iLift

A solution for gyms and sport centers for a better interaction with their users

Documentation

Table of Contents

1. Introduction
2. System specification
   1. System description
   2. Main architecture
   3. Behavior between components of the system
   4. Database specification
3. Deployment specification
   1. Deploy Server-Side
   2. Deploy Client-Side
4. Introduction

We have implemented a solution which aims to resolve problems regarding gym administration such that both the end-user, which is the gym user and the gym administrator can spare time and organize better their schedule on the gym, respectively to make their gym registers easier to use and understand.

Many gyms or sport clubs nowadays are trying to find solutions in order to consume less time and energy on administrating their schedules and to have a clear vision on how used is their gym, but also to have a payment roll so that the administrator would know who needs to pay in order to use their gym. We have compared some gym websites which have the same purpose and we found out that most of them are hard to use and generally they need extra computer skills in order to put them to full use. We though it may be a good idea to make an improvement and make all the process of scheduling courses, handle full training rooms and paying operations much easier, while on the same time much faster than just transforming the job of writing whole pages of gym registers every day into a digitalized job, where the administrator would have to work the same amount and have the same result as keeping paper records.

For example, we have made the process of adding a weekly schedule as simple as selecting a tick. We found out that, statistically, gym administrators would want to schedule a course for a long period of time weekly, twice a month or monthly. So, we added this feature, while keeping the old fashioned way (e.g. add a schedule only for this week), just by selecting the interval on which the administrator would want to schedule to the current week.

Also, we estimated that normally, an end-user would want to do the following operations most often: make an appointment to a course, get a subscription to the gym and verify if the schedule has changed. We managed to do these operations as simple as possible, such that an end-user would not have to do more than 3-4 clicks (excluding the login if he is not already logged in) in order to make one of the operations described above.

For making a subscription to the gym, the end-user is required to pay at the gym. However, in the future, this feature can be extended so that a gym user can pay directly online via secure payment to subscribe to the gym. For this project, we have decided that the user should pay at the gym for the subscriptions/course packages he wants, and the administrator can accept the payment on the website immediately.

In the following technical documentation, we will present the system architecture constructed in order to make everything work, as well as the behavior between components, the database and how the system can be deployed to a gym which accedes in using it for more efficient administration.

1. System Specification

2.a. System description

The system is composed of the front-end component, which is implemented in React, and of the back-end component, implemented in PHP. For data persistence, we have used a MySQL database.

There are two roles which can be used to access the system, the *“user”* role and the *“administrator”* role. In order to access user/administrator features, one must be logged in with a corresponding account which has this role in the database. We have used JWT tokens encrypted with a 256bit key, such that an user cannot access administrator features. For this purpose, on every call on an interface on back-end, if this interface includes admin features, firstly the token is verified so that it will match the role found in the database for the given user. The JWT token is also used in some cases for identification, so that we wouldn’t have to store a *“user\_id”* in the front-end, which would also be a security hazard, as one could identify himself to the system as someone else. The token is validated for this purpose on every single call that involves user/administrator features.

There are some features that are available even if one is not logged in, thus requiring no token. Those calls are not validated on the back-end for a valid token, and are generally very simple operations, for example “get all courses”, “get all trainers”, “see feedbacks for a course” and “see feedbacks for a trainer”.

For security reasons, we also stored hashed passwords in the database. In case of a database leak, there is little chance that someone will easily un-hash any of the passwords in the database. We have used the ‘ripemd160’ algorithm for hashing passwords.

For any operation made on the website, the front-end component needs to call the back-end component, which in turn makes operations on the database, depending on the request. The server responds with a JSON, for example for a call on the server interface */get\_all\_courses.php* would return a JSON of the form:

*{"answer":"Success","courses":[{"id":2,"name":"Boxing Course","description":"Do you want to learn box? This is a great opportunity! ","url\_photo":"..\/uploads\/phpE817.png","number\_of\_feedbacks":9},{"id":3,"name":"Cycling course ","description":"Cycling can be very fun! Come to us! ","url\_photo":"","number\_of\_feedbacks":0}]}*

For communication between front-end and back-end we have used a strict protocol: every JSON response of the server must have an “answer” field, which returns the answer to the server call. This can be “Success”, “Error” or “Warning”. For “Error” and “Warning” answers, the server is guaranteed to include in the response JSON a “reason” field, which describes the exact motive for which that answer was given. The possible reasons will be described in the behavior between components of the system chapter of this documentation. For “Success” answers, the JSON will contain the result of the operation, it might be a list, an element or a descriptive message, depending on the back-end interface called by the front-end. It is worth noting that all calls to the server must be POST calls.

2.b. Main Architecture

We have modelled an architecture diagram, which describes how the system is linked together. Firstly, we will show it in Figure A, then we will explain it.

Figure A – Architecture Diagram

