button.categoryName = category
        button.imageName = image
        button.addTarget(self, action: #selector(categoryButtonClicked), for: .touchUpInside)
        return button
    }

    @objc func categoryButtonClicked(sender: CategoryButton) {
        addMenu!.showHideAddExpenditure(1, category: sender.categoryName, imageName: sender.imageName!)
    }

    func createHolderView(colored color: UIColor) -> UIView {
        let view = UIView()
        view.translatesAutoresizingMaskIntoConstraints = false
        view.backgroundColor = color
        return view
    }
```

```
    // Extra categories in the end:
    lazy var editBudgetButton: UIButton = {
        // this is only a temporary button, the real image needs to be added as png with custom padding.
        let button = UIButton()
        button.translatesAutoresizingMaskIntoConstraints = false
        button.setImage(UIImage(named: "settings-img"), for: .normal)
        button.addTarget(self, action: #selector(editBudgetButtonClicked), for: .touchUpInside)
        return button
    }()

    private func setupUI() {
        self.addSubview(categoriesMainStackView)
        categoriesMainStackView.centerXAnchor.constraint(equalTo: self.centerXAnchor).isActive = true
        categoriesMainStackView.widthAnchor.constraint(equalToConstant: 320).isActive = true
        categoriesMainStackView.heightAnchor.constraint(equalToConstant: 320).isActive = true
        categoriesMainStackView.bottomAnchor.constraint(equalTo: self.bottomAnchor, constant: -70).isActive = true

        let btn1 = createCategoriesButon(for: "housing", with: "housing-img")
        let btn2 = createCategoriesButon(for: "food", with: "food-img")
        let btn3 = createCategoriesButon(for: "transportation", with: "transportation-img")
        let btn4 = createCategoriesButon(for: "utilities", with: "utilities-img")
        let btn5 = createCategoriesButon(for: "personal", with: "personal-img")
        let btn6 = createCategoriesButon(for: "savings", with: "savings-img")
        let btn7 = createCategoriesButon(for: "health", with: "health-img")
        let btn8 = createCategoriesButon(for: "miscellaneous", with: "miscellaneous-img")
        let btn9 = editBudgetButton

        let subView1 = categoriesSubViewFunc(item1: btn1, item2: btn2, item3: btn3)
        let subView2 = categoriesSubViewFunc(item1: btn4, item2: btn5, item3: btn6)
        let subView3 = categoriesSubViewFunc(item1: btn7, item2: btn8, item3: btn9)

        categoriesMainStackView.addArrangedSubview(subView1)
        categoriesMainStackView.addArrangedSubview(subView2)
        categoriesMainStackView.addArrangedSubview(subView3)
    }

    @objc func editBudgetButtonClicked() {
        budgetMenu?.showHideBudgetMenu(1)
    }

}
```

HomeBudgetMenuView.swift

```
//
//  HomeBudgetView.swift
//

import UIKit

class HomeBudgetMenuView: UIView {
    // Edits user budget, bottom left gear icon in Home Screen.

    var parentViewController: HomeCategoriesView?

    let context = (UIApplication.shared.delegate as! AppDelegate).persistentContainer.viewContext

    init(frame: CGRect, parentVC: HomeCategoriesView) {
        super.init(frame: frame)
        self.translatesAutoresizingMaskIntoConstraints = false

        // Problem here.
        setupUI()

        self.parentViewController = parentVC

    }

    //MARK: - (Edit Budget) Functions
    @objc func editButtonClicked() {
        if self.budgetMenuTextField.text ?? "" != "" {
            do {
                let appUser = try context.fetch(User.fetchRequest())[0]
                appUser.dailyBudget = Int64(self.budgetMenuTextField.text!) ?? 0 // setting daily budget to value of text field.

                let newBudget = "\(appUser.dailyBudget)"
                self.budgetMenuCurrentValue.text = newBudget
                self.parentViewController?.refreshBudget2()

                self.budgetMenuTextField.text = ""
                self.endEditing(true)
            }
            catch {
                print("Error fetching data.")
                return
            }
        do {
            try context.save()
        }
        catch {
            print("Error: failed to save data.")
            return
        }
    }
}

//MARK: - UI Elements and Setup

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

var darkBackgroundLayer: UIView = {
    let view = UIView()
    view.translatesAutoresizingMaskIntoConstraints = false
    view.backgroundColor = UIColor(red: 0, green: 0, blue: 0, alpha: 0.4)
    return view
}()

var budgetMenuBackground: UIView = {
    let view = UIView()
    view.translatesAutoresizingMaskIntoConstraints = false
    view.layer.cornerRadius = 5
    view.backgroundColor = UIColor(red: 0.95, green: 0.99, blue: 0.96, alpha: 1.00)
    return view
}()


var budgetMenuTextField: UITextField = {
    let textField = UITextField()
    let centeredParagraphStyle = NSMutableParagraphStyle()
    textField.translatesAutoresizingMaskIntoConstraints = false
    textField.layer.cornerRadius = 10
    textField.clipsToBounds = true
    textField.backgroundColor = .white
    textField.placeholder = "Enter Quantity..."
    textField.setLeftPaddingPoints(20)
    textField.keyboardType = .numberPad
    return textField
```

```
}()

var budgetMenuTitleText: UITextView = {
    let textView = UITextView()
    textView.translatesAutoresizingMaskIntoConstraints = false
    textView.textColor = .black
    textView.backgroundColor = .clear
    textView.isEditable = false
    textView.text = "Edit Budget"
    textView.isScrollEnabled = false
    textView.font = UIFont(name: "Arial", size: 35)
    return textView
}()

var budgetMenuCurrentValue: UITextView = {
    let textView = UITextView()
    textView.isEditable = false
    textView.textAlignment = .center
    textView.translatesAutoresizingMaskIntoConstraints = false
    textView.clipsToBounds = true
    textView.backgroundColor = .clear
    textView.text = "texto"
    textView.font = UIFont(name: "Arial", size: 25)
    textView.textAlignment = .center
    return textView
}()

var budgetMenuEditButton: UIButton = {
    let button = UIButton()
    button.translatesAutoresizingMaskIntoConstraints = false
    button.setImage(UIImage(systemName: "plus"), for: .normal)
    button.tintColor = .white
    button.backgroundColor = UIColor(red: 0.21, green: 0.66, blue: 0.44, alpha: 1.00)
    button.clipsToBounds = true
    button.layer.cornerRadius = 20 // Height ÷ 2, i dunno how to get the height tho
    button.addTarget(self, action: #selector(editButtonClicked), for: .touchUpInside)
    return button
}()

private func setupUI() {
    showHideBudgetMenu(0)

    self.addSubview(darkBackgroundLayer)
    darkBackgroundLayer.heightAnchor.constraint(equalTo: self.heightAnchor, multiplier: 1).isActive = true
```

```
        darkBackgroundLayer.widthAnchor.constraint(equalTo: self.widthAnchor, multiplier: 1).isActive = true
        darkBackgroundLayer.centerYAnchor.constraint(equalTo: self.centerYAnchor).isActive = true
        darkBackgroundLayer.centerXAnchor.constraint(equalTo: self.centerXAnchor).isActive = true

        self.addSubview(budgetMenuBackground)
        budgetMenuBackground.widthAnchor.constraint(equalToConstant: 300).isActive = true
        budgetMenuBackground.heightAnchor.constraint(equalToConstant: 300).isActive = true
        budgetMenuBackground.centerYAnchor.constraint(equalTo: self.centerYAnchor).isActive = true
        budgetMenuBackground.centerXAnchor.constraint(equalTo: self.centerXAnchor).isActive = true
        self.showHideBudgetMenu(0)

        self.addSubview(budgetMenuTitleText)
        budgetMenuTitleText.leftAnchor.constraint(equalTo: budgetMenuBackground.leftAnchor, constant: 10).isActive = true
        budgetMenuTitleText.centerYAnchor.constraint(equalTo: budgetMenuBackground.topAnchor, constant: 50).isActive = true

        // Setting up the current budget
        self.addSubview(budgetMenuCurrentValue)
        budgetMenuCurrentValue.widthAnchor.constraint(equalTo: budgetMenuBackground.widthAnchor, multiplier: 0.7).isActive = true
        budgetMenuCurrentValue.heightAnchor.constraint(equalToConstant: 50).isActive = true
        budgetMenuCurrentValue.centerXAnchor.constraint(equalTo: self.centerXAnchor).isActive = true
        budgetMenuCurrentValue.topAnchor.constraint(equalTo: budgetMenuTitleText.bottomAnchor, constant: 10).isActive = true

        // Problem is here.
        do {
            let appUsers = try context.fetch(User.fetchRequest())
            if appUsers != [] {
                let dailyBudgetValue = appUsers[0].dailyBudget
                budgetMenuCurrentValue.text = "\(dailyBudgetValue)"
            }
//            print(appUsers)

        } catch {
            print("Error setting defaul budget value in this screen.")
        }

        self.addSubview(budgetMenuTextField)
        budgetMenuTextField.delegate = self
        budgetMenuTextField.widthAnchor.constraint(equalTo: budgetMenuBackground.widthAnchor, multiplier: 0.7).isActive = true
        budgetMenuTextField.heightAnchor.constraint(equalToConstant: 50).isActive = true
        budgetMenuTextField.centerXAnchor.constraint(equalTo: self.centerXAnchor).isActive = true
        budgetMenuTextField.topAnchor.constraint(equalTo: budgetMenuCurrentValue.bottomAnchor, constant: 10).isActive = true

        self.addSubview(budgetMenuEditButton)
        budgetMenuEditButton.heightAnchor.constraint(equalToConstant: 40).isActive = true
        budgetMenuEditButton.widthAnchor.constraint(equalToConstant: 100).isActive = true
        budgetMenuEditButton.topAnchor.constraint(equalTo: budgetMenuTextField.bottomAnchor, constant: 20).isActive = true
        budgetMenuEditButton.centerXAnchor.constraint(equalTo: budgetMenuTextField.centerXAnchor).isActive = true

        let gesture = UITapGestureRecognizer(target: self, action:  #selector(self.didTapDarkBackground))
        self.darkBackgroundLayer.addGestureRecognizer(gesture)
    }

    @objc func didTapDarkBackground(sender : UITapGestureRecognizer) {
        self.showHideBudgetMenu(0)
        self.endEditing(true)
    }

    func showHideBudgetMenu(_ value: Int) {
        // Value = 1 to show, 0 to hide.
        if value == 1 {
            self.isHidden = false
        }
        else {
            self.isHidden = true
        }
    }



}

//MARK: - Extensions: Text Field

extension HomeBudgetMenuView: UITextFieldDelegate {
    func textFieldShouldReturn(_ textField: UITextField) -> Bool {
        return true
    }

    func textField(_ textField: UITextField, shouldChangeCharactersIn range: NSRange, replacementString string: String) -> Bool {
        let maxLength = 10
        let currentString: NSString = textField.text! as NSString
        let newString: NSString =  currentString.replacingCharacters(in: range, with: string) as NSString

        return newString.length <= maxLength
    }

}
```

SimplyBudget / Views / Controllers

HomeViewController.swift

```
//
//  ViewController.swift
//


import UIKit
import DropDown
import CoreData


class HomeViewController: UIViewController {
    // Controls the home screen.

    //MARK: - Variables and Init
    // Sub-Views: TimeRangeView, BudgetCounterView, Categories View. Other VC: ExpenditureTableVC
    var timeRangeView: HomeTimeRangeView?
    var budgetCounterView: HomeBudgetCounterView?
    var categoriesView: HomeCategoriesView?
    var userClass: User?
    var expendituresTableVC: ExpendituresTableViewController?

    // Context for database (CoreData).
    let context = (UIApplication.shared.delegate as! AppDelegate).persistentContainer.viewContext


    override func viewDidLoad() {
        super.viewDidLoad()
        view.backgroundColor = .white
        navigationController?.navigationBar.isHidden = true

        budgetCounterView = HomeBudgetCounterView(frame: .zero, parentVC: self)
        timeRangeView = HomeTimeRangeView(frame: .zero, budgetCount: budgetCounterView!)
        categoriesView = HomeCategoriesView(frame: .zero, parentVC: self)
        expendituresTableVC = ExpendituresTableViewController(homeVC: self)

        setupBudget()
        setupHomeUI()
        setupSwipeLeftGesture()
        setupSwipeRightGesture()
    deleteAllData()
    makeFillerData()

}

func deleteAllData() {
        // Deletes data in case database needs to be changed.
        let context = (UIApplication.shared.delegate as! AppDelegate).persistentContainer.viewContext

        do {
            let appUser = try context.fetch(User.fetchRequest())[0]
            let allExpenditures = appUser.totalSpending!
            for expenditure in allExpenditures {
                context.delete(expenditure as! NSManagedObject)
            }
        }
        catch {
            fatalError("Error deleting budget data for table view.")
        }

        do {
            try context.save()
        }
        catch {
            print("Error: failed to save data.")
        }

    }

func makeFillerData() {

    for _ in 0...300 {
        do {
            let appUser = try context.fetch(User.fetchRequest())[0]
            let newExpenditure = Expenditure(context: context)
            newExpenditure.amount = Int64.random(in: 4...30)
            let randomDate = Double.random(in: 1674230218...1697659847)
            newExpenditure.timeStamp = Date(timeIntervalSince1970: randomDate)
            newExpenditure.category = ["housing", "food", "transportation",
                                    "utilities", "personal", "savings", "health", "miscellaneous"].randomElement()!

            appUser.addToTotalSpending(newExpenditure)
        }
        catch {
            print("Error fetching budget data")
        }

        do {
            try context.save()
        }
        catch {
            print("Error: failed to save data.")
        }
    }

}



//MARK: - Functions and UI Setup
func refreshBudget() {
    // Updates the text on BudgetCounterView according to time-range set, by reading database.
    self.budgetCounterView!.progressBarMetricChanged(to: self.timeRangeView!.titleTimeRangeLabel.text!)

    do {
        let appUser = try context.fetch(User.fetchRequest())[0]

        if appUser.totalSpending != [] {
            expendituresTableVC!.tableView.reloadData()
        }

    }
    catch {
        fatalError("Error fetching budget data to refresh view in HomeViewController.")
    }

}

func setupBudget() {
    // If there is no User in database, adds a User with a daily budget of 10. Otherwise, updates UI from database.

    do {
        let appUsers = try context.fetch(User.fetchRequest())

        if appUsers == [] {
            let defaultUser = User(context: self.context)
            defaultUser.dailyBudget = 10
            do {
                try context.save()
            }
        }

        self.refreshBudget()
    }
    catch {
        print("Error fetching budget data.")
    }
}


private func setupHomeUI() {
    // Sets-up UI for Home Screen.
    // Since all sub-views were assigned values in initialization, it is fine to force unwrap optionals.
    view.addSubview(timeRangeView!)
    timeRangeView!.topAnchor.constraint(equalTo: view.topAnchor, constant: 0).isActive = true
    timeRangeView!.heightAnchor.constraint(equalToConstant: 150).isActive = true
    timeRangeView!.widthAnchor.constraint(equalTo: view.widthAnchor, multiplier: 1).isActive = true

    view.addSubview(budgetCounterView!)
    budgetCounterView?.centerXAnchor.constraint(equalTo: view.centerXAnchor).isActive = true
    budgetCounterView?.centerYAnchor.constraint(equalTo: view.centerYAnchor, constant: -135).isActive = true
    budgetCounterView?.heightAnchor.constraint(equalToConstant: 200).isActive = true
    budgetCounterView?.widthAnchor.constraint(equalTo: view.widthAnchor).isActive = true

    view.addSubview(categoriesView!)
    categoriesView?.bottomAnchor.constraint(equalTo: view.bottomAnchor).isActive = true
    categoriesView?.centerXAnchor.constraint(equalTo: view.centerXAnchor).isActive = true
    categoriesView?.heightAnchor.constraint(equalToConstant: 400).isActive = true
    categoriesView?.widthAnchor.constraint(equalTo: view.widthAnchor).isActive = true

    view.addSubview(categoriesView!.addMenu!)
    categoriesView?.addMenu?.center = view.center
    categoriesView?.addMenu?.widthAnchor.constraint(equalTo: view.widthAnchor, multiplier: 1).isActive = true
    categoriesView?.addMenu?.heightAnchor.constraint(equalTo: view.heightAnchor, multiplier: 1).isActive = true

    view.addSubview(categoriesView!.budgetMenu!)
    categoriesView?.budgetMenu?.center = view.center
    categoriesView?.budgetMenu?.widthAnchor.constraint(equalTo: view.widthAnchor, multiplier: 1).isActive = true
    categoriesView?.budgetMenu?.heightAnchor.constraint(equalTo: view.heightAnchor, multiplier: 1).isActive = true
}

//MARK: - Swiped Gestures
func setupSwipeLeftGesture() {
        let swipeLeftGesture = UISwipeGestureRecognizer(target: self, action: #selector(didSwipeLeft))
        swipeLeftGesture.direction = .left
        self.view.addGestureRecognizer(swipeLeftGesture)
    }

    @objc func didSwipeLeft() {
        let analyticsVC = AnalyticsViewController()
        self.navigationController?.pushViewController(analyticsVC, animated: true)
    }

    func setupSwipeRightGesture() {
        let swipeRightGesture = UISwipeGestureRecognizer(target: self, action: #selector(didSwipeRight))
        swipeRightGesture.direction = .right
        self.view.addGestureRecognizer(swipeRightGesture)
    }

    @objc func didSwipeRight() {
        self.navigationController?.pushViewControllerFromLeft(controller: expendituresTableVC!)
    }

    func popCurrentView() {
        self.navigationController?.popViewController(animated: true)
    }
}
```

AnalyticsViewController.swift

```
//
//  AnalyticsViewController.swift
//

import UIKit
import DGCharts
import CoreData

class AnalyticsViewController: UIViewController {
    // Controls analytics view.

    //MARK: - Variables and Init
    var analyticsTitleView: AnalyticsTitleView?
    var allExpenditures: [Expenditure]?
    var allCharts: [Any]?
    var largestCategory: String?
    var chartTitles: [String]?
    var userDailyBudget: Int64?

    var selectedCategory = "Click Chart!"

    // Categories to label the charts.
    let categories = ["housing", "food", "transportation",
                      "utilities", "personal", "savings", "health", "miscellaneous"]

    let context = (UIApplication.shared.delegate as! AppDelegate).persistentContainer.viewContext

    override func viewDidLoad() {
        super.viewDidLoad()
        view.backgroundColor = .white

        refreshDataAndCharts()

        self.analyticsTitleView = AnalyticsTitleView(frame: .zero, parentVC: self)
        setupAnalyticsUI()

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

    // Fills in allExpenditures data everytime view appears.
    override func viewWillAppear(_ animated: Bool) {
        refreshDataAndCharts()
    }

    //MARK: - Data and Setup
    func refreshDataAndCharts() {
        do {
            let appUser = try context.fetch(User.fetchRequest())[0]
            self.userDailyBudget = appUser.dailyBudget
            self.allExpenditures = (appUser.totalSpending?.allObjects as! [Expenditure]).sorted(by: { $0.timeStamp!.compare($1.timeStamp!)
== .orderedDescending
            })
        }
        catch {
            fatalError("Error fetching budget data for table view.")
        }

        // Put all charts here.
        self.allCharts = [
            createRecentSpending(),
            createMostSpentCatMonthlyLineChart(),
            createAllTimeSpendingLineChart(),
            createMonthlySpendingPieChart(),
            createYearlySpendingPieChart(),
        ]

        self.chartTitles = [
            "Recent Spending Under/Over Budget",
            "Most Spent This Month: \(largestCategory?.capitalized ?? "No Data")",
            "Total Daily Spending",
            "Monthly Spending per Category",
            "Yearly Spending per Category",
        ]
    }

    var tableView: UITableView = {
        let tableV = UITableView()
        tableV.translatesAutoresizingMaskIntoConstraints = false
        tableV.backgroundColor = .white
        tableV.allowsSelection = false
        tableV.register(AnalyticsTableViewCell.self, forCellReuseIdentifier: AnalyticsTableViewCell.identifier)
        tableV.rowHeight = 370 // Required because we're not using Auto Layout.
        return tableV
    }()

    //MARK: - Bar Chart
    func createRecentSpending() -> BarChartView {
        // Returns a bar chart showing how much you spent under or over your budget in the last 10 days.
        let barChart = BarChartView()



        // Get all expenditures in the last 10 days.
        var expendituresPerDate: [[Expenditure]] = []

        for _ in 0...9 {
            expendituresPerDate.append([])
        }



        for i in 0...9 {
            let dayOfExpense = Calendar.current.date(byAdding: .day, value: -i, to: Date())!

            for expenditure in self.allExpenditures! {
                if Calendar.current.isDate(expenditure.timeStamp!, inSameDayAs: dayOfExpense) {
```

```
            expendituresPerDate[i].append(expenditure)
        }
    }
}

var recentDailySpending: [Int64] = []

for _ in 0...9 {
    recentDailySpending.append(0) // Look into how to consicely create an empty array of n values in swift
}

// Take the sum of those expenditures per day.
for i in 0...9 {
    for expenditure in expendituresPerDate[i] {
        recentDailySpending[i] += expenditure.amount
    }
}

// We now have recentDailySpending, with the total expenditures in the last 10 days.
var posEntries = [BarChartDataEntry()]
var negEntries = [BarChartDataEntry()]

for i in 0...(recentDailySpending.count - 1) {
    let budgetMinusExpense = self.userDailyBudget! - recentDailySpending[i]
    let newEntry = BarChartDataEntry(x: Double(i), y: Double(budgetMinusExpense))

    if budgetMinusExpense >= 0 {
        posEntries.append(newEntry)
    } else {
        negEntries.append(newEntry)
    }
}

let set1 = BarChartDataSet(entries: posEntries)
let set2 = BarChartDataSet(entries: negEntries)
let data = BarChartData(dataSets: [set1, set2])
set1.drawValuesEnabled = false
set2.drawValuesEnabled = false
barChart.data = data

barChart.xAxis.labelPosition = .bottom
barChart.legend.enabled = false

barChart.chartDescription.text = "Days ago."
```

```
    barChart.chartDescription.font = UIFont(name: "Arial", size: 14)!

    let greenColor: UIColor = UIColor(red: 0.20, green: 0.63, blue: 0.42, alpha: 1.00)
    set1.colors = [greenColor]

    let redColor = UIColor(red: 201/255, green: 36/255, blue: 91/255, alpha: 1.00)
    set2.colors = [redColor]


    return barChart
}


//MARK: - Line Charts
func averageDataIn20(for dailySpendingData:[Double]) -> [ChartDataEntry] {
    // Takes the average of every 20th part of the data and returns these averaged chart entries.

    var entries = [ChartDataEntry()]
    var counter = 0

    for _ in 0...(dailySpendingData.count - 1) {
        // Setting average value to be changed later, and average divisor to be one 20th of the data set.
        var averageValue = 0.0
        let averageDivisor = Int(dailySpendingData.count / 20)

        // Everytime the current day is at a 20th of the whole data, take the average of all the days within that part of the whole data.
        if averageDivisor != 0 {
            if counter % averageDivisor == 0 && counter != 0  {
                for i in 0...averageDivisor-1 {
                    averageValue += dailySpendingData[counter - averageDivisor + i]
                }

                let finalAverage = averageValue/Double(averageDivisor)

                // Every 20th part append this entry to data entries (should have 20 in total).
                entries.append(ChartDataEntry(x: Double(counter), y: finalAverage))

            }
            counter += 1
        }

    }

    // Return entries.
    return entries
}

func styleLineChart(for lineChart: LineChartView,
                    withData setOfData: LineChartDataSet,
                    xBeing descriptionText: String,
                    colored color: UIColor) {

    setOfData.drawCirclesEnabled = false
    setOfData.mode = .cubicBezier
    setOfData.lineWidth = 3

    setOfData.setColor(color)

    lineChart.data?.setDrawValues(false)
    lineChart.legend.enabled = false
    lineChart.leftAxis.enabled = false
    lineChart.rightAxis.setLabelCount(5, force: false)
    lineChart.rightAxis.labelFont = .systemFont(ofSize: 13)

    lineChart.xAxis.labelPosition = .bottom
    lineChart.xAxis.setLabelCount(5, force: false)
    lineChart.xAxis.labelFont = .systemFont(ofSize: 13)

    lineChart.animate(xAxisDuration: 0.5)



    lineChart.chartDescription.text = descriptionText
    lineChart.chartDescription.font = UIFont(name: "Arial", size: 14)!
}

func createMostSpentCatMonthlyLineChart() -> LineChartView {
    let lineChart = LineChartView()

    // First value in the x-axis has to be earliest data entry.
    let userCalendar = Calendar.current

    // In case there are no expenditures
    if self.allExpenditures! != [] {

        // Get first day of month.
        guard let firstDayMonth = userCalendar.date(from: Calendar.current.dateComponents([.year, .month], from: Date())) else {
            fatalError("Error calculating first day of month.")
        }
```

```
// Find the number of days in the month.
let numDays = userCalendar.range(of: .day, in: .month, for: Date())!.count
var expendituresPerDate: [[Expenditure]] = []

// For each day in the month, create an empty array.
for _ in 0...numDays {
    expendituresPerDate.append([])
}

// For each day in the month, fill the empty array with the expenditures in that day of that month.
for i in 0...numDays {
    let dayOfExpense = userCalendar.date(byAdding: .day, value: i, to: firstDayMonth)!

    for expenditure in self.allExpenditures! {
        if userCalendar.isDateInThisMonth(expenditure.timeStamp!) {
            if userCalendar.isDate(expenditure.timeStamp!, inSameDayAs: dayOfExpense) {
                expendituresPerDate[i].append(expenditure)
            }
        }
    }
}

// Find the category with the most spending this month by getting the total for each category in that month.
let totalPerCategory = getTotalPerCategoryMonthly()
var largestCategoryIndex = 0

var counter = 0
for currentCategory in totalPerCategory {

    if currentCategory > totalPerCategory[largestCategoryIndex] {
        largestCategoryIndex = counter
    }
    counter += 1
}

// Add to dailySpendingInMostCategory how much was spent in that category each day.
let largestCategory = self.categories[largestCategoryIndex]
self.largestCategory = largestCategory

var dailySpendingInMostCategory: [Double] = []

for _ in 0...(expendituresPerDate.count - 1) {
    dailySpendingInMostCategory.append(0.0)
```

```
        }

        for i in 0...(expendituresPerDate.count - 1) {
            for expenditure in expendituresPerDate[i] {
                if expenditure.category == largestCategory {
                    dailySpendingInMostCategory[i] += Double(expenditure.amount)
                }
            }
        }

        // Average the data to make a smooth graph.
        let entries = averageDataIn20(for: dailySpendingInMostCategory)

        let set1 = LineChartDataSet(entries: entries)
        let data = LineChartData(dataSet: set1)
        lineChart.data = data

        // Styling
        let magentaColor = UIColor(red: 201/255, green: 36/255, blue: 91/255, alpha: 1.00)
        styleLineChart(for: lineChart, withData: set1, xBeing: "Days in month", colored: magentaColor)

        return lineChart

    } else { return lineChart }
}

func createAllTimeSpendingLineChart() -> LineChartView {
    let lineChart = LineChartView()

    // First value in the x-axis has to be earliest data entry.
    let userCalendar = Calendar.current

    // In case there are no expenditures
    if self.allExpenditures! != [] {

        let startDate = self.allExpenditures!.last!.timeStamp
        let endDate = self.allExpenditures![0].timeStamp

        let numDays = endDate!.interval(ofComponent: .day, fromDate: startDate!)

        var expendituresPerDate: [[Expenditure]] = []

        for _ in 0...numDays {
            expendituresPerDate.append([])
        }

        for i in 0...numDays {
            let dayOfExpense = userCalendar.date(byAdding: .day, value: i, to: startDate!)!

            for expenditure in self.allExpenditures! {
                if userCalendar.isDate(expenditure.timeStamp!, inSameDayAs: dayOfExpense) {
                    expendituresPerDate[i].append(expenditure)
                }
            }
        }

        // Gets total spending per day.
        var totalSpendingPerDay: [Double] = []

        for _ in 0...(expendituresPerDate.count - 1) {
            totalSpendingPerDay.append(0.0)
        }

        for i in 0...(expendituresPerDate.count - 1) {
            for expenditure in expendituresPerDate[i] {
                totalSpendingPerDay[i] += Double(expenditure.amount)
            }
        }

        let entries = averageDataIn20(for: totalSpendingPerDay)

        let set1 = LineChartDataSet(entries: entries)
        let data = LineChartData(dataSet: set1)
        lineChart.data = data

        // Styling
        let blueColor = UIColor(red: 0.22, green: 0.38, blue: 0.83, alpha: 1.00)
        styleLineChart(for: lineChart, withData: set1, xBeing: "Days since using app.", colored: blueColor)

        return lineChart

    } else { return lineChart }
}


//MARK: - Pie Chart Functions
func getTotalPerCategoryMonthly() -> [Int] {
    var totalPerCategoryMonthly: [Int] = []
```

```
        for _ in self.categories {
            totalPerCategoryMonthly.append(0)
        }

        for expenditure in allExpenditures! {
            // For every expenditure: find category and add amount to that categories total spending.
            var categoryIndex = 0
            let userCalendar = Calendar.current

            for category in self.categories {
                if userCalendar.isDateInThisMonth(expenditure.timeStamp!) {
                    if expenditure.category == category {
                        totalPerCategoryMonthly[categoryIndex] += Int(expenditure.amount)
                    }
                }
                categoryIndex += 1
            }
        }

        return totalPerCategoryMonthly
    }

    func stylePieChart(for pieChart: PieChartView) {
        pieChart.animate(xAxisDuration: 0.5, easingOption: .easeInOutCirc)
        pieChart.rotationEnabled = false
        pieChart.legend.enabled = false

        pieChart.centerText = self.selectedCategory.capitalized
        pieChart.centerTextRadiusPercent = 0.95
    }

    func createMonthlySpendingPieChart() -> PieChartView {
        // Creates a MONTHLY spending per category pie chart.

        // Create chart
        let pieChart = PieChartView()
        pieChart.delegate = self

        // Supply Data
        let totalPerCategoryMonthly = getTotalPerCategoryMonthly()


        // For each category add a PieChartDataEntry with the total spending in that category.
        var entries = [PieChartDataEntry()]
        for i in 0...(categories.count - 1) {
            var chartLabel = categories[i].capitalized
            if categories[i] == "transportation" {
                chartLabel = "Transp."
            } else if categories[i] == "miscellaneous" {
                chartLabel = "Misc."
            }

            entries.append(PieChartDataEntry(value: Double(totalPerCategoryMonthly[i]), label: chartLabel))
        }

        let dataSet = PieChartDataSet(entries: entries)

        // Main style
        dataSet.colors = ChartColorTemplates.pastel()
        dataSet.drawValuesEnabled = false

        let data = PieChartData(dataSet: dataSet)
        pieChart.data = data

        // Other styling
        stylePieChart(for: pieChart)

        return pieChart
    }

    func createYearlySpendingPieChart() -> PieChartView {
        // Creates a YEARLY spending per category pie chart.

        // Create chart
        let pieChart = PieChartView()
        pieChart.delegate = self

        // Supply Data
        var totalPerCategoryYearly: [Int] = []
        for _ in self.categories {
            totalPerCategoryYearly.append(0)
        }

        for expenditure in allExpenditures! {
            // For every expenditure: find category and add amount to that categories total spending.
            var categoryIndex = 0
            let userCalendar = Calendar.current
        for category in categories {
            if userCalendar.isDateInThisYear(expenditure.timeStamp!) {
                if expenditure.category == category {
                    totalPerCategoryYearly[categoryIndex] += Int(expenditure.amount)
                }
                categoryIndex += 1
            }

        }
    }

    // For each category add a PieChartDataEntry with the total spending in that category.
    var entries = [PieChartDataEntry()]

    for i in 0...(categories.count - 1) {
        var chartLabel = categories[i].capitalized
        if categories[i] == "transportation" {
            chartLabel = "Transp."
        } else if categories[i] == "miscellaneous" {
            chartLabel = "Misc."
        }

        entries.append(PieChartDataEntry(value: Double(totalPerCategoryYearly[i]), label: chartLabel))
    }

    let dataSet = PieChartDataSet(entries: entries)

    // Main style
    dataSet.colors = ChartColorTemplates.pastel()
    dataSet.drawValuesEnabled = false

    let data = PieChartData(dataSet: dataSet)
    pieChart.data = data

    // Other styling
    stylePieChart(for: pieChart)

    return pieChart
}


//MARK: - Functions and UI Setup
func popAnalyticsView() {
        self.navigationController?.popViewController(animated: true)
    }


    func setupAnalyticsUI() {
        view.backgroundColor = .white

        view.addSubview(analyticsTitleView!)
        analyticsTitleView!.topAnchor.constraint(equalTo: view.topAnchor, constant: 0).isActive = true
        analyticsTitleView!.heightAnchor.constraint(equalToConstant: 120).isActive = true
        analyticsTitleView!.widthAnchor.constraint(equalTo: view.widthAnchor, multiplier: 1).isActive = true

        view.addSubview(tableView)
        self.tableView.topAnchor.constraint(equalTo: analyticsTitleView!.bottomAnchor, constant: 20).isActive = true
        self.tableView.bottomAnchor.constraint(equalTo: view.bottomAnchor, constant: 0).isActive = true
        self.tableView.rightAnchor.constraint(equalTo: view.rightAnchor, constant: 0).isActive = true
        self.tableView.leftAnchor.constraint(equalTo: view.leftAnchor, constant: 0).isActive = true
    }

}

//MARK: - Extensions: Table and Chart

extension AnalyticsViewController: UITableViewDelegate, UITableViewDataSource {
    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return self.allCharts?.count ?? 0
    }

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

        guard let cell = tableView.dequeueReusableCell(withIdentifier: AnalyticsTableViewCell.identifier, for: indexPath) as?
AnalyticsTableViewCell else {
            fatalError("The table view could not dequeue a ExpendituresTableCell.")
        }

        let currentChart = self.allCharts![indexPath.row]
        let title = self.chartTitles![indexPath.row]

        if currentChart is PieChartView {
            cell.configurePieChart(title: title, pieChart: currentChart as! PieChartView)
        }
        else if currentChart is LineChartView {
            let lineChart = currentChart as! LineChartView
            cell.configureLineChart(title: title, lineChart: lineChart)
            if lineChart.chartDescription.text == "Days in month" {
                let usersCalendar = Calendar.current
                let month = usersCalendar.component(.month, from: Date())
                let monthName = DateFormatter().monthSymbols[month - 1]
                lineChart.chartDescription.text = "Days in \(monthName)"
            }
        }
        else if currentChart is BarChartView {
            cell.configureBarChart(title: title, barChart: currentChart as! BarChartView)
        }

        return cell
    }
}

extension AnalyticsViewController: ChartViewDelegate {
    func chartValueSelected(_ chartView: ChartViewBase, entry: ChartDataEntry, highlight: Highlight) {
        let valueLabel: String = entry.value(forKey: "label")! as! String
        let valueNum: Int = Int(entry.value(forKey: "value")! as! Double)

        // Goes through all charts available, and check to see which one is the being clicked on.
        for chart in allCharts! {
            if chartView.isEqual(chart) {
                // If it is a piechart then display data in center.
                guard let currentChart = chart as? PieChartView else { return }
                currentChart.centerText = """
                    \(valueLabel)
                    \(valueNum.addCommas())
                    """
            }
        }
    }

    func chartValueNothingSelected(_ chartView: ChartViewBase) {
        for chart in allCharts! {
            if chartView.isEqual(chart) {
                // If it is a piechart then display data in center.
                guard let currentChart = chart as? PieChartView else { return }
                currentChart.centerText = ""
            }
        }
    }
}
```

ExpendituresTableViewController.swift

```
//
//  ExpendituresTableViewController.swift
//

import UIKit

class ExpendituresTableViewController: UIViewController {

    //MARK: - Variables and Init
    var expTableTitleView: ExpendituresTableTitleView?
    var homeViewController: HomeViewController?
    var allExpenditures: [Expenditure]?

    let context = (UIApplication.shared.delegate as! AppDelegate).persistentContainer.viewContext

    // Requires a way to communicate with homeVC, as it changes data.
    init(homeVC: HomeViewController?) {
        super.init(nibName: nil, bundle: nil)
        homeViewController = homeVC
    }

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

    override func viewDidLoad() {
        super.viewDidLoad()

        self.expTableTitleView = ExpendituresTableTitleView(frame: .zero, parentVC: self)
        setupTableExpendituresUI()

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

    //MARK: - Table View
    var tableView: UITableView = {
        let tableV = UITableView()
        tableV.translatesAutoresizingMaskIntoConstraints = false
        tableV.backgroundColor = .white
        tableV.allowsSelection = false
        tableV.register(ExpendituresTableCell.self, forCellReuseIdentifier: ExpendituresTableCell.identifier)
        return tableV
    }()

    //MARK: - UI Setup and Functions
    func popExpendituresTableView() {
        self.navigationController?.popViewControllerFromRight()
    }

    func setupTableExpendituresUI() {
        view.backgroundColor = .white

        view.addSubview(expTableTitleView!)
        expTableTitleView!.topAnchor.constraint(equalTo: view.topAnchor, constant: 0).isActive = true
        expTableTitleView!.heightAnchor.constraint(equalToConstant: 130).isActive = true
        expTableTitleView!.widthAnchor.constraint(equalTo: view.widthAnchor, multiplier: 1).isActive = true

        view.addSubview(tableView)
        self.tableView.topAnchor.constraint(equalTo: expTableTitleView!.bottomAnchor, constant: 20).isActive = true
        self.tableView.bottomAnchor.constraint(equalTo: view.bottomAnchor, constant: 0).isActive = true
        self.tableView.rightAnchor.constraint(equalTo: view.rightAnchor, constant: 20).isActive = true
        self.tableView.leftAnchor.constraint(equalTo: view.leftAnchor, constant: 0).isActive = true
    }
}

extension ExpendituresTableViewController: UITableViewDelegate, UITableViewDataSource {
    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        // How many table rows we want.
        do {
            let appUser = try context.fetch(User.fetchRequest())[0]
            return appUser.totalSpending?.count ?? 0
        }
        catch {
            fatalError("Error fetching budget data for table view.")
        }
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        // Pick what custom cell and pass in data.
        do {
            let appUser = try context.fetch(User.fetchRequest())[0]
            self.allExpenditures = (appUser.totalSpending?.allObjects as! [Expenditure]).sorted(by: { $0.timeStamp!.compare($1.timeStamp!)
== .orderedDescending
            })
            guard let cell = tableView.dequeueReusableCell(withIdentifier: ExpendituresTableCell.identifier, for: indexPath) as?
ExpendituresTableCell else {
                fatalError("The table view could not dequeue a ExpendituresTableCell.")
            }

            let cellCategory = allExpenditures![indexPath.row].category
            let cellAmount = allExpenditures![indexPath.row].amount
            let cellTime = allExpenditures![indexPath.row].timeStamp

            cell.configure(category: cellCategory!, amount: cellAmount, timeStamp: cellTime!)
            return cell
        }
        catch {
            fatalError("Error fetching budget data for table view.")
        }
    }

    func tableView(_ tableView: UITableView, editingStyleForRowAt indexPath: IndexPath) -> UITableViewCell.EditingStyle {
        return .delete
    }

    func tableView(_ tableView: UITableView, commit editingStyle: UITableViewCell.EditingStyle, forRowAt indexPath: IndexPath) {
        if editingStyle == .delete {
            tableView.beginUpdates()
            let expenditureToRemove = self.allExpenditures![indexPath.row] // Table was made, allExpenditures != nil
            self.context.delete(expenditureToRemove)

            // Saving data and refreshing budget.
            do {
                try context.save()
            }
            catch {
                print("Error: failed to save data.")
            }

            homeViewController!.refreshBudget()

            tableView.deleteRows(at: [indexPath], with: .fade)
            tableView.endUpdates()
        }
    }


}
```

SimplyBudget

Extensions.swift

```
//
//  Extensions.swift
//

import Foundation
import UIKit

extension UITextField {
    func setLeftPaddingPoints(_ amount:CGFloat){
        let paddingView = UIView(frame: CGRect(x: 0, y: 0, width: amount, height: self.frame.size.height))
        self.leftView = paddingView
        self.leftViewMode = .always
    }
    func setRightPaddingPoints(_ amount:CGFloat) {
        let paddingView = UIView(frame: CGRect(x: 0, y: 0, width: amount, height: self.frame.size.height))
        self.rightView = paddingView
        self.rightViewMode = .always
    }
}

extension Calendar {
    private var currentDate: Date { return Date() }

    func isDateInThisMonth(_ date: Date) -> Bool {
        return isDate(date, equalTo: currentDate, toGranularity: .month)
    }

    func isDateInThisYear(_ date: Date) -> Bool {
        return isDate(date, equalTo: currentDate, toGranularity: .year)
    }
}

extension UINavigationController {
    func pushViewControllerFromLeft(controller: UIViewController) {
        let transition = CATransition()
        transition.duration = 0.3
        transition.type = CATransitionType.push
        transition.subtype = CATransitionSubtype.fromLeft
        transition.timingFunction = CAMediaTimingFunction(name: CAMediaTimingFunctionName.easeInEaseOut)
        view.window!.layer.add(transition, forKey: kCATransition)
        pushViewController(controller, animated: false)
    }

    func popViewControllerFromRight() {
        let transition = CATransition()
        transition.duration = 0.3
        transition.type = CATransitionType.push
        transition.subtype = CATransitionSubtype.fromRight
        transition.timingFunction = CAMediaTimingFunction(name: CAMediaTimingFunctionName.easeInEaseOut)
        view.window!.layer.add(transition, forKey: kCATransition)
        popViewController(animated: false)
    }
}


extension Date {

    func interval(ofComponent comp: Calendar.Component, fromDate date: Date) -> Int {

        let currentCalendar = Calendar.current

        guard let start = currentCalendar.ordinality(of: comp, in: .era, for: date) else { return 0 }
        guard let end = currentCalendar.ordinality(of: comp, in: .era, for: self) else { return 0 }

        return end - start
    }
}

extension Int {
    func addCommas() -> String {
        let numberFormatter = NumberFormatter()
        numberFormatter.numberStyle = .decimal
        return numberFormatter.string(from: NSNumber(value: self))!
    }
}
```

AppDelegate.swift

```
//
//  AppDelegate.swift
//

import UIKit
import CoreData
```

```
@main
class AppDelegate: UIResponder, UIApplicationDelegate {



   func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) ->
Bool {
       // Override point for customization after application launch.
       return true
   }

   // MARK: UISceneSession Lifecycle

   func application(_ application: UIApplication, configurationForConnecting connectingSceneSession: UISceneSession, options:
UIScene.ConnectionOptions) -> UISceneConfiguration {
       // Called when a new scene session is being created.
       // Use this method to select a configuration to create the new scene with.
       return UISceneConfiguration(name: "Default Configuration", sessionRole: connectingSceneSession.role)
   }

   func application(_ application: UIApplication, didDiscardSceneSessions sceneSessions: Set<UISceneSession>) {
       // Called when the user discards a scene session.
       // If any sessions were discarded while the application was not running, this will be called shortly after
application:didFinishLaunchingWithOptions.
       // Use this method to release any resources that were specific to the discarded scenes, as they will not return.
   }

   // MARK: - Core Data stack
       lazy var persistentContainer: NSPersistentContainer = {
           let container = NSPersistentContainer(name: "FamillyBudget")
           container.loadPersistentStores(completionHandler: { (storeDescription, error) in
               if let error = error as NSError? {
                   fatalError("Unresolved error \(error), \(error.userInfo)")
               }
           })
           return container
       }()
       // MARK: - Core Data Saving support
       func saveContext () {
           let context = persistentContainer.viewContext
           if context.hasChanges {
               do {
                   try context.save()
```

```
            } catch {
                let nserror = error as NSError
                fatalError("Unresolved error \(nserror), \(nserror.userInfo)")
            }
        }
    }


}
```

SceneDelegate.swift

```
//
//  SceneDelegate.swift
//

import UIKit

class SceneDelegate: UIResponder, UIWindowSceneDelegate {

    var window: UIWindow?


    func scene(_ scene: UIScene, willConnectTo session: UISceneSession, options connectionOptions: UIScene.ConnectionOptions) {
        // Use this method to optionally configure and attach the UIWindow `window` to the provided UIWindowScene `scene`.
        // If using a storyboard, the `window` property will automatically be initialized and attached to the scene.
        // This delegate does not imply the connecting scene or session are new (see `application:configurationForConnectingSceneSession`
instead).

        guard let windowScene = (scene as? UIWindowScene) else { return }
        let window = UIWindow(windowScene: windowScene)
        window.rootViewController = UINavigationController(rootViewController: HomeViewController())
        self.window = window
        self.window?.makeKeyAndVisible()
    }

    func sceneDidDisconnect(_ scene: UIScene) {
        // Called as the scene is being released by the system.
        // This occurs shortly after the scene enters the background, or when its session is discarded.
        // Release any resources associated with this scene that can be re-created the next time the scene connects.
        // The scene may re-connect later, as its session was not necessarily discarded (see `application:didDiscardSceneSessions` instead).
    }

    func sceneDidBecomeActive(_ scene: UIScene) {
        // Called when the scene has moved from an inactive state to an active state.
        // Use this method to restart any tasks that were paused (or not yet started) when the scene was inactive.
    }

    func sceneWillResignActive(_ scene: UIScene) {
        // Called when the scene will move from an active state to an inactive state.
        // This may occur due to temporary interruptions (ex. an incoming phone call).
    }

    func sceneWillEnterForeground(_ scene: UIScene) {
        // Called as the scene transitions from the background to the foreground.
        // Use this method to undo the changes made on entering the background.
    }

    func sceneDidEnterBackground(_ scene: UIScene) {
        // Called as the scene transitions from the foreground to the background.
        // Use this method to save data, release shared resources, and store enough scene-specific state information
        // to restore the scene back to its current state.
    }


}
```