```swift
//
//  PlayChallengeRESTWebAPIModelAccessStrategy.swift
//  f30
//
//  Created by David on 05/02/2018.
//  Copyright © 2018 com.smartfoundation. All rights reserved.
//

import SFCore
import SFModel
import SFSocial
import SFSerialization
import SFNet
import f30Core
import f30Model

/// A strategy for accessing the PlayChallenge model data using a REST Web API
public class PlayChallengeRESTWebAPIModelAccessStrategy:
 RESTWebAPIModelAccessStrategyBase {

    // MARK: - Initializers

    private override init() {
        super.init()
    }

    public override init(connectionString: String,
                         storageDateFormatter: DateFormatter) {
        super.init(connectionString: connectionString,
                   storageDateFormatter: storageDateFormatter,
                   tableName: "PlayChallenges")

    }


    // MARK: - Private Methods

    fileprivate func runQuery(byIsActiveYN isActiveYN: Bool, relativeMemberID:
     String, playGameID: String, collection: ProtocolModelItemCollection,
     oncomplete completionHandler: @escaping ([String : Any]?, Error?) -> Void) {

        #if DEBUG

            if (ApplicationFlags.flag(key: "LoadPlayChallengesDummyDataYN")) {

                self.selectDummy(byIsActiveYN: isActiveYN, relativeMemberID:
                 relativeMemberID, playGameID: playGameID, collection: collection,
                 oncomplete: completionHandler)

                return

            }

        #endif

        // Create the dataWrapper
        let dataWrapper:            DataJSONWrapper = DataJSONWrapper()
        dataWrapper.setParameterValue(key: "\
         (PlayChallengeDataParameterKeys.IsActiveYN)", value: "\
         (BoolHelper.toInt(value: isActiveYN))")

        // Create processResponse completion handler
        let processResponseCompletionHandler: (([String:Any]?, URLResponse?,
         Error?) -> Void) =
        {
            (data, response, error) -> Void in  // [weak self]

            // Call the completion handler
            completionHandler(data, error)
        }

        // Create processResponse
        let processResponse:        ((NSMutableData?, URLResponse?, Error?) ->
         Void) = self.getProcessResponse(oncomplete:
         processResponseCompletionHandler)

        // Create restApiHelper
        let restApiHelper:          RESTApiHelper = RESTApiHelper(processResponse:
         processResponse, mode: RESTApiHelperMode.CompletionHandler)

        // Get the Url
        var urlString:              String =
         NSLocalizedString("PlayChallengesSelectbyIsActiveYNPlayGameID",
         tableName: "RESTWebAPIConfig", comment: "")
        urlString                   = String(format: urlString,
                                             relativeMemberID,
                                             playGameID)

        // Call the REST Api
        restApiHelper.call(urlString: urlString, httpMethod: .POST, data:
         dataWrapper)

    }

    fileprivate func getPlayChallengeObjectiveModelAdministrator() ->
     PlayChallengeObjectiveModelAdministrator? {

        // Get modelAdministratorProvider
        let modelAdministratorProvider:     ProtocolModelAdministratorProvider? =
         self.delegate?.modelAccessStrategy(getModelAdministratorProvider: self)

        guard (modelAdministratorProvider != nil) else { return nil }

        // Get PlayChallengeObjectiveModelAdministrator
        let pcoma:
         PlayChallengeObjectiveModelAdministrator? =
         modelAdministratorProvider!.getModelAdministrator(key:
         "PlayChallengeObjectives") as? PlayChallengeObjectiveModelAdministrator

        return pcoma

    }
    // MARK: - Override Methods

    public override func getRelationalDataWrapper(fromItem item:
     ProtocolModelItem) -> DataJSONWrapper? {

        let result:                             DataJSONWrapper? =
         DataJSONWrapper()

        // Create playChallengeObjectiveWrappers
        let playChallengeObjectiveWrappers:     DataJSONWrapper =
         DataJSONWrapper()
        playChallengeObjectiveWrappers.ID = "PlayChallengeObjectives"
        result?.Items.append(playChallengeObjectiveWrappers)

        // Go through each item
        for pco in
         self.getPlayChallengeObjectiveModelAdministrator()!.collection!.items! {

            let pco = pco as! PlayChallengeObjective

            playChallengeObjectiveWrappers.Items.append(pco.copyToWrapper())

        }

        return result

    }

    public override func setRelationalItems(fromWrapper relationalDataWrapper:
     DataJSONWrapper, for item: ProtocolModelItem, originalID: String) {

        // Get playChallengeObjectiveWrappers
        var playChallengeObjectiveWrappers:     DataJSONWrapper? = nil

        // Go through each item
        for item in relationalDataWrapper.Items {

            if (item.ID == "PlayChallengeObjectives") {

                playChallengeObjectiveWrappers = item

            }

        }

        guard (playChallengeObjectiveWrappers != nil) else { return }

        #if DEBUG

            if (ApplicationFlags.flag(key: "SaveDummyDataYN")) {

                var item                        = item as ProtocolModelItem
                item.id                         = UUID().uuidString

                // Go through each item
            for pco in
             self.getPlayChallengeObjectiveModelAdministrator()!.collection!.i
             tems! {

                let pco = pco as! PlayChallengeObjective

                pco.id                      = UUID().uuidString
                pco.playChallengeID         = item.id
                pco.status                  = .unmodified

            }

            return

        }

    #endif

    // Go through each item
    for pcow in playChallengeObjectiveWrappers!.Items {

        // Get OriginalID
        let oid:                String? = pcow.getParameterValue(key:
         "OriginalID")

        guard (oid != nil) else { continue }

        // Get PlayChallengeObjective item
        let pco:                PlayChallengeObjective? =
         self.getPlayChallengeObjectiveModelAdministrator()!.collection!.getIt
         em(id: oid!) as? PlayChallengeObjective

        guard (pco != nil) else { continue }

        // Update PlayChallengeObjective
        pco!.id                 = pcow.ID
        pco!.playChallengeID    = item.id
        pco!.status             = .unmodified

    }

}


// MARK: - Dummy Data Methods

fileprivate func selectDummy(byIsActiveYN isActiveYN: Bool, relativeMemberID:
 String, playGameID: String, collection: ProtocolModelItemCollection,
 oncomplete completionHandler: @escaping ([String : Any]?, Error?) -> Void) {

    let responseString  = NSLocalizedString("byIsActiveYN_playGameID",
     tableName: "PlayChallengesDummyRESTWebAPIResponse", comment: "")

    // Convert the response to JSON dictionary
    let data:           [String:Any]? = JSONHelper.stringToJSON(jsonString:
     responseString) as? [String:Any]
```

```
        // Process the data
        let returnData:     [String:Any]? =
         self.processRESTWebAPIResponse(responseData: data!)

        // Call the completion handler
        completionHandler(returnData, nil)

    }


}

// MARK: - Extension ProtocolPlayChallengeModelAccessStrategy

extension PlayChallengeRESTWebAPIModelAccessStrategy:
 ProtocolPlayChallengeModelAccessStrategy {

    // MARK: - Public Methods

    public func select(byIsActiveYN isActiveYN: Bool, relativeMemberID: String,
     playGameID: String, collection: ProtocolModelItemCollection, oncomplete
     completionHandler: @escaping ([String : Any]?, ProtocolModelItemCollection?,
     Error?) -> Void) {

        // Create completion handler
        let runQueryCompletionHandler: (([String:Any]?, Error?) -> Void) =
         self.getRunQueryCompletionHandler(collection: collection, oncomplete:
         completionHandler)

        // Run the query
        self.runQuery(byIsActiveYN: isActiveYN, relativeMemberID:
         relativeMemberID, playGameID: playGameID, collection: collection,
         oncomplete: runQueryCompletionHandler)

    }

}
```