Rafic Galli
rafic.galli@eoc.ch

Il nostro percorso

01 **L’inizio di tutto**
Qual è stata l’idea dietro questa presentazione?

02 **Spaghetti Architecture**
Prima la scrittura del codice e poi la definizione del design

03 **Onion Architecture**
Prima la definizione del design e poi la scrittura del codice

04 **La Demo**
Fammi vedere il gioco!

05 **Conclusione**
Meglio scrivere prima il codice o definire il design?

Il nostro percorso

01 L’inizio di tutto
Qual è stata l’idea dietro questa presentazione?

02 Spaghetti Architecture
Prima la scrittura del codice e poi la definizione del design

03 Onion Architecture
Prima la definizione del design e poi la scrittura del codice

04 La Demo
Fammi vedere il gioco!

05 Conclusione
Meglio scrivere prima il codice o definire il design?

L’inizio di tutto

L’inizio di tutto

party fever

L’inizio di tutto

Swift 3.0

Swift 5

L’inizio di tutto

Unsupported Swift Version

The target “PartyFever!” contains source code developed with Swift 3.x. This version of Xcode does not support building or migrating Swift 3.x targets.

Use Xcode 10.1 to migrate the code to Swift 4.

OK

In order to use “Xcode”, you need to update to the latest version.

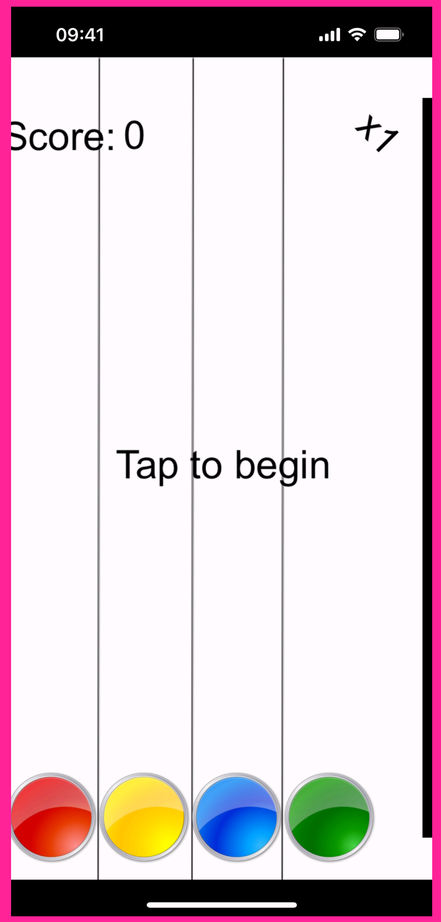
The version of Xcode installed on this Mac is not compatible with macOS Sonoma. Download the latest version for free from the App Store.

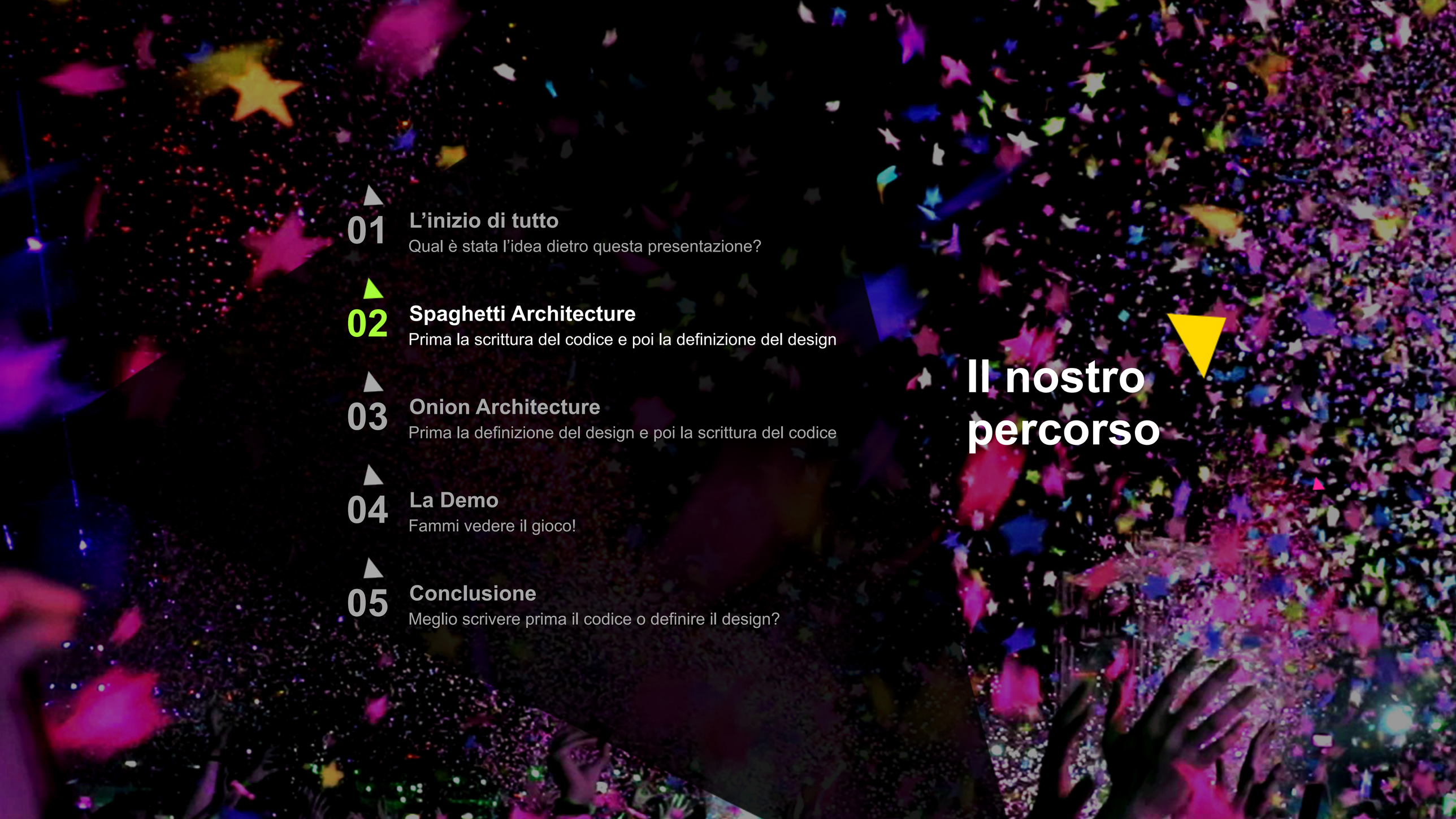
Search App Store

Cancel

L’inizio di tutto

L’inizio di tutto





Il nostro percorso

01 **L’inizio di tutto**
Qual è stata l’idea dietro questa presentazione?

02 **Spaghetti Architecture**
Prima la scrittura del codice e poi la definizione del design

03 **Onion Architecture**
Prima la definizione del design e poi la scrittura del codice

04 **La Demo**
Fammi vedere il gioco!

05 **Conclusione**
Meglio scrivere prima il codice o definire il design?

Target interfaces

Delivery Infrastructure

Application Services

Domain Entities

Game View

Game ViewController

GameEngine

ReportView

Report ViewController

Highscore View

Highscore ViewController

Playlist ViewController

Playlist View

Score

TapNote
PressNote
HoldNote
SwipeNote
ShiftNote

Song

Note

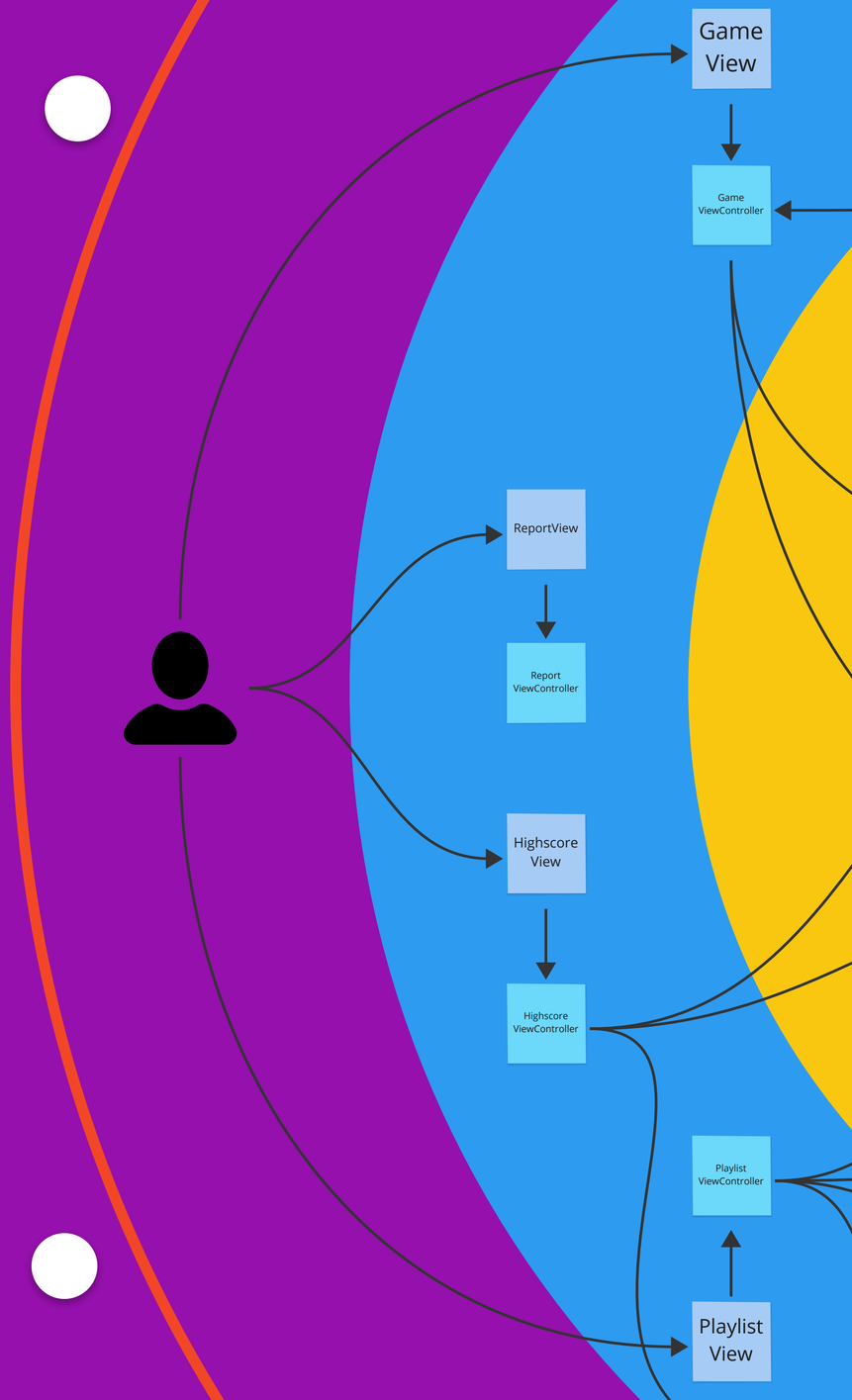
Song Factory

Gestione della partita musicale

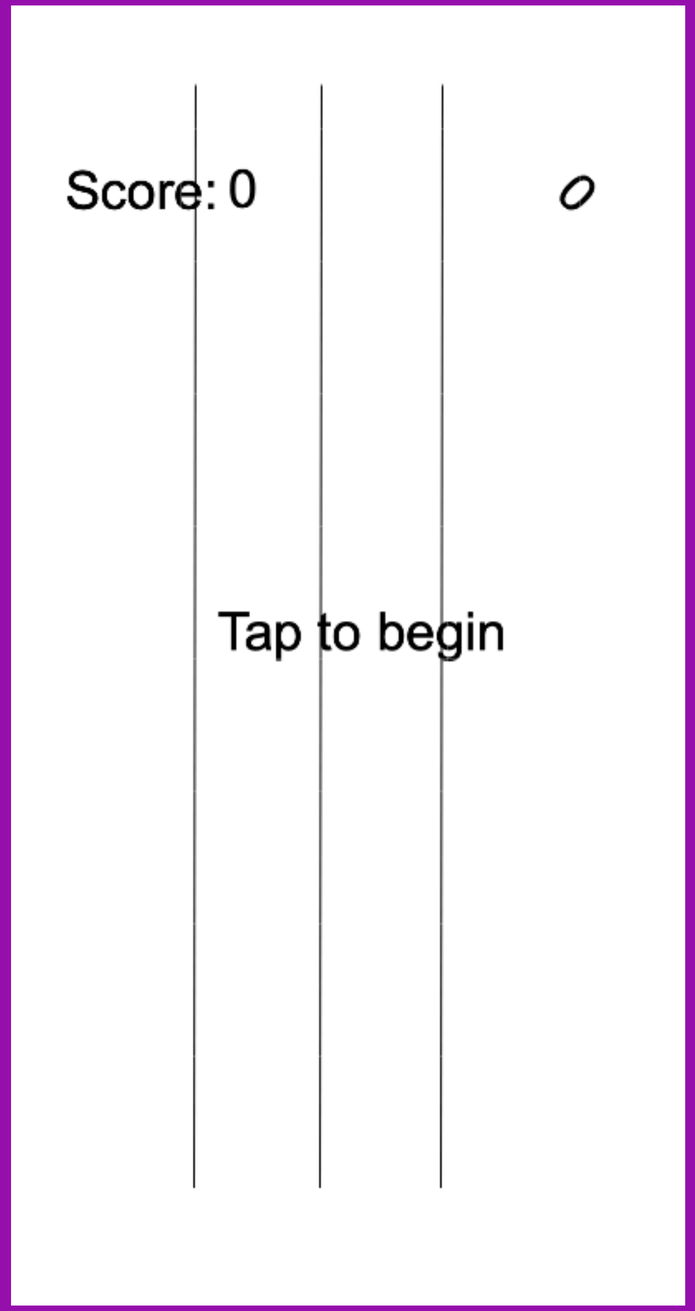
Valutazione della partita giocata

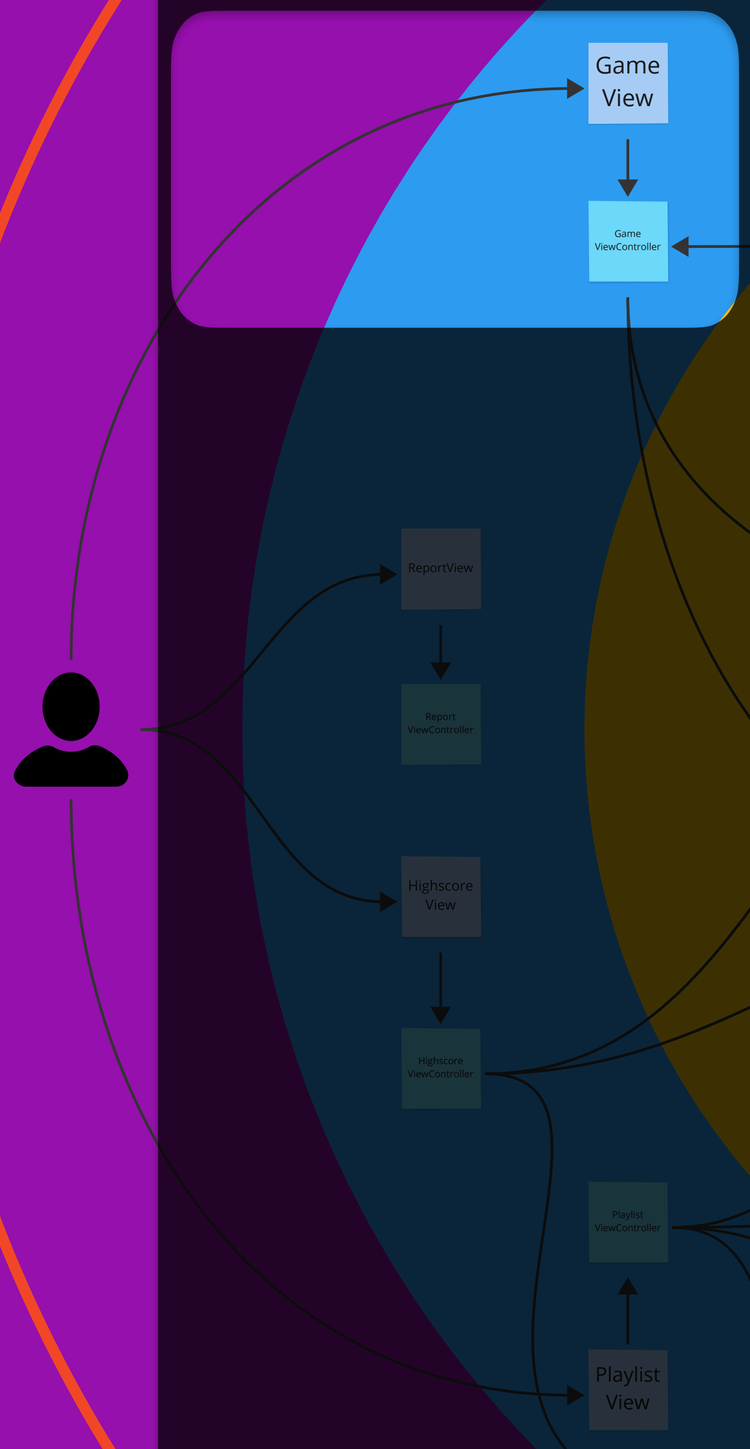
Lista dei punteggi dei giocatori

Selezione della musica da giocare

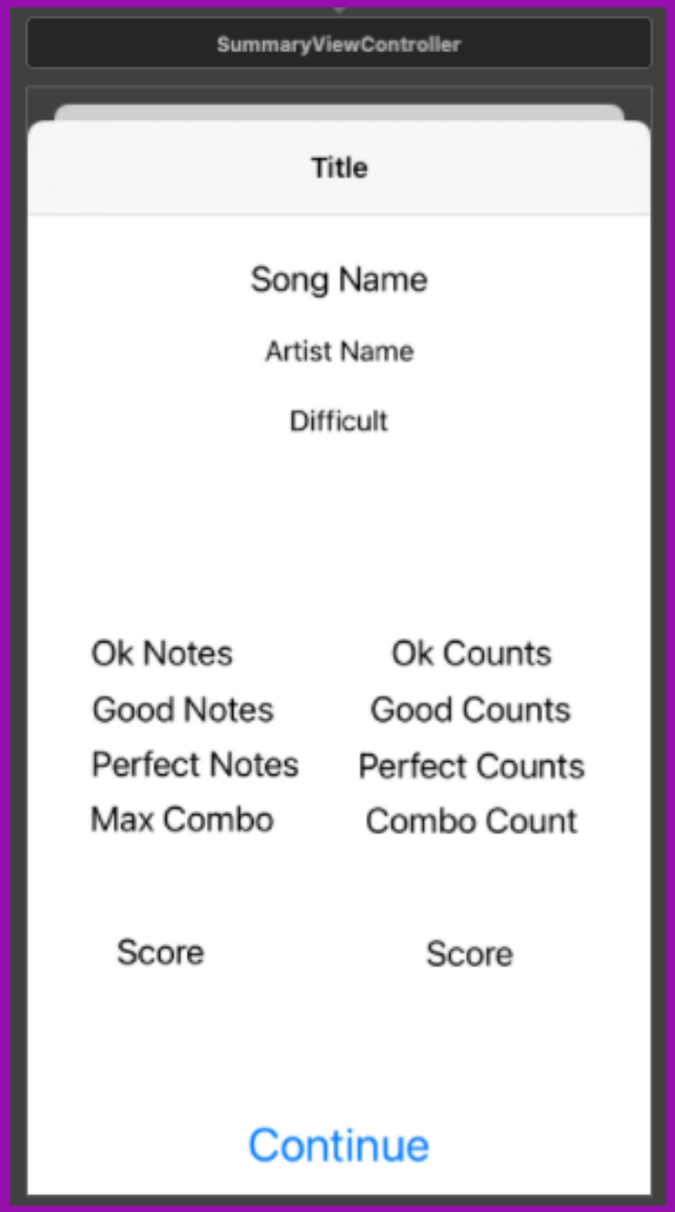


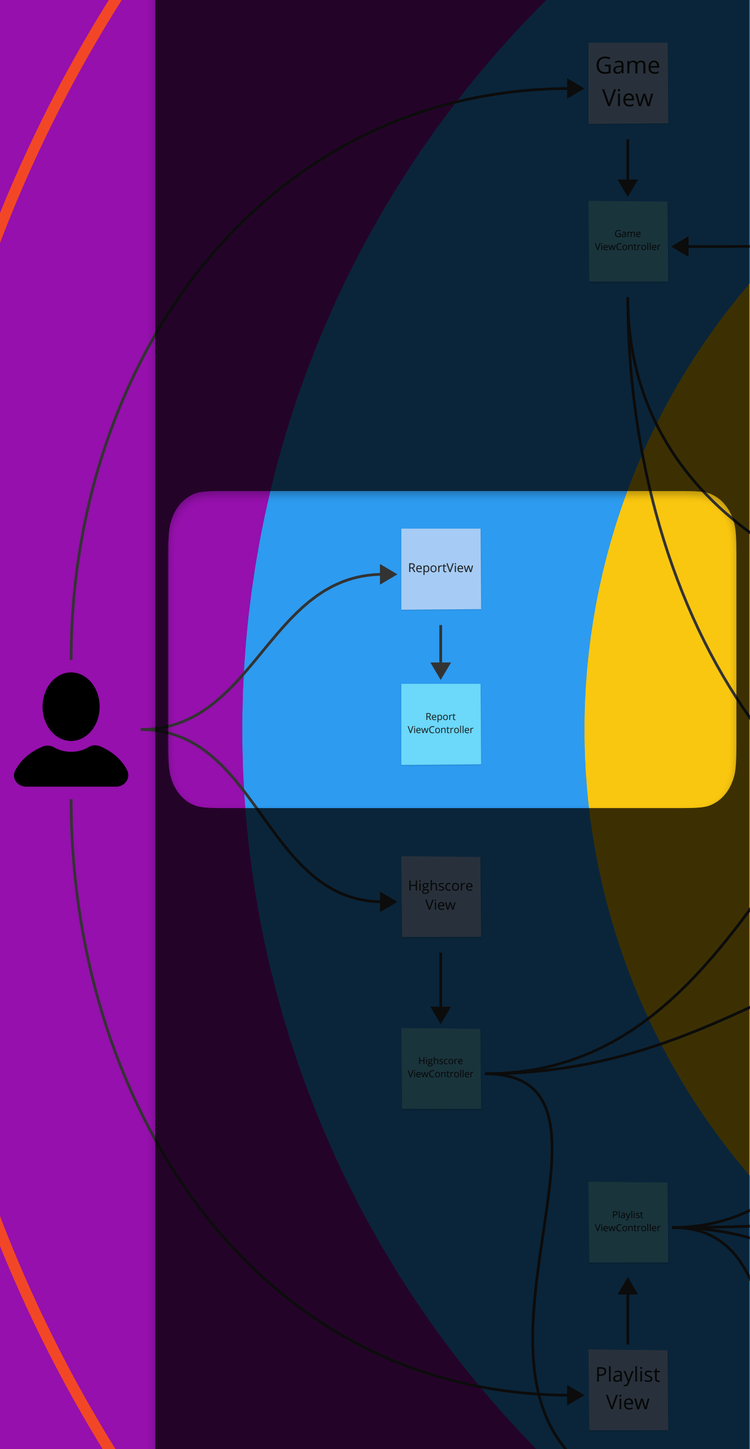
```swift
class GameViewController: UIViewController {

    var song: Song?                                 // The selected song from the user
    var difficult: String?                          // The selected difficult from the user
    var score: Score?                               // The final score made by the user
    var gameScene: GameScene?                       // The class charged to manage the game
    var username: String?                           // The username saved with the score
    var gameView: SKView?                           // The view displayed to the user
    var highscoreController: HighscoreViewController?   // The reference of the previous controller
    var okCounts: Int!                              // Number of OK quality
    var goodCounts: Int!                            // Number of Good quality
    var perfectCounts: Int!                         // Number of Perfect quality
    var maxCombo: Int!                              // Max sequence of correct actions

    @IBOutlet weak var navigationBar: UINavigationBar!  // Reference to the navigation bar

    // Method called when the view is loaded
    override func viewDidLoad() {
        super.viewDidLoad()

        // Get the scene from the project directory
        if let scene = GameScene(fileNamed:"GameScene") {
            let skView = self.view as! SKView       // Getting the view
            skView.showsFPS = false                 // Hiding the FPS
            skView.showsNodeCount = false           // Hiding the number of nodes in the view
            skView.showsPhysics = false             // Don't manage physics in this view

            scene.song = song
            scene.difficult = difficult
            scene.controller = self
            gameScene = scene
            gameView = skView

            // Sprite Kit applies additional optimizations to improve rendering performance
            skView.ignoresSiblingOrder = false

            // Set the scale mode to scale to fit the window
            scene.scaleMode = .aspectFill

            skView.presentScene(gameScene)          // Showing the view
        }
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }

    // Method that end the current game
    func endGame () {
        highscoreController?.saveScore(newScore: score!)        // Saving the current score
        performSegue(withIdentifier: "endGame", sender: self)   // Showing the summary view
    }

    // Method that initialize the summary controller with informations obtained in the game part
    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        gameScene?.stopMusic()                                  // Stopping the current song

        if let summaryController = segue.destination as? SummaryViewController {

            summaryController.songName = song?.name
            summaryController.artistName = song?.artist
            summaryController.difficult = difficult
            summaryController.score = score?.score
            summaryController.okCounts = okCounts
            summaryController.goodCounts = goodCounts
            summaryController.perfectCounts = perfectCounts
            summaryController.maxCombo = maxCombo
        }
    }

}
```





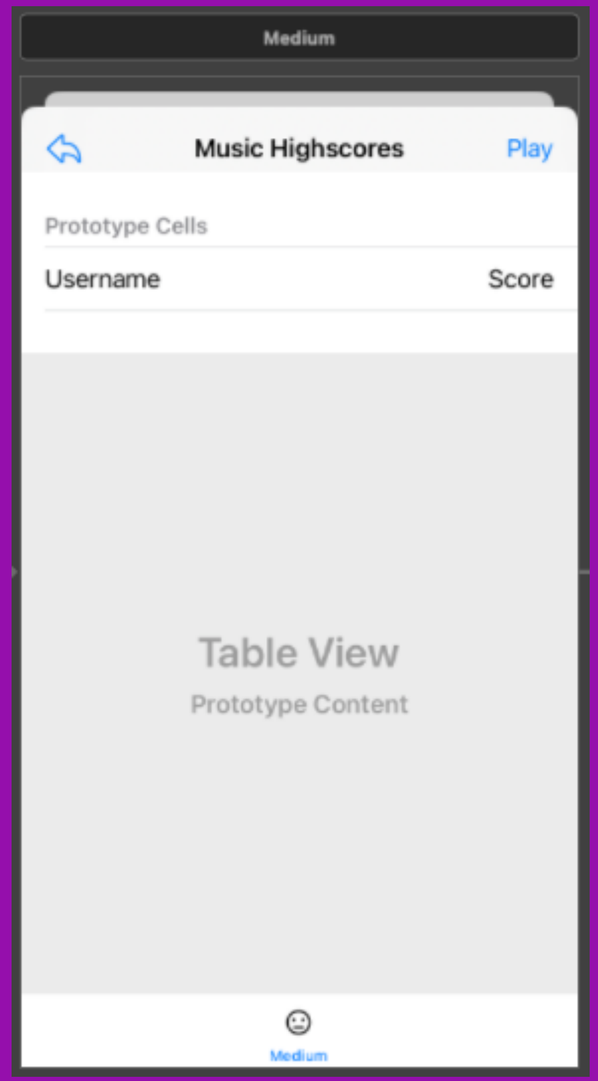
```swift
class SummaryViewController: UIViewController {

    var songName: String!
    var artistName: String!
    var difficult: String!
    var okCounts: Int!
    var goodCounts: Int!
    var perfectCounts: Int!
    var maxCombo: Int!
    var score: Int!

    @IBOutlet weak var songNameLabel: UILabel!
    @IBOutlet weak var artistNameLabel: UILabel!
    @IBOutlet weak var difficultLabel: UILabel!
    @IBOutlet weak var okCountsLabel: UILabel!
    @IBOutlet weak var goodCountsLabel: UILabel!
    @IBOutlet weak var perfectCountsLabel: UILabel!
    @IBOutlet weak var maxComboLabel: UILabel!
    @IBOutlet weak var scoreLabel: UILabel!

    // Setting informations when the view is loaded
    override func viewDidLoad() {
        super.viewDidLoad()

        songNameLabel.text = songName
        artistNameLabel.text = artistName
        difficultLabel.text = difficult
        okCountsLabel.text = String(okCounts)
        goodCountsLabel.text = String(goodCounts)
        perfectCountsLabel.text = String(perfectCounts)
        maxComboLabel.text = String(maxCombo)
        scoreLabel.text = String(score)
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }

}
```

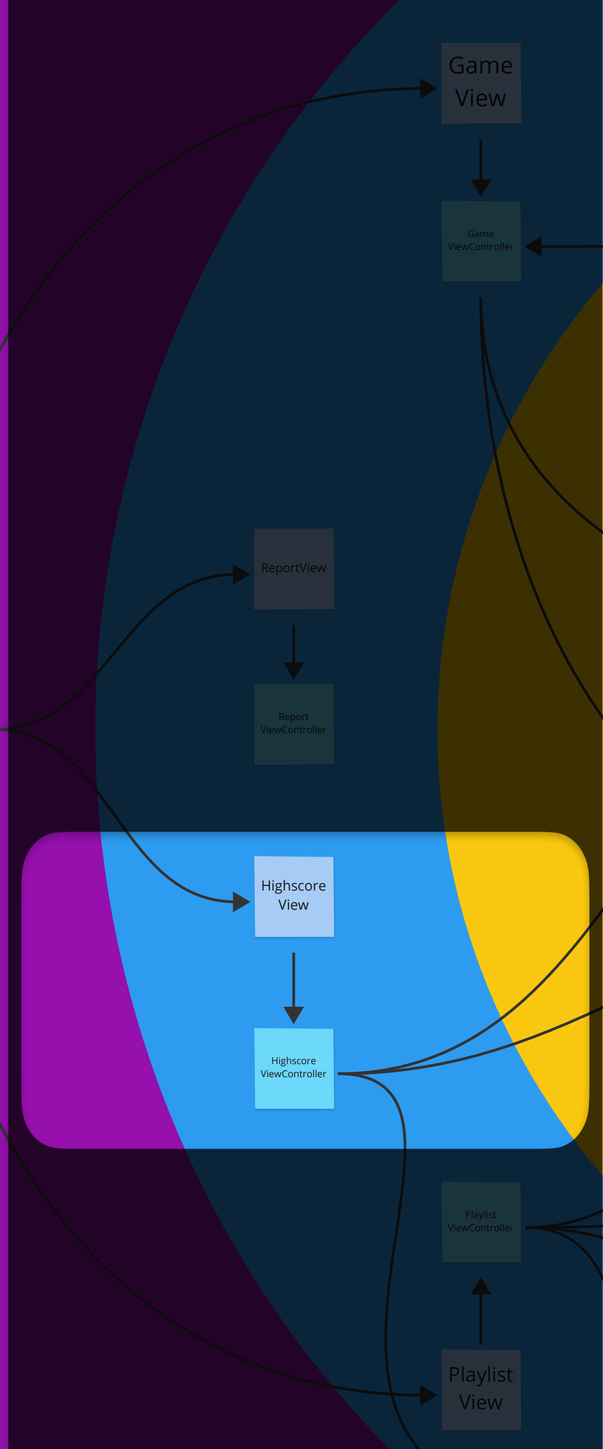




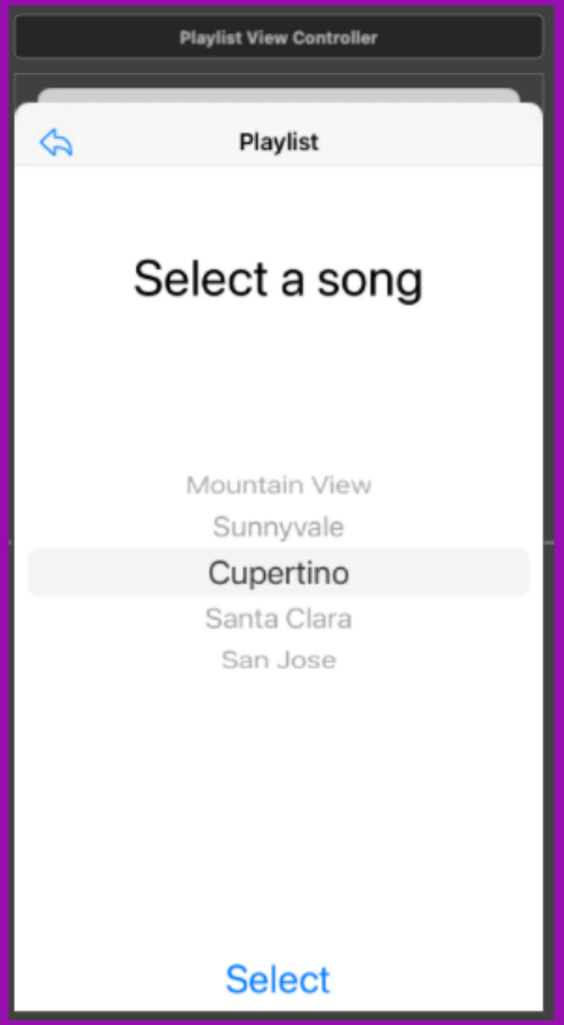
```swift
class HighscoreViewController: UIViewController, UITableViewDelegate, UITableViewDataSource {

    let highscoreTableName = "Highscore"
    let idScoreKey = "idScore"
    let idSongKey = "idSong"
    let usernameKey = "username"
    let difficultKey = "difficult"
    let scoreKey = "score"

    var song: Song?
    var scores = [Score]()
    var difficult: String?
    var audioPlayer: AVPlayer?

    @IBOutlet weak var scoreTable: UITableView!

    // Initializing the table with score data
    override func viewDidLoad() {
        super.viewDidLoad()

        // Retrieving informations by requesting the database
        loadScores(theSongId: (song?.id)!, theDifficult: difficult!)
        scoreTable.delegate = self
        scoreTable.dataSource = self
        scoreTable.reloadData()
        scoreTable.allowsSelection = false
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }

    @IBAction func normalBackward(_ sender: Any) {
        dismiss(animated: true, completion: nil)
    }

    @IBAction func hardBackward(_ sender: Any) {
        dismiss(animated: true, completion: nil)
    }

    // Saving the score when returning in this view
    @IBAction func unwindToHighscoreController(_ sender: UIStoryboardSegue) {
        if let sender = sender.source as? GameViewController {
            if sender.score != nil {
                saveScore(newScore: sender.score!)
            }
        }
    }

    // Initializing the game controller when leaving this view
    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        stopMusic()

        let dest = segue.destination as? GameViewController
        dest?.song = song
        dest?.difficult = difficult
        dest?.highscoreController = self
    }

    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return scores.count
    }

    func numberOfSections(in tableView: UITableView) -> Int {
        return 1
    }

    // Setting the cell of the score table
    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier: "scoreCell", for: indexPath) as! ScoreCell
        cell.username.text = scores[indexPath.row].username
        cell.score.text = String(describing: scores[indexPath.row].score)

        return cell
    }
```

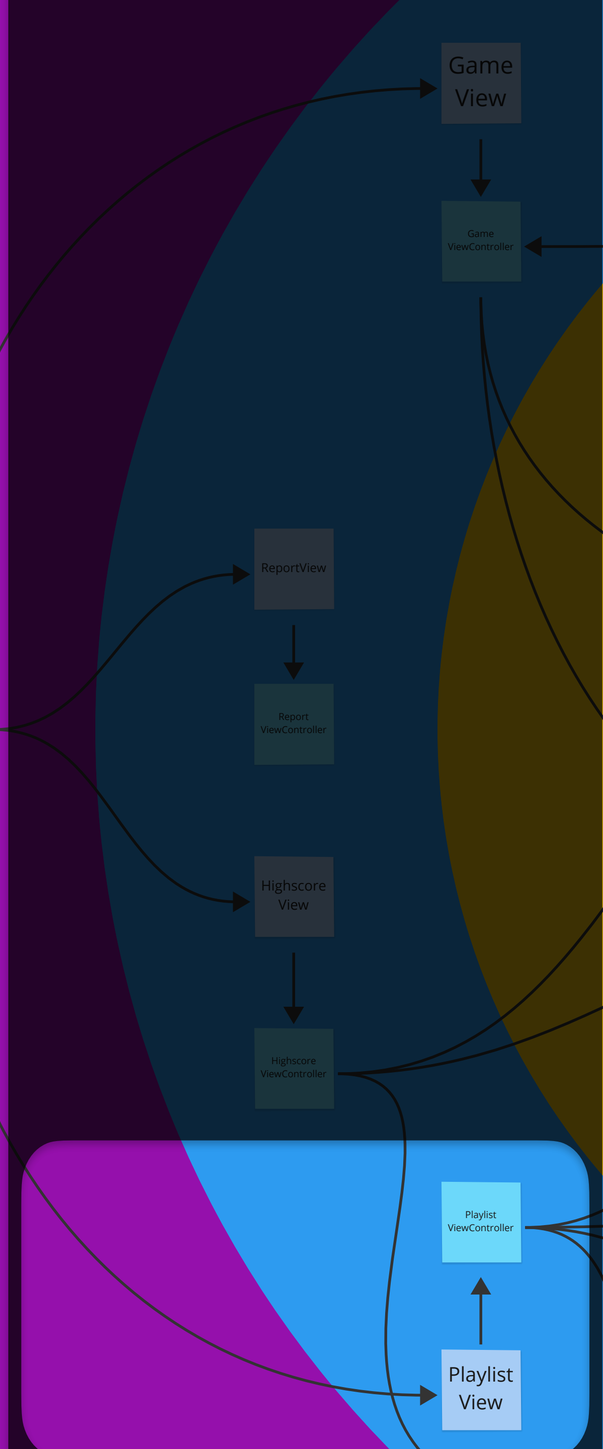
```swift
    // Method that saves a given score in the database
    func saveScore (newScore: Score) {
        let appDelegate = UIApplication.shared.delegate as! AppDelegate
        let managedContext = appDelegate.persistentContainer.viewContext
        let entity = NSEntityDescription.entity(forEntityName: highscoreTableName, in: managedContext)
        let score = NSManagedObject(entity: entity!, insertInto: managedContext)

        score.setValue(newScore.idScore, forKey: idScoreKey)
        score.setValue(newScore.idSong, forKey: idSongKey)
        score.setValue(newScore.username, forKey: usernameKey)
        score.setValue(newScore.difficult, forKey: difficultKey)
        score.setValue(newScore.score, forKey: scoreKey)

        do {
            try managedContext.save()
            let newIndex = IndexPath(row: scores.count, section: 0)
            scores.append(newScore)
            scoreTable.insertRows(at: [newIndex], with: .bottom)
        } catch let error as NSError {
            print("Could not save \(error), \(error.userInfo)")
        }
    }

    // Method that load all scores from the database
    private func loadScores (theSongId: Int, theDifficult: String) {
        let appDelegate = UIApplication.shared.delegate as! AppDelegate
        let managedContext = appDelegate.persistentContainer.viewContext
        let fetchRequest = NSFetchRequest<NSFetchRequestResult>(entityName: highscoreTableName)

        do {
            let results = try managedContext.fetch(fetchRequest) as! [NSManagedObject]

            for result in results {
                let songId = result.value(forKey: idSongKey) as! Int
                let difficult = result.value(forKey: difficultKey) as! String

                // Filtering results by keeping only the desired scores
                if songId == theSongId && difficult == theDifficult {
                    let id = result.value(forKey: idScoreKey) as! Int
                    let username = result.value(forKey: usernameKey) as! String
                    let score = result.value(forKey: scoreKey) as! Int
                    scores.append(Score(
                        theIdScore: id,
                        theIdSong: songId,
                        theUsername: username,
                        theDifficult: difficult,
                        theScore: score))
                }
            }
            // Sorting the list from the better to the worse score
            scores.sort(by: {$0.score > $1.score})

        } catch let error as NSError {
            print("Could not fetch \(error), \(error.userInfo)")
        }
    }

    // Stopping the current song
    private func stopMusic () {
        audioPlayer?.pause()
        audioPlayer = nil
    }

}
```



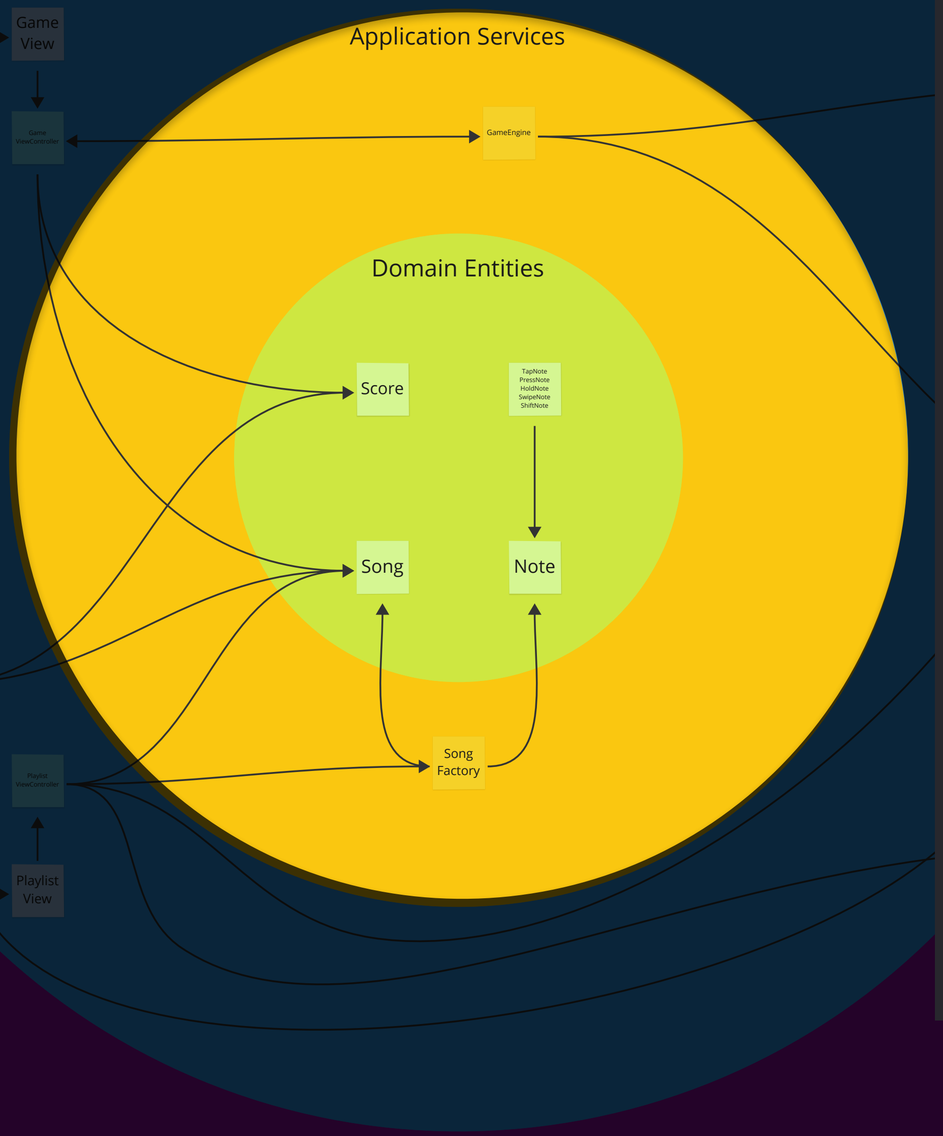


```swift
class PlaylistViewController: UIViewController, UIPickerViewDelegate, UIPickerViewDataSource {

    var songList = [Song]()                     // The list of available songs
    var audioPlayer: AVPlayer?                  // A audio player for the previews

    let factory = SongFactory()                 // The factory that creates the songs

    @IBOutlet weak var playlist: UIPickerView!

    // Method that is called when the view is created. Filling the list of the picker view
    override func viewDidLoad() {
        super.viewDidLoad()

        initSongList()                  // Creating the songs for this application

        playlist.delegate = self
        playlist.dataSource = self
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }

    // Method called when leaving this view and initializing the view for the scores
    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        if let dest = segue.destination as? DifficultTabController {
            dest.song = songList[playlist.selectedRow(inComponent: 0)]
            dest.audioPlayer = audioPlayer
        }
    }

    @IBAction func unwindToPlaylistController(_ sender: UIStoryboardSegue) {
    }

    func numberOfComponents(in pickerView: UIPickerView) -> Int {
        return 1
    }

    func pickerView(_ pickerView: UIPickerView, numberOfRowsInComponent component: Int) -> Int {
        return songList.count
    }

    func pickerView(_ pickerView: UIPickerView, rowHeightForComponent component: Int) -> CGFloat {
        return 70
    }

    // Method used to custom a single picker view cell
    func pickerView(_ pickerView: UIPickerView, viewForRow row: Int, forComponent component: Int, reusing view: UIView?) -> UIView {
        stopMusic()

        // Getting the size of a single cell in the UIPickerView
        let cellSize = pickerView.rowSize(forComponent: component)

        var newView = view
        var pickerLabel: UILabel?
        var pickerImage: UIImageView?

        pickerLabel = UILabel()
        pickerImage = UIImageView()

        if newView == nil {
            newView = UIView()
        }

        // Getting name of the song anf artwork
        pickerLabel?.text = songList[row].name
        pickerImage = UIImageView(image: songList[row].artwork)

        // Colorating each cell with a different color (rainbow effect)
        let hue = CGFloat(row)/CGFloat(songList.count)
        pickerImage?.backgroundColor = UIColor(hue: hue, saturation: 1.0, brightness: 1.0, alpha: 1.0)
        pickerLabel?.backgroundColor = UIColor(hue: hue, saturation: 1.0, brightness: 1.0, alpha: 1.0)

        // Placing them in a specific position: first the artwork and then the name of the song
        pickerImage?.frame.size = CGSize(width: cellSize.height, height: cellSize.height)
        let labelXPosition = (pickerImage?.frame.origin.x)! + (pickerImage?.frame.width)!
        let labelYPosition = pickerImage?.frame.origin.y
        pickerLabel?.frame = CGRect(origin: CGPoint(x: labelXPosition, y: labelYPosition!), size: CGSize(width: cellSize.width - (pickerImage?.frame.size.width)!, height: cellSize.height))

        pickerLabel?.textAlignment = .center

        // Adding the artwork and the name of the song in the cell view
        newView?.addSubview(pickerLabel!)
        newView?.addSubview(pickerImage!)

        startMusic(index: row)                  // Starting the preview song

        return newView!
    }

    // Function that initialize the song list by creating at this moment the note list
    private func initSongList () {
        songList.append(factory.createOdeToJoySong())
        songList.append(factory.createAmericanIdiotSong())
        songList.append(factory.createRealDoneSong())
        songList.append(factory.createFreeBirds())
        songList.append(factory.createCrossingField())
        songList.append(factory.createHello())
        songList.append(factory.createSpazzmaticaPolka())
    }

    // Function that starts a given music in the PickerView
    func startMusic (index: Int) {
        if audioPlayer == nil {
            let url = Bundle.main.url(forResource: songList[index].fileName, withExtension: "mp3")
            audioPlayer = AVPlayer(url: url!)
            audioPlayer?.play()
        }
    }

    // Method that stops the current song
    func stopMusic () {
        audioPlayer?.pause()
        audioPlayer = nil
    }
}
```



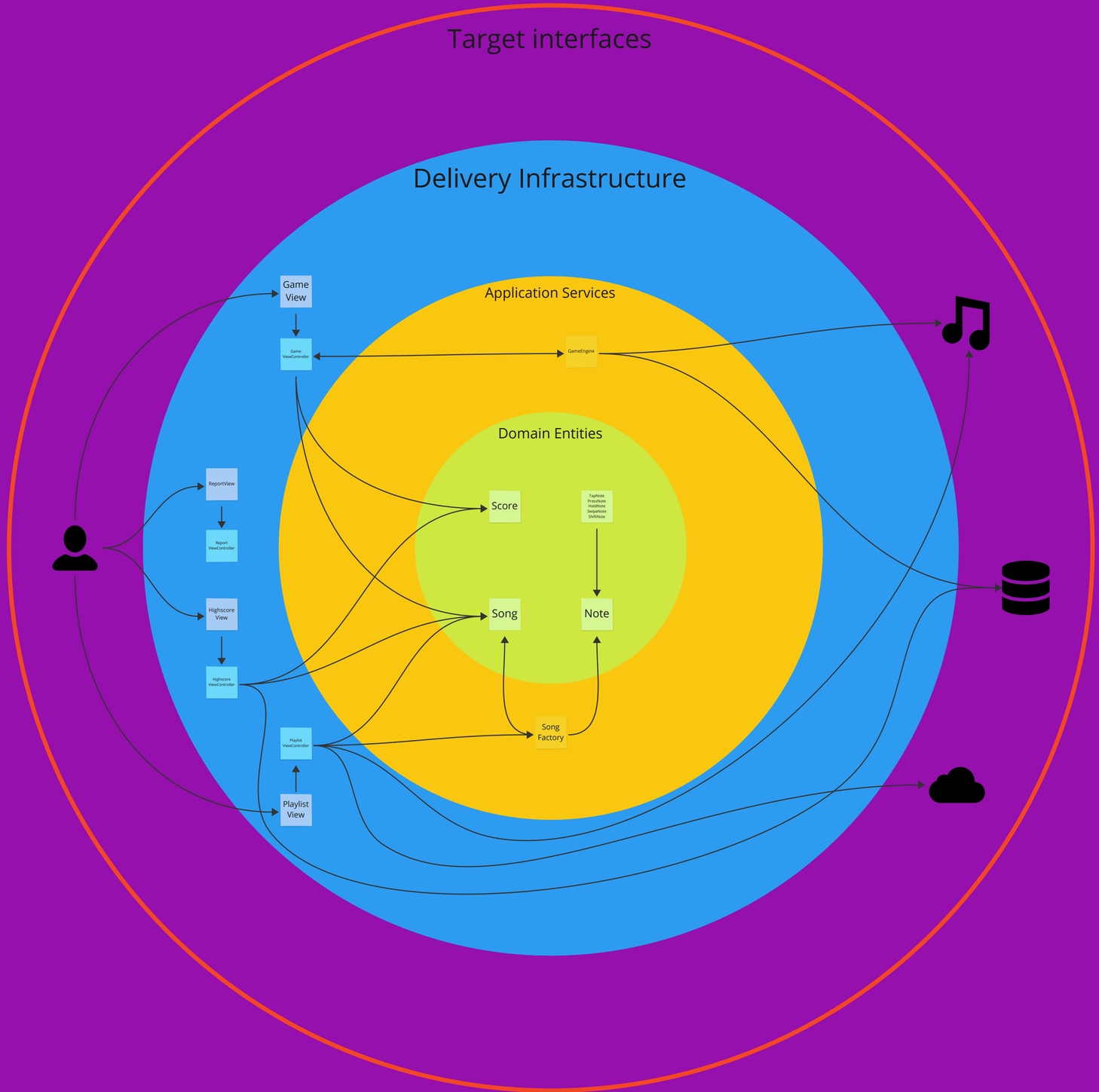


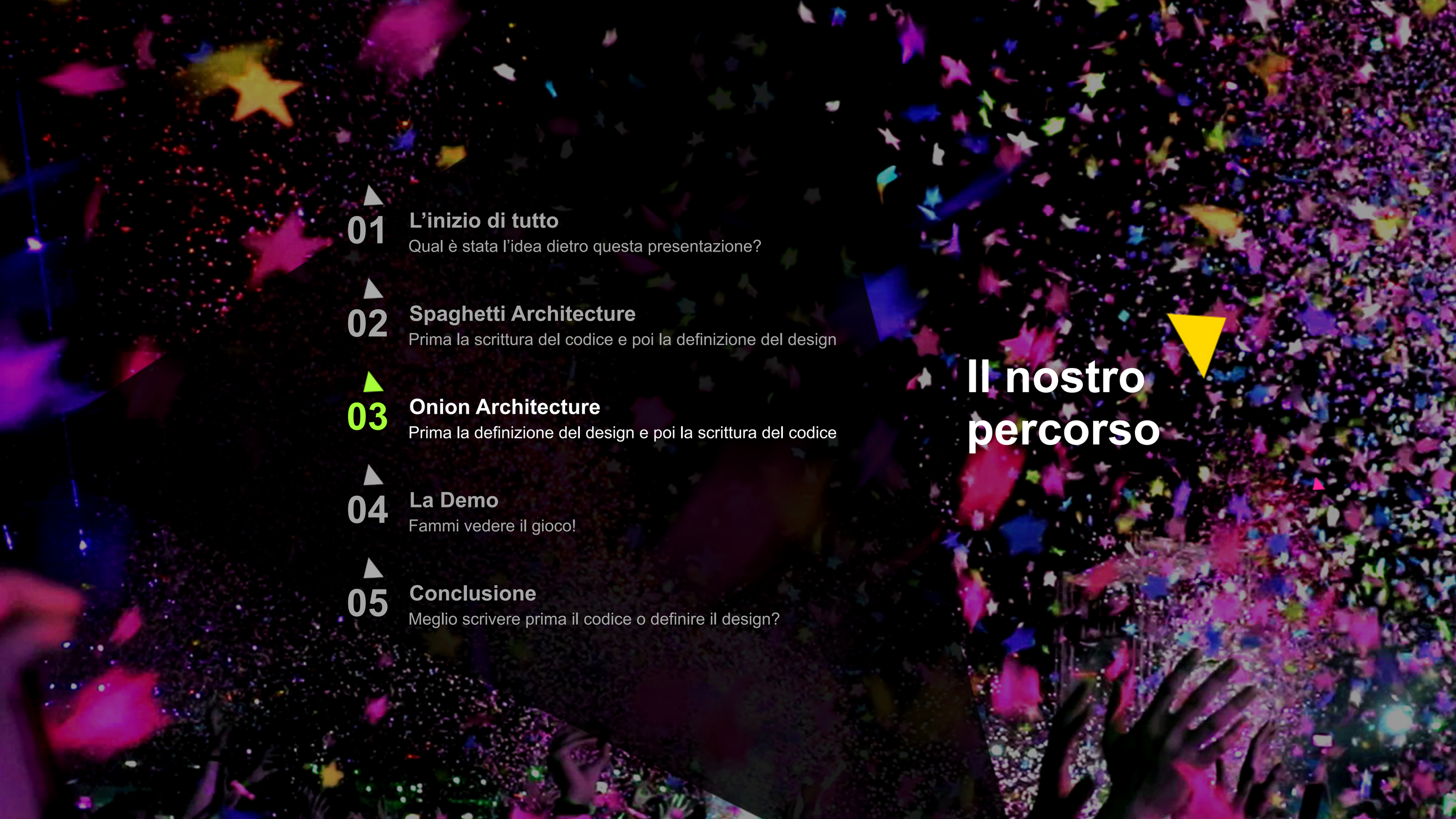
Delivery Infrastructure




```swift
class GameScene: SKScene, UITextFieldDelegate, OnNoteMissed {

    let FEVER_KEY = "fever"                 // Identifier for the fever mode

    let TOUCH = 0                           // Constant defining a touch from the user
    let SWIPE = 1                           // Constant defining a swipe from the user

    let LEFT_TO_RIGHT = 0                   // Constant indicating a swipe from left to right
    let RIGHT_TO_LEFT = 1                   // Constant indicating a swipe from right to left

    let BUTTON1 = CGPoint(x: 50, y: 50)     // Position of the first button (red)
    let BUTTON2 = CGPoint(x: 125, y: 50)    // Position of the second button (yellow)
    let BUTTON3 = CGPoint(x: 200, y: 50)    // Position of the third button (blue)
    let BUTTON4 = CGPoint(x: 275, y: 50)    // Position of the fourth button (green)

    var controller: GameViewController?     // The parent controller
    var activePressNote: PressNote?         // Viariable containing the reference of the current PressNote
    var song: Song?                         // The song that will be played
    var noteList = [Note]()                 // The list of notes that will be inserted in the view
    var audioPlayer: AVPlayer?              // The player charged to play the song
    var score: Score?                       // The final score of the player
    var difficult: String?                  // The previous selected difficult
    var username: String?                   // The username inserted at the end of the game
    var points = 0                          // The current score
    var scoreFactor = 1                     // Current factor (sequence of correct notes)
    var okCounts: Int?                      // Number of "Ok" actions performed
    var goodCounts: Int?                    // Number of "Good" actions performed
    var perfectCounts: Int?                 // Number of "Perfect" actions performed
    var maxCombo: Int?                      // The max sequence of correct actions
    var comboCount: Int?                    // The current sequence of correct actions
    var isStarted = false                   // Variable indicating if the game is started or not
    var isPressed: Bool?                    // Variable indicating if the user is pressing a PressNote
    var endGame = false                     // Variable indicating if the game is terminated or not
    var textLabel: SKLabelNode?             // Label containing the text for the score
    var scoreLabel: SKLabelNode?            // Label containing the current score
    var factorLabel: SKLabelNode?           // Label containing the current factor
    var progressBar: SKSpriteNode?          // Progress bar indicating how much time is left
    var startLabel: SKLabelNode?            // Label used to indicate the user to tap to begin the game
    var onFireLabel: SKLabelNode?           // Label containing the Fever mode description
    var fireList = [SKSpriteNode]()         // Variable containing fires for the Fever mode
    var fontSize: CGFloat?                  // Font size for each label in the view
    var swipeDirection: Int?                // Variable indicating the direction of a swipe

    // Function equivalent to viewDidLoad() of a normal controller
    override func didMove(to view: SKView) {
        super.didMove(to: view)

        // Charging the list of notes base on the selected difficult
        if difficult == "Medium" {
            noteList = (song?.normalNoteList)!
        }
        else if difficult == "Hard" {
            noteList = (song?.hardNoteList)!
        }

        // Charging the player with the selected song
        let odeToJoyUrl = Bundle.main.url(forResource: song?.fileName, withExtension: "mp3")
        audioPlayer = AVPlayer(url: odeToJoyUrl!)

        // Adding the swipe gesture for both left and right directions
        let swipeRight:UISwipeGestureRecognizer = UISwipeGestureRecognizer(target: self, action: #selector(self.parseNoteSwiped))
        swipeRight.direction = .right
        view.addGestureRecognizer(swipeRight)

        let swipeLeft:UISwipeGestureRecognizer = UISwipeGestureRecognizer(target: self, action: #selector(self.parseNoteSwiped))
        swipeLeft.direction = .left
        view.addGestureRecognizer(swipeLeft)

        // Getting labels from the game view
        textLabel = self.childNode(withName: "text") as! SKLabelNode?
        scoreLabel = self.childNode(withName: "score") as! SKLabelNode?
        factorLabel = self.childNode(withName: "factor") as! SKLabelNode?
        startLabel = self.childNode(withName: "startLabel") as! SKLabelNode?
```

```swift
        //Configuration of the labels
        scoreLabel?.horizontalAlignmentMode = .left
        factorLabel?.horizontalAlignmentMode = .center
        scoreLabel?.color = UIColor.cyan
        factorLabel?.text = "x1"

        fontSize = textLabel?.fontSize

        // Initializing values for the summary view
        okCounts = 0
        goodCounts = 0
        perfectCounts = 0
        maxCombo = 0
        comboCount = 0

        // Setting the begin of the game
        isPressed = false
        endGame = false

        // Inserting the progress bar on the right and the four color buttons
        initProgressBar()
        initButtons()
    }

    // Implementing the protocol to know when the user misses a musical note
    func onNoteMissed(finalPosition: CGPoint, note: Note) {
        touchFailed(btnPosition: finalPosition)     // the note in this x position is a missed action

        // Deleting all components from the PressNote by searching for the relative HoldNote
        if let pressNote = note as? HoldNote {
            pressNote.parentNote?.endNote?.removeFromParent()
            pressNote.parentNote?.rectangle?.removeFromParent()
        }
    }

    // Function that is called when the user tap on the screen
    override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
        let touch = touches.first                           // Getting only the first touch
        let positionTouch = touch?.location(in: self)       // Getting the position of the touch
        let touchedNodes = self.nodes(at: positionTouch!)   // Getting the nodes touched by the user

        // Iteration used to know if the user begin a PressNote by pressing on a HoldNote
        for node in touchedNodes {
            if node is HoldNote {
                let holdNote = node as! HoldNote
                if (holdNote.parentNote?.isBeginNote(note: holdNote))! {
                    isPressed = true
                    activePressNote = holdNote.parentNote
                    activePressNote?.rectangle?.color = UIColor.yellow
                    holdNote.removeFromParent()
                }
            }
        }
    }
```


Target interfaces
Delivery Infrastructure
Application Services
Domain Entities
Game View
Game ViewController
GameEngine
ReportView
Report ViewController
Highscore View
Highscore ViewController
Playlist ViewController
Playlist View
Score
TapNote
PressNote
HoldNote
SwipeNote
ShiftNote
Song
Note
Song Factory



Il nostro percorso

01 **L’inizio di tutto**
Qual è stata l’idea dietro questa presentazione?

02 **Spaghetti Architecture**
Prima la scrittura del codice e poi la definizione del design

03 **Onion Architecture**
Prima la definizione del design e poi la scrittura del codice

04 **La Demo**
Fammi vedere il gioco!

05 **Conclusione**
Meglio scrivere prima il codice o definire il design?

Target interfaces

Delivery Infrastructure

Application Services

Domain Entities

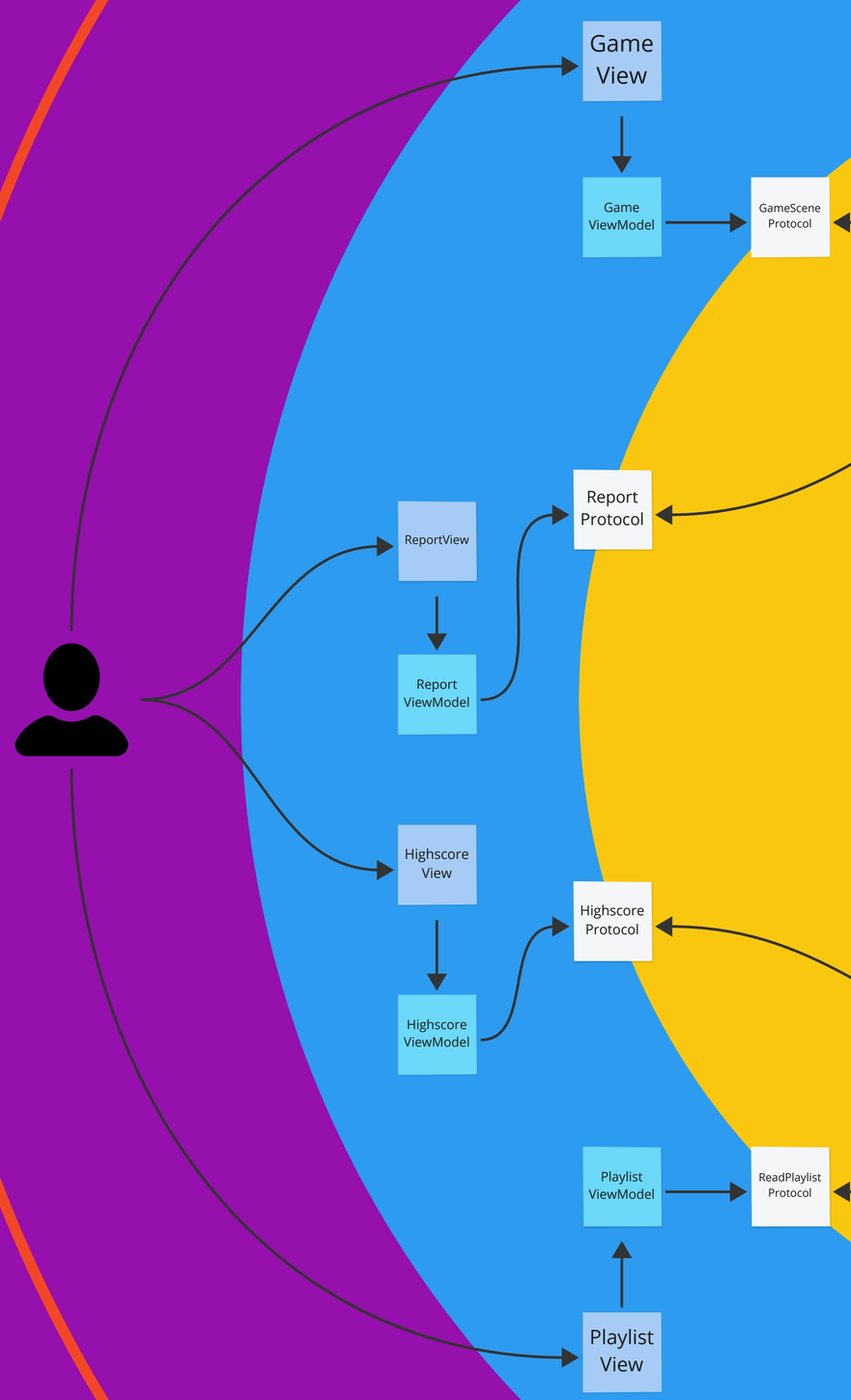
Game View
Game ViewModel
GameScene Protocol
GameEngine
LevelService
ReportService
MusicPlayer Service
MusicPlayer Protocol
Music Player
MusicPlayer Mock
ReportView
Report ViewModel
Report Protocol
Report
Level
NoteFactory
TapNote
PressNote
HoldNote
SwipeNote
ShiftNote
Score
Song
Note
ScoreService
ScoreRepository Protocol
Score Repository
ScoreRepository MockStub
Highscore View
Highscore ViewModel
Highscore Protocol
Playlist ViewModel
ReadPlaylist Protocol
Playlist View
MusicService
MusicRepository Protocol
MusicRepository Stub
Music Repository

Gestione della partita musicale

Valutazione della partita giocata

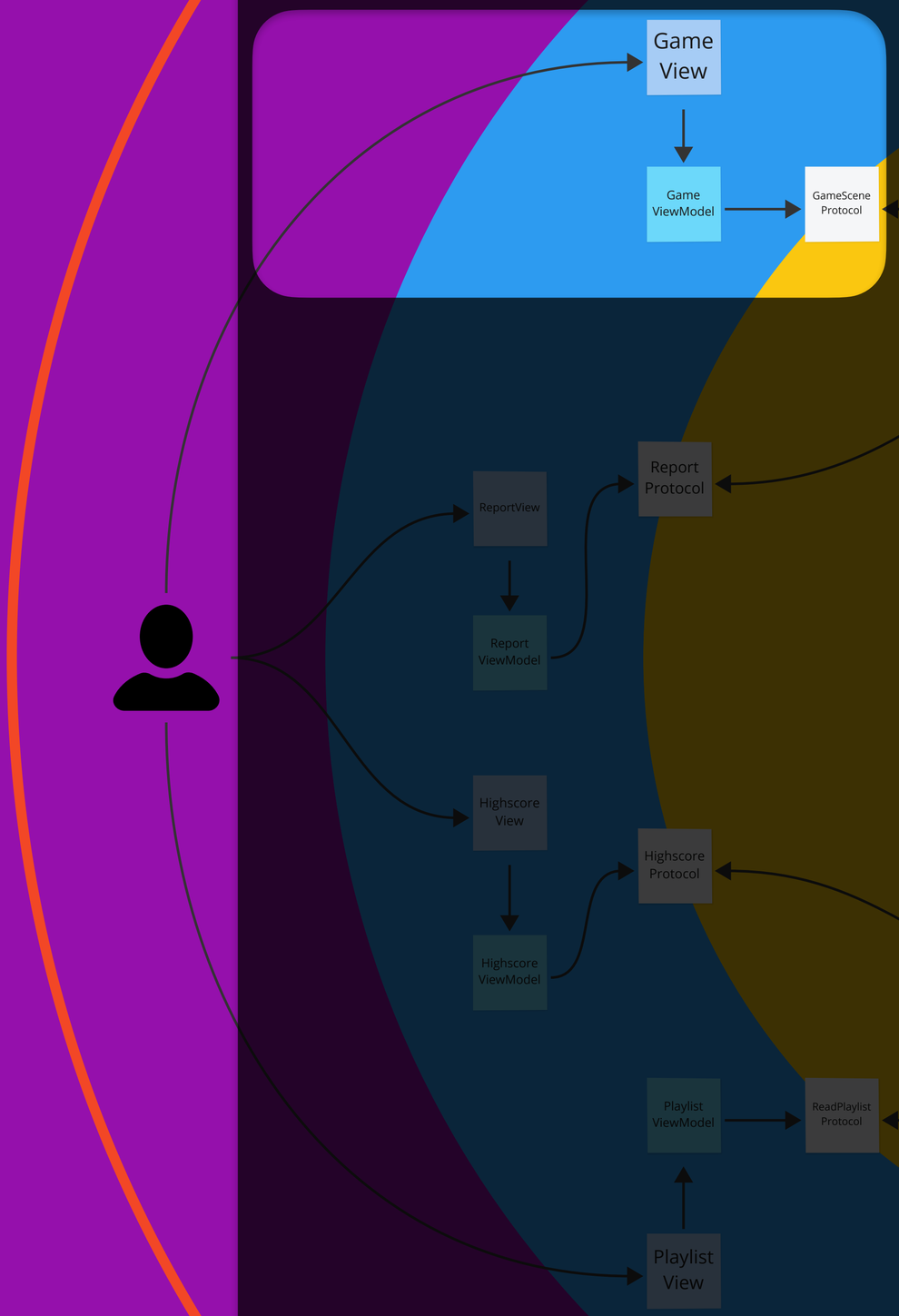
Lista dei punteggi dei giocatori

Selezione della musica da giocare



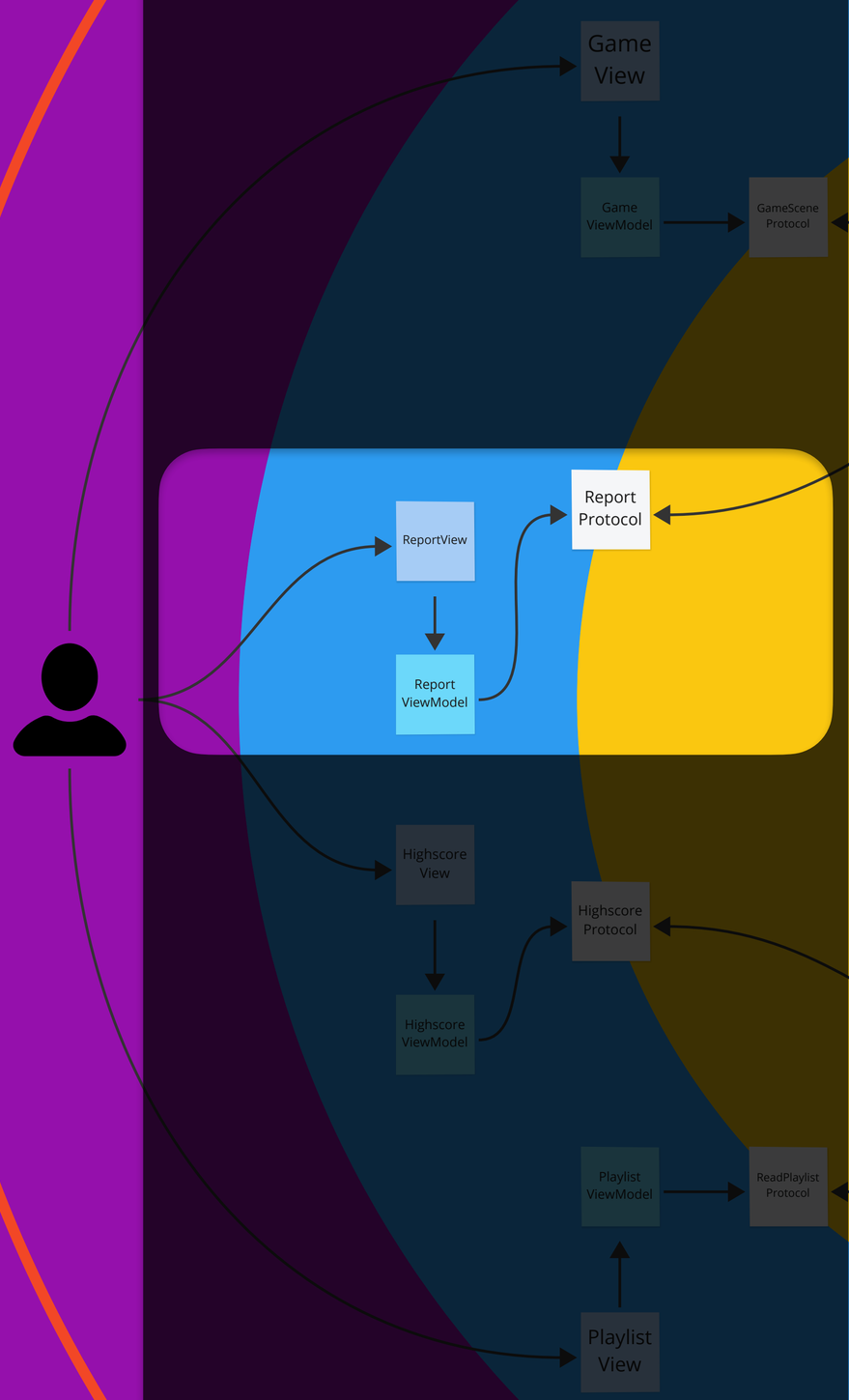
```swift
protocol GameSceneProtocol {
    func renderGame()
}
```

```swift
class GameViewModel : ObservableObject {
    var gameScene: SKScene {
        let scene = GameScene(fileNamed: "GameScene")!
        scene.song = ViewOrchestrator.shared.getSelectedSong()
        scene.difficult = ViewOrchestrator.shared.getSelectedDifficulty()
        scene.scaleMode = .aspectFill
        return scene
    }
}
```



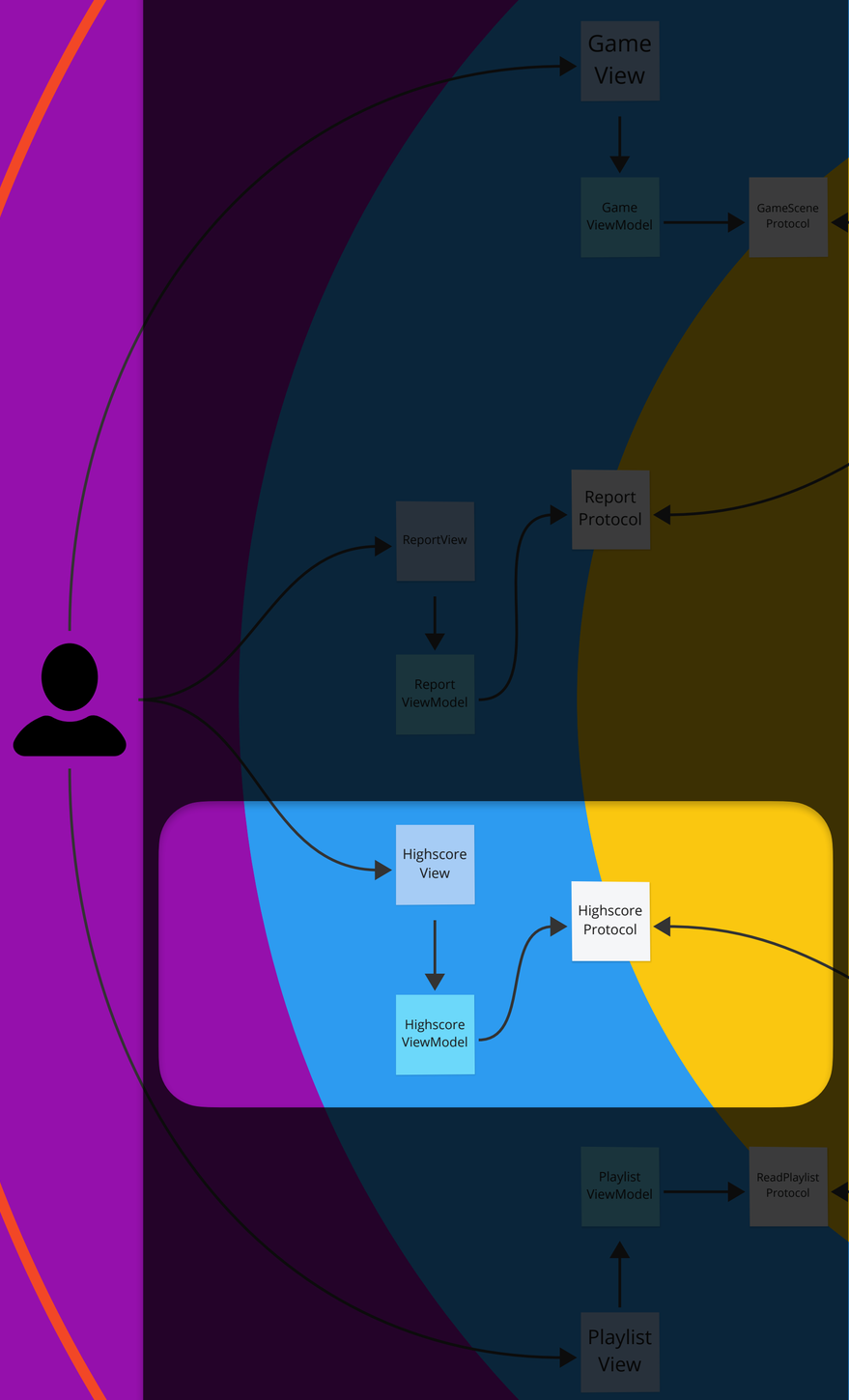
```swift
protocol ReportProtocol {
    func save(score: Score)
    func getReport() -> Report
}
```

```swift
class ReportViewModel : ObservableObject {
    @Published var report: Report?

    private let reportService: ReportProtocol = ReportService()

    func generateReport() {
        report = reportService.getReport()
    }

    func backToPlaylist() {
        ViewOrchestrator.shared.backToPlaylist()
    }
}
```



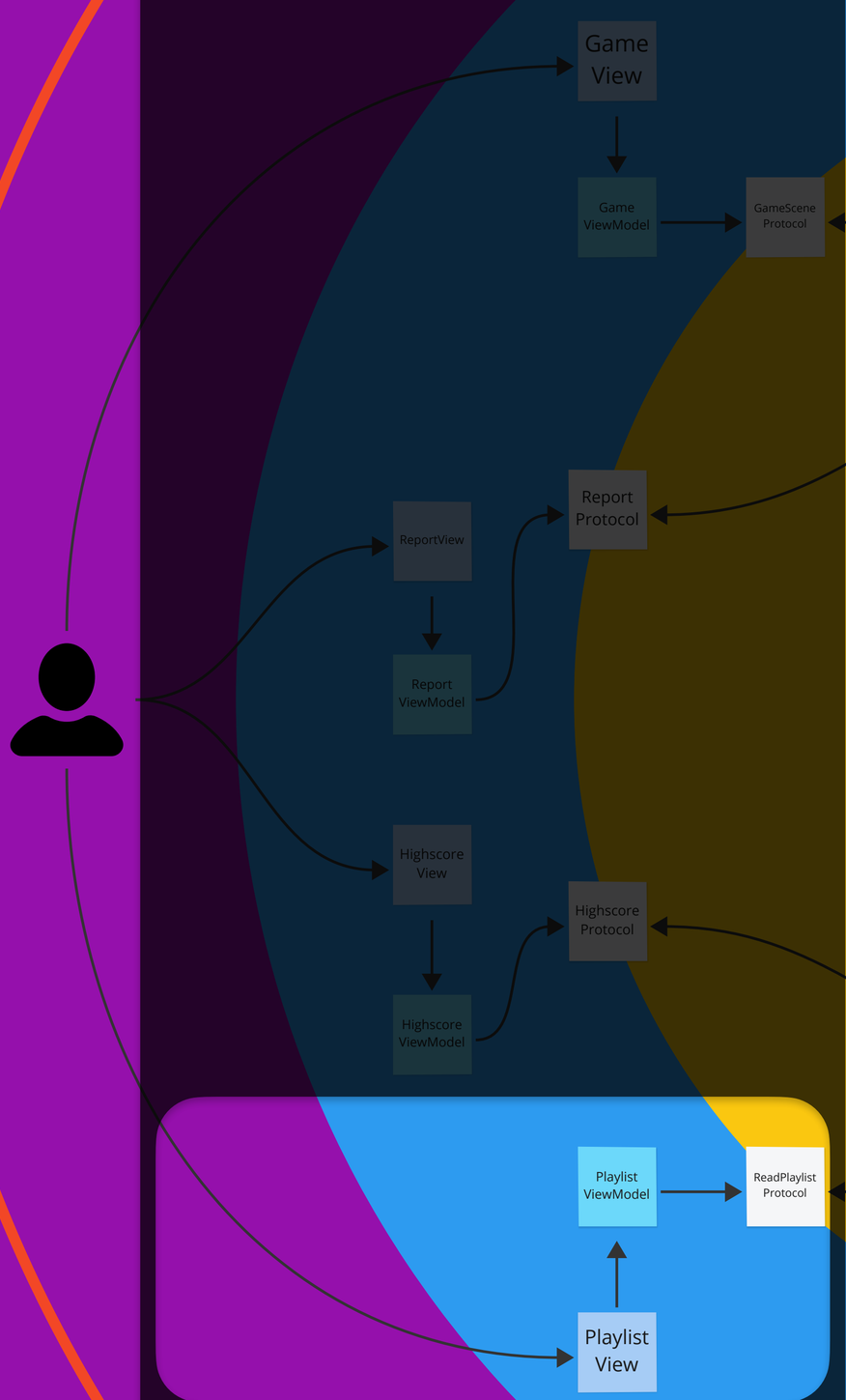
```swift
protocol HighscoreProtocol {
    func getScores(for song: SongSelection, difficulty: Difficulty) -> [Score]
}
```

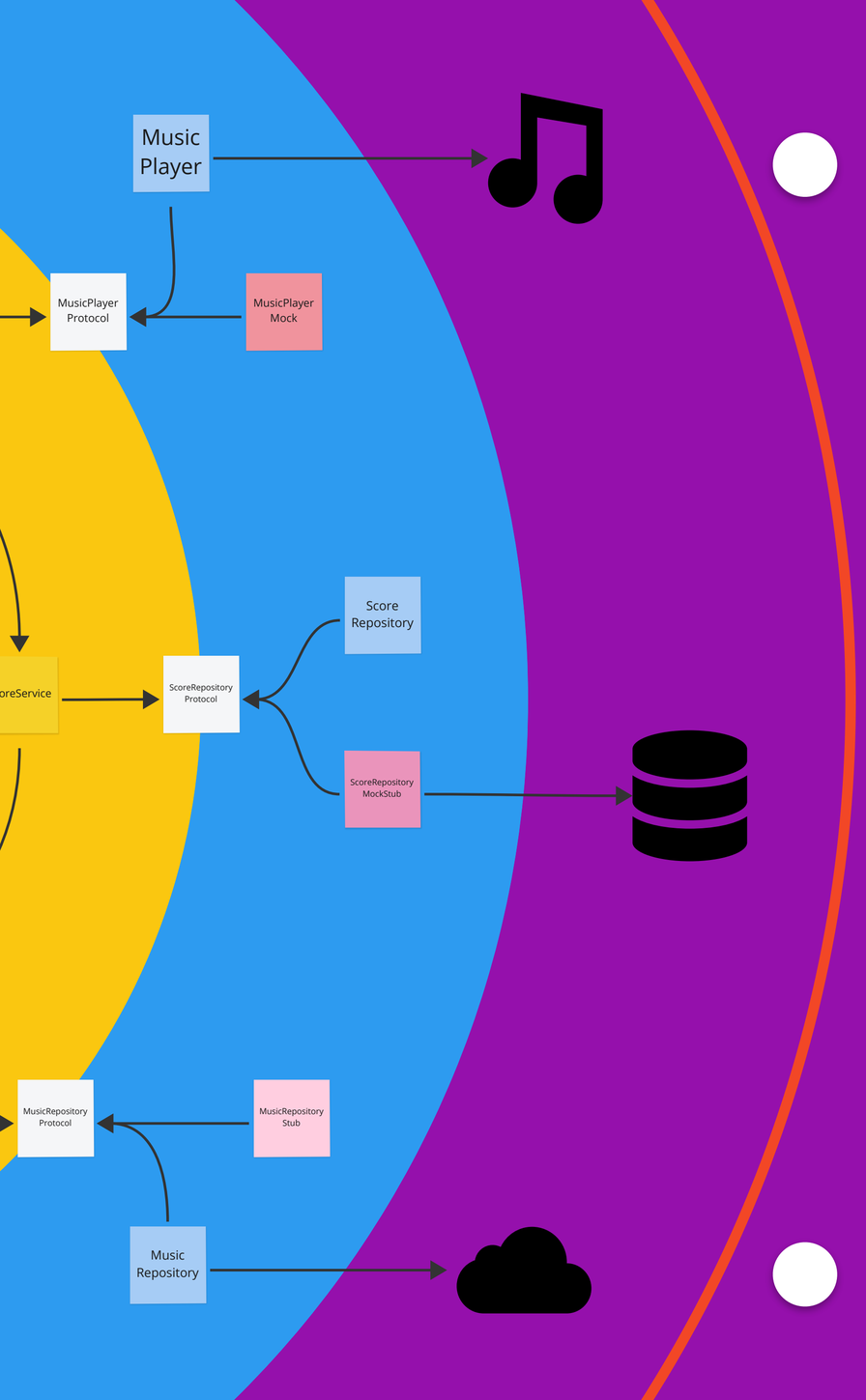
```swift
import Foundation

class HighscoreViewModel : ObservableObject {
    @Published var difficulty = Difficulty.easy
    @Published var scores = [Score]()

    private let scoreService: HighscoreProtocol = ScoreService()

    func fetchScores() {
        let song = ViewOrchestrator.shared.getSelectedSong()
        scores = scoreService.getScores(for: song, difficulty: difficulty)
    }

    func play() {
        ViewOrchestrator.shared.playWith(difficulty: difficulty)
    }
}
```



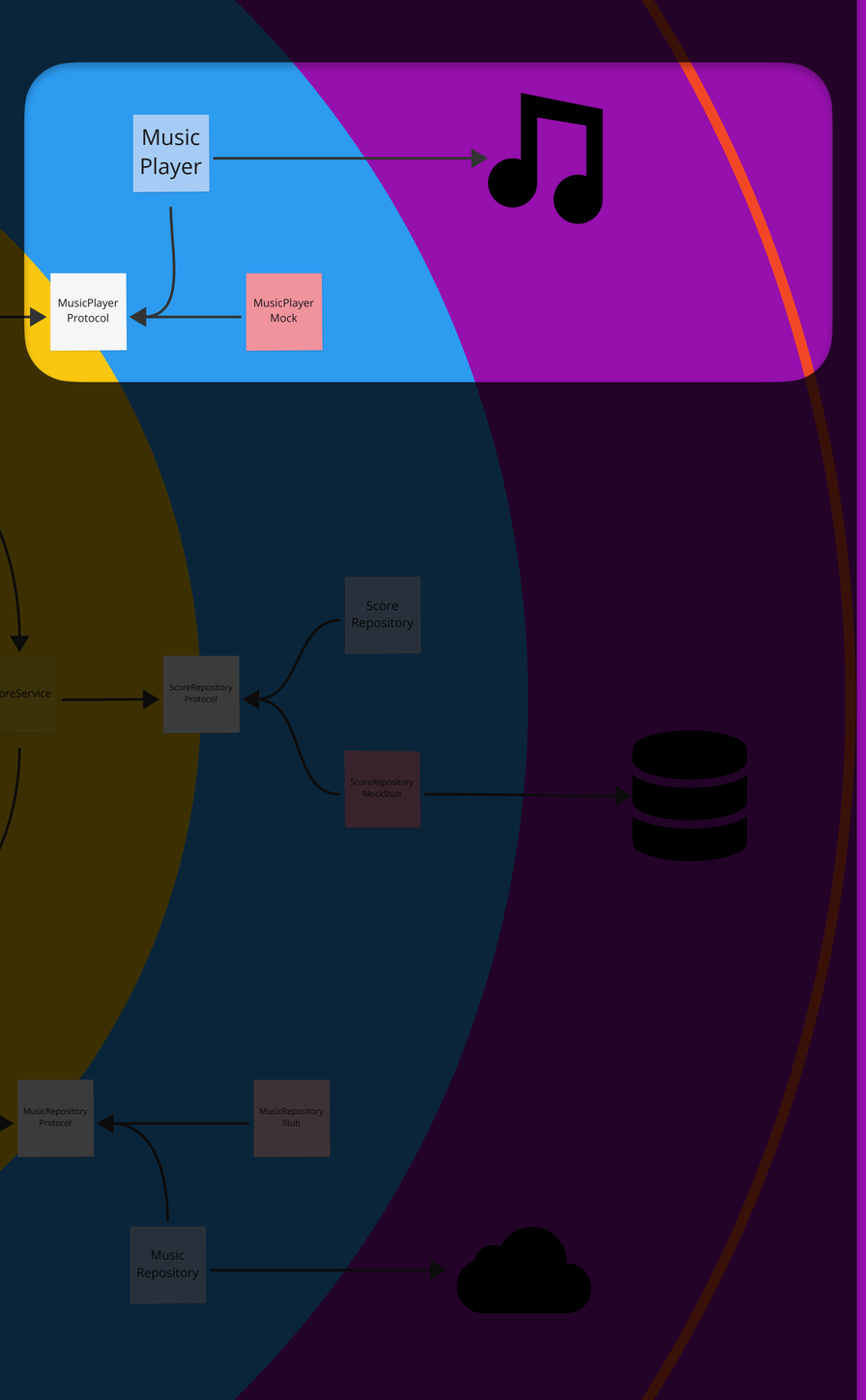
```swift
protocol ReadPlaylistProtocol {
    func getPlaylist() -> [SongDto]
}
```

```swift
class PlaylistViewModel : ObservableObject {
    @Published var playlist = [SongDto]()

    private let musicService: ReadPlaylistProtocol = MusicService()

    func fetchPlaylist() {
        playlist = musicService.getPlaylist()
    }

    func select(song: SongDto) {
        ViewOrchestrator.shared.selectSongToPlay(song: song)
    }
}
```



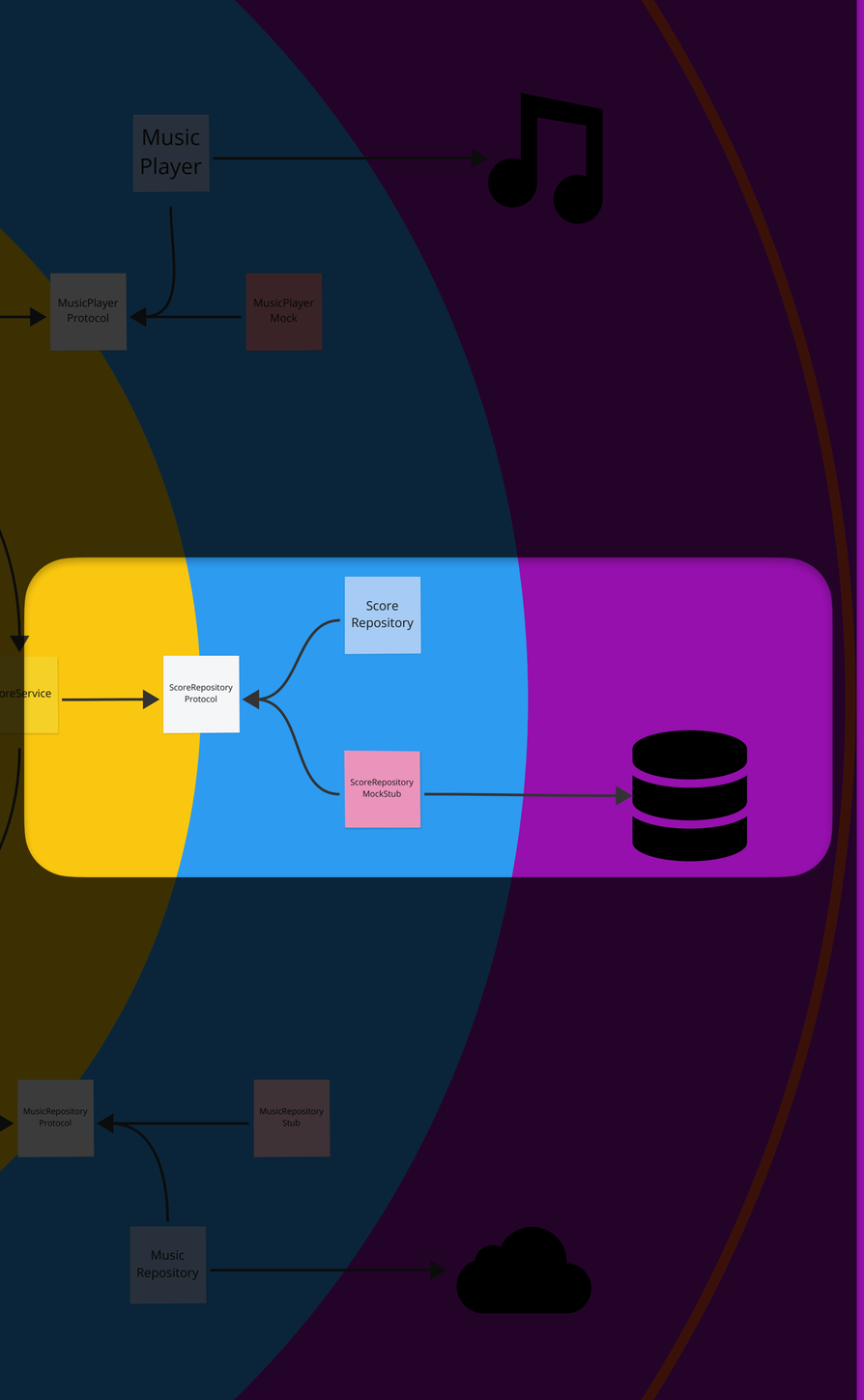


- Riproduzione della musica
- Salvataggio degli score dei giocatori
- Lista di musiche disponibili nel gioco



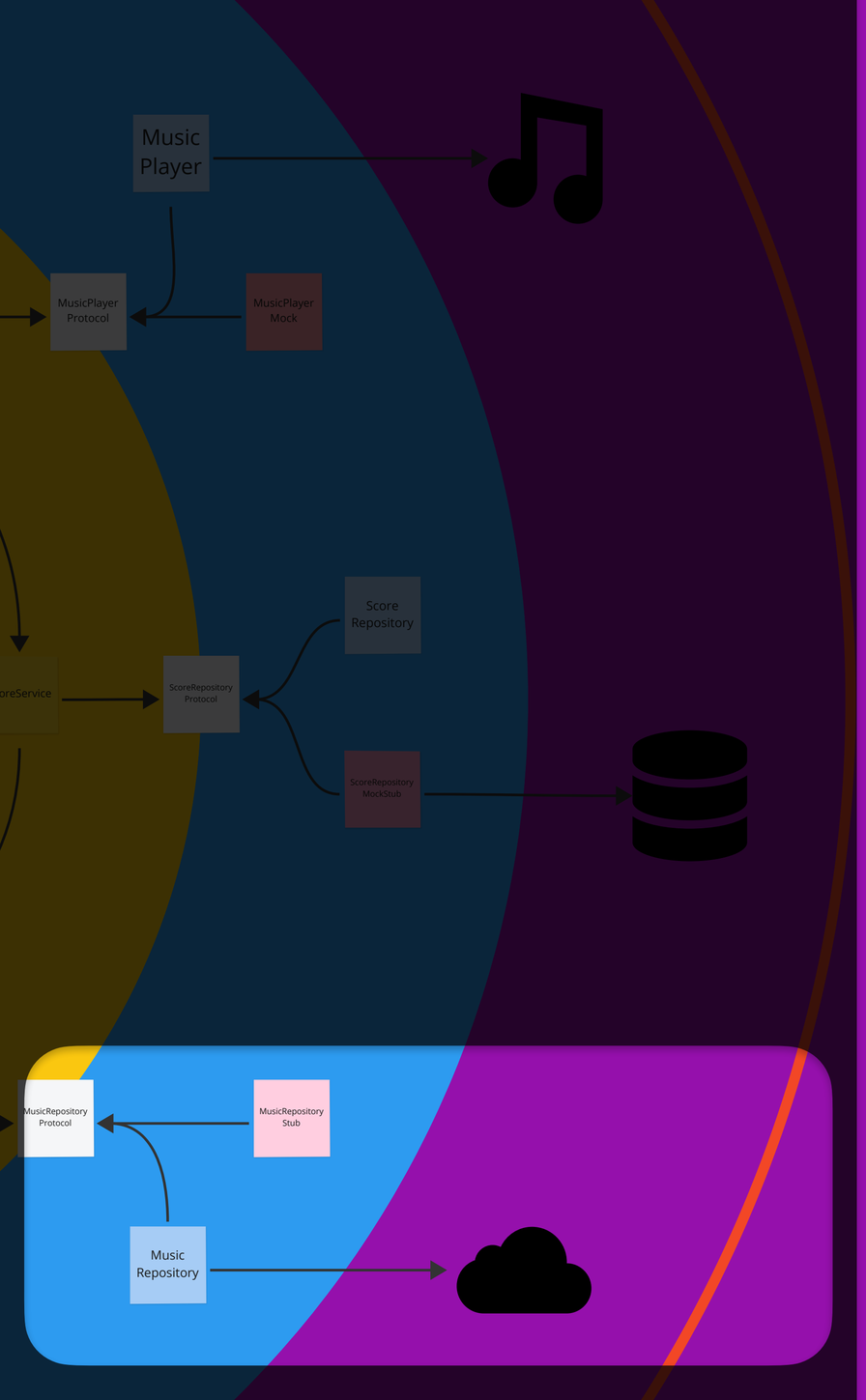

```swift
protocol MusicPlayerProtocol {
    func play(song: Song)
    func resume()
    func pause()
    func stop()
}
```

```swift
class MusicPlayer : MusicPlayerProtocol {
    var audioPlayer: AVPlayer?

    func play(song: Song) {
        if audioPlayer != nil {
            stop()
        }

        let url = Bundle.main.url(forResource: song.fileName, withExtension: "mp3")
        audioPlayer = AVPlayer(url: url!)
        audioPlayer?.play()
    }

    func resume() {
        audioPlayer?.play()
    }

    func pause() {
        audioPlayer?.pause()
    }

    func stop() {
        audioPlayer?.pause()
        audioPlayer = nil
    }
}
```




```swift
protocol ScoreRepositoryProtocol {
    func addScore(score: Score)
    func getScores() -> [Score]
}
```

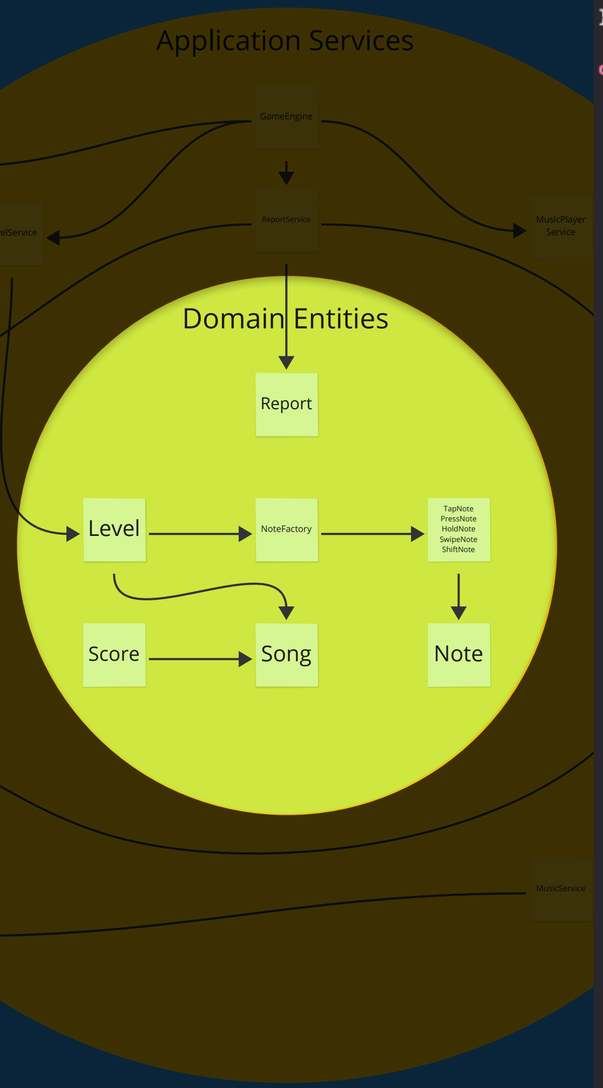
```swift
final class ScoreRepository : ScoreRepositoryProtocol {
    @MainActor
    static let shared = ScoreRepository()

    var sharedModelContainer: ModelContainer = {
        let schema = Schema([Score.self])
        let modelConfiguration = ModelConfiguration(schema: schema, isStoredInMemoryOnly: false)

        do {
            return try ModelContainer(for: schema, configurations: [modelConfiguration])
        } catch {
            fatalError("Could not create ModelContainer: \(error)")
        }
    }()

    private let modelContainer: ModelContainer
    private let modelContext: ModelContext

    @MainActor
    private init() {
        let schema = Schema([Score.self])
        let modelConfiguration = ModelConfiguration(schema: schema, isStoredInMemoryOnly: false)
        self.modelContainer = try! ModelContainer(for: schema, configurations: [modelConfiguration])
        self.modelContext = modelContainer.mainContext
    }

    func addScore(score: Score) {
        modelContext.insert(score)

        do {
            try modelContext.save()
        } catch {
            fatalError(error.localizedDescription)
        }
    }

    func getScores() -> [Score] {
        do {
            return try modelContext.fetch(FetchDescriptor<Score>())
        } catch {
            fatalError(error.localizedDescription)
        }
    }
}
```




```swift
protocol MusicRepositoryProtocol {
    func getPlaylist() -> [SongDto]
}
```

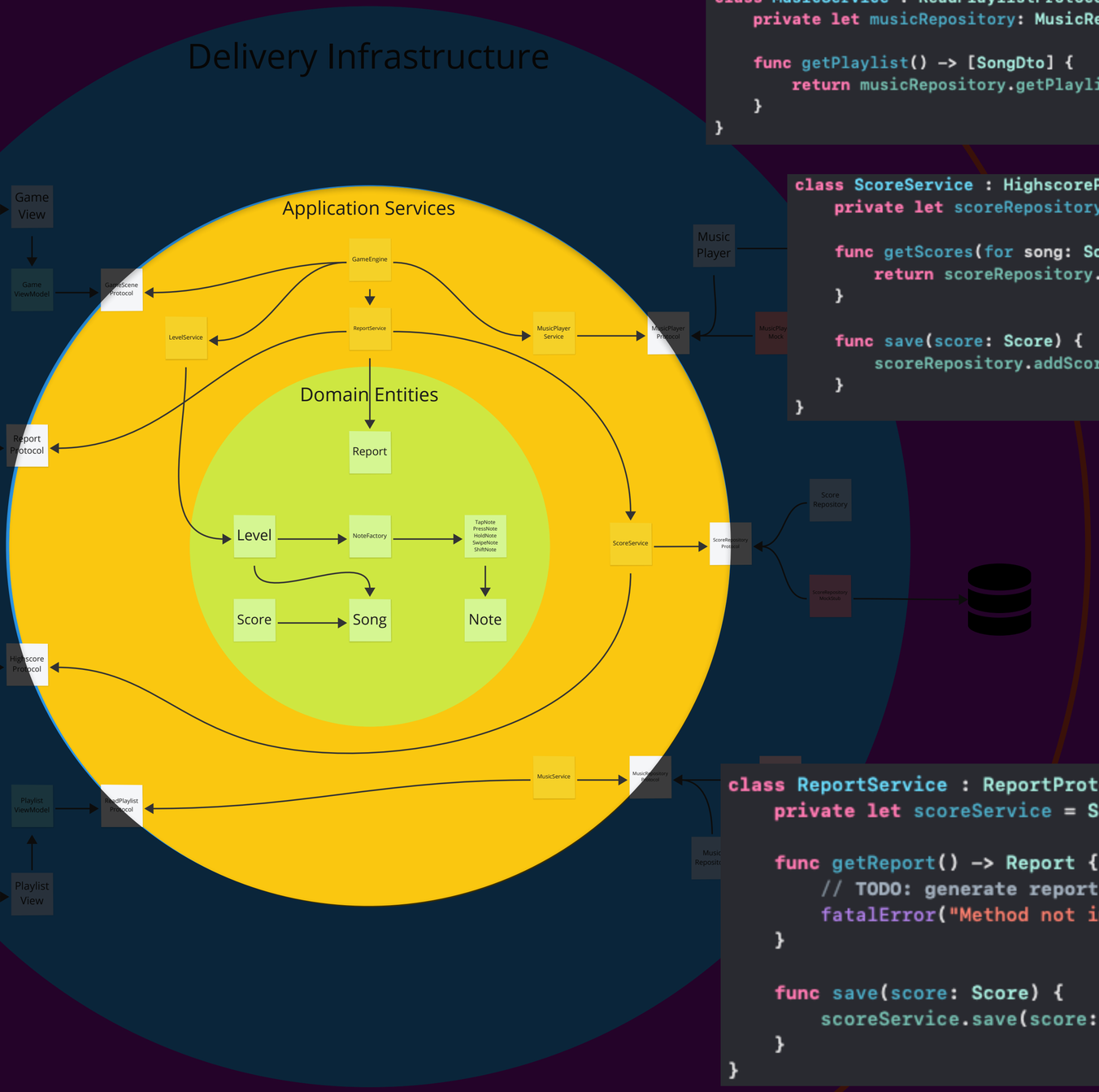
```swift
class MusicRepository : MusicRepositoryProtocol {
    private let songs: [SongDto]

    init() {
        self.songs = [
            SongDto(id: 0, name: "American Idiot", artist: "Green Day", cover: UIImage(named: "American Idiot")!),
            SongDto(id: 1, name: "Crossing Field", artist: "Lisa", cover: UIImage(named: "Crossing Field")!),
            SongDto(id: 2, name: "Free Bird", artist: "Lynyrd Skynyrd", cover: UIImage(named: "Free Bird")!),
            SongDto(id: 3, name: "Hello", artist: "OMFG", cover: UIImage(named: "Hello")!),
            SongDto(id: 4, name: "Inno alla Gioia", artist: "Beethoven", cover: UIImage(named: "Inno alla Gioia")!),
            SongDto(id: 5, name: "Real Gone", artist: "Sheryl Gone", cover: UIImage(named: "Real Gone")!),
            SongDto(id: 6, name: "Spazzmatica Polka", artist: "Kevin MacLeod", cover: UIImage(named: "Spazzmatica Polka")!)
        ]
    }

    func getPlaylist() -> [SongDto] {
        return songs
    }
}
```

Delivery Infrastructure




```swift
protocol NoteMissedProtocol {
    func onNoteMissed(finalPosition: CGPoint, note: Note)
}

class Note: SKSpriteNode {
    var fromPoint: CGPoint
    var toPoint: CGPoint
    var startingTime: TimeInterval
    var duration: TimeInterval
    var onNoteMissedProtocol: NoteMissedProtocol?

    init(fromPoint: CGPoint, toPoint: CGPoint, startingTime: TimeInterval, duration:
        TimeInterval, texture: String, size: CGSize) {
        self.fromPoint = fromPoint
        self.toPoint = toPoint
        self.startingTime = startingTime
        self.duration = duration

        super.init(texture: SKTexture(imageNamed: texture), color: .clear, size: size)
        super.position = fromPoint
    }

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

    func move() {
        super.position = fromPoint

        // Calculating the time to reach the bottom of the view
        let distBeginBtn = abs(toPoint.y - fromPoint.y)
        let newTime = CGFloat(duration * 700)
        let t = newTime / distBeginBtn

        // Creating the animation: wait, move to the bottom of the view, cease to exist,
        // notify the listener
        let vector = CGVector(dx: toPoint.x - fromPoint.x, dy: 0 - fromPoint.y)
        self.run(SKAction.sequence([
            SKAction.wait(forDuration: startingTime),
            SKAction.move(by: vector, duration: TimeInterval(t)),
            SKAction.removeFromParent(),
            SKAction.run { self.onNoteMissedProtocol?.onNoteMissed(finalPosition:
                self.toPoint, note: self) }
            ]))
    }
}
```

```swift
import Foundation
import SpriteKit

class HoldNote : Note {

    let START_Y: Double = 700           // The top of the view
    let END_Y: Double = 50              // The bottom of the view

    var parentNote: PressNote?          // Reference of the parent note

    // Initializer: Setting this note as a normal musical note and giving also the parent
    init(xPosition: Double, theStartingTime: TimeInterval, theDuration: TimeInterval,
        theParent: PressNote) {
        super.init(fromPoint: CGPoint(x: xPosition, y: START_Y), toPoint: CGPoint(x:
            xPosition, y: END_Y), startingTime: theStartingTime, duration: theDuration,
            texture: "default_artwork", size: CGSize(width: 50, height: 50))
        parentNote = theParent
    }

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

}
```




```
class MusicService : ReadPlaylistProtocol {
    private let musicRepository: MusicRepositoryProtocol = MusicRepository()

    func getPlaylist() -> [SongDto] {
        return musicRepository.getPlaylist()
    }
}
```

```
class ScoreService : HighscoreProtocol {
    private let scoreRepository: ScoreRepositoryProtocol = ScoreRepository.shared

    func getScores(for song: SongSelection, difficulty: Difficulty) -> [Score] {
        return scoreRepository.getScores()
    }

    func save(score: Score) {
        scoreRepository.addScore(score: score)
    }
}
```

```
class MusicPlayerService {
    private let musicPlayer: MusicPlayerProtocol = MusicPlayer()

    func play() {
        let song = ViewOrchestrator.shared.getSelectedSong()
        musicPlayer.play(song: song)
    }

    func resume() {
        musicPlayer.resume()
    }

    func pause() {
        musicPlayer.pause()
    }

    func stop() {
        musicPlayer.stop()
    }
}
```

```
class ReportService : ReportProtocol {
    private let scoreService = ScoreService()

    func getReport() -> Report {
        // TODO: generate report
        fatalError("Method not implemented")
    }

    func save(score: Score) {
        scoreService.save(score: score)
    }
}
```

Il nostro percorso

01 **L’inizio di tutto**
Qual è stata l’idea dietro questa presentazione?

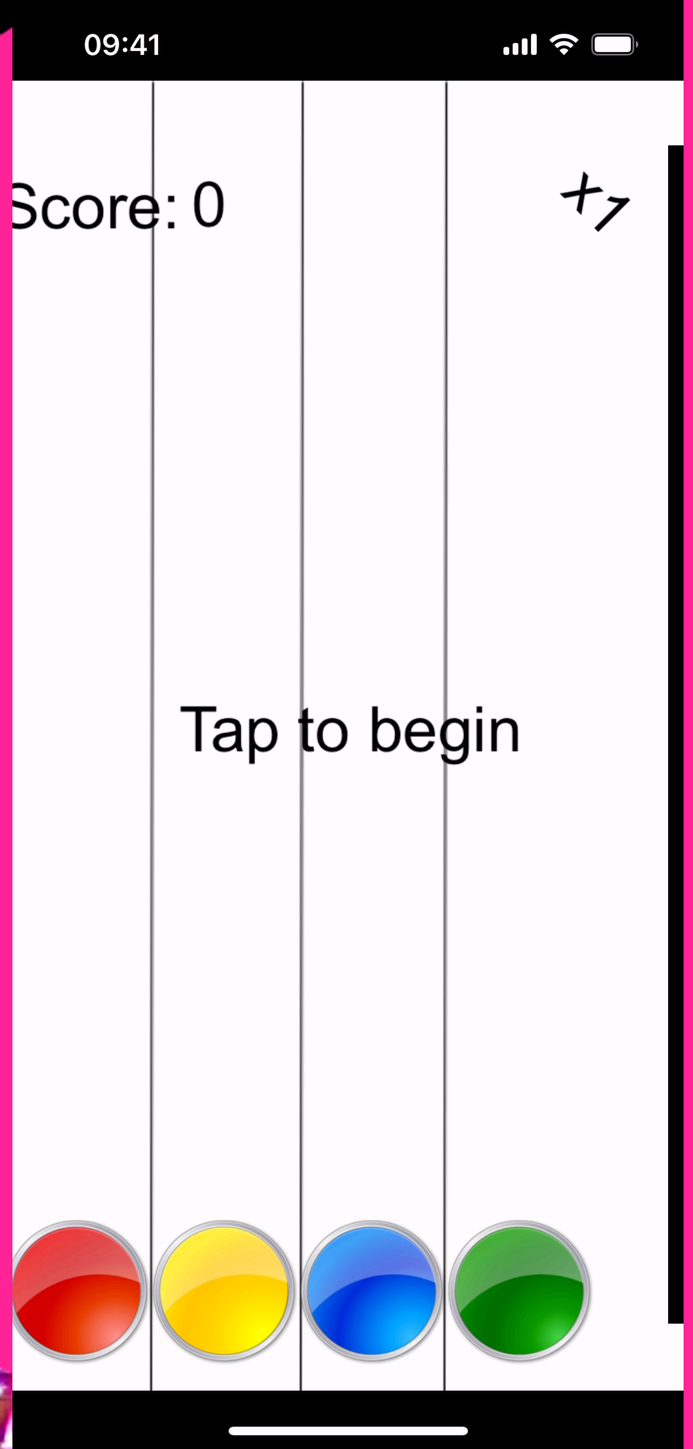
02 **Spaghetti Architecture**
Prima la scrittura del codice e poi la definizione del design

03 **Onion Architecture**
Prima la definizione del design e poi la scrittura del codice

04 **La Demo**
Fammi vedere il gioco!

05 **Conclusione**
Meglio scrivere prima il codice o definire il design?

La Demo



Il nostro percorso

01 **L’inizio di tutto**
Qual è stata l’idea dietro questa presentazione?

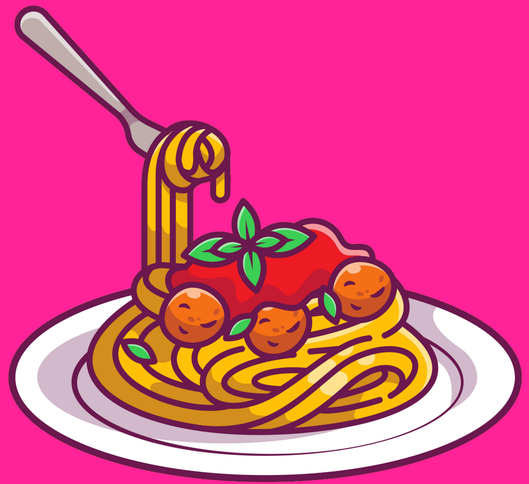
02 **Spaghetti Architecture**
Prima la scrittura del codice e poi la definizione del design

03 **Onion Architecture**
Prima la definizione del design e poi la scrittura del codice

04 **La Demo**
Fammi vedere il gioco!

05 **Conclusione**
Meglio scrivere prima il codice o definire il design?

Conclusione




Grazie!