MATES Computer Science

Senior Capstone Project Final Report

Project Title	Athena
Team Members	Alex Henbest
Link to Github	https://github.com/AlexBH74/Athena/tree/main

I. Executive Summary

My project is an Apple game for daily trivia. The app draws from a csv file of trivia questions and multiple choice answers. When the app opens there is a login and create account screen. Once you log in the first time you are immediately brought to the home screen upon opening the app. After logging in there is an onboarding screen that has slides that lists the game's summary and general instructions. After clicking next past all of the slides the screen changes and there is an interface with easy, medium, and hard modes. Each player will be able to play one question for each mode each day. Once the user presses the mode of the question they will be able to click a start button. A timer will start after the button is clicked and the screen will display the question with four multiple choice answers. The timer can be paused but the trivia question is blocked from view if the game is stopped. The question and timer are

saved until the day is up or the question is answered. At the end whether or not you answered correctly and your time will be displayed in a pop-up. That difficulty will then be locked until the date changes.

II. Detailed Summary

The original inspirations for my project came from Dr. Bixler's daily brain teasers. I originally intended to make a brainteaser game, but this proved to be too difficult for the time frame. After this discovery, I reverted to a daily trivia game. As a huge nerd myself I have always been into trivia. I love solving questions and learning new facts so this game was the perfect fit for me.

My project was mainly sourced in Apple's custom code language, Swift, and the project was hosted in Apple's code editor app designed around Swift, XCode. There was also some minor work done in Firebase and Python.

- Swift/XCode: https://www.swift.org/install/macos/ (XCode Version 14.0)
- Python: https://www.python.org/downloads/
- Firebase: https://console.firebase.google.com/u/0/

During the first sprint, I started by familiarizing myself with Swift and Xcode, learning the layout of both interfaces. I then connected my project to Github and figured out how to commit and push my code. After some initial exploration of XCode, I moved on to coding. I followed a video tutorial to create a food order app to learn Swift fundamentals. While following the tutorial, I also worked on my own project to retain knowledge. I built a launch screen and made

significant progress on the onboarding slides controller. Below are some photos and descriptions from my first progress period.

First iPhone image is the main storyboard where the slideshow design is located. It has an image collection up top with two labels and a next button below

The second iPhone is the launch screen story storyboard and contains the loading screen design.

Below is a struct class that creates that designates the structure of each element in the onboarding slideshow

```
8  import UIKit
9
10 // Struct representing an individual slide in the onboarding flow
11 struct OnboardingSlide {
12     let title: String          // Title of the slide
13     let description: String    // Description of the slide
14     let image: UIImage         // Image displayed on the slide
15 }
```

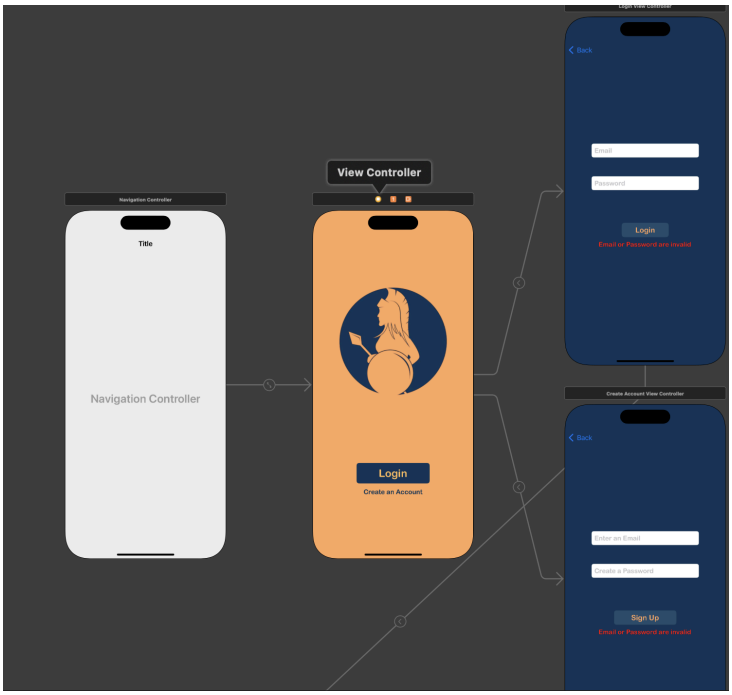
```swift
// Importing the UIKit framework which provides fundamental building blocks for iOS app development
import UIKit

// Declaring a class named OnboardingViewController, subclassed from UIViewController, responsible for managing the onboarding flow
class OnboardingViewController: UIViewController {

    // Declaring IBOutlet properties representing UI elements
    @IBOutlet weak var collectionView: UICollectionView! // UICollectionView to display slides
    @IBOutlet weak var nextButton: UIButton! // Button to proceed to the next slide
    @IBOutlet weak var pageControl: UIPageControl! // Page control to indicate the current slide

    // Declaring a property to hold onboarding slides
    var slides: [OnboardingSlide] = []

    // Method called after the view controller's view is loaded into memory
    override func viewDidLoad() {
        super.viewDidLoad()

        // Setting up the collection view delegate and data source
        collectionView.delegate = self
        collectionView.dataSource = self

        // Initializing the array of slides with dummy data
        slides = [
            OnboardingSlide(title: "Title 1", description: "Description 1", image: ),
            OnboardingSlide(title: "Title 2", description: "Description 2", image: ),
            OnboardingSlide(title: "Title 3", description: "Description 3", image: )
        ]
    }

    // Action method called when the next button is clicked
    @IBAction func nextBtnClicked(_ sender: Any) {
        // Implement code to handle navigation to the next slide
    }

}
// Extension of OnboardingViewController to conform to UICollectionViewDelegate, UICollectionViewDataSource, and UICollectionViewDelegateFlowLayout protocols
extension OnboardingViewController: UICollectionViewDelegate, UICollectionViewDataSource, UICollectionViewDelegateFlowLayout {

    // Method to specify the number of items in the collection view
    func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {
        return slides.count
    }

    // Method to configure and return cells for the collection view
    func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) -> UICollectionViewCell {
        let cell = collectionView.dequeueReusableCell(withReuseIdentifier: OnboardingCollectionViewCell.identifier, for: indexPath) as! OnboardingCollectionViewCell
        cell.setup(slides[indexPath.row])
        return cell
    }

    // Method to specify the size of items in the collection view
    func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout, sizeForItemAt indexPath: IndexPath) -> CGSize {
        return CGSize(width: collectionView.frame.width, height: collectionView.frame.height)
    }
}
```

Above is the onboarding view controller class that is the backend to the view controller within the main storyboard does

```swift
import UIKit

// Custom collection view cell for displaying onboarding slides
class OnboardingCollectionViewCell: UICollectionViewCell {

    // Static property to define a reusable identifier for the cell
    static let identifier = String(describing: OnboardingCollectionViewCell.self)

    // Outlets for UI elements within the cell
    @IBOutlet weak var slideimageView: UIImageView!      // Image view for the slide image
    @IBOutlet weak var slideTitle: UILabel!              // Label for the slide title
    @IBOutlet weak var slideDescription: UILabel!        // Label for the slide description

    // Method to configure the cell with data from an OnboardingSlide
    func setup(_ slide: OnboardingSlide) {
        slideimageView.image = slide.image            // Set the image view's image
        slideTitle.text = slide.title                 // Set the title label's text
        slideDescription.text = slide.description     // Set the description label's text
    }
}
```

Above is the class that controls the cells of the collection view and designates what type of data each cell in the collection view displays

During the second sprint I tackled the login function using Firebase for my backend database. Next, I attempted to build a database of brainteasers. This proved challenging. Creating my own brain teasers that were both difficult and codable would be time-consuming, while premade options were too easy. I pivoted to trivia instead. After a week of wrestling with MySQL download and operation, I successfully uploaded a trivia CSV to a MySQL database. Below are some photos from this period.



To the left is my main login interface attached to my login and create account screens

To the right is the onboarding screen where the slideshow design is located. I edited the colors and sizing of everything

```swift
import UIKit
import Firebase

class LoginViewController: UIViewController {

    @IBOutlet weak var usernameTextField: UITextField!
    @IBOutlet weak var passwordTextField: UITextField!
    @IBOutlet weak var invalidText: UILabel!

    override func viewDidLoad() {
        super.viewDidLoad()
        self.invalidText.isHidden = true
        // Do any additional setup after loading the view.
    }

    @IBAction func loginClicked(_ sender: Any) {
        guard let email = usernameTextField.text else { return }
        guard let password = passwordTextField.text else { return }

        Auth.auth().signIn(withEmail: email, password: password) { firebaseResult, error in
            if let e = error {
                self.invalidText.isHidden = false
            }
            else {
                self.performSegue(withIdentifier: "goToNext", sender: self)
            }
        }
    }

}
```

```swift
import UIKit
import Firebase

class CreateAccountViewController: UIViewController {

    @IBOutlet weak var usernameTextField: UITextField!
    @IBOutlet weak var passwordTextField: UITextField!
    @IBOutlet weak var invalidText: UILabel!

    override func viewDidLoad() {
        super.viewDidLoad()
        self.invalidText.isHidden = true
        // Do any additional setup after loading the view.
    }

    @IBAction func signupClicked(_ sender: Any) {
        guard let email = usernameTextField.text else { return }
        guard let password = passwordTextField.text else { return }

        Auth.auth().createUser(withEmail: email, password: password) { firebaseResult, error in
            if let e = error {
                self.invalidText.isHidden = false
            }
            else {
                self.performSegue(withIdentifier: "goToNext", sender: self)
            }
        }
    }
}
```

The two view controller classes above display the functions used within both the login and create account view controllers. They are the same code with the only difference being where they are connected to on the main storyboard

The third sprint saw slower progress than anticipated. Having confirmed CSV uploads to MySQL, I focused on data cleaning. With Python, I pulled questions from an online database, creating a 1000+ question CSV. Google Sheets formulas tackled some data errors, but manual review was necessary for capitalization and punctuation correction across all questions and answers. This consumed most of the week. After a successful MySQL upload, I shifted to researching and attempting the connection between MySQL and Xcode. Below are pictures from this sprint.

```sql
-- Create a new database called trivia
create database trivia_mc;

-- Create a table called questions in the trivia database with three columns of text
create table trivia_mc.questions(
    category varchar(100),
    difficulty varchar(20),
    question varchar(3000),
    correctAnswer varchar(1000),
    incorrect1 varchar(1000),
    incorrect2 varchar(1000),
    incorrect3 varchar(1000)
);

-- Disable strict mode in MySQL session to allow for only warnings where certain errors occur
SET @@SESSION.sql_mode = REPLACE(@@SESSION.sql_mode, 'STRICT_TRANS_TABLES', '');

-- Load data from a CSV file located at 'C:/ProgramData/MySQL/MySQL Server 8.0/Data/trivia_mc.csv' into the trivia.questions table
LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Data/trivia_mc.csv'
INTO TABLE trivia_mc.questions
FIELDS TERMINATED BY ","   -- The data in the CSV file is separated by commas
IGNORE 1 ROWS;   -- The first row is ignored as it contains column headers
```

category	difficulty	question	correctAnswer	incorrect1	incorrect2	incorrect3
The Arts	Easy	"In Edward Lear's poem, which bird sang to the ...	The Owl	The Robin	The Swan	The Albatross
The Arts	Easy	Which author wrote 'Harry Potter and the Philos...	J. K. Rowling	Christopher Tolkien	Philip Pullman	C. S. Lewis
The Arts	Easy	In which book series does Frodo Baggins appear?	The Lord of the Rings	Harry Potter	Red Rising	The Hunger Games
The Arts	Easy	Which author wrote 'The History of Middle-Earth'?	J. R. R. Tolkien	G. K. Chesterton	Philip Pullman	C. S. Lewis
The Arts	Easy	Who is the American artist who uses Campbell's ...	Andy Warhol	Jackson Pollock	Gertrude Stein	Aaron Copland
The Arts	Easy	In which book is Bilbo Baggins the main character?	The Hobbit	Charlottes Web	Alices Adventures in Wonderland	Bluebeard
The Arts	Easy	"In E.B. White's classic children's book 'Charlott...	A Spider	A Pig	A Sheep	A Horse
The Arts	Easy	Which author wrote 'The Hobbit'?	J. R. R. Tolkien	G. K. Chesterton	Philip Pullman	C. S. Lewis
The Arts	Easy	Which author wrote 'A Midsummer Night's Dream'?	William Shakespeare	Arthur C. Clarke	Isaac Newton	Enid Blyton
The Arts	Easy	Which author wrote 'Harry Potter and the Deat...	J. K. Rowling	Christopher Tolkien	Philip Pullman	C. S. Lewis
The Arts	Easy	"Which piece of written work starts with the line...	Lolita	Les Liaisons dange...	And Then There Were None	Ulysses
The Arts	Easy	Which author wrote 'The Origin of Species'?	Charles Darwin	Percy Bysshe Shelley	H. Rider Haggard	Robert Louis Stev...
The Arts	Easy	What is the word for a composition made of cut ...	Collage	Watercolor	Sculpture	Pointilism
The Arts	Easy	Who wrote A Christmas Carol'?	Charles Dickens	Thomas Hardy	James Joyce	Emily Brontë
The Arts	Easy	Which piece of written work starts with the line '...	The Hobbit	Charlottes Web	Alices Adventures in Wonderland	Bluebeard
The Arts	Easy	In which book series does Prince Caspian appear?	The Chronicles of Na...	Voyages Extraordi...	Harry Potter	Percy Jackson & t...
The Arts	Easy	In which book series does Thalia Grace appear?	Percy Jackson & the ...	Twilight	Voyages Extraordinaires	Harry Potter
The Arts	Easy	"Who said 'But, soft! what light through yonder ...	Romeo	Juliet	Hamlet	Paris
The Arts	Easy	What word describes a painting executed in a si...	Monochrome	Magenta	Unicolor	Solohue
The Arts	Easy	In which book series does Lord Voldemort appear?	Harry Potter	The Lord of the Ri...	The Inheritace Series	The Hunger Games
The Arts	Easy	What is the word given to a flat board used by ...	A palette	An eisel	A square	A pitcher

Above is the code and output for my MySQL database containing all 1036 rows in the csv.

```
url1 = "https://the-trivia-api.com/v2/questions?categories=science&difficulties=easy&limit=50"
response = requests.get(url1)
if response.status_code == 200:
    easy_science = response.json()
else:
    print("Failed to retrieve questions. Status code:", response.status_code)


url2 = "https://the-trivia-api.com/v2/questions?categories=science&difficulties=medium&limit=50"
response = requests.get(url2)
if response.status_code == 200:
    med_science = response.json()
else:
    print("Failed to retrieve questions. Status code:", response.status_code)


url3 = "https://the-trivia-api.com/v2/questions?categories=science&difficulties=hard&limit=50"
response = requests.get(url3)
if response.status_code == 200:
    hard_science = response.json()
else:
    print("Failed to retrieve questions. Status code:", response.status_code)


science = easy_science + med_science + hard_science
```

Above is the python code used to draw the questions and answers from the online database (This is only for one category).

```
list1 = arts + generalKnowledge + geography + history + science + sports
df = pd.DataFrame(list1)
df = df.drop(columns = ['id', 'type', 'regions', 'isNiche'])
df
```

```
df.to_csv('trivia.csv', index=False)
```

Above is the python code that condensed the lists for each category into a single list that was converted to a dataframe, and then into a csv.

During sprint #4, I ran into issues with MySQL. Privacy settings embedded in my MacBook would not allow for remote php access and were too time consuming to figure out, so I put MySQL on pause and just locally imported the CSVs into my project. In order to do so I had to create a custom dataframe model to import my CSV data in a structured format. I then assigned values from the dataframe to specific objects in the EasyGameViewController using indexing. This displayed the category at the top and the question with answer buttons below. I implemented a randomizer to select a row and display its values. Additionally, I wrote code to

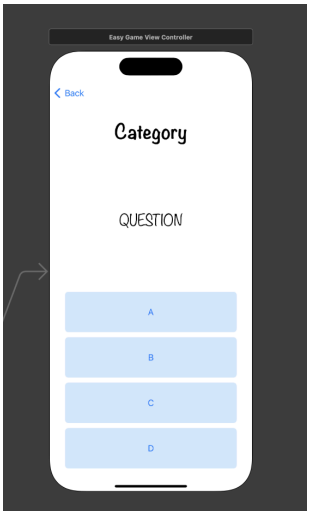
prevent the randomizer from selecting the same row twice. Finally, I added logic to display "Correct!" or "Incorrect!" based on the chosen answer.

```
category*difficulty*question*correctAnswer*incorrect1*incorrect2*incorrect3
The Arts*Easy*In Edward Lear's poem, which bird sang to the pussycat?*The Owl*The Robin*The Swan*The Albatross
The Arts*Easy*Which author wrote 'Harry Potter and the Philosopher's Stone'?*J. K. Rowling*Christopher Tolkien*Philip Pullman*C. S. Lewis
The Arts*Easy*In which book series does Frodo Baggins appear?*The Lord of the Rings*Harry Potter*Red Rising*The Hunger Games
The Arts*Easy*Which author wrote 'The History of Middle-Earth'?*J. R. R. Tolkien*G. K. Chesterton*Philip Pullman*C. S. Lewis
The Arts*Easy*Who is the American artist who uses Campbell's Soup cans in his pop art?*Andy Warhol *Jackson Pollock*Gertrude Stein*Aaron Copland
The Arts*Easy*In which book is Bilbo Baggins the main character?*The Hobbit*Charlottes Web*Alices Adventures in Wonderland*Bluebeard
The Arts*Easy*In E.B. White's classic children's book 'Charlotte's Web', what kind of animal is Charlotte?*A Spider*A Pig*A Sheep*A Horse
The Arts*Easy*Which author wrote 'The Hobbit'?*J. R. R. Tolkien*G. K. Chesterton*Philip Pullman*C. S. Lewis
The Arts*Easy*Which author wrote 'A Midsummer Night's Dream'?*William Shakespeare*Arthur C. Clarke*Isaac Newton*Enid Blyton
The Arts*Easy*Which author wrote 'Harry Potter and the Deathly Hallows'?*J. K. Rowling*Christopher Tolkien*Philip Pullman*C. S. Lewis
The Arts*Easy*Which piece of written work starts with the line 'Lolita, light of my life, fire of my loins.'?*Lolita*Les Liaisons dangereuses*And Then There Were None*Ulysses
The Arts*Easy*Which author wrote 'The Origin of Species'?*Charles Darwin*Percy Bysshe Shelley*H. Rider Haggard*Robert Louis Stevenson
The Arts*Easy*What is the word for a composition made of cut and pasted pieces of materials?*Collage*Watercolor*Sculpture*Pointilism
The Arts*Easy*Who wrote A Christmas Carol'?*Charles Dickens*Thomas Hardy*James Joyce*Emily Brontë
The Arts*Easy*Which piece of written work starts with the line 'In a hole in the ground there lived a hobbit.'?*The Hobbit*Charlottes Web*Alices Adventures in Wonderland*Bluebeard
The Arts*Easy*In which book series does Prince Caspian appear?*The Chronicles of Narnia*Voyages Extraordinaires*Harry Potter*Percy Jackson & the Olympians
The Arts*Easy*In which book series does Thalia Grace appear?*Percy Jackson & the Olympians*Twilight*Voyages Extraordinaires*Harry Potter
The Arts*Easy*Who said 'But, soft! what light through yonder window breaks'?*Romeo*Juliet*Hamlet*Paris
The Arts*Easy*What word describes a painting executed in a single color?*Monochrome*Magenta*Unicolor*Solohue
The Arts*Easy*In which book series does Lord Voldemort appear?*Harry Potter*The Lord of the Rings*The Inheritace Series*The Hunger Games
```

Above is the new look of the csv file I imported

```swift
import UIKit

struct DataFrame {
    var columns: [String]
    var rows: [[String]]

    init(fromCSVFile fileName: String) {
        guard let url = Bundle.main.url(forResource: fileName, withExtension: "csv"),
              let csvData = try? String(contentsOf: url) else {
            self.columns = []
            self.rows = []
            return
        }

        let lines = csvData.components(separatedBy: .newlines)

        if let headerLine = lines.first {
            self.columns = headerLine.components(separatedBy: "*")
        } else {
            self.columns = []
        }

        self.rows = lines.dropFirst().compactMap { line in
            let columns = line.components(separatedBy: "*")
            return columns.count > 1 && !columns.allSatisfy { $0.isEmpty } ? columns : nil
        }
    }
}
```

Above is model for splitting the csv into a data frame



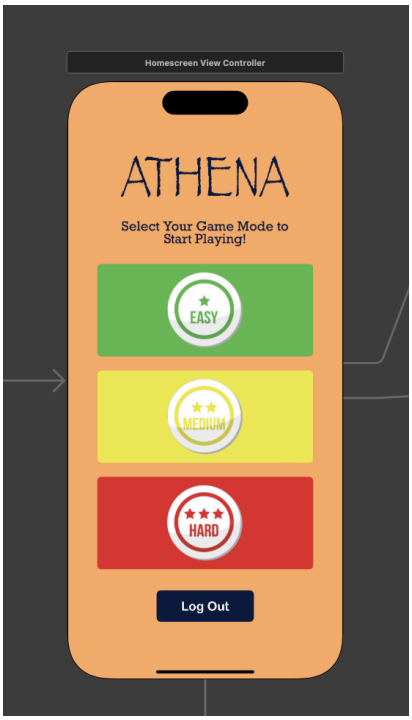
The view controller to the left is the base look of the EasyGameViewController

```swift
class EasyGameViewController: UIViewController {
    @IBOutlet weak var questionText: UILabel!
    @IBOutlet weak var categoryLabel: UILabel!
    @IBOutlet weak var titleAnswer1: UIButton!
    @IBOutlet weak var titleAnswer2: UIButton!
    @IBOutlet weak var titleAnswer3: UIButton!
    @IBOutlet weak var titleAnswer4: UIButton!

    private var dataFrame: DataFrame?
    private var correctAnswer: String?

    private var trivia: [triviaScreen] = []

    private var usedIndexes: Set<Int> {
        get {
            if let storedIndexes = UserDefaults.standard.object(forKey: "usedIndexes") as? [Int] {
                return Set(storedIndexes)
            } else {
                return []
            }
        }
        set {
            UserDefaults.standard.set(Array(newValue), forKey: "usedIndexes")
        }
    }


    override func viewDidLoad() {
        super.viewDidLoad()

        loadDataFrameFromCSV()
        displayRandomTrivia()
    }


    func loadDataFrameFromCSV() {
        dataFrame = DataFrame(fromCSVFile: "trivia_mc - Easy")
        if let dataFrame = dataFrame {
            print(dataFrame.columns)
        }
    }
func displayRandomTrivia() {
    let numberOfRows = dataFrame?.rows.count ?? 0
    var availableIndexes = Array(0..<numberOfRows)
    var answerRandomizer = [3, 4, 5, 6]

    // Remove the used indexes from the available indexes
    availableIndexes.removeAll(where: { usedIndexes.contains($0) })

    // If there are no more available indexes, reset the usedIndexes set
    if availableIndexes.isEmpty {
        usedIndexes.removeAll()
        availableIndexes = Array(0..<numberOfRows)
    }

    // Randomly select a row index
    if let i = availableIndexes.randomElement() {
        print("Randomly selected row index: \(i)")

        usedIndexes.insert(i)
        usedIndexes.removeAll() //comment out if i want used indexes to save

        let j = answerRandomizer.shuffled()
        print(j)

        let category = dataFrame!.rows[i][0]
        let question = dataFrame!.rows[i][2]
        correctAnswer = dataFrame!.rows[i][3]
        let answer1 = dataFrame!.rows[i][j[0]]
        let answer2 = dataFrame!.rows[i][j[1]]
        let answer3 = dataFrame!.rows[i][j[2]]
        let answer4 = dataFrame!.rows[i][j[3]]

        trivia = [triviaScreen(category: category, question: question, answer1: answer1, answer2: answer2, answer3: answer3, answer4: answer4)]

        categoryLabel.text = trivia.first?.category
        questionText.text = trivia.first?.question
        titleAnswer1.setTitle(trivia.first?.answer1, for: .normal)
        titleAnswer2.setTitle(trivia.first?.answer2, for: .normal)
        titleAnswer3.setTitle(trivia.first?.answer3, for: .normal)
        titleAnswer4.setTitle(trivia.first?.answer4, for: .normal)
    }
```

```swift

    @IBAction func aClicked(_ sender: Any) {
        let answer = correctAnswer
        if answer == titleAnswer1.titleLabel?.text {
            print("Correct!")
        }
        else {
            print("Incorrect!")
        }
    }
    @IBAction func bClicked(_ sender: Any) {
        let answer = correctAnswer
        if answer == titleAnswer2.titleLabel?.text {
            print("Correct!")
        }
        else {
            print("Incorrect!")
        }
    }
    @IBAction func cClicked(_ sender: Any) {
        let answer = correctAnswer
        if answer == titleAnswer3.titleLabel?.text {
            print("Correct!")
        }
        else {
            print("Incorrect!")
        }
    }
    @IBAction func dClicked(_ sender: Any) {
        let answer = correctAnswer
        if answer == titleAnswer4.titleLabel?.text {
            print("Correct!")
        }
        else {
            print("Incorrect!")
        }
    }
}
```

Above is the entire code for the EasyGameViewController

During the fifth sprint I developed the timer functionality in stages. First, I tackled getting the timer to count and the pause/play button working. After completing the stopwatch, I implemented the blur effect for the pause button press. Then, I blurred the initial screen, added a start time button, and linked it to starting the timer and displaying a trivia question. Finally, I adjusted the view hierarchy to ensure elements appeared correctly when needed. After all this was finished, I moved on to displaying the post-answer popups. Below are photos of this progress period.



Above to the left is my home screen view controller. Above to the right is my easy view controller with the two container views off to the side.

```swift
    private var correctTimes: [String] {
        get {
            if let storedTimes = UserDefaults.standard.object(forKey: "easyCorrectTimes") as? [String] {
                return storedTimes
            } else {
                return []
            }
        }
        set {
            UserDefaults.standard.set(newValue, forKey: "easyCorrectTimes")
        }
    }


    override func viewDidLoad() {
        super.viewDidLoad()

        self.blurEffect.isHidden = false
        self.startButton.isHidden = false
        self.pausedLabel.isHidden = true
        self.correctPopUp.isHidden = true
        self.incorrectPopUp.isHidden = true

        loadDataFrameFromCSV()
        displayRandomTrivia()

        correctPopUp.layer.cornerRadius = 25
        correctPopUp.clipsToBounds = true
        incorrectPopUp.layer.cornerRadius = 25
        incorrectPopUp.clipsToBounds = true
    }

```

Above is the code for saving the correct times and hiding certain objects.

```swift

    @IBAction func startClicked(_ sender: Any) {
        startCounting()
        self.startButton.isHidden = true
    }


    @IBAction func aClicked(_ sender: Any) {
        let answer = correctAnswer
        if answer == titleAnswer1.titleLabel?.text {
            correct = true
            incorrect = false
        } else {
            incorrect = true
            correct = false
        }
        stopCounting()

        if incorrect == true {
            print("Incorrect!")
            resetTimer()
            self.incorrectPopUp.isHidden = false
        } else if correct == true {
            print("Correct!")
            correctTimes.append(timeString)
            //correctTimes = [] //comment out to make correct times save
            print(correctTimes)
            UserDefaults.standard.set(true, forKey: "correctShowing")
            self.correctPopUp.isHidden = false

        }
    }
```

Above is the code for the start button one of the answer buttons

```
240     func startCounting() {
241         timerCounting = true
242         timer = Timer.scheduledTimer(timeInterval: 1, target: self, selector: #selector(timerCounter), userInfo: nil,
                repeats: true)
243         let pauseImage = UIImage(named: "pause")
244         pauseplayImage.image = pauseImage
245         self.blurEffect.isHidden = true
246     }
247
248     func stopCounting() {
249         timerCounting = false
250         timer.invalidate()
251         self.blurEffect.isHidden = false
252     }
253
254     func resetTimer() {
255         self.num = 0
256         self.timer.invalidate()
257         self.timerLabel.text = self.makeTimeString(minutes: 0, seconds: 0)
258     }
259
260     @objc func timerCounter() -> Void {
261         num = num + 1
262         let time = secondsToMinutesSeconds(seconds: num)
263         timeString = makeTimeString(minutes: time.0, seconds: time.1)
264         if timeString == "60:00" {
265             incorrect = true
266             print("Incorrect!")
267             stopCounting()
268             resetTimer()
269         } else {
270             self.timerLabel.text = timeString
271         }
272     }
```

Above is the code for the start/stop timer functions, the reset functions, and the actual counter used for the to run the stopwatch

```
273
274     @IBAction func pauseClicked(_ sender: Any) {
275         if(timerCounting) {
276             stopCounting()
277             let playImage = UIImage(named: "play")
278             pauseplayImage.image = playImage
279             self.pausedLabel.isHidden = false
280             view.bringSubviewToFront(pauseplayImage)
281             view.bringSubviewToFront(pauseButton)
282             view.bringSubviewToFront(timerLabel)
283         } else {
284             startCounting()
285             self.pausedLabel.isHidden = true
286             view.sendSubviewToBack(timerLabel)
287             view.sendSubviewToBack(pauseButton)
288             view.sendSubviewToBack(pauseplayImage)
289             view.sendSubviewToBack(darkerSpace)
290         }
291     }
292
293     func secondsToMinutesSeconds(seconds: Int) -> (Int, Int) {
294         return (((seconds % 3600) / 60), ((seconds % 3600) % 60))
295     }
296
297     func makeTimeString(minutes: Int, seconds: Int) -> String {
298         timeString = ""
299         timeString += String(format: "%02d", minutes)
300         timeString += ":"
301         timeString += String(format: "%02d", seconds)
302         return timeString
303     }
304 }
```

Above is the code for the pause/play button and the code that calculates and returns the time as minutes and seconds. It then makes that time a string and returns it back.

During my final sprint I cleaned up the project really nicely. First, I implemented a timer that updates the "correct answer" pop-up every 0.1 seconds, displaying the most recent time for each difficulty. This timer resets when the user returns to the home screen. Next, I created medium and hard difficulty levels by copying and modifying code from the easy view controller. While tedious, it wasn't overly complex. To limit users to one question per day per difficulty, I utilized a function that checks the current date against the last answered question date. If they match, the button locks, preventing further attempts that day. I also ensured the timer persists before answering a question. This prevents users from exiting and resetting their time. Two timer defaults were added: one saves the seconds upon pressing the "start clock" button, and the other prevents resetting until the question is answered or a new day begins. Finally, I implemented code to save the question index until answered or a new day starts. An if/else statement checks a "indexSaves" boolean flag. If true, it returns the saved index. Like the timer saving, this flag is only reset when the question is answered or the day changes.

```
20      override func viewDidLoad() {
21          super.viewDidLoad()
22
23          timer = Timer.scheduledTimer(timeInterval: 0.1, target: self, selector: #selector(correctDisplayCounter), userInfo: nil, repeats: true)
24      }
25
26      @objc func correctDisplayCounter() -> Void {
27          let display = UserDefaults.standard.bool(forKey: "easyCorrectShowing")
28          if display == true {
29              let correctTimes = UserDefaults.standard.object(forKey: "easyCorrectTimes") as? [String]
30
31              let index = correctTimes!.count
32
33              if index != 0 {
34                  lastInsertedTime = correctTimes![index-1]
35
36                  time = "Time - " + lastInsertedTime
37
38                  self.timeLabel.text = time
39              }
40          }
41      }
42
43      @IBAction func homeClicked(_ sender: Any) {
44          UserDefaults.standard.set(false, forKey: "easyCorrectShowing")
45          timer.invalidate()
46          goToHomescreen()
47      }
```

Above is the timer function that updates and displays the time on the easy correct view controller. The same code is in the other two difficulties as well.

```swift
class HomescreenViewController: UIViewController {

    private var currentDate = Date()
    let format = DateFormatter()
    private var easy = UserDefaults.standard.bool(forKey: "easyDone")
    private var medium = UserDefaults.standard.bool(forKey: "mediumDone")
    private var hard = UserDefaults.standard.bool(forKey: "hardDone")

    @IBOutlet weak var questionLabel: UILabel!
    @IBOutlet weak var easyLockImage: UIImageView!
    @IBOutlet weak var mediumLockImage: UIImageView!
    @IBOutlet weak var hardLockImage: UIImageView!
    @IBOutlet weak var easyBtn: UIButton!
    @IBOutlet weak var medBtn: UIButton!
    @IBOutlet weak var hardBtn: UIButton!

    override func viewDidLoad() {
        super.viewDidLoad()

        self.easyLockImage.isHidden = true
        self.mediumLockImage.isHidden = true
        self.hardLockImage.isHidden = true

        format.dateFormat = "yyyy-MM-dd"
        let date = format.string(from: currentDate)
```

Above in the home screen view controller are the date function variables and then the formatter that converts the date into a string.

```swift

        //UserDefaults.standard.set(nil, forKey: "easyLastDate") //comment out
        var easyLastDate = UserDefaults.standard.string(forKey: "easyLastDate")
        if easyLastDate == nil {
            easyLastDate = "2024-01-01"
            print(easyLastDate!)
        } else {
            print(easyLastDate!)
        }

        if date == easyLastDate {
            print(date)
            easy = UserDefaults.standard.bool(forKey: "easyDone")
        } else {
            print(date)
            UserDefaults.standard.set(false, forKey: "easyDone")
            if date != UserDefaults.standard.string(forKey: "easyLastReset") {
                UserDefaults.standard.set(false, forKey: "easyIndexSaves")
                UserDefaults.standard.set(0, forKey: "easyTimerNum")
                UserDefaults.standard.set(date, forKey: "easyLastReset")
            }
            easy = UserDefaults.standard.bool(forKey: "easyDone")
        }

        if easy == true {
            easyBtn.isEnabled = false
            self.easyLockImage.isHidden = false
        } else {
            easyBtn.isEnabled = true
            print(easy)
        }
```

Above is the code that makes it so only one easy question can be answered per day. The code is the same for the medium and hard difficulties.

```
@objc func timerCounter() -> Void {
    timeSaves = UserDefaults.standard.bool(forKey: "easyTimeSaves")

    if timeSaves == true {
        num = UserDefaults.standard.integer(forKey: "easyTimerNum")
        UserDefaults.standard.set(false, forKey: "easyTimeSaves")
        print(num)
    }

    num = num + 1
    UserDefaults.standard.set(num, forKey: "easyTimerNum")

    let time = secondsToMinutesSeconds(seconds: num)
    timeString = makeTimeString(minutes: time.0, seconds: time.1)
    if timeString == "60:00" {
        stopCounting()
        answerIncorrect()
    } else {
        self.timerLabel.text = timeString
    }
}
```

Above is the modified code for the easy timer that allows the timer to be saved. The code is the same for the medium and hard difficulties.

```
private func answerCorrect() {
    print("Correct!")
    correctTimes.append(timeString)
    correctTimes = [] //comment out to make correct times save
    print(correctTimes)
    UserDefaults.standard.set(true, forKey: "easyCorrectShowing")
    UserDefaults.standard.set(true, forKey: "easyDone")
    UserDefaults.standard.set(currentDate, forKey: "easyLastDate")
    UserDefaults.standard.set(0, forKey: "easyTimerNum")
    UserDefaults.standard.set(false, forKey: "easyTimeSaves")
    UserDefaults.standard.set(false, forKey: "easyIndexSaves")
    UserDefaults.standard.set(currentDate, forKey: "easyLastReset")
    self.correctPopUp.isHidden = false
    navigationItem.setHidesBackButton(true, animated: true)
    self.homeBtn.isHidden = true
}

private func answerIncorrect() {
    print("Incorrect!")
    resetTimer()
    UserDefaults.standard.set(true, forKey: "easyIncorrectShowing")
    UserDefaults.standard.set(true, forKey: "easyDone")
    UserDefaults.standard.set(currentDate, forKey: "easyLastDate")
    UserDefaults.standard.set(0, forKey: "easyTimerNum")
    UserDefaults.standard.set(false, forKey: "easyIndexSaves")
    UserDefaults.standard.set(currentDate, forKey: "easyLastReset")
    UserDefaults.standard.set(false, forKey: "easyTimeSaves")
    self.incorrectPopUp.isHidden = false
    navigationItem.setHidesBackButton(true, animated: true)
    self.homeBtn.isHidden = true
}
```

Above is the modified code for easy answer correct and incorrect functions. There are some added UserDefaults. The code is the same for the medium and hard difficulties.

The scope changed drastically from the start of the project. Originally I intended to have my databases all set up with questions and User data, but instead they are both half finished and were pushed to the side for more important aspects. Also intended to have a stats screen which was altered into a reach goal and never even started. If I had another month the first step would be saving any user defaults to the Firestore database instead of locally. Then I would work on hooking up MySQL for the questions.

III. Demo Video

Linked to Google Classroom and stored in the GitHub repository.

IV. Reflection and Closing Thoughts

Taking on swift coding was a challenge that I was not entirely prepared for. Learning a new language while trying to incorporate so many different aspects proved time consuming. I spent many weekends on Stack Overflow trying to sort through errors that ended up being caused by one constraint or word. If I could go back to January, I probably would have coded a web app instead since I could have incorporated more aspects of my project that I wanted to complete. The main project idea would not have changed though, and I think I would have still reverted to trivia instead of brain teasers.

Swift is a challenging coding language not just because it was new but because it is updated so frequently. This means that much of the information you find online is deprecated, so you have to sift through a ton of code before you find the right bits and pieces. There are also

two types of ways to code in Swift so there is all the junk from SwiftUI coding that does not work for my project either.

Now for something that went well. I had much success with completing a fully working app. I learned in pieces, but eventually those pieces came together into something concrete. Although not everything went according to plan, I learned a great deal and was able to make a working product that resembles my original idea. It's hard to pick and choose specific things that went well, but I believe the project as a whole to be a success.

I plan to make some more progress on my app before I quit. There are still a few things I absolutely want to get done, so I plan to work at least into the Summer. As far as next year goes, the only tweak I would make to the course is to make sure kids working on the same thing are in the same block. Not just kids on the same project, but kids working with the same code so that information can be passed and things can be learned quicker. I know this is not entirely possible, but it would be nice to try and work out.