```swift
//
//  StudentViewController.swift
//  AttendanceApplication
//

import UIKit
import CoreLocation
import Alamofire

class StudentViewController: UIViewController, CLLocationManagerDelegate {

    // MARK: IBOutlets

    @IBOutlet weak var studentView: UIView!

    @IBOutlet weak var studentName: UILabel!
    @IBOutlet weak var studentEmail: UILabel!
    @IBOutlet weak var closestBeaconName: UILabel!
    @IBOutlet weak var beaconDescription: UILabel!

    @IBOutlet weak var statusLabel: UILabel!

    // MARK: Private Properties

    var locationManager = CLLocationManager()

    var threadStarted = false
    var backgroundTask: UIBackgroundTaskIdentifier = UIBackgroundTaskInvalid

    private let timeInterval: Int = 300 // Time interval set to 300 seconds (5
     minutes)

    private var timer: Timer!
    private var currentBeacon: (Int8, Int8) = (0, 0)
    private let beaconRegion = CLBeaconRegion(proximityUUID: UUID(uuidString:
     "DCEF54A2-31EB-467F-AF8E-350FB641C97B")!, identifier: "SchoolBeacon")

    // MARK: UIViewController methods

    override func viewDidLoad() {
        super.viewDidLoad()

        locationManager = CLLocationManager()
        locationManager.delegate = self
        locationManager.requestAlwaysAuthorization()
        locationManager.allowsBackgroundLocationUpdates = true
        locationManager.startUpdatingLocation()

        studentView.backgroundColor = UIColor.orange
        // Get the information about the student from the server, update the labels
        getInfo()
        // Start the scanner with the CLLocation Manager
        startScanning()
        // Start the timer which will activate the location updater at a interval of
         timeInterval
        startUpdateIntervalTimer()
        // An inital update for the view
        updateLocation(major_key: currentBeacon.0, minor_key: currentBeacon.1)
    }

    override func viewWillDisappear(_ animated: Bool) {
        performLogout()
    }

    // MARK: IBAction methods

    @IBAction func logoutButton(_ sender: Any) {
        /*
         * Method connected to the logout button in the storyboard. Calls seperate
         * method that performs the logout.
         */
        performLogout()
    }

    // MARK: Internal methods

    internal func startScanning() {
        beaconRegion.notifyEntryStateOnDisplay = true
        studentView.backgroundColor = UIColor.green
        locationManager.startMonitoring(for: beaconRegion)
        locationManager.startRangingBeacons(in: beaconRegion)
    }

    internal func stopScanning() {
        beaconRegion.notifyEntryStateOnDisplay = false
        locationManager.stopMonitoring(for: beaconRegion)
        locationManager.stopRangingBeacons(in: beaconRegion)
    }

    internal func locationManager(_ manager: CLLocationManager, didRangeBeacons
     beacons: [CLBeacon], in region: CLBeaconRegion) {
        if beacons.count > 0 {
            currentBeacon = (beacons[0].major.int8Value, beacons[0].minor.int8Value)
        } else {
            currentBeacon = (0, 0)
        }
    }

    @objc func updateIntervalTimer() {
//        if currentBeacon.1 != 0 {
            updateLocation(major_key: currentBeacon.0, minor_key: currentBeacon.1)
//        }
    }

    internal func updateLocation(major_key: Int8, minor_key: Int8) {
        /*
         * Method that is called at a timed interval to update the students location
          in relation
         * to beacons currently around him/her. Sends a POST request with the major
          and minor keys
         * of the database as arguments.
         *
         * returns: A JSON response with a variable representing if it was
          successful, and a
         *          variable with the reason why. As well as a variable that
          requires a login
         *          if the current session has expired.
         */
        let parameters: Parameters = [
            "type": "student.update_location",
            "args": [
                // Username is added by the session manager
                "username": "",
                "major": major_key,
                "minor": minor_key
            ]
        ]

        Alamofire.request(HTTPHelper.url, method: .post, parameters: parameters,
         encoding: JSONEncoding.default).responseJSON {
            response in

            switch response.result {
            case .failure( _):
                self.indicateInactiveState(reason: "Failure connecting to server.")
                self.closestBeaconName.text = "Unable to find beacon."
                self.beaconDescription.text = "Unable to find zone."
                return

            case .success(let data):
                // First make sure a dictionary is recieved: Data validation
                guard let json = data as? [String : AnyObject] else {
                    // Print statement for debugging purposes, not seen by users.
                    print("Failed to get expected dictionary from webserver.")
                    return
                }

                // Then make sure that key/value pairs are correct: Data validation
                guard let success = json["successful"] as? Int, let reason =
                 json["reason"] as? String, let closestBeacon =
                 json["closest_beacon"] as? String, let currentZone =
                 json["beacon_description"] as? String else {
                    // Print statement for debugging purposes, not seen by users.
                    print("Failed to get expected data from webserver")
                    return
                }

                if success == 1 {
                    // If successful in updating location, update the user's view
                     with their zone and closest beacon.
                    self.closestBeaconName.text = closestBeacon
                    self.beaconDescription.text = currentZone
                    self.indicateActiveState()
                } else {
                    // If unsuccesful in updating location, let user know with
                     reason provided by server and update status.
                    self.closestBeaconName.text = "Unable to find beacon."
                    self.beaconDescription.text = "Unable to find zone."
                    self.indicateInactiveState(reason: reason)
                }
            }
        }
}

internal func getInfo() {
    let parameters: Parameters = [
        "type": "student.get_info",
        "args": [
            // Username is added by the session manager
            "username": ""
        ]
    ]

    Alamofire.request(HTTPHelper.url, method: .post, parameters: parameters,
     encoding: JSONEncoding.default).responseJSON {
        response in

        switch response.result {
        case .failure( _):
            self.indicateInactiveState(reason: "Failure connecting to server.")
            return

        case .success(let data):
            // First make sure a dictionary is recieved: Data validation
            guard let json = data as? [String : AnyObject] else {
                // Print statement for debugging purposes, not seen by users.
                print("Failed to get expected dictionary from webserver.")
                return
            }

            // Then make sure that key/value pairs are correct: Data validation
            guard let success = json["successful"] as? Int, let reason =
             json["reason"] as? String, let name = json["name"] as? String, let
             email = json["email"] as? String else {
                // Print statement for debugging purposes, not seen by users.
                print("Failed to get expected data from webserver")
                return
            }

            if success == 1 {
                // If successful update studentName and studentEmail label.
                self.studentName.text = name
                self.studentEmail.text = email
                self.indicateActiveState()
                } else {
                    // If unsuccesful in fetching user info , let user know with
                     reason provided by server and update status.
                    self.indicateInactiveState(reason: reason)
                }
            }
        }
    }

    internal func indicateInactiveState(reason: String){
        self.statusLabel.text = reason
        self.studentView.backgroundColor = UIColor.red
    }

    internal func indicateActiveState(){
        self.statusLabel.text = "Active"
        self.studentView.backgroundColor = UIColor.green
    }

    internal func performLogout() {
        /*
         * Method that stops active processes and segues back to the login view.
         * Exists as a seperate method from logoutButton() so that it can be
         * called if a session has expired.
         */
        self.stopUpdateIntervalTimer()
        self.stopScanning()
        dismiss(animated: true, completion: nil)
    }

    internal func startUpdateIntervalTimer() {
        timer = Timer.scheduledTimer(timeInterval: TimeInterval(self.timeInterval),
         target: self, selector: #selector(self.updateIntervalTimer), userInfo: nil,
         repeats: true)
    }

    internal func stopUpdateIntervalTimer() {
        timer.invalidate()
    }
}
```