**MTD367**

**iOS App Development**

**July 2024**

**ECA**

|  |  |
| --- | --- |
| Name: | Yang Xian Wei Shawn |
| T-Group | T01 |
| Date Submitted | 7 October 2024 Monday |

**Question 1**

**Question 1(a) (10 marks)**

**ANS:**

|  |
| --- |
| **Correct and Functional Code:**  **import** UIKit  // This function calculates the Levenshtein distance between two strings.  // The Levenshtein distance is the minimum number of single-character edits required to change one string into the other.  /\*  This statement was replaced:  func LDist(\_ s1: string, s2.count)    Corrected function declaration  \*/  **func** LDist(\_ s1: String, \_ s2: String) -> Int  {  // Get the lengths of the input strings.  **let** (lenS1, lenS2) = (s1.count, s2.count)    /\*  This statement was replaced:  let dp = Array(repeating: Array(repeating: 0, count: lenS2 + 1), count: lenS1 + 1)    Change 'let' to 'var' to make the variable mutable    Initialize a 2D array to store the minimum edit distances.  \*/  **var** dp = Array(repeating: Array(repeating: 0, count: lenS2 + 1), count: lenS1 + 1)    // Iterate over the characters of s1 (including an empty substring at the beginning).  **for** i **in** 0...lenS1  {  // Iterate over the characters of s2 (including an empty substring at the beginning).  **for** j **in** 0...lenS2  {    // dp[i][j] represents the Levenshtein distance between the first i characters of s1 and the first j characters of s2.    **if** i == 0  {  // Initialize first row  dp[i][j] = j  }    **else** **if** j == 0  {  // Initialize first column  dp[i][j] = i  }  **else** {  dp[i][j] = min(  /\*  Del  Deletion: Delete the last character of s1.  \*/  dp[i - 1][j] + 1,    /\*  Ins  Insertion: Insert the last character of s2.  \*/  dp[i][j - 1] + 1,    /\*  ????  Substitution: If the last characters of s1 and s2 are different, substitute one for the other.  \*/  dp[i - 1][j - 1] + (s1[s1.index(  s1.startIndex,  offsetBy: i - 1  )] == s2[s2.index(  s2.startIndex,  offsetBy: j - 1)] ? 0 : 1)  )  }  }  }    /\*  This Statement was replaced:  --> dp[lenS1][lenS2]    Added return statement    Return the Levenshtein distance between the two strings.  \*/  **return** dp[lenS1][lenS2]  }  // Example Usage  **let** input1 = "Kitchen"  **let** input2 = "Chicken"  **let** output = LDist(input1, input2)  /\*  This Statement was replaced:    print('\(input1)' -> '\(input2)' = \(output).")    Print the result.  \*/  print(" '\(input1)' -> '\(input2)' = \(output).")  // 'Kitchen' -> 'Chicken' = 4  **Output:**  A black rectangular object with a black border  Description automatically generated  **Explanation of Code Corrections:**   1. The original function declaration “**func** LDist(\_ s1: string, s2.count)” had errors.   The new function declaration is “**func** LDist(\_ s1: String, \_ s2: String) -> Int {“  The parameter “\_ s1: string” should instead have been renamed to “(\_ s1: String” where the capitalized String is an acceptable Swift data type. The s1 variable represents string variable  The parameter “s2.count” is incorrect and instead it should be changed to “\_ s2: String” where an underscore was added in place of a explicit argument label and String data type was also used so that the s2 variable would become a second string variable.  An explicit return type “-> Int” was added to the function declaration to indicate that the function must return an integer value which represents the Levenshtein distance.   1. The original array “**let** dp = Array(repeating: Array(repeating: 0, count: lenS2 + 1), count: lenS1 + 1)” was immutable.   The new array “**var** dp = Array(repeating: Array(repeating: 0, count: lenS2 + 1), count: lenS1 + 1)” is mutable.  The array “dp” was declared with a type “let” which made it a constant that has immutable values assigned to it..  Since the values within the array “dp” needs to be modified in the function, the array should instead be declared with a type of “var” to make it a variable so that mutable values would be assigned to it.   1. The original function did not explicitly provide a return statement and instead returned this statement “--> dp[lenS1][lenS2]” which is incorrect. Since we explicitly require the function to provide a return statement that returns an integer value, I have added this statement “**return** dp[lenS1][lenS2]” to replace the original statement. 2. This is the original print statement “print('\(input1)' -> '\(input2)' = \(output).")”   This is the updated print statement “print(" '\(input1)' -> '\(input2)' = \(output).")”  The original print statement is incorrect because the first quotation mark ‘ is incorrectly used and instead it should be replaced with a double quotation mark as seen in the updated print statement.  **Clarification of Code Comments:**   1. The code comment “Del” for the line “dp[i - 1][j] + 1,” has been corrected to “Deletion: Delete the last character of s1.” which represents the code operation to delete the last element from the first string “s1”. 2. The code comment “Ins” for the line “dp[i][j - 1] + 1,” has been corrected to “Insertion: Insert the last character of s2.” which represents the code operation to insert the last element from the second string “s2” into the first string which is ”s1” 3. The code comment “????” for the line “dp[i - 1][j - 1] + (s1[s1.index(s1.startIndex, offsetBy: i - 1)] == s2[s2.index(s2.startIndex, offsetBy: j - 1)] ? 0 : 1)” has been corrected to “Substitution: If the last characters of s1 and s2 are different, substitute one for the other.” |

**Question 1(b) (10 marks)**

**ANS:**

|  |
| --- |
| **Explanation of how the Code works:**  The function “LDist” is used to systematically consider single-character edits or code operations such as the insertion, deletion and substitution operations as well as apply dynamic programming to obtain and use results to compute the Levenshtein distance between two strings.  The Levenshtein distance represents the minimum number of single-character edits which refers to the various code operations such as the insertion, deletion and substitution operations that were previously mentioned above.  The Levenshtein distance function has 4 key stages, initialization, base cases, dynamic programming and result.  **1st Stage: Initialization**  In the initialization stage, the function “LDist” takes 2 strings “s1” and “s2” as parameters and within the function, the tuple “(s1.count, s2.count)” contains the length of both of the string values and stores them into a constant tuple “let (lenS1, lenS2)”.  A 2D array called “dp” is defined as a dynamic programming table that will store the minimum edit distances.  The “dp[i][j]” is the substring that would store the Levenshtein distance between the first element at position “[i]” of the first string “s1” and the first element at position “[j]” of the second string “s2”.  **2nd Stage: Base Cases**  In the base cases stage, the first two for loops are used to manage the base cases by iterating through the elements of strings “s1” and “s2” to check whether anyone of them are empty.  The statement “**if** i == 0” checks if the string “s1” is empty and if this is true, then the Levenshtein distance would be the length of string “s2” which is referring to “”lenS2”.  The statement “**else if j == 0**” checks if the string ”s2” is empty and if this is true, then the Levenshtein distance would be the length of string “s1” which is referring to “”lenS1”.  **3rd Stage: Dynamic Programming**  In the dynamic programming stage, for each pair of elements at positions “[i]” and “[j]”, if the elements at the current positions from the 2 string values “s1” and “s2” match, no single-character edits would be required because the Levenshtein distance matches the distance between the substrings that do not have those elements at the positions “[i]” and “[j]”.  If the elements at the positions “[i]” and “[j]’ do not match the 3 minimum number of single-character edits or code operations need to be considered which are the Deletion, Insertion and Substitution operations.  The deletion operation which is represented by the code “dp[i - 1][j] + 1” involves removing the last element from the first substring “s1” and calculating the distance between “s1” and the second substring “s2”.  For the deletion operation, the “dp” array would be increased by 1.  The insertion operation which is represented by the code “dp[i][j - 1] + 1” involves inserting the last element from the second substring “s2” into the substring “s1” and calculating the distance between “s1” and the second substring “s2”.  For the insertion operation, the “dp” array would also be increased by 1.  The substitution operation which is represented by the code “dp[i - 1][j - 1] + (s1[s1.index(s1.startIndex, offsetBy: i - 1)] == s2[s2.index(s2.startIndex, offsetBy: j - 1)] ? 0 : 1)” involves substituting the last element of the first substring “s1” with the last element of the second substring “s2” and calculating the distance the substrings “s1” and “s2”.  For the substitution operation, the “dp” array would also be increased by 1 if the elements are not the same but if they are the same, increase by 0.  The minimum distance from exploring these 3 code operations would be used as the Levenshtein distance for the substring “dp[i][j]”.  **4th Stage: Result**  In the result stage, after the for loops have been computed, the function would return the value “dp[lenS1][lenS2]” that would store into the bottom-right corner of the “dp” array the Levenshtein distance between the strings “s1” and “s2”.  **Example that gives an output = 4 when one of the two inputs is “MTD367 Swift”:**  **// Example Usage**  let input1 = "MTD367 Swift"  let input2 = "MT36 Swf"  let output = LDist(input1, input2)  /\*  This Statement was replaced:  print('\(input1)' -> '\(input2)' = \(output).")  Print the result.  \*/  print(" '\(input1)' -> '\(input2)' = \(output).")  **Output:**  A black rectangular object with a black border  Description automatically generated |

**Question 2**

**Question 2(a) (15 marks)**

**ANS:**

|  |
| --- |
| **LaunchScreen:**  **A screen shot of a phone case  Description automatically generated**  **Treasure Trekker Singapore App Workflow in Xcode:**  **A screenshot of a cell phone  Description automatically generatedA screenshot of a cell phone  Description automatically generated**  **Simulator Flow:**  **A screen shot of a phone  Description automatically generated A screenshot of a phone  Description automatically generated**  **A screen shot of a phone  Description automatically generated A screen shot of a phone  Description automatically generated A screen shot of a phone  Description automatically generated**  **A screen shot of a phone  Description automatically generated A screenshot of a cell phone  Description automatically generated A screenshot of a cell phone  Description automatically generated**  **A screenshot of a cell phone  Description automatically generated A screen shot of a phone  Description automatically generated A screenshot of a phone  Description automatically generated**  **A screenshot of a cell phone  Description automatically generated A screenshot of a phone  Description automatically generated A white rectangular frame with a black border  Description automatically generated**  **A screenshot of a cell phone  Description automatically generated A screenshot of a phone  Description automatically generated A screenshot of a phone  Description automatically generated**  **Treasure Trekker Singapore App Workflow in Xcode:**  **1. Overview**  The app I designed called Treasure Trekkers Singapore is an app that encourages users explore specific parts of Singapore by using 2 longitudinal and latitude sliders user interface widgets to simulate the effect of walking or running on the map to various locations that identified by pins and when users reach the specified pin locations, users would be granted points that they can use to win attractive prizes.  Users would also have their own profile page where they use the notes feature to write about their experiences about exploring various locations on the map and users would also be able to share photos in the profile view by displaying photos that are taken from the simulator photo gallery.  **2. Workflow and Features**   * Login Page:   When users first access the app, they will be greeted by a login page where they can input their email address, password and phone number into several UITextFields. Each input field has text validation for example for the email address, you can only input a valid email address, for the password field it requires 1 uppercase letter, 1 lowercase letter and 1 number, for the phone number field, it requires 8 numbers to be considered a valid phone number. When users have input the correct information, the submit button would be enabled then they would need to swipe right the button.   * Treasure Locations Tab   After the login page, users would see a Treasure Locations tab which is one of several tabs as of part of a tab bar view controller and in this tab which uses a UICollectionView and UICollectionViewCell, users can view information about the various treasure locations in our app using UIImageView and UILabels such as their image, name, challenge time levels, description and points granted for reaching each of them.   * Treasure Selection Tab   In the Treasure Selection tab, a UITableView, UITableViewCell and UISwitch is used to allow users to choose either to generate 5 random pins or select specified pins that they can generate can on the map. After a user have selected the pin locations, they would also have to select the corresponding challenge time levels for each pin which are easy, medium and hard which will allocate users with different amounts of time depending on the level of difficulty.   * Exploration Map Tab   When users select on the Exploration Map tab, they will see the map view that uses the MKMapView that would automatically zoom into an area around SUSS in Singapore and users would notice that the Start Exploration button which uses a UIButton is disabled.  After users have chosen the treasure locations, in the exploration map tab, they would notice that the button for exploration has been enabled and they would need to double tap the button to start an exploration. When the button is double tap the following actions would be triggered using MKPointAnnotation:   * + If users selected the Random Treasure Locations option, 5 random pins would be displayed on the map view.      * + If users instead selected other specific pins such as SIM Global Education and Lien Ying Chow Library, those pins would be displayed on the map view instead of 5 random pins.   + When users have chosen either 1 of the 2 options and have double tapped the Start Exploration button, they would notice that another pin called aAnnotation would also appear in the map view and they can use the 2 UISlider widgets just below the map view in the interface to move the aAnnotation pin in longitude and latitude directions to the treasure locations.   + When users have chosen either 1 of the 2 options and have double tapped the Start Exploration button, they would notice that there would be 2 timers that would immediately start counting, the 1st timer counts the amount of time used and the 2nd timer counts the amount of time remaining for the duration of the exploration.   + When the aAnnotation pin hits one of the other pins such as SIM Global Education or SIM Lien Ying Chow Library, users would be displayed with an alert message stating that they have reached a treasure location.   + Users would notice that their points would increase by 4 whenever the aAnnotation pin hits a treasure location and users would also notice they can only be granted the points once for each treasure location hit during the duration of the exploration.   + When an exploration has started users would notice that the exploration button text has changed to End Exploration and when the users double tap the button, the exploration would be ended and the 2 timers would stop counting. When the exploration has ended, the exploration button text would also revert to Start Exploration which would allow the user to begin a new exploration.   + During an exploration, if the user does not move the 2 sliders for 1 minute, the user would receive an alert message stating that they have stopped for too long, and they should continue moving.   + During an exploration, if the time remaining reaches the last 10 minutes mark, the user would receive an alert message stating that they only have 10 minutes left to complete the exploration.   + During an exploration, if the time remaining reaches 0, the user would receive an alert message stating that their time is up and the 2 timers would stop counting. * Status View Page   When an exploration has ended, the user would notice a button called Status View would appear and when they select the button, the user would be direct to a Status View page where UILabels are used to show statistical data about their exploration such as time used, time remaining, distance covered and current velocity.   * Gift Shop Tab   When users select on the Gift Shop tab, they will see a catalogue of gifts that uses a UICollectionView and UICollectionViewCell to display information using UIImageView, UILabels and UIButton about gifts such as SGD vouchers, watches, movie tickets and electronic devices and each gift also has a button with the text Redeem Gifts. Each gift contains information such as a gift image, name and points required to redeem the gift.  When users attain a certain amount of points that matches the points required for a gift in the gift shop, for example if the user has 8 points, the user would be eligible to redeem a $5 SGD Voucher and the button with the text Redeem Gift would be enabled within the UICollectionViewCell.  After user has returned from the gift redemption details page, the user would notice that their points has been reduced by the amount of points they used to redeem a gift for example if the user had 8 points, after redeeming a gift, their points value would drop to 0 points.   * Gift Redemption Details Page   When a user selects the Redeem Gift button when it is enabled, the user would be directed to a Gift Redemption page that uses UILabels, UITextFields and a UIButton where the user would have to input their email address and phone number followed by selecting the Submit button. After the user have selected the Submit button, the user would receive a message stating that they have successfully redeemed a gift. The user can now return to the gift shop page.   * User Profile Tab   When the user selects the User Profile tab, they would notice that the gift that they had redeemed is displayed in the first segment of a UISegmentControl and in a UITableView and UITableViewCell that uses a UIImageView and UILabels to display information such as gift image, name and points required to redeem the gift.  When the user selects the second segment, they would be greeted by the text that uses a UILabel stating that no image has been added yet, instead users can users can select on the Post a Photo button that uses a UIButton which will trigger a request for access to the simulator photo gallery. In the simulator photo gallery, the user can select on a photo and upload the photo to the 2nd segment within a UICollectionView and UICollectionViewCell. Users may also display multiple images to the 2nd segment UICollectionView.  When users select on the Write a Post button that uses a UIButton, they would be directed to a notes page.   * Notes Page   In the notes page that uses a UITableView, UITableViewCell and UILabels, users can select the + button that uses a UIButton and will be directed to another page that uses a UITextField and UITextView where they can input a post title and description followed by selecting the Save button that uses a UIButton to save their post. The user would then be returned to the initial notes page where they can see the record of their post. The notes page can be used by users to publish their experiences about the explorations that they were involved in. |

**Question 2(b) (65 marks)**

**ANS:**

|  |
| --- |
| **New functional feature created in our app:** **A gift redemption summary in our user profile view.**  **Rationale for Gift Redemption Summary:**  **Why this feature should be created:**  I created this feature because at first when the user selects a gift in the gift shop view and then keys in their email address and phone number in the gift redemption details view, they will then receive an alert message stating that they have successfully redeemed a gift and when they return to the gift shop view their accumulated points would be reduced based on the number of points the user used to redeem their gift.  However subsequently, if users were to forget about their gift redemptions, there would be no way for users to check their gift redemptions and users would not be able to track gift redemptions.  A gift redemption summary in the user profile page would provide users with a record of gifts that users have redeemed with information such as the gift image, name and points used which.  The gift redemption summary can also be used by users to showcase the gifts that they have redeemed to their peers and provide users with further motivation to continue using the app to redeem more gifts reaching more pins on the map and accumulate more points. Therefore, such gift redemption summary would further provide people with greater motivation to participate in outdoor activities such as running or walking to the pins on the map to attain points and redeem attractive prizes.  **How this feature can be created:**  The gift redemption summary feature applies a struct data structure named “RedeemedGift” to store important data relating to each redeemed gift such as the gift name, image and points required. This data for each redeemed gift is further stored in an array called “ExplorationData” to manage a record of each user gift redemptions.  The gift redemption summary feature makes use of a table view to display its information and in each table cell, a redeemed gift is displayed with relevant information such as their respective gift name, image and points required.  The gift redemption summary feature also makes use of a Swift UI @ObservedObject variable called “explorationData” that is used to program the gift redemption summary table view to automatically update whenever a user redeems a gift from the gift shop view.  The gift redemption summary feature also involves creating an instance called “RedeemedGift” and adding it to an array called “redeemedGifts” in our class called “ExplorationData” each time a user successfully redeems a gift by inputting their email address and phone number in my gift redemption controller called “RedemptionDetailsViewController”.  1. **EntryViewController Source Code:**  **import** UIKit  **class** EntryViewController: UIViewController {  // Outlets for the title input field and the note text view  **@IBOutlet** **var** titleField: UITextField! // Reference to a UITextField for entering the note's title  **@IBOutlet** **var** noteField: UITextView! // Reference to a UITextView for entering the note's content  // Optional closure to be called when the note is saved  // This will be set by the presenting view controller to handle the saved note data  **public** **var** completion: ((String, String) -> Void)?  //  // // Called when the view controller's view is loaded into memory  **override** **func** viewDidLoad() {  **super**.viewDidLoad()  // // Make the titleField the first responder, so the keyboard appears automatically  titleField.becomeFirstResponder()  //  // // Create a "Save" button on the right side of the navigation bar  // // When tapped, it will trigger the didTapSave function  navigationItem.rightBarButtonItem = UIBarButtonItem(title: "Save", style: .done, target: **self**, action: **#selector**(didTapSave))  }  // Function called when the "Save" button is tapped  **@objc** **func** didTapSave() {  // Check if there's text in both the titleField and noteField  **if** **let** text = titleField.text, !text.isEmpty, !noteField.text.isEmpty {  // If so, call the completion handler with the title and note text  completion?(text, noteField.text)  }  // If either field is empty, do nothing (no saving)  }  }  2. **ExplorationData Class Source Code:**  /\*  We'll use a singleton pattern to hold the data that needs to be shared between the two view controllers.    This singleton will be accessible from anywhere in your app.  \*/  **import** Foundation  // This class uses the singleton pattern to store and manage data shared across the application during an exploration.  **class** ExplorationData: ObservableObject  {  // **MARK: - Published Properties**    // These properties, when changed, will automatically update any parts of your UI that observe them.  // The user's current score  @Published **var** score: Int = 0    // An array to store redeemed gifts (assuming you have a RedeemedGift type)  @Published **var** redeemedGifts: [RedeemedGift] = []    // **MARK: - Shared Instance**    // This creates a single, shared instance of the ExplorationData class that can be accessed from anywhere in your app.  **static** **let** shared = ExplorationData()    // **MARK: - Properties**    **var** timeUsed: String = ""    **var** timeRemaining: String = ""    **var** distanceCovered: Double = 0    **var** currentVelocity: Double = 0    // **MARK: - Private Initializer**    // This private initializer ensures that only one instance of ExplorationData can be created.  **private** **init**() {}  }  3. **ExplorationViewController Source Code:**  **import** UIKit  **import** MapKit  **import** CoreLocation  // used to create audio video players and play  **import** AVFoundation  // used for create AV Player View -- acceleration audio, forward...  **import** AVKit  // If you are using SwiftUI elements  **import** SwiftUI  **class** ExplorationViewController: UIViewController, MKMapViewDelegate, TreasureSelectionDelegate {  // **MARK: - Outlets**    **@IBOutlet** **weak** **var** explorationMap: MKMapView!    **@IBOutlet** **weak** **var** explorationBtn: UIButton!    **@IBOutlet** **weak** **var** timeUsedLabel: UILabel!    **@IBOutlet** **weak** **var** resetTimerBtn: UIButton!    **@IBOutlet** **weak** **var** timeRemainingLabel: UILabel!    **@IBOutlet** **weak** **var** resetExplorationBtn: UIButton!  **@IBOutlet** **weak** **var** longSlider: UISlider!    **@IBOutlet** **weak** **var** latSlider: UISlider!    **@IBOutlet** **weak** **var** leftRightLabel: UILabel!    **@IBOutlet** **weak** **var** upDownLabel: UILabel!    **@IBOutlet** **weak** **var** statusViewBtn: UIButton!    **@IBOutlet** **weak** **var** pointsLabel: UILabel!    // **MARK: - Properties**  // Timer for tracking time used  **var** timer\_one:Timer = Timer()    // Timer for tracking time remaining  **var** timer\_two:Timer = Timer()    // Time used for random treasures (in seconds)  **var** randomTreasures\_timeUsedCount : Int = 0    // Changed variable name  **var** randomTreasures\_timeRemainingCount : Int = 0    **var** timerCounting : Bool = **false**    // Declare the flag  **var** hasPresentedAlert = **false**    **var** treasure\_timeUsedCount : Int = 0    **var** treasure\_timeRemainingCount: Int = 0    // Flag for initial start  **var** initialStart = **true**    // Location manager and annotations    // Manages location services  **let** location\_man = CLLocationManager()    // An annotation that will be moved  **let** aAnnotation = MKPointAnnotation()    // An annotation created on long press  **var** Annotation1 = MKPointAnnotation()    // Used for timing annotation movement  **var** Timestamp = 0.0    // Add a set to keep track of collided annotations  **var** collidedAnnotations: Set<MKPointAnnotation> = []    // Inactivity tracking  **var** inactivityTimer: Timer?    **var** inactivityDuration: TimeInterval = 0    // Declare variables to hold URLs for audio and video files  **let** file\_src = Bundle.main.url(forResource: "bensound-rebelsofourownkind", withExtension: "mp3")    // Audio player object  **var** playerAudio: AVAudioPlayer!    // Distance and score tracking  **var** previousLocation: CLLocationCoordinate2D?    **var** totalDistance: Double = 0    **var** score: Int = 0  **override** **func** viewDidLoad() {  **super**.viewDidLoad()    // Map view setup    view.addSubview(explorationMap)    explorationMap.frame = view.bounds    explorationMap.delegate = **self**    // Configure location manager for best accuracy and request authorization  location\_man.desiredAccuracy = kCLLocationAccuracyBest    location\_man.requestAlwaysAuthorization()    // Show the user's location on the map  explorationMap.showsUserLocation = **true**    // Initially hide all pins  explorationMap.removeAnnotations(explorationMap.annotations)    // UI setup    // Lock the button  explorationBtn.isEnabled = **false**    // Access the Tab Bar Controller and then the TreasureSelectionViewController  **if** **let** tabBarController = **self**.tabBarController,  **let** viewControllers = tabBarController.viewControllers {  **for** viewController **in** viewControllers {  **if** **let** treasureSelectionVC = viewController **as**? TreasureSelectionViewController {  // Set the delegate  treasureSelectionVC.delegate = **self**  **break**  }  }    // Create the double-tap gesture recognizer for the exploration button  **let** doubleTap = UITapGestureRecognizer(target: **self**, action: **#selector**(explorationBtnDoubleTapped(\_:)))    doubleTap.numberOfTapsRequired = 2    explorationBtn.addGestureRecognizer(doubleTap)  }    // Hide the button initially  resetExplorationBtn.isHidden = **true**    longSlider.isHidden = **true**  latSlider.isHidden = **true**    leftRightLabel.isHidden = **true**  upDownLabel.isHidden = **true**    // Start the inactivity timer  resetInactivityTimer()    resetTimerBtn.isHidden = **true**    statusViewBtn.isHidden = **true**    // Initialize the audio player  **if** **let** path = file\_src  {  // Try to create an audio player with the provided URL  playerAudio = **try**! AVAudioPlayer(contentsOf: path)  }  **else**  {  // Print an error message if the audio file is not found  print("Audio file not found")  }    // Prepare the audio player for playback  playerAudio.prepareToPlay()    // Print the audio duration to the console (for debugging purposes)  print(playerAudio.duration)  }    **override** **func** viewDidAppear(\_ animated: Bool)  {  // Set up the initial annotation when the view appears  aAnnotation.title = "Some where interesting"    aAnnotation.coordinate = CLLocationCoordinate2D(latitude: 1.3324, longitude: 103.775)  // Center the map on the annotation  **let** regionATMyloc = MKCoordinateRegion(  center: aAnnotation.coordinate,  latitudinalMeters: 1000,  longitudinalMeters: 1000  )    explorationMap.setRegion(regionATMyloc, animated: **true**)  }    // **MARK: - Slider Actions**    // IBActions for slider value changes  **@IBAction** **func** longChangedS(\_ sender: UISlider)  {  **let** newLocation = CLLocationCoordinate2D(latitude: aAnnotation.coordinate.latitude, longitude: 103.775 + Double(sender.value - 0.5) \* 0.01)    totalDistance += calculateDistance(from: aAnnotation.coordinate, to: newLocation)  aAnnotation.coordinate = newLocation  previousLocation = newLocation  // Adjust the annotation's longitude based on the slider value  aAnnotation.coordinate.longitude = 103.775 + Double(sender.value - 0.5) \* 0.01    // Check for collision after updating longitude  checkCollision()    // Reset the inactivity timer  resetInactivityTimer()  }    **@IBAction** **func** latChangedS(\_ sender: UISlider)  {  **let** newLocation = CLLocationCoordinate2D(latitude: 1.3324 + Double(sender.value - 0.5) \* 0.01, longitude: aAnnotation.coordinate.longitude)  // Calculate distance  totalDistance += calculateDistance(from: aAnnotation.coordinate, to: newLocation)  // Update annotation location  aAnnotation.coordinate = newLocation    previousLocation = newLocation  // Adjust the annotation's latitude based on the slider value  aAnnotation.coordinate.latitude = 1.3324 + Double(sender.value - 0.5) \* 0.01    // Check for collision after updating latitude  checkCollision()    // Reset the inactivity timer  resetInactivityTimer()  }    // **MARK: - Inactivity Timer**    **func** resetInactivityTimer() {  inactivityTimer?.invalidate()  inactivityDuration = 0  inactivityTimer = Timer.scheduledTimer(timeInterval: 1, target: **self**, selector: **#selector**(updateInactivityTimer), userInfo: **nil**, repeats: **true**)  }    **@objc** **func** updateInactivityTimer() {  inactivityDuration += 1    // 60 seconds of inactivity  **if** inactivityDuration >= 60 {  showInactivityAlert()    // Reset inactivityDuration after showing the alert  inactivityDuration = 0  }  }    **func** showInactivityAlert() {  // Check if the ExplorationViewController is currently visible  **if** **self**.tabBarController?.selectedViewController == **self** {  // Show an alert to the user about inactivity  **let** alertStoppedTooLong = UIAlertController(  title: "Stopped Too Long",  message: "You have stopped too long, please resume walking.",  preferredStyle: .alert  )  alertStoppedTooLong.addAction(UIAlertAction(title: "Cancel", style: .cancel, handler: { (\_) **in**  // Do nothing  }))  alertStoppedTooLong.addAction(UIAlertAction(title: "Okay", style: .destructive))  present(alertStoppedTooLong, animated: **true**)  }  }    // **MARK: - Annotation Movement (Currently Not Used)**  **func** checkCollision() {  **for** annotation **in** explorationMap.annotations {  // Skip the aAnnotation itself  **if** annotation === aAnnotation { **continue** }  // Check if the distance between the two annotations is less than a threshold (e.g., 5 meters)  **let** distance = calculateDistance(from: aAnnotation.coordinate, to: annotation.coordinate)    **if** distance < 10 {    // Check if this annotation has already collided  **guard** !collidedAnnotations.contains(annotation **as**! MKPointAnnotation) **else** { **continue** }  // Increment the score  score += 4    pointsLabel.text = "\(score)" // Update the pointsLabel  ExplorationData.shared.score = score // Update the shared score  // Collision detected! Show the alert  **let** alertLocationReached = UIAlertController(  title: "Reached Treasure Location",  message: "You have arrived at a Treasure Location",  preferredStyle: .alert  )    alertLocationReached.addAction(UIAlertAction(title: "Cancel", style: .cancel, handler: { (\_) **in**  //do nothing  }))    alertLocationReached.addAction(UIAlertAction(title: "Okay", style: .destructive))    present(alertLocationReached, animated: **true**)  // Optionally, you can remove the collided annotation or perform other actions here    // Add the collided annotation to the set  collidedAnnotations.insert(annotation **as**! MKPointAnnotation)  // Exit the loop after finding a collision  **break**  }  }  }    **func** calculateDistance(from coordinate1: CLLocationCoordinate2D, to coordinate2: CLLocationCoordinate2D) -> Double  {  **let** location1 = CLLocation(latitude: coordinate1.latitude, longitude: coordinate1.longitude)  **let** location2 = CLLocation(latitude: coordinate2.latitude, longitude: coordinate2.longitude)    **return** location1.distance(from: location2)    // Distance in meters  }    // 2. Implement the delegate methods  **func** didUnlockExploration() {  // Unlock the button  explorationBtn.isEnabled = **true**  }    **func** didLockExploration() {  // Lock the button  explorationBtn.isEnabled = **false**  }    **@IBAction** **func** resetTimerTapped(\_ sender: UIButton)  {  // Reset the initialStart flag when exploration is reset  initialStart = **true**    **let** alert = UIAlertController(title: "Reset Timer?", message: "Are you sure you would like to reset the Timer?", preferredStyle: .alert)  alert.addAction(UIAlertAction(title: "Cancel", style: .cancel, handler: { (\_) **in**  // do nothing  }))    alert.addAction(UIAlertAction(title: "Yes", style: .default, handler: { (\_) **in**  **self**.randomTreasures\_timeUsedCount = 0  **self**.timer\_one.invalidate()  **self**.timeUsedLabel.text = **self**.makeTimeString(hours: 0, minutes: 0, seconds: 0)  **self**.explorationBtn.setTitle("Start Exploration", for: .normal)  **self**.explorationBtn.setTitleColor(UIColor.blue, for: .normal)  }))    **self**.present(alert, animated: **true**, completion: **nil**)  }    **@IBAction** **func** resetExplorationTapped(\_ sender: **Any**)  {  longSlider.isHidden = **true**  latSlider.isHidden = **true**    leftRightLabel.isHidden = **true**  upDownLabel.isHidden = **true**      score = 0 // Reset the score  pointsLabel.text = "0" // Update the pointsLabel    **let** alert = UIAlertController(title: "Start Exploration?", message: "Are you sure you would like to reset the Exploration?", preferredStyle: .alert)    alert.addAction(UIAlertAction(title: "Cancel", style: .cancel, handler: { (\_) **in**  // do nothing  }))    alert.addAction(UIAlertAction(title: "Yes", style: .default, handler: { (\_) **in**    // 1. Reset the map view  **self**.explorationMap.removeAnnotations(**self**.explorationMap.annotations)    // 2. Reset the explorationBtn  **self**.explorationBtn.isEnabled = **false**  **self**.explorationBtn.setTitle("Start Exploration", for: .normal)  **self**.explorationBtn.setTitleColor(UIColor.blue, for: .normal)    // 3. Reset the TimerLabel  **self**.randomTreasures\_timeUsedCount = 0  **self**.timer\_one.invalidate()  **self**.timeUsedLabel.text = **self**.makeTimeString(hours: 0, minutes: 0, seconds: 0)    // 4. Reset the treasure selection switches  **if** **let** tabBarController = **self**.tabBarController,  **let** viewControllers = tabBarController.viewControllers {  **for** viewController **in** viewControllers {  **if** **let** treasureSelectionVC = viewController **as**? TreasureSelectionViewController {  treasureSelectionVC.resetSwitches() // Call a function to reset switches in TreasureSelectionViewController  }  }  }    // 5. Reset the timeRemainingLabel  **self**.timeRemainingLabel.text = **self**.makeTimeString(hours: 0, minutes: 0, seconds: 0)    **self**.resetTimerBtn.isHidden = **true**    // Hide the resetExplorationBtn after reset  **self**.resetExplorationBtn.isHidden = **true**  }))  **self**.present(alert, animated: **true**, completion: **nil**)  // Clear the collidedAnnotations set when resetting the exploration  collidedAnnotations.removeAll()  }    **@IBAction** **func** explorationBtnDoubleTapped(\_ sender: UIButton)  {  totalDistance = 0 // Reset distance when a new exploration starts  // Add the annotation to the map  explorationMap.addAnnotation(aAnnotation)    // Show the resetExplorationBtn when the exploration starts  longSlider.isHidden = **false**  latSlider.isHidden = **false**    leftRightLabel.isHidden = **false**  upDownLabel.isHidden = **false**    resetTimerBtn.isHidden = **false**    resetExplorationBtn.isHidden = **false**    // Access the Tab Bar Controller and then the TreasureSelectionViewController  **guard** **let** tabBarController = **self**.tabBarController,  **let** viewControllers = tabBarController.viewControllers **else** { **return** }    **for** viewController **in** viewControllers {  **if** **let** treasureSelectionVC = viewController **as**? TreasureSelectionViewController {    // Get the selected locations and their difficulty from TreasureSelectionViewController  **let** selectedLocations = treasureSelectionVC.selectedLocations    **if** selectedLocations.isEmpty {  // No location selected, handle this case (e.g., show an alert)  **return**  }    // Clear existing pins on the map  explorationMap.removeAnnotations(explorationMap.annotations)    **if** selectedLocations.count == 1 && selectedLocations.keys.contains(0) {    // Random location selected  \_ = treasurelocations[0] // Get the Random Treasure Locations object    **let** difficulty = selectedLocations[0]! // Get the selected difficulty    // Set timeRemainingCount only on initial start  **if** initialStart {  **switch** difficulty {  **case** .easy:  // randomTreasures\_timeRemainingCount = 11 \* 60 // 11 minute in seconds    // randomTreasures\_timeRemainingCount = 1 \* 60 // 1 minute in seconds    randomTreasures\_timeRemainingCount = 5 \* 60 \* 60 // 5 hours in seconds    **case** .medium:  randomTreasures\_timeRemainingCount = 2 \* 60 \* 60 + 30 \* 60 // 2.5 hours in seconds  **case** .hard:  randomTreasures\_timeRemainingCount = 1 \* 60 \* 60 + 15 \* 60 // 1.25 hours in seconds  }  initialStart = **false** // Set the flag to false after initial start  }    // Shuffle the treasurelocations array to randomize  **let** shuffledLocations = treasurelocations.shuffled()    // Add only the first 5 pins from the shuffled array  **for** i **in** 0..<5 {  **let** location = shuffledLocations[i]  **let** pin = MKPointAnnotation()  pin.coordinate = location.coordinate  pin.title = location.location  pin.subtitle = location.description  explorationMap.addAnnotation(pin)  explorationMap.addAnnotation(aAnnotation)    }  }    **else** {  **for** (row, difficulty) **in** selectedLocations  {  **let** location = treasurelocations[row] // Get the TreasureLocation object    // Add the pin to the map  **let** pin = MKPointAnnotation()  pin.coordinate = location.coordinate  pin.title = location.location  pin.subtitle = location.description  explorationMap.addAnnotation(pin)  explorationMap.addAnnotation(aAnnotation)    // Set timeRemainingCount only on initial start  **if** initialStart {  **switch** difficulty {  **case** .easy:    // randomTreasures\_timeRemainingCount = 1 \* 60 // 1 minute  // randomTreasures\_timeRemainingCount = 11 \* 60 // 11 minute    randomTreasures\_timeRemainingCount = 1 \* 60 \* 60 // 5 hours in seconds    **case** .medium:  randomTreasures\_timeRemainingCount = 30 \* 60 // 30 minutes in seconds  **case** .hard:  randomTreasures\_timeRemainingCount = 15 \* 60 // 15 minutes in seconds  }    // Set the flag to false after initial start  initialStart = **false**  }  }  }    **break**  }  }    **if**(timerCounting)  {  timerCounting = **false**    timer\_one.invalidate()  timer\_two.invalidate()    explorationBtn.setTitle("Start Exploration", for: .normal)  explorationBtn.setTitleColor(UIColor.blue, for: .normal)    // Stop playing the audio  playerAudio.stop()    statusViewBtn.isHidden = **false**  }  **else**  {  timerCounting = **true**  explorationBtn.setTitle("End Exploration", for: .normal)  explorationBtn.setTitleColor(UIColor.red, for: .normal)    // Initialize timers here  timer\_one = Timer.scheduledTimer(timeInterval: 1, target: **self**, selector: **#selector**(timerCounter), userInfo: **nil**, repeats: **true**)  timer\_two = Timer.scheduledTimer(timeInterval: 1, target: **self**, selector: **#selector**(timeRemainingCounter), userInfo: **nil**, repeats: **true**)    // Start playing the audio from the beginning  playerAudio.currentTime = 0  playerAudio.play()  }  }    // Map  **func** mapView(\_ mapView: MKMapView, viewFor annotation: MKAnnotation) -> MKAnnotationView?  {  **guard** annotation **is** MKPointAnnotation **else** { **return** **nil** }    **var** annotationView = explorationMap.dequeueReusableAnnotationView(withIdentifier: "custom")    **if** annotationView == **nil** {  // Create the view  annotationView = MKAnnotationView(  annotation: annotation,  reuseIdentifier: "custom"  )  annotationView!.rightCalloutAccessoryView = UIButton(type: .infoDark)  annotationView?.canShowCallout = **true**    annotationView?.calloutOffset = CGPoint(x: -10, y: -10)  }  **else** {  // Reuse the annotation view  annotationView?.annotation = annotation  }    // Set the hand-drawn pin image for aAnnotation  **if** annotation === aAnnotation {  annotationView?.image = UIImage(named: "icons8-pin-80")    } **else** {  annotationView?.image = UIImage(named: "icons8-pin-48") // For other annotations    }    **return** annotationView  }    // Handle annotation selection  **func** mapView(\_ mapView: MKMapView, didSelect view: MKAnnotationView)  {  **if** **let** annoteTitle = view.annotation?.title  {  print("tapped on:", annoteTitle!)  }  }    // Handle taps on the callout accessory  **func** mapView(\_ mapView: MKMapView, annotationView view: MKAnnotationView, calloutAccessoryControlTapped control: UIControl)  {  **let** alertWin = UIAlertController(  title: "Treasure Location",  message: "There is hidden treasure here.",  preferredStyle: .alert  )    alertWin.addAction(UIAlertAction(title: "Cancel", style: .cancel, handler: { (\_) **in**  //do nothing  }))    alertWin.addAction(UIAlertAction(  title: "Okay",  style: .destructive  )  )  present(alertWin, animated: **true**)  }    **@objc** **func** timerCounter() -> Void  {  randomTreasures\_timeUsedCount = randomTreasures\_timeUsedCount + 1  **let** time = secondsToHoursMinutesSeconds(seconds: randomTreasures\_timeUsedCount)  **let** timeString = makeTimeString(hours: time.0, minutes: time.1, seconds: time.2)  timeUsedLabel.text = timeString      // Update the shared data  ExplorationData.shared.timeUsed = timeString    ExplorationData.shared.distanceCovered = totalDistance        // Calculate velocity in meters per second  **let** velocity = randomTreasures\_timeUsedCount > 0 ? totalDistance / Double(randomTreasures\_timeUsedCount) : 0    ExplorationData.shared.currentVelocity = velocity    }    **func** secondsToHoursMinutesSeconds(seconds: Int) -> (Int, Int, Int)  {  **return** ((seconds / 3600), ((seconds % 3600) / 60),((seconds % 3600) % 60))  }    **func** makeTimeString(hours: Int, minutes: Int, seconds : Int) -> String  {  **var** timeString = ""  timeString += String(format: "%02d", hours)  timeString += " : "  timeString += String(format: "%02d", minutes)  timeString += " : "  timeString += String(format: "%02d", seconds)  **return** timeString  }    **@objc** **func** timeRemainingCounter() -> Void  {  **if** randomTreasures\_timeRemainingCount > 0 {  randomTreasures\_timeRemainingCount -= 1  **let** time = secondsToHoursMinutesSeconds(seconds: randomTreasures\_timeRemainingCount)  **let** timeString = makeTimeString(hours: time.0, minutes: time.1, seconds: time.2)  timeRemainingLabel.text = timeString    // Update the shared data  ExplorationData.shared.timeRemaining = timeString  }    **if** treasure\_timeRemainingCount > 0 {  treasure\_timeRemainingCount -= 1  **let** time = secondsToHoursMinutesSeconds(seconds: treasure\_timeRemainingCount)  **let** timeString = makeTimeString(hours: time.0, minutes: time.1, seconds: time.2)  timeRemainingLabel.text = timeString    // Update the shared data  ExplorationData.shared.timeRemaining = timeString  }      // Check for 10 minutes remaining  **if** randomTreasures\_timeRemainingCount == 10 \* 60 && !hasPresentedAlert {  hasPresentedAlert = **true**  **let** alertExplorationEnd = UIAlertController(  title: "10 minutes left",  message: "You have 10 minutes left, please complete your exploration.",  preferredStyle: .alert  )  alertExplorationEnd.addAction(UIAlertAction(title: "Cancel", style: .cancel, handler: { (\_) **in**  // Do nothing  }))  alertExplorationEnd.addAction(UIAlertAction(title: "Okay", style: .destructive))  present(alertExplorationEnd, animated: **true**)  }    **if** randomTreasures\_timeRemainingCount == 0 {  timer\_two.invalidate()  // Use a boolean flag to track if the alert has been presented  **if** !hasPresentedAlert { // Check the flag  hasPresentedAlert = **true** // Set the flag to true  // Show the alert when timeRemainingLabel reaches 0  **let** alertExplorationEnd = UIAlertController(  title: "Times Up",  message: "Your allotted time has ended.",  preferredStyle: .alert  )    alertExplorationEnd.addAction(UIAlertAction(title: "Cancel", style: .cancel, handler: { (\_) **in**  // Do nothing  }))    alertExplorationEnd.addAction(UIAlertAction(title: "Okay", style: .destructive))  present(alertExplorationEnd, animated: **true**)  }  }  }  }  4. **GiftRedeemedTableViewCell Class Source Code:**  **import** UIKit  // This class defines a custom table view cell to display information about redeemed gifts.  **class** GiftRedeemdTableViewCell: UITableViewCell {  // **MARK: - Outlets**  **@IBOutlet** **weak** **var** iconImageView: UIImageView!    **@IBOutlet** **weak** **var** gift\_redeemed: UILabel!    **@IBOutlet** **weak** **var** points\_required: UILabel!  }  5. **Gifts Swift File Source Code:**  **import** UIKit  // Defines a structure to hold information about each gift  **struct** Gift {  **let** title: String  **let** points\_needed: Int  **let** image: UIImage  // Add imageName property  **let** imageName: String  }  // Creates an array of Gift objects, each representing a different gift in the gift shop  **let** gifts: [Gift] = [  Gift(title: "$5 SGD Voucher", points\_needed: 8, image: imageLiteral(resourceName: "gift voucher"), imageName: "gift voucher"),    Gift(title: "$5 SGD Voucher", points\_needed: 8, image: imageLiteral(resourceName: "gift voucher"), imageName: "gift voucher"),    Gift(title: "$10 SGD Voucher", points\_needed: 12, image: imageLiteral(resourceName: "gift voucher"), imageName: "gift voucher"),    Gift(title: "$10 SGD Voucher", points\_needed: 12, image: imageLiteral(resourceName: "gift voucher"), imageName: "gift voucher"),    Gift(title: "$20 SGD Voucher", points\_needed: 16, image: imageLiteral(resourceName: "gift voucher"), imageName: "gift voucher"),    Gift(title: "$20 SGD Voucher", points\_needed: 16, image: imageLiteral(resourceName: "gift voucher"), imageName: "gift voucher"),    Gift(title: "Creative Zen Air DOT", points\_needed: 20, image: imageLiteral(resourceName: "Creative Zen Air DOT"), imageName: "Creative Zen Air DOT"),    Gift(title: "Cyxus Aviator Polarized Sunglasses", points\_needed: 24, image: imageLiteral(resourceName: "Cyxus Aviator Polarized Sunglasses"), imageName: "Creative Live Cam Sync 1080p V2"),    Gift(title: "2 GV Tickets for Transformers One", points\_needed: 20, image: imageLiteral(resourceName: "2 GV Tickets for Transformers One"), imageName: "2 GV Tickets for Transformers One"),    Gift(title: "Creative Live Cam Sync 1080p V2", points\_needed: 24, image: imageLiteral(resourceName: "Creative Live Cam Sync 1080p V2"), imageName: "Creative Live Cam Sync 1080p V2"),    Gift(title: "Casio Sports Watch F91W-1D", points\_needed: 16, image: imageLiteral(resourceName: "Casio Digital Watch F91W-1D"), imageName: "Casio Digital Watch F91W-1D")  ]  6. **GiftsCollectionViewCell Class Source Code:**  **import** UIKit  **class** GiftsCollectionViewCell: UICollectionViewCell {  // **MARK: - Outlets**    **@IBOutlet** **weak** **var** giftImageView: UIImageView!    **@IBOutlet** **weak** **var** giftTitleLabel: UILabel!    **@IBOutlet** **weak** **var** giftPriceLabel: UILabel!    **@IBOutlet** **weak** **var** giftBtn: UIButton!    // **MARK: - Properties**    **var** currentPoints: Int = 0    // Reference to the RedemptionDetailsViewController  **var** redemptionVC: RedemptionDetailsViewController!    // Stores the index path of the cell  **var** indexPath: IndexPath!    // **MARK: - Setup Function**    **func** setup(with gift: Gift) {  giftImageView.image = gift.image    giftTitleLabel.text = gift.title    giftPriceLabel.text = "\(gift.points\_needed)"    // Enable/disable the redeem button based on whether the user has enough points  giftBtn.isEnabled = currentPoints >= gift.points\_needed  }    // **MARK: - Actions**    **@IBAction** **func** giftBtnTapped(\_ sender: UIButton)  {  print("Redeem Gift Button Tapped")    // Set the selected gift in the redemptionVC  redemptionVC.selectedGift = gifts[indexPath.row]    // In GiftsCollectionViewCell, in the giftBtnTapped method, set the delegate of redemptionVC to the GiftShopViewController instance:  redemptionVC.delegate = **self**.window?.rootViewController **as**? GiftShopViewController    // Present the redemptionVC modally  **self**.window?.rootViewController?.present(redemptionVC, animated: **true**, completion: **nil**)  }  }  7. **GiftShopViewController Source Code:**  **import** UIKit  // This view controller manages the gift shop where users can redeem gifts using their points.  **class** GiftShopViewController: UIViewController {  // **MARK: - Outlets**    **@IBOutlet** **weak** **var** currentPointsLabel: UILabel!    **@IBOutlet** **weak** **var** giftCollectionView: UICollectionView!  // **MARK: - View Lifecycle**    **override** **func** viewDidLoad() {  **super**.viewDidLoad()    // Display the initial score from ExplorationData  currentPointsLabel.text = "\(ExplorationData.shared.score)"    // Set up the collection view  giftCollectionView.dataSource = **self**  giftCollectionView.delegate = **self**  giftCollectionView.collectionViewLayout = UICollectionViewFlowLayout()  }  **override** **func** viewWillAppear(\_ animated: Bool) {  **super**.viewWillAppear(animated)    // Update the score label with the latest value from ExplorationData  currentPointsLabel.text = "\(ExplorationData.shared.score)"    // Refresh the collection view to update button states (enable/disable based on points)  giftCollectionView.reloadData()  }  }  // **MARK: - UICollectionViewDataSource**  **extension** GiftShopViewController: UICollectionViewDataSource {    // Returns the number of gifts to display in the collection view  **func** collectionView(\_ collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {  **return** gifts.count  }    // Configures and returns a cell to display in the collection view  **func** collectionView(\_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) -> UICollectionViewCell {    **let** cell = collectionView.dequeueReusableCell(withReuseIdentifier: "GiftsCollectionViewCell", for: indexPath) **as**! GiftsCollectionViewCell    // Configure the cell with gift data  cell.setup(with: gifts[indexPath.row])    // Provide the current points to the cell  cell.currentPoints = ExplorationData.shared.score  // Instantiate RedemptionDetailsViewController  **let** redemptionVC = **self**.storyboard!.instantiateViewController(withIdentifier: "RedemptionDetailsViewController") **as**! RedemptionDetailsViewController    // Set the redemptionVC property of the cell  cell.redemptionVC = redemptionVC    // Set the indexPath property of the cell  cell.indexPath = indexPath  **return** cell  }  }  // **MARK: - UICollectionViewDelegateFlowLayout**  **extension** GiftShopViewController: UICollectionViewDelegateFlowLayout {    // Defines the size of each item (gift cell) in the collection view  **func** collectionView(\_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout, sizeForItemAt indexPath: IndexPath) -> CGSize {  **return** CGSize(width: 180, height: 300)  }  }  // **MARK: - UICollectionViewDelegate**  **extension** GiftShopViewController: UICollectionViewDelegate {    // Called when an item (gift cell) in the collection view is selected  **func** collectionView(\_ collectionView: UICollectionView, didSelectItemAt indexPath: IndexPath) {  print(gifts[indexPath.row].title)  }  }  // **MARK: - RedemptionDelegate**  // Make GiftShopViewController conform to the RedemptionDelegate protocol:  **extension** GiftShopViewController: RedemptionDelegate {    **func** didRedeemGift() {  // Update the points label and refresh the collection view after a gift is redeemed  currentPointsLabel.text = "\(ExplorationData.shared.score)"  giftCollectionView.reloadData()  }  }  8. **LoginViewController Source Code:**  **import** UIKit  **class** LoginViewController: UIViewController  {  // **MARK: - Outlets**  // These connect to UI elements in your Storyboard    **@IBOutlet** **weak** **var** emailTF: UITextField!    **@IBOutlet** **weak** **var** passwordTF: UITextField!    **@IBOutlet** **weak** **var** phoneTF: UITextField!    **@IBOutlet** **weak** **var** emailError: UILabel!    **@IBOutlet** **weak** **var** passwordError: UILabel!    **@IBOutlet** **weak** **var** phoneError: UILabel!    **@IBOutlet** **weak** **var** submitButton: UIButton!    // **MARK: - View Lifecycle**    **override** **func** viewDidLoad() {  **super**.viewDidLoad()  // Do any additional setup after loading the view.    // Create the swipe gesture recognizer  **let** swipeRight = UISwipeGestureRecognizer(target: **self**, action: **#selector**(submitButtonSwiped(\_:)))  swipeRight.direction = .right // Set the desired swipe direction  submitButton.addGestureRecognizer(swipeRight)    // Reset the form to its initial state  resetForm()  }    // **MARK: - Actions**  // These functions are called when the corresponding UI elements are interacted with  **@IBAction** **func** emailChanged(\_ sender: UITextField)  {  **if** **let** email = emailTF.text  {  // Validate the email address  **if** **let** errorMessage = invalidEmail(email)  {  emailError.text = errorMessage  emailError.isHidden = **false**  }  **else**  {  emailError.isHidden = **true**  }  }    // Check if the form is valid  checkForValidForm()  }      **@IBAction** **func** passwordChanged(\_ sender: UITextField)  {  **if** **let** password = passwordTF.text  {  **if** **let** errorMessage = invalidPassword(password)  {  passwordError.text = errorMessage  passwordError.isHidden = **false**  }  **else**  {  passwordError.isHidden = **true**  }  }    // Check if the form is valid  checkForValidForm()  }    **@IBAction** **func** phoneChanged(\_ sender: UITextField)  {  **if** **let** phoneNumber = phoneTF.text  {  // Validate the phone number  **if** **let** errorMessage = invalidPhoneNumber(phoneNumber)  {  phoneError.text = errorMessage  phoneError.isHidden = **false**  }  **else**  {  phoneError.isHidden = **true**  }  }    // Check if the form is valid  checkForValidForm()  }    // This function is called when the submit button is swiped right  **@IBAction** **func** submitButtonSwiped(\_ sender: **Any**)  {  // MARK - Using Story board    // Instantiate the TabBarController from the storyboard  **let** storyboard = **self**.storyboard?.instantiateViewController(withIdentifier: "TabBarController") **as**! UITabBarController  // Push the TabBarController onto the navigation stack  **self**.navigationController?.pushViewController(storyboard, animated: **true**)    // Reset the form to its initial state  resetForm()  }    // **MARK: - Helper Functions**  // These functions perform specific tasks within the view controller    // Resets the form to its initial state  **func** resetForm()  {  submitButton.isEnabled = **false**    emailError.isHidden = **false**  phoneError.isHidden = **false**  passwordError.isHidden = **false**    emailError.text = "Required"  phoneError.text = "Required"  passwordError.text = "Required"    emailTF.text = ""  passwordTF.text = ""  phoneTF.text = ""  }    // Validates the email address  **func** invalidEmail(\_ value: String) -> String?  {  **let** reqularExpression = "[A-Z0-9a-z.\_%+-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,64}"  **let** predicate = NSPredicate(format: "SELF MATCHES %@", reqularExpression)  **if** !predicate.evaluate(with: value)  {  **return** "Invalid Email Address"  }    **return** **nil**  }    // Validates the password  **func** invalidPassword(\_ value: String) -> String?  {  **if** value.count < 8  {  **return** "Password must be at least 8 characters"  }  **if** containsDigit(value)  {  **return** "Password must contain at least 1 digit"  }  **if** containsLowerCase(value)  {  **return** "Must contain 1 lowercase character"  }  **if** containsUpperCase(value)  {  **return** "Must contain 1 uppercase character"  }  **return** **nil**  }    // Checks if the string contains a digit  **func** containsDigit(\_ value: String) -> Bool  {  **let** reqularExpression = ".\*[0-9]+.\*"  **let** predicate = NSPredicate(format: "SELF MATCHES %@", reqularExpression)  **return** !predicate.evaluate(with: value)  }    // Checks if the string contains a lowercase character  **func** containsLowerCase(\_ value: String) -> Bool  {  **let** reqularExpression = ".\*[a-z]+.\*"  **let** predicate = NSPredicate(format: "SELF MATCHES %@", reqularExpression)  **return** !predicate.evaluate(with: value)  }    // Checks if the string contains an uppercase character  **func** containsUpperCase(\_ value: String) -> Bool  {  **let** reqularExpression = ".\*[A-Z]+.\*"  **let** predicate = NSPredicate(format: "SELF MATCHES %@", reqularExpression)  **return** !predicate.evaluate(with: value)  }    // Validates the phone number  **func** invalidPhoneNumber(\_ value: String) -> String?  {  **let** set = CharacterSet(charactersIn: value)  **if** !CharacterSet.decimalDigits.isSuperset(of: set)  {  **return** "Phone Number must contain only digits"  }    **if** value.count != 8  {  **return** "Phone Number must have 8 Digits"  }  **return** **nil**  }    // Checks if the form is valid and enables/disables the submit button accordingly  **func** checkForValidForm()  {  **if** emailError.isHidden && passwordError.isHidden && phoneError.isHidden  {  submitButton.isEnabled = **true**  }  **else**  {  submitButton.isEnabled = **false**  }  }  }  9. **NotesViewController Source Code:**  **import** UIKit  **class** NoteViewController: UIViewController {  // Outlets for displaying the note's title and content  **@IBOutlet** **var** titleLabel: UILabel! // Reference to a UILabel for displaying the note's title  **@IBOutlet** **var** noteLabel: UITextView! // Reference to a UITextView for displaying the note's content  // Public variables to hold the note's title and content.  // These will be set by the previous view controller when navigating to this one  **public** **var** noteTitle: String = ""  **public** **var** note: String = ""  // Called when the view controller's view is loaded into memory  **override** **func** viewDidLoad() {  **super**.viewDidLoad()    // Set the text of the titleLabel and noteLabel with the passed-in values  titleLabel.text = noteTitle  noteLabel.text = note  }  }  10. **PhotoCollectionViewCell Class Source Code:**  **import** UIKit  // This class defines a custom collection view cell to display photos.  **class** PhotoCollectionViewCell: UICollectionViewCell {    // A static identifier for reusing cells  **static** **let** identifier = "PhotoCollectionViewCell"    // A private UIImageView to display the photo within the cell  **private** **let** imageView: UIImageView = {  **let** imageView = UIImageView()    // Set content mode to fill the entire image view  imageView.contentMode = .scaleAspectFill    // Clip any content that goes beyond the bounds  imageView.clipsToBounds = **true**    **return** imageView  }()    // **MARK: - Initializers**    // This initializer is called when creating the cell programmatically  **override** **init**(frame: CGRect) {  **super**.init(frame: frame)    // Add the imageView to the cell's content view  contentView.addSubview(imageView)    // Create an array of UIImages (using optional chaining to handle potential nil images)  **let** images = [  UIImage(named: "1. Dover MRT Station"),    UIImage(named: "2. Blk 21 Dover Crescent Playground"),    UIImage(named: "3. One-north Park"),    UIImage(named: "4. National University Hospital (NUH)"),    UIImage(named: "5. Singapore Polytechnic (SP)"),    UIImage(named: "6. The Japanese Cemetery Park")  ].compactMap({ $0 })    // Set a random image from the array to the imageView  imageView.image = images.randomElement()  }    // This initializer is required when using the cell from a Storyboard/XIB  **required** **init**?(coder: NSCoder) {  // This should not be called, as we're creating the cell programmatically  fatalError()  }    // **MARK: - Layout**    // This function is called when the cell's layout needs to be updated  **override** **func** layoutSubviews() {  **super**.layoutSubviews()    // Make the imageView fill the entire cell  imageView.frame = contentView.bounds  }    // **MARK: - Reuse**    // This function is called when the cell is about to be reused  **override** **func** prepareForReuse() {  **super**.prepareForReuse()  // You can add any cleanup code here if needed (e.g., resetting the image)  }    // **MARK: - Public Function**    // Sets the image of the cell's imageView  **public** **func** setImage(\_ image: UIImage) {  imageView.image = image  }  }  11. **ProfileFirstViewController Source Code:**  **import** UIKit  **import** Foundation  // If you are using SwiftUI elements  **import** SwiftUI  // This view controller manages the first view in the user's profile, likely displaying a list of redeemed gifts.  **class** ProfileFirstViewController: UIViewController {  // **MARK: - Outlets**    // Outlet for the table view  **@IBOutlet** **weak** **var** redeemedGiftsTableView: UITableView!  // **MARK: - Properties**  // An observed object to access shared data  @ObservedObject **var** explorationData = ExplorationData.shared    // **MARK: - View Lifecycle**    **override** **func** viewDidLoad() {  **super**.viewDidLoad()  // Set the table view's data source  redeemedGiftsTableView.dataSource = **self**  }    **override** **func** viewWillAppear(\_ animated: Bool) {  **super**.viewWillAppear(animated)    // Reload the table view data when the view appears  redeemedGiftsTableView.reloadData()  }  }  // **MARK: - UITableViewDataSource**  // This extension provides data to the table view  **extension** ProfileFirstViewController: UITableViewDataSource  {  // Returns the number of redeemed gifts to display in the table view  **func** tableView(\_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {  **return** explorationData.redeemedGifts.count  }  // Configures and returns a cell to display in the table view  **func** tableView(\_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {    **let** cell = redeemedGiftsTableView.dequeueReusableCell(withIdentifier: "cell", for: indexPath) **as**! GiftRedeemdTableViewCell    **let** redeemedGift = explorationData.redeemedGifts[indexPath.row]  // Configure the cell with redeemedGift data  cell.gift\_redeemed.text = redeemedGift.title    cell.points\_required.text = "\(redeemedGift.points)"    // Assuming you store image names  cell.iconImageView.image = UIImage(named: redeemedGift.imageName)    **return** cell  }    // This function currently returns 0, which means the cells will not be visible.  // You likely want to return an appropriate height for your cells here.  **func** tableView(\_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {  **return** 0  }  }  12. **ProfileNotesViewController Source Code:**  **import** UIKit  **class** ProfileNotesViewController: UIViewController, UITableViewDelegate, UITableViewDataSource {  **@IBOutlet** **weak** **var** testLabel: UILabel!    // Outlets for the table view and label in the UI  **@IBOutlet** **var** table: UITableView! // Reference to the UITableView in the storyboard  **@IBOutlet** **var** label: UILabel! // Reference to a UILabel (likely for placeholder text when no notes exist)    // Array to store the notes. Each note is a tuple with a title and the actual note content  **var** models: [(title: String, note: String)] = []    **override** **func** viewDidLoad() {  **super**.viewDidLoad()    // Set the title of the navigation bar  title = "Notes"  // Set this view controller as the delegate and data source for the table view  // This means this class will handle providing data to the table and responding to user interactions  table.delegate = **self**  table.dataSource = **self**  }    // Action triggered when the "New Note" button is tapped  **@IBAction** **func** didTapNewnote() {  // Attempt to instantiate the "EntryViewController" from the storyboard  **guard** **let** vc = storyboard?.instantiateViewController(withIdentifier: "new") **as**? EntryViewController **else** {  **return** // If instantiation fails, do nothing and exit the function  }  // Set the title of the new note view controller  vc.title = "New Note"  // Disable large titles for this view controller  vc.navigationItem.largeTitleDisplayMode = .never  // Set a completion handler to be called when the new note is saved  vc.completion = { noteTitle, note **in**  // Navigate back to the root view controller (this view controller)  **self**.navigationController?.popToRootViewController(animated: **true**)  // Append the new note to the models array  **self**.models.append((title: noteTitle, note: note))  // Hide the label (likely a placeholder) and show the table  **self**.label.isHidden = **true**  **self**.table.isHidden = **false**  // Reload the table view to display the new note  **self**.table.reloadData()  }  // Push the new note view controller onto the navigation stack  navigationController?.pushViewController(vc, animated: **true**)  }  // **MARK: - Table View Data Source Methods**  // Return the number of rows (notes) to display in the table  **func** tableView(\_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {  **return** models.count  }  // Configure and return a cell to display at the specified index path  **func** tableView(\_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {  // Dequeue a reusable cell with the identifier "cell"  **let** cell = tableView.dequeueReusableCell(withIdentifier: "cell", for: indexPath)  // Set the title and note text for the cell  cell.textLabel?.text = models[indexPath.row].title  cell.detailTextLabel?.text = models[indexPath.row].note  **return** cell  }  // **MARK: - Table View Delegate Methods**  // Called when a row (note) is selected  **func** tableView(\_ tableView: UITableView, didSelectRowAt indexPath: IndexPath)  {  // Deselect the row with animation  tableView.deselectRow(at: indexPath, animated: **true**)  // Get the selected note model  **let** model = models[indexPath.row]  // Attempt to instantiate the "NoteViewController" from the storyboard  **guard** **let** vc = storyboard?.instantiateViewController(identifier: "note") **as**? NoteViewController **else** {  **return**  }  // Disable large titles for this view controller  vc.navigationItem.largeTitleDisplayMode = .never  // Set the title of the note view controller  vc.title = "Note"  // Pass the note title and content to the note view controller  vc.noteTitle = model.title  vc.note = model.note  // Push the note view controller onto the navigation stack  navigationController?.pushViewController(vc, animated: **true**)  }  }  13. **ProfileSecondViewController Source Code:**  **import** UIKit  // This view controller manages a collection view to display photos.  **class** ProfileSecondViewController: UIViewController, UICollectionViewDelegate, UICollectionViewDataSource, UICollectionViewDelegateFlowLayout {    // **MARK: - Outlets**  **@IBOutlet** **weak** **var** noImageLabel: UILabel!    // **MARK: - Properties**    // The collection view to display photos  **private** **let** collectionView = UICollectionView(  frame: .zero,  collectionViewLayout: UICollectionViewFlowLayout()  )    // An array to store the images to be displayed  **private** **var** images: [UIImage] = []    // **MARK: - View Lifecycle**  **override** **func** viewDidLoad() {  **super**.viewDidLoad()    // Initially hide the "no images" label  noImageLabel.isHidden = **true**    // Register the PhotoCollectionViewCell for use in the collection view  collectionView.register(PhotoCollectionViewCell.**self**, forCellWithReuseIdentifier: PhotoCollectionViewCell.identifier)  // Set the collection view's delegate  collectionView.delegate = **self**    // Set the collection view's data source  collectionView.dataSource = **self**    // Add the collection view to the view controller's view  view.addSubview(collectionView)    // This line of code tells the view to rearrange its subviews so that noImageLabel is placed at the top of the stack, ensuring it's visible above other elements like the collectionView and the UIView container.  view.bringSubviewToFront(noImageLabel)  }    // Called when the view's layout has been calculated and subviews are about to be positioned  **override** **func** viewDidLayoutSubviews() {  **super**.viewDidLayoutSubviews()    // Make the collection view fill the entire view  collectionView.frame = view.bounds  }    // **MARK: - Public Function**    // Adds an image to the images array and reloads the collection view  **func** addImage(\_ image: UIImage) {  images.append(image)  collectionView.reloadData()    // Hide the label after adding an image  noImageLabel.isHidden = **true**  }    // **MARK: - UICollectionViewDataSource**  // These functions provide data to the collection view  // Returns the number of photos to display  **func** collectionView(\_ collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {    **if** images.count == 0 {  // Show the label if no images  noImageLabel.isHidden = **false**  } **else** {  // Hide the label if images are present  noImageLabel.isHidden = **true**  }    **return** images.count  }  // Returns a configured cell for the collection view  **func** collectionView(\_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) -> UICollectionViewCell  {  **let** cell = collectionView.dequeueReusableCell(withReuseIdentifier: PhotoCollectionViewCell.identifier, for: indexPath) **as**! PhotoCollectionViewCell    // Set the image for the cell  cell.setImage(images[indexPath.item])  **return** cell  }  // **MARK: - UICollectionViewDelegateFlowLayout**  // These functions control the layout of the collection view  // Sets the size of each photo item in the collection view  **func** collectionView(\_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout, sizeForItemAt indexPath: IndexPath) -> CGSize  {  **return** CGSize(  width: (view.frame.size.width/3)-3,  height: (view.frame.size.width/3)-3  )  }    // Sets the spacing between items in the same row  **func** collectionView(\_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout, minimumInteritemSpacingForSectionAt section: Int) -> CGFloat {  **return** 1  }    // Sets the spacing between rows  **func** collectionView(\_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout, minimumLineSpacingForSectionAt section: Int) -> CGFloat {  **return** 1  }    // Sets the insets for the collection view section  **func** collectionView(\_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout, insetForSectionAt section: Int) -> UIEdgeInsets  {  **return** UIEdgeInsets(top: 1, left: 1, bottom: 1, right: 1)  }    // Called when an item in the collection view is selected  **func** collectionView(\_ collectionView: UICollectionView, didSelectItemAt indexPath: IndexPath) {  // Deselect the item with animation  collectionView.deselectItem(at: indexPath, animated: **true**)    print("Selected section \(indexPath.section) and row \(indexPath.row)")  }  }  14. **RedeemedGifts Swift File Source Code:**  **import** UIKit  // We need a way to store the redeemed gifts. Let's create a new structure to hold the redemption details:  **struct** RedeemedGift: Codable { // Make it Codable for storage  **let** title: String  **let** points: Int  **let** redemptionDate: Date  **let** imageName: String // Store the image name or identifier  }  15. **RedemptionDetailsViewController Source Code:**  **import** UIKit  // Protocol that allows this view controller to communicate with the GiftShopViewController  **protocol** RedemptionDelegate: AnyObject {  // Called when a gift is successfully redeemed  **func** didRedeemGift()  }  **class** RedemptionDetailsViewController: UIViewController {    // **MARK: - Outlets**    **@IBOutlet** **weak** **var** giftRedemptionEmailTF: UITextField!    **@IBOutlet** **weak** **var** giftRedemptionPhoneTF: UITextField!    **@IBOutlet** **weak** **var** giftRedemptionEmailError: UILabel!    **@IBOutlet** **weak** **var** giftRedemptionPhoneError: UILabel!    **@IBOutlet** **weak** **var** giftRedemptionSubmitBtn: UIButton!    // **MARK: - Properties**    // Pass gift data to RedemptionDetailsViewController  **var** selectedGift: Gift?    // In RedemptionDetailsViewController, add a delegate property:  **weak** **var** delegate: RedemptionDelegate?  // **MARK: - View Lifecycle**  **override** **func** viewDidLoad() {  **super**.viewDidLoad()  // Reset the form to its initial state  resetForm()  }    // **MARK: - Actions**    // These functions are called when the corresponding UI elements are interacted with  **@IBAction** **func** emailChanged(\_ sender: UITextField)  {  **if** **let** email = giftRedemptionEmailTF.text  {  // Validate the email address  **if** **let** errorMessage = invalidEmail(email)  {  giftRedemptionEmailError.text = errorMessage  giftRedemptionEmailError.isHidden = **false**  }  **else**  {  giftRedemptionEmailError.isHidden = **true**  }  }    // Check if the form is valid  checkForValidForm()  }    **@IBAction** **func** phoneChanged(\_ sender: UITextField)  {  **if** **let** phoneNumber = giftRedemptionPhoneTF.text  {  // Validate the phone number  **if** **let** errorMessage = invalidPhoneNumber(phoneNumber)  {  giftRedemptionPhoneError.text = errorMessage  giftRedemptionPhoneError.isHidden = **false**  }  **else**  {  giftRedemptionPhoneError.isHidden = **true**  }  }    // Check if the form is valid  checkForValidForm()  }    **@IBAction** **func** submitButtonTapped(\_ sender: **Any**)  {  **if** **let** gift = selectedGift {  // Deduct points for the redeemed gift  ExplorationData.shared.score -= gift.points\_needed    // Create a RedeemedGift object and add it to the redeemedGifts array in ExplorationData  **let** redeemedGift = RedeemedGift(title: gift.title, points: gift.points\_needed, redemptionDate: Date(), imageName: gift.imageName)  ExplorationData.shared.redeemedGifts.append(redeemedGift)  }    // Notify the delegate (GiftShopViewController) that a gift has been redeemed  delegate?.didRedeemGift()  // Show an alert to confirm gift redemption  **let** alertGiftRedemption = UIAlertController(  title: "Gift Redeemed",  message: "Congratulations, you have redeemed a gift.",  preferredStyle: .alert  )    alertGiftRedemption.addAction(UIAlertAction(title: "Cancel", style: .cancel, handler: { (\_) **in**  // Do nothing  }))    alertGiftRedemption.addAction(UIAlertAction(title: "Okay", style: .destructive))    present(alertGiftRedemption, animated: **true**)    // Reset the form after submission  resetForm()  }    // **MARK: - Helper Functions**  // These functions perform specific tasks within the view controller    // Resets the form to its initial state  **func** resetForm()  {  giftRedemptionSubmitBtn.isEnabled = **false**    giftRedemptionEmailError.isHidden = **false**  giftRedemptionPhoneError.isHidden = **false**    giftRedemptionEmailError.text = "Required"  giftRedemptionPhoneError.text = "Required"    giftRedemptionEmailTF.text = ""  giftRedemptionPhoneTF.text = ""  }    // Validates the email address  **func** invalidEmail(\_ value: String) -> String?  {  **let** reqularExpression = "[A-Z0-9a-z.\_%+-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,64}"  **let** predicate = NSPredicate(format: "SELF MATCHES %@", reqularExpression)  **if** !predicate.evaluate(with: value)  {  **return** "Invalid Email Address"  }    **return** **nil**  }  **func** containsDigit(\_ value: String) -> Bool  {  **let** reqularExpression = ".\*[0-9]+.\*"  **let** predicate = NSPredicate(format: "SELF MATCHES %@", reqularExpression)  **return** !predicate.evaluate(with: value)  }    **func** containsLowerCase(\_ value: String) -> Bool  {  **let** reqularExpression = ".\*[a-z]+.\*"  **let** predicate = NSPredicate(format: "SELF MATCHES %@", reqularExpression)  **return** !predicate.evaluate(with: value)  }    **func** containsUpperCase(\_ value: String) -> Bool  {  **let** reqularExpression = ".\*[A-Z]+.\*"  **let** predicate = NSPredicate(format: "SELF MATCHES %@", reqularExpression)  **return** !predicate.evaluate(with: value)  }    **func** invalidPhoneNumber(\_ value: String) -> String?  {  **let** set = CharacterSet(charactersIn: value)  **if** !CharacterSet.decimalDigits.isSuperset(of: set)  {  **return** "Phone Number must contain only digits"  }    **if** value.count != 8  {  **return** "Phone Number must have 8 Digits"  }  **return** **nil**  }    // Checks if the form is valid and enables/disables the submit button accordingly  **func** checkForValidForm()  {  **if** giftRedemptionEmailError.isHidden && giftRedemptionPhoneError.isHidden  {  giftRedemptionSubmitBtn.isEnabled = **true**  }  **else**  {  giftRedemptionSubmitBtn.isEnabled = **false**  }  }  }  16. **StatusViewController Source Code:**  **import** UIKit  **class** StatusViewController: UIViewController {    // **MARK: - Outlets**    **@IBOutlet** **weak** **var** StatusTimedUsedLabel: UILabel!    **@IBOutlet** **weak** **var** StatusTimeRemainingLabel: UILabel!    **@IBOutlet** **weak** **var** StatusDistanceCoveredLabel: UILabel!    **@IBOutlet** **weak** **var** StatusCurrentVelocityLabel: UILabel!    // **MARK: - View Lifecycle**    **override** **func** viewDidLoad() {  **super**.viewDidLoad()  // Retrieve and display the initial data from the shared ExplorationData singleton  StatusTimedUsedLabel.text = ExplorationData.shared.timeUsed    StatusTimeRemainingLabel.text = ExplorationData.shared.timeRemaining  **let** distanceMeters = ExplorationData.shared.distanceCovered    **let** distanceKm = distanceMeters / 1000    StatusDistanceCoveredLabel.text = String(format: "%.2f Kilometres", distanceKm)    StatusCurrentVelocityLabel.text = String(format: "%.2f m/s", ExplorationData.shared.currentVelocity)  }    **override** **func** viewWillAppear(\_ animated: Bool) {  **super**.viewWillAppear(animated)    // Refresh the labels with the latest data from ExplorationData whenever the view appears  StatusTimedUsedLabel.text = ExplorationData.shared.timeUsed    StatusTimeRemainingLabel.text = ExplorationData.shared.timeRemaining    **let** distanceMeters = ExplorationData.shared.distanceCovered    **let** distanceKm = distanceMeters / 1000    StatusDistanceCoveredLabel.text = String(format: "%.2f Kilometres", distanceKm)    StatusCurrentVelocityLabel.text = String(format: "%.2f m/s", ExplorationData.shared.currentVelocity)  }  }  17. **Treasure Swift File Source Code:**  **import** UIKit  // Defines a Treasure struct to hold information about each treasure location  **struct** Treasure  {  // Name of the location  **let** location: String    // Image of the location  **let** image: UIImage    // Time limit for finding the treasure  **let** timeLimit: String    // Reward for finding the treasure  **let** reward: String    // Description of the treasure or task  **let** description: String  }  // Creates an array of Treasure objects, each representing a different treasure location  **let** treasures: [Treasure] = [    // Treasure Location 1  Treasure(  location: "SIM Global Education",  image: imageLiteral(resourceName: "1. SIM Global Education"),  timeLimit: "Challenge Time Levels (Easy/Medium/Hard - Minutes): 60 / 30 / 15",  reward: "Points: 4",  description: "Description: Find the plaque commemorating SIM's founding."  ),    // Treasure Location 2  Treasure(  location: "Lien Ying Chow Library",  image: imageLiteral(resourceName: "2. Lien Ying Chow Library"),  timeLimit: "Challenge Time Levels (Easy/Medium/Hard - Minutes): 60 / 30 / 15",  reward: "Points: 4",  description: "Description: How many floors does the library have?"  ),    // Treasure Location 3  Treasure(  location: "Ngee Ann Poly Futsal Court",  image: imageLiteral(resourceName: "3. Ngee Ann Poly Futsal Court"),  timeLimit: "Challenge Time Levels (Easy/Medium/Hard - Minutes): 60 / 30 / 15",  reward: "Points: 4",  description: "Description: Take a panoramic photo of the entire court."  ),    // Treasure Location 4  Treasure(  location: "King Albert Park",  image: imageLiteral(resourceName: "4. King Albert Park"),  timeLimit: "Challenge Time Levels (Easy/Medium/Hard - Minutes): 60 / 30 / 15",  reward: "Points: 4",  description: "Description: Locate the playground within the park."  ),    // Treasure Location 5  Treasure(  location: "Ngee Ann Poly Block 50",  image: imageLiteral(resourceName: "5. Ngee Ann Poly Block 50"),  timeLimit: "Challenge Time Levels (Easy/Medium/Hard - Minutes): 60 / 30 / 15",  reward: "Points: 4",  description: "Description: Find the directory board in the building."  ),    // Treasure Location 6  Treasure(  location: "Old Jurong Railway",  image: imageLiteral(resourceName: "6. Old Jurong Railway"),  timeLimit: "Challenge Time Levels (Easy/Medium/Hard - Minutes): 60 / 30 / 15",  reward: "Points: 4",  description: "Description: Take a selfie with the railway station."  ),    // Treasure Location 7  Treasure(  location: "King Albert Lodge",  image: imageLiteral(resourceName: "7. King Albert Lodge"),  timeLimit: "Challenge Time Levels (Easy/Medium/Hard - Minutes): 60 / 30 / 15",  reward: "Points: 4",  description: "Description: How many windows on the front facade?"  ),    // Treasure Location 8  Treasure(  location: "Methodist Girls' School",  image: imageLiteral(resourceName: "8. Methodist Girls' School"),  timeLimit: "Challenge Time Levels (Easy/Medium/Hard - Minutes): 60 / 30 / 15",  reward: "Points: 4",  description: "Description: Locate the main entrance to MGS."  ),    // Treasure Location 9  Treasure(  location: "Rail Corridor (Bukit Timah)",  image: imageLiteral(resourceName: "9. Rail Corridor (Bukit Timah)"),  timeLimit: "Challenge Time Levels (Easy/Medium/Hard - Minutes): 60 / 30 / 15",  reward: "Points: 4",  description: "Description: Find a marker post along the Rail Corridor."  ),  // Treasure Location 10  Treasure(  location: "Singapore Institute of Technology (SIT@NP)",  image: imageLiteral(resourceName: "10. Singapore Institute of Technology (SIT@NP)"),  timeLimit: "Challenge Time Levels (Easy/Medium/Hard - Minutes): 60 / 30 / 15",  reward: "Points: 4",  description: "Description: Find the SIT logo."  )    ]  18. **TreasureCollectionViewCell Class Source Code:**  **import** UIKit  // This class defines the layout and behavior of a single cell within your Treasure Collection View  **class** TreasureCollectionViewCell: UICollectionViewCell  {  // **MARK: - Outlets**  // Connect to the UI elements within your cell's design in the Storyboard    // Displays the treasure image  **@IBOutlet** **weak** **var** treasureImageView: UIImageView!    // Shows the treasure location name  **@IBOutlet** **weak** **var** treasureLocationLabel: UILabel!    // Shows the time limit for the treasure  **@IBOutlet** **weak** **var** treasureTimeLimitLabel: UILabel!    // Provides a description of the treasure or task  **@IBOutlet** **weak** **var** treasureDescriptionLabel: UILabel!    // Displays the reward for finding the treasure  **@IBOutlet** **weak** **var** treasureRewardLabel: UILabel!    // **MARK: - Setup Function**  // This function configures the cell with data from a Treasure object    **func** setup(with treasure: Treasure) {  treasureImageView.image = treasure.image    treasureLocationLabel.text = treasure.location    treasureTimeLimitLabel.text = treasure.timeLimit    treasureRewardLabel.text = treasure.reward    treasureDescriptionLabel.text = treasure.description  }  }  19. **TreasureCustomTableViewCell Class Source Code:**  **import** UIKit  // This class defines a custom table view cell for selecting treasure locations and their difficulty levels  **class** TreasureCustomTableViewCell: UITableViewCell  {  // **MARK: - Outlets**    // Connect to UI elements in the cell's design in the Storyboard  **@IBOutlet** **weak** **var** treasureLocationLabel: UILabel!    **@IBOutlet** **weak** **var** selectLocationLabel: UILabel!    **@IBOutlet** **weak** **var** selectLocationSwitch: UISwitch!    **@IBOutlet** **weak** **var** challengeTimeLevelsLabel: UILabel!    **@IBOutlet** **weak** **var** easyLabel: UILabel!    **@IBOutlet** **weak** **var** easySwitch: UISwitch!    **@IBOutlet** **weak** **var** mediumLabel: UILabel!    **@IBOutlet** **weak** **var** mediumSwitch: UISwitch!    **@IBOutlet** **weak** **var** hardLabel: UILabel!    **@IBOutlet** **weak** **var** hardSwitch: UISwitch!    // **MARK: - Cell Lifecycle**    **override** **func** awakeFromNib()  {  **super**.awakeFromNib()    // Set labels with proper text    // Rename labels  selectLocationLabel.text = "Select Location"  challengeTimeLevelsLabel.text = "Challenge Time Levels"  easyLabel.text = "Easy"  mediumLabel.text = "Medium"  hardLabel.text = "Hard"    // Set all switches to off initially  selectLocationSwitch.isOn = **false**  easySwitch.isOn = **false**  mediumSwitch.isOn = **false**  hardSwitch.isOn = **false**    // Hide challenge options initially  hideChallengeOptions()  }    // **MARK: - Actions**  // These functions are called when UI elements are interacted with  // Called when the "Select Location" switch is toggled  **@IBAction** **func** selectLocationSwitchToggled(\_ sender: UISwitch)  {  **if** sender.isOn  {  // Show difficulty options when location is selected  showChallengeOptions()  } **else** {  // Hide difficulty options when location is deselected  hideChallengeOptions()  }  }    // Called when the "Easy" switch is toggled  **@IBAction** **func** easySwitchToggled(\_ sender: UISwitch)  {  **if** sender.isOn  {  // Ensure only one difficulty is selected at a time  mediumSwitch.isOn = **false**  hardSwitch.isOn = **false**  }  }    // Called when the "Medium" switch is toggled  **@IBAction** **func** mediumSwitchToggled(\_ sender: UISwitch)  {  **if** sender.isOn  {  // Ensure only one difficulty is selected at a time  easySwitch.isOn = **false**  hardSwitch.isOn = **false**  }  }    // Called when the "Hard" switch is toggled  **@IBAction** **func** hardSwitchToggled(\_ sender: UISwitch)  {  **if** sender.isOn  {  // Ensure only one difficulty is selected at a time  mediumSwitch.isOn = **false**  easySwitch.isOn = **false**  mediumSwitch.isOn = **false**  }  }    // **MARK: - Helper Functions**  // These functions perform specific tasks within the cell  // Shows the difficulty level options  **func** showChallengeOptions() {  challengeTimeLevelsLabel.isHidden = **false**  easyLabel.isHidden = **false**  easySwitch.isHidden = **false**  mediumLabel.isHidden = **false**  mediumSwitch.isHidden = **false**  hardLabel.isHidden = **false**  hardSwitch.isHidden = **false**  }  // Hides the difficulty level options  **func** hideChallengeOptions() {  challengeTimeLevelsLabel.isHidden = **true**  easyLabel.isHidden = **true**  easySwitch.isHidden = **true**  mediumLabel.isHidden = **true**  mediumSwitch.isHidden = **true**  hardLabel.isHidden = **true**  hardSwitch.isHidden = **true**  }  }  20. **TreasureSelection Swift File Source Code:**  **import** UIKit  **import** MapKit  **import** CoreLocation  // Defines a structure to hold information about each treasure location  **struct** TreasureLocation  {  // Name of the location  **let** location: String    // Coordinates of the location  **let** coordinate: CLLocationCoordinate2D    // Description of the task or challenge  **let** description: String    // Difficulty level (easy, medium, hard)  **var** difficulty: Difficulty    // Time limit to find the treasure (in seconds)  **var** timeLimit: Int    // Reward points for finding the treasure  **var** reward: Int  }  // Creates an array of TreasureLocation objects, each representing a different treasure location  **var** treasurelocations: [TreasureLocation] = [    // Treasure Location 1  TreasureLocation(  location: "Random Treasure Locations",  coordinate: CLLocationCoordinate2D(latitude: 1.2867,longitude: 103.7910),  description: "Challenge Time Levels (Easy/Medium/Hard - Minutes): 60 / 30 / 15",  // "Challenge Time Levels (Easy/Medium/Hard - Minutes): 300 / 150 / 75"  difficulty: Difficulty.easy,  timeLimit: 1,  reward: 1  ),    // 1. SIM Global Education Coordinates  TreasureLocation(  location: "SIM Global Education",  coordinate: CLLocationCoordinate2D(  latitude: 1.3294,  longitude: 103.7762  ),  description: "Find the plaque commemorating SIM's founding.",  // "Challenge Time Levels (Easy/Medium/Hard - Minutes): 60 / 30 / 15"  difficulty: .easy,  timeLimit: 1,  reward: 1  ),    // 2. Lien Ying Chow Library Coordinates  TreasureLocation(  location: "Lien Ying Chow Library",  coordinate: CLLocationCoordinate2D(  latitude: 1.333586,  longitude: 103.776868  ),  description: "How many floors does the library have?",  // "Challenge Time Levels (Easy/Medium/Hard - Minutes): 60 / 30 / 15"  difficulty: .easy,  timeLimit: 1,  reward: 1  ),    // 3. Ngee Ann Poly Futsal Court Coordinates  TreasureLocation(  location: "Ngee Ann Poly Futsal Court",  coordinate: CLLocationCoordinate2D(  latitude: 1.332060,  longitude: 103.777084  ),  description: "Take a panoramic photo of the entire court.",  // "Challenge Time Levels (Easy/Medium/Hard - Minutes): 60 / 30 / 15"  difficulty: .easy,  timeLimit: 1,  reward: 1  ),    // 4. King Albert Park Coordinates  TreasureLocation(  location: "King Albert Park",  coordinate: CLLocationCoordinate2D(  latitude: 1.334044,  longitude: 103.779700  ),  description: "Locate the playground within the park.",  // "Challenge Time Levels (Easy/Medium/Hard - Minutes): 60 / 30 / 15"  difficulty: .easy,  timeLimit: 1,  reward: 1  ),    // 5. Ngee Ann Poly Block 50 Coordinates  TreasureLocation(  location: "Ngee Ann Poly Block 50",  coordinate: CLLocationCoordinate2D(  latitude: 1.331557,  longitude: 103.773971  ),  description: "Find the directory board in the building.",  // "Challenge Time Levels (Easy/Medium/Hard - Minutes): 60 / 30 / 15"  difficulty: .easy,  timeLimit: 1,  reward: 1  ),    // 6. Old Jurong Railway  TreasureLocation(  location: "Old Jurong Railway",  coordinate: CLLocationCoordinate2D(  latitude: 1.328323,  longitude: 103.780313  ),  description: "Take a selfie with the railway station.",  // "Challenge Time Levels (Easy/Medium/Hard - Minutes): 60 / 30 / 15"  difficulty: .easy,  timeLimit: 1,  reward: 1  ),    // 7. King Albert Lodge Coordinates  TreasureLocation(  location: "King Albert Lodge",  coordinate: CLLocationCoordinate2D(  latitude: 1.336467,  longitude: 103.780312  ),  description: "How many windows on the front facade?",  // "Challenge Time Levels (Easy/Medium/Hard - Minutes): 60 / 30 / 15"  difficulty: .easy,  timeLimit: 1,  reward: 1  ),    // 8. Maju Camp Coordinates  TreasureLocation(  location: "Methodist Girls' School",  coordinate: CLLocationCoordinate2D(  latitude: 1.332852,  longitude: 103.783459  ),  description: "Locate the main entrance to MGS.",  // "Challenge Time Levels (Easy/Medium/Hard - Minutes): 60 / 30 / 15"  difficulty: .easy,  timeLimit: 1,  reward: 1  ),    // 9. Rail Corridor (Bukit Timah) Coordinates  TreasureLocation(  location: "Rail Corridor (Bukit Timah)",  coordinate: CLLocationCoordinate2D(  latitude: 1.331612,  longitude: 103.781242  ),  description: "Find a marker post along the Rail Corridor.",  // "Challenge Time Levels (Easy/Medium/Hard - Minutes): 60 / 30 / 15"  difficulty: .easy,  timeLimit: 1,  reward: 1  ),    // 10. Singapore Institute of Technology (SIT@NP) Coordinates  TreasureLocation(  location: "Singapore Institute of Technology (SIT@NP)",  coordinate: CLLocationCoordinate2D(  latitude: 1.334106,  longitude: 103.774496  ),  description: "Find the SIT logo.",  // "Challenge Time Levels (Easy/Medium/Hard - Minutes): 60 / 30 / 15"  difficulty: .easy,  timeLimit: 1,  reward: 1  )  ]  // Defines an enumeration for the difficulty levels of treasure locations  **enum** Difficulty  {  **case** easy, medium, hard  }  21. **TreasureSelectionViewController Source Code:**  **import** UIKit  // 1. Define the delegate protocol  // This protocol allows this view controller to communicate with other parts of your app  **protocol** TreasureSelectionDelegate: AnyObject  {  // Called when an exploration becomes available  **func** didUnlockExploration()    // Called when an exploration is no longer available  **func** didLockExploration()  }  **class** TreasureSelectionViewController: UIViewController, UITableViewDataSource, UITableViewDelegate {    // **MARK: - Outlets**    **@IBOutlet** **weak** **var** table: UITableView!    // **MARK: - Properties**    // Dictionary to store selected locations and their difficulties  **var** selectedLocations: [Int: Difficulty] = [:]    // Flag to indicate if the "Random" location is selected  **var** isRandomLocationSelected = **false**    // Delegate to communicate with other view controllers  **weak** **var** delegate: TreasureSelectionDelegate?  // **MARK: - View Lifecycle**    **override** **func** viewDidLoad() {  **super**.viewDidLoad()    // Set the data source for the table view  table.dataSource = **self**    // Set the delegate for the table view  table.delegate = **self**  }    // **MARK: - UITableViewDataSource**  // These functions provide data to the table view  // Returns the number of treasure locations to display in the table view  **func** tableView(\_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int  {  **return** treasurelocations.count  }    // Configures and returns a cell to display in the table view  **func** tableView(\_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell  {  // Get the TreasureLocation for this row  **let** sunset = treasurelocations[indexPath.row]    // Set the location label text    **let** cell = table.dequeueReusableCell(withIdentifier: "cell", for: indexPath) **as**! TreasureCustomTableViewCell    // Set the location label text  cell.treasureLocationLabel.text = sunset.location  // Add targets for switches to call the switchToggled function when their value changes  cell.selectLocationSwitch.addTarget(**self**, action: **#selector**(switchToggled(\_:)), for: .valueChanged)    cell.easySwitch.addTarget(**self**, action: **#selector**(switchToggled(\_:)), for: .valueChanged)    cell.mediumSwitch.addTarget(**self**, action: **#selector**(switchToggled(\_:)), for: .valueChanged)    cell.hardSwitch.addTarget(**self**, action: **#selector**(switchToggled(\_:)), for: .valueChanged)    // Set the switch state based on the selectedLocations dictionary  **if** **let** difficulty = selectedLocations[indexPath.row] {  cell.selectLocationSwitch.isOn = **true**  cell.showChallengeOptions()  **switch** difficulty {  **case** .easy:  cell.easySwitch.isOn = **true**  cell.mediumSwitch.isOn = **false**  cell.hardSwitch.isOn = **false**  **case** .medium:  cell.easySwitch.isOn = **false**  cell.mediumSwitch.isOn = **true**  cell.hardSwitch.isOn = **false**  **case** .hard:  cell.easySwitch.isOn = **false**  cell.mediumSwitch.isOn = **false**  cell.hardSwitch.isOn = **true**  }  } **else** {  cell.selectLocationSwitch.isOn = **false**  cell.hideChallengeOptions()  }  // Disable/enable cells based on isRandomLocationSelected  **if** isRandomLocationSelected && indexPath.row > 0  {  // Disable other locations if "Random" is selected  cell.selectLocationSwitch.isEnabled = **false**  } **else** **if** !isRandomLocationSelected && indexPath.row > 0 {  // Only disable other locations if Random is selected  cell.selectLocationSwitch.isEnabled = **true**  } **else** {  // Always enable the "Random" location cell  cell.selectLocationSwitch.isEnabled = **true**  }    **return** cell  }    // Sets the height of each row in the table view  **func** tableView(\_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {  **return** 200  }    // **MARK: - Actions**    // Called when a switch is toggled  **@objc** **func** switchToggled(\_ sender: UISwitch) {  // Get the cell that contains the switch that was toggled  **let** cell = sender.superview?.superview **as**! TreasureCustomTableViewCell // Access the custom cell    // Check which difficulty switch was toggled  **if** sender == cell.easySwitch {  print("easy switch toggled")  } **else** **if** sender == cell.mediumSwitch {  print("medium switch toggled")  } **else** **if** sender == cell.hardSwitch {  print("hard switch toggled")  }  // Get the index path of the cell  **guard** **let** indexPath = table.indexPath(for: cell) **else** { **return** }    // "Random Treasure Locations" cell  **if** indexPath.row == 0 {  isRandomLocationSelected = cell.selectLocationSwitch.isOn    // Disable/enable other cells  **for** i **in** 1..<treasurelocations.count {  table.cellForRow(at: IndexPath(row: i, section: 0))?.isUserInteractionEnabled = !isRandomLocationSelected  }    } **else** {  // Other treasure location cells  **if** cell.selectLocationSwitch.isOn {  // Disable the "Random Treasure Locations" cell  isRandomLocationSelected = **false**    table.cellForRow(at: IndexPath(row: 0, section: 0))?.isUserInteractionEnabled = **false**    **var** difficulty: Difficulty    **if** cell.easySwitch.isOn { difficulty = .easy }    **else** **if** cell.mediumSwitch.isOn { difficulty = .medium }    **else** { difficulty = .hard }    selectedLocations[indexPath.row] = difficulty  // Update the difficulty in the treasurelocations array  treasurelocations[indexPath.row].difficulty = difficulty  } **else** {  // Check if any other location is selected before enabling "Random Treasure Locations"  **var** anyOtherLocationSelected = **false**    **for** i **in** 1..<treasurelocations.count **where** i != indexPath.row {  **if** **let** otherCell = table.cellForRow(at: IndexPath(row: i, section: 0)) **as**? TreasureCustomTableViewCell,  otherCell.selectLocationSwitch.isOn {  anyOtherLocationSelected = **true**  **break**  }  }    table.cellForRow(at: IndexPath(row: 0, section: 0))?.isUserInteractionEnabled = !anyOtherLocationSelected    // Remove the location from selectedLocations  selectedLocations.removeValue(forKey: indexPath.row)  }  }    // Check if the selectLocationSwitch and at least one difficulty switch is on  **if** cell.selectLocationSwitch.isOn &&  (cell.easySwitch.isOn || cell.mediumSwitch.isOn || cell.hardSwitch.isOn) {  delegate?.didUnlockExploration()  } **else** {  delegate?.didLockExploration()  }  // Update the selectedLocations dictionary  **if** cell.selectLocationSwitch.isOn {  **let** difficulty: Difficulty  **if** cell.easySwitch.isOn { difficulty = .easy }  **else** **if** cell.mediumSwitch.isOn { difficulty = .medium }  **else** { difficulty = .hard }  selectedLocations[indexPath.row] = difficulty  } **else** {  selectedLocations.removeValue(forKey: indexPath.row)  }  }  // **MARK: - Helper Functions**    // Resets all switches to their initial state  **func** resetSwitches() {  selectedLocations = [:] // Clear the selectedLocations dictionary  isRandomLocationSelected = **false**  // Reload the table view to reflect the changes  table.reloadData()  }  }  22. **TreasureViewController Source Code:**  **import** UIKit  **class** TreasureViewController: UIViewController  {  // **MARK: - Outlets**    // Outlet for the collection view that displays treasures  **@IBOutlet** **weak** **var** collectionView: UICollectionView!    // **MARK: - View Lifecycle**  **override** **func** viewDidLoad() {  **super**.viewDidLoad()  // Do any additional setup after loading the view.    // Set the data source and delegate for the collection view  collectionView.dataSource = **self**  collectionView.delegate = **self**    // Set the collection view layout to a flow layout  collectionView.collectionViewLayout = UICollectionViewFlowLayout()  }  }  // **MARK: - UICollectionViewDataSource**  // This extension provides data to the collection view  **extension** TreasureViewController: UICollectionViewDataSource  {  // Returns the number of treasures to display in the collection view  **func** collectionView(\_ collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {  **return** treasures.count  }    // Configures and returns a cell to display in the collection view  **func** collectionView(\_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) -> UICollectionViewCell  {  // Dequeue a reusable cell with the specified identifier  **let** cell = collectionView.dequeueReusableCell(withReuseIdentifier: "TreasureCollectionViewCell", for: indexPath) **as**! TreasureCollectionViewCell    // Configure the cell with data from the corresponding Treasure object  cell.setup(with: treasures[indexPath.row])    **return** cell  }  }  // **MARK: - UICollectionViewDelegateFlowLayout**  // This extension allows you to customize the layout of items in the collection view  **extension** TreasureViewController: UICollectionViewDelegateFlowLayout {  **func** collectionView(\_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout, sizeForItemAt indexPath: IndexPath) -> CGSize {  **return** CGSize(width: 392, height: 440)  }  }  // **MARK: - UICollectionViewDelegate**  // This extension handles user interaction with the collection view  **extension** TreasureViewController: UICollectionViewDelegate {  **func** collectionView(\_ collectionView: UICollectionView, didSelectItemAt indexPath: IndexPath) {  print(treasures[indexPath.row].location)  }  }  23. **UserProfileViewController Source Code:**  **import** UIKit  // This view controller manages the user's profile, which has two segments controlled by a UISegmentedControl.  **class** UserProfileViewController: UIViewController {  // **MARK: - Outlets**    // Segmented control to switch between profile views  **@IBOutlet** **weak** **var** control: UISegmentedControl!    // Container view to hold the currently active profile view  **@IBOutlet** **weak** **var** containerView: UIView!  **@IBOutlet** **weak** **var** postPhotoBtn: UIButton!    // **MARK: - Properties**  // Stores the image selected by the user  **var** selectedImage: UIImage?    // **MARK: - Child View Controllers**    // These are the two view controllers that will be displayed in the container view  **private** **lazy** **var** firstViewController: ProfileFirstViewController = {  **let** storyboard = UIStoryboard(name: "Main", bundle: Bundle.main)  **var** viewController = storyboard.instantiateViewController(identifier: "ProfileFirstViewController") **as**! ProfileFirstViewController  // Add the first view controller as a child  **self**.add(asChildViewController: viewController)  **return** viewController  }()    **private** **lazy** **var** secondViewController: ProfileSecondViewController = {  **let** storyboard = UIStoryboard(name: "Main", bundle: Bundle.main)  **var** viewController = storyboard.instantiateViewController(identifier: "ProfileSecondViewController") **as**! ProfileSecondViewController  // Add the second view controller as a child  **self**.add(asChildViewController: viewController)  **return** viewController  }()    // **MARK: - View Lifecycle**  **override** **func** viewDidLoad() {  **super**.viewDidLoad()    // Set the background color of the segmented control  control.backgroundColor = .cyan    // Initially display the first view controller  updateView()  }  // **MARK: - Actions**  **@IBAction** **func** didChangeSegment(\_ sender: UISegmentedControl) {  // Update the displayed view controller when the segment changes  updateView()  }    **@IBAction** **func** postPhotoBtnDidTap(\_ sender: UIButton)  {  // Allow the user to select a photo from their library  **let** vc = UIImagePickerController()  vc.sourceType = .photoLibrary  vc.delegate = **self**  vc.allowsEditing = **true**  present(vc, animated: **true**)  }  // **MARK: - Helper Functions**    // These functions manage adding and removing child view controllers  // Updates the displayed view controller based on the selected segment  **private** **func** updateView() {  **if** control.selectedSegmentIndex == 0 {  // Remove the second view controller  remove(asChildViewController: secondViewController)    // Add the first view controller  add(asChildViewController: firstViewController)  } **else** **if** control.selectedSegmentIndex == 1 {  // Remove the first view controller  remove(asChildViewController: firstViewController)    // Add the second view controller  add(asChildViewController: secondViewController)  }  }  // Adds a view controller as a child to this view controller  **private** **func** add(asChildViewController viewController: UIViewController) {  addChild(viewController)  containerView.addSubview(viewController.view)  viewController.view.frame = containerView.bounds  viewController.view.autoresizingMask = [.flexibleWidth, .flexibleHeight]  viewController.didMove(toParent: **self**)  }  // Removes a child view controller from this view controller  **private** **func** remove(asChildViewController viewController: UIViewController) {  viewController.willMove(toParent: **nil**)  viewController.view.removeFromSuperview()  viewController.removeFromParent()  }  }  // **MARK: - UIImagePickerControllerDelegate, UINavigationControllerDelegate**  // This extension handles the image picker delegate methods  **extension** UserProfileViewController: UIImagePickerControllerDelegate, UINavigationControllerDelegate {    **func** imagePickerControllerDidCancel(\_ picker: UIImagePickerController)  {  // Dismiss the image picker if canceled  picker.dismiss(animated: **true**, completion: **nil**)  }  **func** imagePickerController(\_ picker: UIImagePickerController, didFinishPickingMediaWithInfo info: [UIImagePickerController.InfoKey : **Any**])  {  // Dismiss the image picker  picker.dismiss(animated: **true**, completion: **nil**)  **guard** **let** image = info[UIImagePickerController.InfoKey.editedImage] **as**? UIImage **else**  {  // Return if no edited image is found  **return**  }  **self**.selectedImage = image  // Pass the selected image to the ProfileSecondViewController if it's the active view  **if** control.selectedSegmentIndex == 1  {  **if** **let** secondVC = **self**.children.first(where: { $0 **is** ProfileSecondViewController }) **as**? ProfileSecondViewController {  secondVC.addImage(image)  }  }  }  } |