

# Progetto di sistemi aperti e distribuiti

Università degli studi di Perugia a.a. 2020/2021

*Prof. Sergio Tasso*



*Progettazione Web Service:*

## SISTEMA DI PRENOTAZIONI PALESTRA



**FITNESS CLUB**

***Cristian Cosci***

310898

***Fabrizio Fagiolo***

310818

# Indice

## 1. Introduzione

- 1.1. *Descrizione del progetto*

## 2. Analisi dei requisiti

- 2.1. *Committente*
- 2.2. *Analista*

## 3. Tecnologie utilizzate

- 3.1. *Back-end*
- 3.2. *Front-end*

## 4. Implementazione

- 4.1. *Descrizione Database per l'archiviazione dei dati*
- 4.2. *Back-end*
- 4.3. *Front-end*

## 5. Esempio di utilizzo

# 1.Introduzione

## 1.1 Descrizione del progetto

Ci è stata affidata la progettazione e l'implementazione di un Web Service che offra servizi di prenotazione per una nota catena di palestre. In particolare è richiesto che sia predisposto un servizio che consenta ai clienti, iscritti con un abbonamento, di prenotare una sessione di allenamento direttamente da casa e in completa autonomia. Il cliente avrà quindi possibilità di visionare tra le varie sedi della catena di palestre, di selezionarne una e vedere i possibili intervalli d'orario in cui allenarsi e prenotarsi in uno di quelli disponibili.

Si necessita anche di una finestra di registrazione in cui i clienti possano creare un account per interagire con il servizio in modo immediato all'accesso successivo effettuato mediante login.

Inoltre è richiesta l'implementazione di un'interfaccia per le sedi delle palestre, mediante la quale i proprietari possono registrare la propria sede ed inserire i relativi dati di intervalli orari per sessione di allenamento.

La piattaforma è stata studiata in modo tale da fornire il tracciamento dei possibili contagi di covid-19. Infatti il web service fornisce la capacità ai clienti iscritti alla piattaforma di segnalare la propria positività o un contatto con un caso di covid-19. In questo modo le palestre in cui il cliente si è allenato hanno possibilità di contattare gli altri clienti che possono esservi stati in contatto e operare le adeguate misure di quarantena ed eventualmente tamponi.

## 2.ANALISI DEI REQUISITI

### 2.1 Committente

Sono qui riportati i requisiti richiesti in fase di analisi da parte del committente del servizio:

- Sistema di registrazione degli utenti
- Sistema di login degli utenti (già registrati alla piattaforma)
- Sistema di registrazione per le sedi delle palestre
- Sistema di gestione degli abbonamenti da parte dei proprietari delle palestre
- Sistema di prenotazione per i clienti con lista delle palestre disponibili
- Sistema di gestione e visione delle segnalazioni

### 2.1 Analista

Sono qui riportati i requisiti necessari per il servizio richiesto rispecchiando il ruolo di analista:

- Interfaccia principale del Web Service in cui si avrà distinzione tra Fruitore e Fornitore del servizio
- Area di registrazione per il cliente
- Area di login per il cliente
- Area di registrazione per la sede della palestra
- Area di login per la sede della palestra
- Interfaccia di prenotazione per il cliente. In cui sarà possibile:
  - *Prenotare una sessione*
  - *Vedere le palestre disponibili e relativi orari*
- Interfaccia di controllo dei dati relativi alle palestre.
- Interfaccia di controllo dei dati relativi ai clienti registrati.
- Interfaccia per effettuare segnalazioni lato cliente
- Interfaccia di gestione delle segnalazioni lato palestra
- Implementazione di un database relazionale con le rispettive tabelle per gestire:
  - *Clienti registrati*
  - *Sedi palestre registrate*
  - *Prenotazioni*
  - *Segnalazioni*

## 3. TECNOLOGIE UTILIZZATE

### 3.1 BACK-END

Per implementare il Web Service si è deciso di operare secondo lo schema architetturale di tipo **REST (REpresentational State Transfer)**. Quest'ultimo infatti è uno stile di architettura per la progettazione di sistemi hypermedia distribuiti, molto comune nella progettazione e sviluppo di Web Services. Sono così stati seguiti i 6 vincoli che REST presenta con lo scopo di fungere da "guida" per la progettazione dell'architettura. Infatti un'interfaccia deve soddisfare tutti e 6 i vincoli per essere definita di tipo RESTful.

**1. Client-server**

**2. Stateless**

**3. Cacheable**

**4. Uniform Interface**

**5. Layered System**

**6. Code on demand**

Ci siamo quindi basati nel definire una struttura che implementasse le principali operazioni CRUD mediante la gestione delle varie richieste HTTP ricevute.



Il server dovrà anche essere in grado di dare risposta di avvenuta gestione della richiesta ricevuta, per fare ciò sono stati utilizzati dei messaggi di risposta di tipo JSON, ma anche i semplici e fondamentali codici di risposta:

## Codici Comuni

- 200 (OK) : Operazione eseguita con successo;
- 201 (Created): Successo con conseguente creazione di una nuova risorsa del servizio (ad esempio in caso di inserimenti in POST di prodotti)
- 400 (Bad Request): La richiesta non è stata formulata correttamente
- 401 (Unauthorized): Deve essere eseguita correttamente l'autenticazione
- 403 (Forbidden): La richiesta è corretta, ma la risorsa richiesta è inaccessibile
- 404 (Not Found): La risorsa non è stata trovata
- 500 (Internal Server Error): Errore generico interno al server
- 501 (Not Implemented): Il tipo di richiesta non è stato implementato nel server

Per l'implementazione del server si necessitava di un linguaggio di programmazione che permettesse la creazione di sistema in grado di intercettare le varie richieste http ricevute e agire di conseguenza, eseguendo le operazioni richieste e fornire una risposta al client. Si è così scelto di adoperare **Node.js**



Node.js è un ambiente di runtime JavaScript open source, multiplatforma e back-end che viene eseguito sul motore V8 ed esegue il codice JavaScript all'esterno di un browser Web. Node.js consente agli sviluppatori di utilizzare JavaScript per scrivere strumenti da riga di comando e per lo scripting lato server. Da qui parte l'implementazione del server utilizzando varie librerie e framework, tra cui:



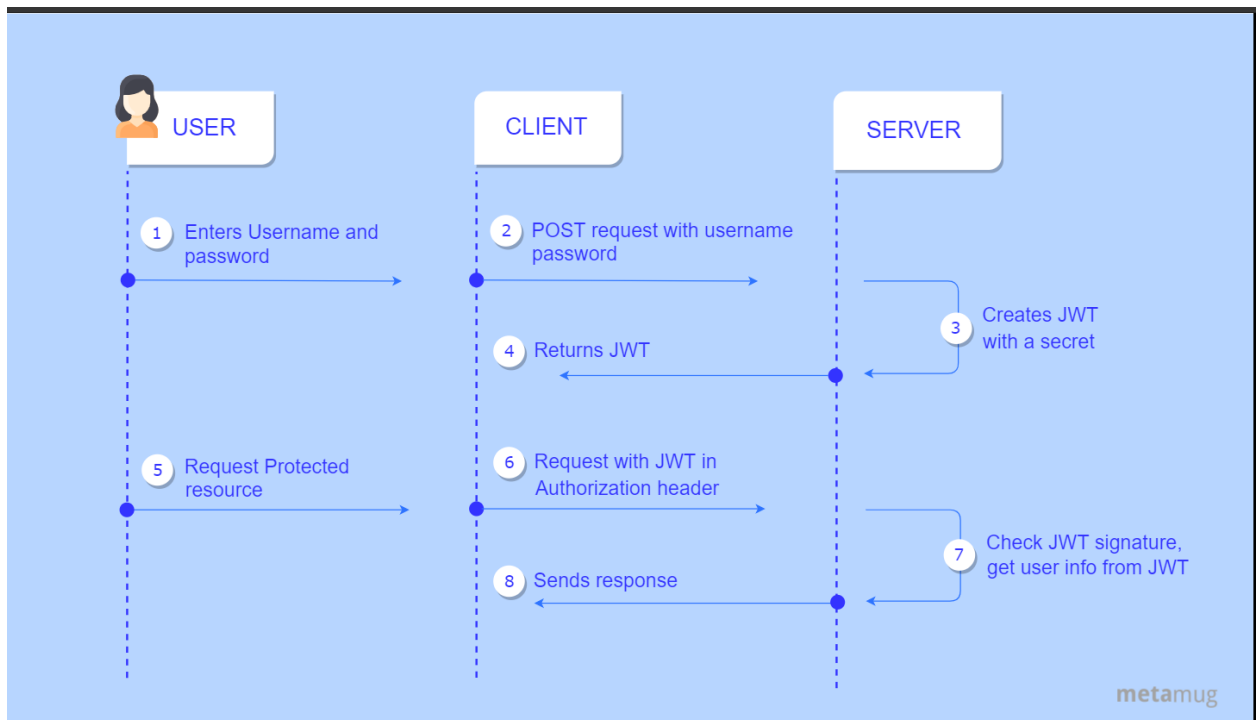
**Express.js, o semplicemente Express**, è un framework per applicazioni web per Node.js, open source sotto Licenza MIT. È stato progettato per creare web application e API ed ormai definito il server framework standard de facto per Node.js. Quest'ultimo infatti permetterà l'implementazione del servizio basato su API.

Per quanto riguarda il mantenimento dei dati di clienti, palestre, prenotazioni e segnalazioni si necessita di un database in grado di permetterci di conservare le varie informazioni. In combo ad express si utilizza quindi **MongoDb** il quale mediante la libreria **mongoose** permette una semplice ed efficace comunicazione tra il server e il database.



MongoDB è un programma di database orientato ai documenti multiplatforma. Classificato come un programma di database NoSQL, MongoDB utilizza documenti simili a JSON con schemi opzionali.

Ovviamente è necessario anche un sistema di autenticazione in grado di controllare che l'utente sia chi dichiara di essere e che le varie richieste siano a lui permesse. Per implementare questa funzionalità si è scelto di utilizzare il sistema di **JWT (Json Web Token)**. JSON Web Token è uno standard Internet proposto per la creazione di dati con firma opzionale e / o crittografia opzionale il cui payload contiene JSON che afferma un certo numero di attestazioni. I token vengono firmati utilizzando un segreto privato o una chiave pubblica / privata. Il funzionamento è così spiegato:



Per ultimo è necessario definire il modello in cui le risposte sono fornite dal server, ed in questo caso si è scelto ovviamente di sfruttare un sistema basato su JSON per il trasferimento di dati. **JSON, acronimo di JavaScript Object Notation**, è un formato adatto all'interscambio di dati fra applicazioni client/server.

## 3.2 FRONT-END

Per lo sviluppo del client sono state usate le seguenti tecnologie:

- HTML5
- CSS3
- JQUERY
- COOKIE

### HTML5

L'HTML5 è un linguaggio di markup per la strutturazione delle pagine web, pubblicato come W3C Recommendation da ottobre 2014.



HTML5 all'interno del progetto è stato utilizzato per comporre "l'ossatura" del nostro front-end.

### CSS3

Il CSS (sigla di Cascading Style Sheets, in italiano fogli di stile a cascata) è un linguaggio usato per definire la formattazione di documenti HTML, XHTML e XML, ad esempio i siti web e relative pagine web.



### JQUERY

jQuery è una **libreria JavaScript per applicazioni web**, distribuita come software libero, distribuito sotto i termini della Licenza MIT. Nasce con l'obiettivo di semplificare la selezione, la manipolazione, la gestione degli eventi e l'animazione di elementi DOM in pagine HTML, nonché semplificare l'uso di funzionalità AJAX, la gestione degli eventi e la manipolazione dei CSS.

Le sue caratteristiche permettono agli sviluppatori JavaScript di astrarre le interazioni a basso livello con i contenuti delle pagine HTML.



### CORE

**Il core di jQuery fornisce:**

I costruttori per l'utilizzo della libreria stessa:

- Per ottenere elementi tramite un selettore
- Per ottenere un elemento referenziandolo come parametro



- Per creare ex novo un elemento partendo da codice HTML grezzo

I metodi e le proprietà per accedere agli elementi contenuti in un oggetto jQuery:

- Per conoscere il numero di elementi (funzione **size()** oppure proprietà **length**)
- Per iterare ogni elemento (**funzione each()**)
- Per conoscere il selettore utilizzato o l'elemento DOM referenziato (proprietà **selector** o **context**)
- Per ottenere e manipolare elementi nativi (**funzioni get()** e **index()**)

I metodi per creare e utilizzare liste e code (di oggetti e funzioni)

I metodi per estendere il framework mediante plugin (**funzione extend()** e **fn.extend()**)

I metodi per eseguire animazioni mediante le funzioni **show()**, **hide()** e **animate()**

## EVENTI

Il framework riconosce gli oggetti di tipo event e provvede a modificare le loro proprietà rendendoli uniformi, semplificando la loro gestione, la loro propagazione, e fornendo un'utile modalità per impedire al browser di continuare l'esecuzione (ad esempio sulla **onclick** di un link). L'assegnazione di eventi quali **click**, **load**, **mouseover** è **gestita in maniera efficace e non invadente**.

## AJAX

La gestione delle chiamate asincrone (AJAX) è davvero semplificata, e sono fornite le funzioni:

- Per caricare contenuti dinamicamente
- Funzione di caricamento semplice
- Funzione di caricamento di codice HTML con inserimento automatico
- Per eseguire richieste asincrone (**con metodo GET/POST**)
- Per l'interazione con JavaScript
- Funzione per **caricare un oggetto JSON**
- Funzione per **caricare un file JavaScript remoto ed eseguirlo automaticamente**

Anche gli eventi AJAX sono gestibili in modo semplificato, per il completamento dei form di immissione, la gestione degli errori e l'invio dei dati.

## COOKIE

Vengono utilizzati dalle applicazioni web lato server per archiviare e recuperare informazioni a lungo termine sul lato client.



I server inviano i cookie nella risposta HTTP al client e ci si aspetta che i web browser salvino e inviino i cookie al server, ogni qual volta si facciano richieste aggiuntive al web server.

Tale riconoscimento permette di realizzare meccanismi di autenticazione usati ad esempio per i login; di memorizzare dati utili alla sessione di navigazione, come le preferenze sull'aspetto grafico o linguistico del sito; di tracciare la navigazione dell'utente, ad esempio per fini statistici o pubblicitari; di associare dati memorizzati dal server, ad esempio il contenuto del carrello di un negozio elettronico.

Le impostazioni possono essere personalizzate per abilitarli o bloccarli sempre, entro un determinato periodo di permanenza, per filtrare i siti in base a liste bianche e liste nere, e per filtrare cookie che sono utilizzati dallo stesso server o anche da collegamenti (spesso pubblicitari) a siti ospitati su server differenti.

È da notare che il funzionamento dei cookie è totalmente dipendente dal browser di navigazione che l'utente usa.

## 4.IMPLEMENTAZIONE

### 4.1 DESCRIZIONE DATABASE PER L'ARCHIVIAZIONE DEI DATI

Per quanto riguarda il database in mongoDB si opera secondo l'architettura a modelli. In cui ogni modello rappresenta una struttura dati che andrà poi a rappresentare un'entità che sarà mantenuta nel db sottoforma di schema (tabella). Qui di seguito verranno mostrati i vari modelli che raffigurano le strutture dati in gioco.

- **CLIENTE**

```
var clienteSchema = mongoose.Schema({
  firstName: {
    type: String,
    required: true
  },
  lastName: {
    type: String,
    required: true
  },
  email: String,
  password: String,
  permissionLevel: Number
});
```

- **PALESTRA**

```
mongoose = require('mongoose');

// Setup schema
var palestraSchema = mongoose.Schema({
  name: {
    type: String,
    required: true
  },
  address: {
    type: String,
    required: true
  },
  postal_code: String,
  email: String,
  password: String,
  orario_apertura: String,
  orario_chiusura: String,
  permissionLevel: Number
});
```

- **PRENOTAZIONE**

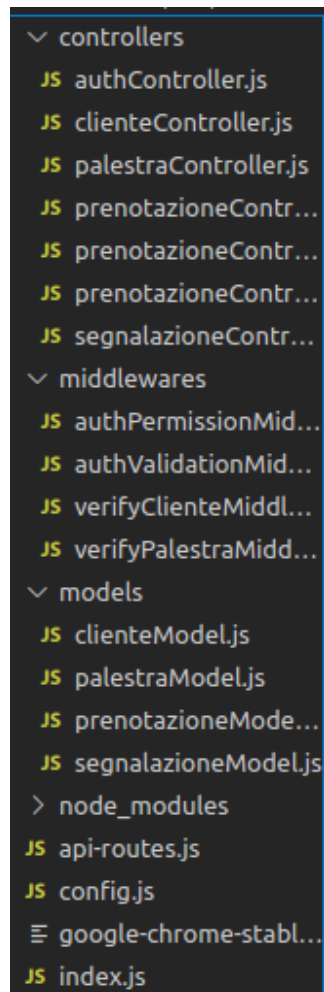
```
// Setup schema
var prenotazioneSchema = mongoose.Schema({
  id_cliente: {
    type: String,
    required: true
  },
  id_palestra: {
    type: String,
    required: true
  },
  name_palestra: {
    type: String,
  },
  phone: String,
  orario_prenotazione: {
    type: Date,
  }
});
```

- **SEGNALAZIONE**

```
var segnalazioneSchema = mongoose.Schema({
  id_cliente: {
    type: String
  }
});
```

## 4.2 BACK-END

La struttura del server sarà divisa in 3 componenti, i controllers (risolvono le richieste), middlewares (gestione dell'autenticazione), models (le strutture dati nel db).



Nell'implementare il web service si ha innanzitutto la necessita di definire le route con cui interagiranno i rispettivi client mediante le API effettuando le richieste di tipo HTTP. Ogni richiesta verrà poi distinta inizialmente dalla route a cui si fa riferimento e in un secondo passaggio in base al metodo HTTP utilizzato nella richiesta. Qui sotto si riportano delle route relative al server.

```

6 //login cliente authentication
7 router.route('/cliente/login')
8   .post(
9     VerifyUserMiddleware.hasAuthValidFields,
10    VerifyUserMiddleware.isPasswordAndUserMatch,
11    authController.login)
12
13 //refresh cliente authentication
14 router.route('/cliente/login/refresh')
15   .post(
16     AuthValidationMiddleware.validJWTNeeded,
17     AuthValidationMiddleware.verifyRefreshBodyField,
18     AuthValidationMiddleware.validRefreshNeeded,
19     authController.login)
20
21 // Define cliente Controller
22 var clienteController = require('./controllers/clienteController');
23
24 //routes cliente
25 router.route('/cliente/signup')
26   .post(clienteController.new)
27
28 router.route('/cliente')
29   .get(
30     AuthValidationMiddleware.validJWTNeeded,
31     PermissionMiddleware.minimumPermissionLevelRequired(FREE),
32     PermissionMiddleware.onlySameUserOrAdminCanDoThisAction,
33     clienteController.view)
34   .patch(
35     AuthValidationMiddleware.validJWTNeeded,
36     PermissionMiddleware.minimumPermissionLevelRequired(FREE),
37     PermissionMiddleware.onlySameUserOrAdminCanDoThisAction,
38     clienteController.patchById)
39   .put(
40     AuthValidationMiddleware.validJWTNeeded,
41     PermissionMiddleware.minimumPermissionLevelRequired(FREE),
42     PermissionMiddleware.onlySameUserOrAdminCanDoThisAction,
43     clienteController.patchById)
44
45 //route per la palestra
46 router.route('/palestra')
47   .get(
48     AuthValidationMiddleware.validJWTNeeded,
49     PermissionMiddleware.minimumPermissionLevelRequired(FREE),
50     PermissionMiddleware.onlySameUserOrAdminCanDoThisAction,
51     palestraController.view)
52   .patch(
53     AuthValidationMiddleware.validJWTNeeded,
54     PermissionMiddleware.minimumPermissionLevelRequired(FREE),
55     PermissionMiddleware.onlySameUserOrAdminCanDoThisAction,
56     palestraController.patchById)
57   .put(
58     AuthValidationMiddleware.validJWTNeeded,
59     PermissionMiddleware.minimumPermissionLevelRequired(FREE),
60     PermissionMiddleware.onlySameUserOrAdminCanDoThisAction,
61     palestraController.patchById)
62   .delete(
63     AuthValidationMiddleware.validJWTNeeded,
64     PermissionMiddleware.minimumPermissionLevelRequired(FREE),
65     palestraController.removeById);
66
67 // Define cliente Controller
68 var segnalazioneController = require('./controllers/segnalazioneController');
69
70 // route per la palestra
71 router.route('/segnalazioni')
72   .get(segnalazioneController.index)
73
74 //routes cliente segnalazioni
75 router.route('/cliente/segnalazione')
76   .get(
77     AuthValidationMiddleware.validJWTNeeded,
78     PermissionMiddleware.minimumPermissionLevelRequired(FREE),
79     PermissionMiddleware.onlySameUserOrAdminCanDoThisAction,
80     segnalazioneController.view)
81   .post(
82     AuthValidationMiddleware.validJWTNeeded,
83     PermissionMiddleware.minimumPermissionLevelRequired(FREE),
84     PermissionMiddleware.onlySameUserOrAdminCanDoThisAction,
85     segnalazioneController.new)
86   .delete(
87     AuthValidationMiddleware.validJWTNeeded,
88     PermissionMiddleware.minimumPermissionLevelRequired(FREE),
89     segnalazioneController.delete);
90
91 // route per palestra segnalazioni
92 router.route('/palestra/prenotazioni/segnalazione')
93   .post(
94     AuthValidationMiddleware.validJWTNeeded,
95     PermissionMiddleware.minimumPermissionLevelRequired(FREE),
96     PermissionMiddleware.onlySameUserOrAdminCanDoThisAction,
97     prenotazioneControllerPalestra.segnalazione)
98
99 router.route('/cliente/prenotazioni')
100   .post(
101     AuthValidationMiddleware.validJWTNeeded,
102     PermissionMiddleware.minimumPermissionLevelRequired(FREE),
103     PermissionMiddleware.onlySameUserOrAdminCanDoThisAction,
104     prenotazioneControllerCliente.new)
105   .get(
106     AuthValidationMiddleware.validJWTNeeded,
107     PermissionMiddleware.minimumPermissionLevelRequired(FREE),
108     PermissionMiddleware.onlySameUserOrAdminCanDoThisAction,
109     prenotazioneControllerCliente.view)
110   .patch(
111     AuthValidationMiddleware.validJWTNeeded,
112     PermissionMiddleware.minimumPermissionLevelRequired(FREE),
113     PermissionMiddleware.onlySameUserOrAdminCanDoThisAction,
114     prenotazioneControllerCliente.update)
115   .put(
116     AuthValidationMiddleware.validJWTNeeded,
117     PermissionMiddleware.minimumPermissionLevelRequired(FREE),
118     PermissionMiddleware.onlySameUserOrAdminCanDoThisAction,
119     prenotazioneControllerCliente.update)
120   .delete(
121     AuthValidationMiddleware.validJWTNeeded,
122     PermissionMiddleware.minimumPermissionLevelRequired(FREE),
123     PermissionMiddleware.onlySameUserOrAdminCanDoThisAction,
124     prenotazioneControllerCliente.delete);

```

```

1 //route per la palestra
2 router.route('/palestra')
3   .get(
4     AuthValidationMiddleware.validJWTNeeded,
5     PermissionMiddleware.minimumPermissionLevelRequired(FREE),
6     PermissionMiddleware.onlySameUserOrAdminCanDoThisAction,
7     palestraController.view)
8   .patch(
9     AuthValidationMiddleware.validJWTNeeded,
10    PermissionMiddleware.minimumPermissionLevelRequired(FREE),
11    PermissionMiddleware.onlySameUserOrAdminCanDoThisAction,
12    palestraController.patchById)
13   .put(
14     AuthValidationMiddleware.validJWTNeeded,
15     PermissionMiddleware.minimumPermissionLevelRequired(FREE),
16     PermissionMiddleware.onlySameUserOrAdminCanDoThisAction,
17     palestraController.patchById)
18   .delete(
19     AuthValidationMiddleware.validJWTNeeded,
20     PermissionMiddleware.minimumPermissionLevelRequired(FREE),
21     palestraController.removeById);
22
23 // Define cliente Controller
24 var segnalazioneController = require('./controllers/segnalazioneController');
25
26 // route per la palestra
27 router.route('/segnalazioni')
28   .get(segnalazioneController.index)
29
30 //routes cliente segnalazioni
31 router.route('/cliente/segnalazione')
32   .get(
33     AuthValidationMiddleware.validJWTNeeded,
34     PermissionMiddleware.minimumPermissionLevelRequired(FREE),
35     PermissionMiddleware.onlySameUserOrAdminCanDoThisAction,
36     segnalazioneController.view)
37   .post(
38     AuthValidationMiddleware.validJWTNeeded,
39     PermissionMiddleware.minimumPermissionLevelRequired(FREE),
40     PermissionMiddleware.onlySameUserOrAdminCanDoThisAction,
41     segnalazioneController.new)
42   .delete(
43     AuthValidationMiddleware.validJWTNeeded,
44     PermissionMiddleware.minimumPermissionLevelRequired(FREE),
45     segnalazioneController.delete);
46
47 // route per palestra segnalazioni
48 router.route('/palestra/prenotazioni/segnalazione')
49   .post(
50     AuthValidationMiddleware.validJWTNeeded,
51     PermissionMiddleware.minimumPermissionLevelRequired(FREE),
52     PermissionMiddleware.onlySameUserOrAdminCanDoThisAction,
53     prenotazioneControllerPalestra.segnalazione)
54
55 router.route('/cliente/prenotazioni')
56   .post(
57     AuthValidationMiddleware.validJWTNeeded,
58     PermissionMiddleware.minimumPermissionLevelRequired(FREE),
59     PermissionMiddleware.onlySameUserOrAdminCanDoThisAction,
60     prenotazioneControllerCliente.new)
61   .get(
62     AuthValidationMiddleware.validJWTNeeded,
63     PermissionMiddleware.minimumPermissionLevelRequired(FREE),
64     PermissionMiddleware.onlySameUserOrAdminCanDoThisAction,
65     prenotazioneControllerCliente.view)
66   .patch(
67     AuthValidationMiddleware.validJWTNeeded,
68     PermissionMiddleware.minimumPermissionLevelRequired(FREE),
69     PermissionMiddleware.onlySameUserOrAdminCanDoThisAction,
70     prenotazioneControllerCliente.update)
71   .put(
72     AuthValidationMiddleware.validJWTNeeded,
73     PermissionMiddleware.minimumPermissionLevelRequired(FREE),
74     PermissionMiddleware.onlySameUserOrAdminCanDoThisAction,
75     prenotazioneControllerCliente.update)
76   .delete(
77     AuthValidationMiddleware.validJWTNeeded,
78     PermissionMiddleware.minimumPermissionLevelRequired(FREE),
79     PermissionMiddleware.onlySameUserOrAdminCanDoThisAction,
80     prenotazioneControllerCliente.delete);

```

In questa configurazione si può anche notare che prima di essere gestita dal relativo controller, ogni richiesta viene presa in carica dai middlewares incaricati di gestire l'autenticazione dell'utente. Solo dopo che il client risulta essere autenticato ha diritto all'esecuzione della richiesta da parte del server. Per confermare che il client sia autenticato si procede infatti al controllo della validità del JWT del client.

Qui sotto sono riportati delle parti di codice in cui si processa in prima fase il login del client e successivamente il controllo sul fatto che il client si sia effettivamente loggato prima di eseguire determinate operazioni.

```

exports.isPasswordAndUserMatch = (req, res, next) => {
  Cliente.findByEmail(req.body.email)
    .then((user)=>{
      if(!user[0]){
        res.status(404).send({});
      }else{
        let passwordFields = user[0].password.split('$');
        let salt = passwordFields[0];
        let hash = crypto.createHmac('sha512', salt).update(req.body.password).digest("base64");
        if (hash === passwordFields[1]) {
          req.body = {
            userId: user[0]._id,
            email: user[0].email,
            permissionLevel: user[0].permissionLevel,
            provider: 'email',
            name: user[0].firstName + ' ' + user[0].lastName,
          };
          return next();
        } else {
          return res.status(400).send({errors: ['Invalid e-mail or password']});
        }
      }
    })
};

```

```

exports.validJWTNeeded = (req, res, next) => {
  if (req.headers['authorization']) {
    try {
      let authorization = req.headers['authorization'].split(' ');
      if (authorization[0] !== 'Bearer') {
        return res.status(401).send();
      } else {
        req.jwt = jwt.verify(authorization[1], secret);
        return next();
      }
    } catch (err) {
      return res.status(403).send();
    }
  } else {
    return res.status(401).send();
  }
};

```

La parte fondamentale dell'implementazione di un web service sta quindi nei controller, i quali prendono in carico le richieste ed eseguono le operazioni per esaudirle. Come già detto si distinguerà tra i vari metodi HTTP. Qui sotto sono riportati degli esempi di come vengono gestite alcune richieste da parte dei client.

```

// Handle create cliente actions
exports.new = function (req, res) {
  let salt = crypto.randomBytes(16).toString('base64');
  let hash = crypto.createHmac('sha512', salt).update(req.body.password).digest("base64");
  req.body.password = salt + "$" + hash;
  req.body.permissionLevel = 1;
  Cliente.createUser(req.body)
    .then((result) => {
      res.status(201).send({id: result._id});
    });
};

exports.getById = (req, res) => {
  Cliente.findById(req.body.userId)
    .then((result) => {
      res.status(200).send(result);
    });
};

exports.view = function (req, res) {
  Cliente.find({
    _id : req.headers.id
  }, function (err, cliente) {
    if (err)
      res.send(err);
    res.json({
      message: 'Cliente details loading..',
      data: cliente
    });
  });
};

```

```

exports.patchById = function (req, res) {
  Cliente.findOne({
    _id : req.headers.id
  }, function (err, cliente) {
    if (err)
      res.send(err);

    if(req.body.firstName){
      cliente.firstName = req.body.firstName;
    }
    if(req.body.lastName){
      cliente.lastName = req.body.lastName;
    }
    if(req.body.email){
      cliente.email = req.body.email;
    }
    if(req.body.password){
      let salt = crypto.randomBytes(16).toString('base64');
      let hash = crypto.createHmac('sha512', salt).update(req.body.password).digest("base64");
      req.body.password = salt + "$" + hash;
      cliente.password = req.body.password;
    }
    // save the contact and check for errors
    cliente.save(function (err) {
      if (err)
        res.json(err);
      res.json({
        message: 'Cliente Info updated',
        data: cliente
      });
    });
  });
};

```

```

const jwtSecret = require('../config.js').jwt_secret, jwt = require('jsonwebtoken');
const crypto = require('crypto');
const uuid = require('uuid');

//if login credentials is correct it respond with jwt token
exports.login = (req, res) => {
  try {
    let refreshId = req.body.userId + jwtSecret;
    let salt = crypto.randomBytes(16).toString('base64');
    let hash = crypto.createHmac('sha512', salt).update(refreshId).digest("base64");
    req.body.refreshKey = salt;
    let token = jwt.sign(req.body, jwtSecret);
    let b = Buffer.from(hash);
    let refresh_token = b.toString('base64');
    res.status(201).send({accessToken: token, refreshToken: refresh_token, id : req.body.userId});
  } catch (err) {
    res.status(500).send({errors: err});
  }
};

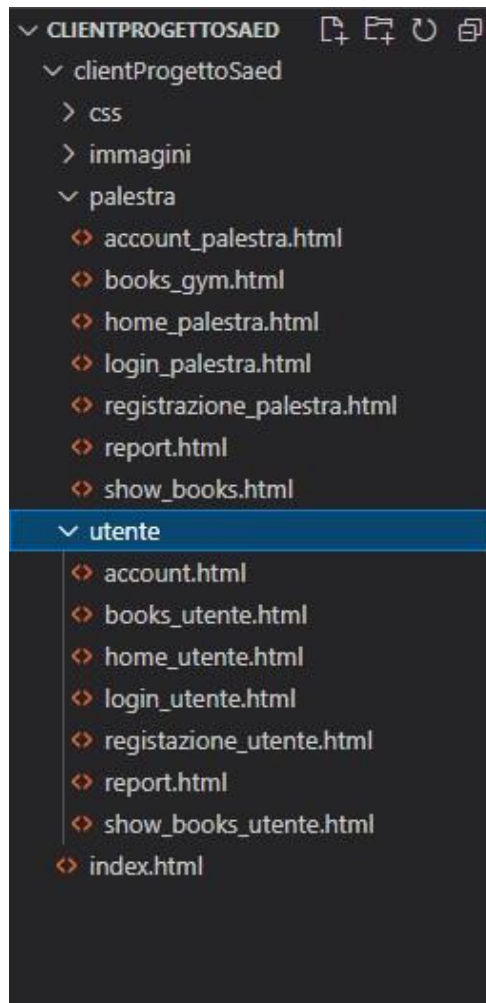
//refresh the jwt token
exports.refresh_token = (req, res) => {
  try {
    req.body = req.jwt;
    let token = jwt.sign(req.body, jwtSecret);
    res.status(201).send({id: token});
  } catch (err) {
    res.status(500).send({errors: err});
  }
};

```

## 4.3 FRONT-END

Per prima cosa nello sviluppo del lato client è stata introdotta una struttura gerarchica per distinguere appunto le 2 figure:





Ovviamente fuori dalle 2 figure abbiamo l'indice del nostro sito dove possiamo "navigare".

## LATO UTENTE

- **Registrazione Utente**

La prima cosa che andiamo a fare è registrare l'utente all'interno del web server, per fare questo andiamo a fare una richiesta POST alla route nella foto sottostante.

```

<script>
    $('#form').submit(function(e){
        e.preventDefault();
        $.ajax({
            url: 'http://localhost:8080/fitness_club/cliente/signup',
            method: 'POST',
            data: $(this).serialize(),
            success: function(reponse){
                console.log(reponse);
                window.location = "login_utente.html"
            },
            error: function(reponse){
                console.log(reponse);
            }
        })
    });
</script>

```

I dati all'interno della richiesta post vengono presi dalla form presente nel codice html seguente, e la richiesta viene attivata dopo la pressione del bottone.

L'eventuale successo/insuccesso è poi segnalato nella console, dove avremmo la risposta del server.

```

<form id="form">
    <br>
    <div class="form-floating">
        <input id="firstName" name="firstName" type="text" class="form-control" placeholder="FirstName" name="FirstName">
    </div>

    <br>
    <div class="form-floating">
        <input id="lastName" name="lastName" type="text" class="form-control" placeholder="LastName" name="LastName">
    </div>

    <br>
    <div class="form-floating">
        <input id="email" type="email" name="email" class="form-control" placeholder="name@example.com" name="email">
    </div>

    <br>
    <div class="form-floating">
        <input id="password" name="password" type="password" class="form-control" placeholder="password" name="password">
    </div>

    <br>
    <button class="w-100 btn btn-lg btn-primary" type="submit" id="butsave">Register USER</button>
</form>

```

Una volta effettuata la registrazione viene ricaricata la pagina e l'utente viene reindirizzato alla pagina di log-in.

## 🔍 Login Utente

Dopo aver registrato l'utente si va ad effettuare il log-in.

```

<script>
    $('#form').submit(function(e){
        e.preventDefault();
        $.ajax({
            url: 'http://localhost:8080/fitness_club/cliente/login',
            method: 'POST',
            data: $(this).serialize(),
            success: function(response){
                console.log(response.statusCode);
                localStorage.setItem('user', JSON.stringify(response));
                localStorage.setItem('id', JSON.stringify(response.id));
                localStorage.setItem('jwt', JSON.stringify(response.accessToken));
                console.log("user" + localStorage.getItem('user'));
                console.log("id " + localStorage.getItem('id'));
                console.log("jwt" + localStorage.getItem('jwt'));

                cname = 'id';
                cvalue = response.id;
                cname1 = 'token';
                ctoken = response.accessToken;
                document.cookie = cname + "=" + cvalue + "; ";
                document.cookie = cname1 + "=" + ctoken + "; ";
                window.location = "home_utente.html";
            },
            error: function(response){
                console.log(response);
                var text = '<h4 class="yellow"> Incorrect USERNAME or PASSWORD </h4>';
                var text2 = '<h4 class="yellow"> TRY AGAIN! </h4>';
                $(".mypanel2").html(text);
                $(".mypanel3").html(text2);
            }
        })
    });
</script>

```

Anche nel log-in andiamo ad effettuare una richiesta POST alla route che si vede nella foto precedente, la quale è presente sul server.

Oltre alla richiesta POST nello script si va a creare anche un cookie che si occupa di salvare la sessione di log-in di quel determinato utente, ovviamente una volta fatto il cookie viene cancellato.

Sul cookie vengono salvati l'id dell'utente e il codice di autenticazione, inoltre si va a controllare lo stato della risposta del server per verificare se, l'utente ha immesso le giuste credenziali.

Se tutto è andato a buon fine l'utente viene reindirizzato alla sua determinata home.

```

<h1 class="black">LOG IN USER</h1>

<main class="form-signin">
  <form id="form">
    <br>
    <div class="form-floating">
      <input id="email" type="email" name="email" class="form-control" placeholder="name@example.com">
    </div>

    <br>
    <div class="form-floating">
      <input id="password" type="password" name="password" class="form-control" placeholder="password">
    </div>
    <br>
    <p class="black"><b>ARE YOU READY?</b></p>
    <button class="w-100 btn btn-lg btn-primary" type="submit" role="button">Log in</button>
  </form>
  <div class="inner cover">
    <di class="mypanel2"></di>
    <di class="mypanel3"></di>
  </div>

```

I dati all'interno della richiesta post vengono presi dalla form presente nel codice html seguente, e la richiesta viene attivata dopo la pressione del bottone.

## 🔍 Home utente

Nella home dell'utente viene letto il cookie creato in precedenza e stampato in console.

```

<script>
function getCookie(cname) {
  let name = cname + "=";
  let decodedCookie = decodeURIComponent(document.cookie);
  let ca = decodedCookie.split(';');
  for(let i = 0; i <ca.length; i++) {
    let c = ca[i];
    while (c.charAt(0) == ' ') {
      c = c.substring(1);
    }
    if (c.indexOf(name) == 0) {
      return c.substring(name.length, c.length);
    }
  }
  return "";
}
let x = getCookie("token")
console.log(x);
</script>

```

L'utente ha poi 3 scelte che stanno su 3 bottoni, dove può :

- **Prenotare una sessione di allenamento in una data palestra**

Qui vengono fatte 2 richieste la prima per prenotare appunto la sessione ed è una richiesta post che si attiva come le altre al click del bottone. La seconda è una richiesta GET che stampa le relative palestre dove può andare ad allenarsi l'utente.

```

$('#form').submit(function(e){
e.preventDefault();
$.ajax({
type: 'POST',
url: 'http://localhost:8080/fitness_club/cliente/prenotazioni',
crossDomain: true,
dataType: 'json',
headers:
{
Authorization : "Bearer " + jwt,
id : id_utente
},
data: $(this).serialize(),
success: function(data){
$("#risposta").html("Prenotazione effettuata con successo!!!");
},
error:function(data){
console.log(data);
}
})
});
</script>
<script>
$.ajax({
type: 'GET',
url: 'http://localhost:8080/fitness_club/palestre',
crossDomain: true,
dataType: 'json',
success: function(data) {
var text = '<tr><th>ID</th><th>Nome palestra</th><th>Indirizzo</th><th>ORARIO APERTURA</th><th>ORARIO CHIUSURA</th></tr>'
for (let i = 0; i < data.length; i++) {
text = text + `
<tr><td>${data[i].id}</td>` +
`<td>${data[i].name}</td>` +
`<td>${data[i].address} ${data[i].postal_code}</td>` +
`<td>${data[i].orario_apertura}</td>` +
`<td>${data[i].orario_chiusura}</td>` +
`</tr>`
}
}
});
</script>

```

Anche qui per mantenere la sessione attiva si utilizzano i cookie che andremo a leggere con la funzione vista in precedenza.

Anche qua i dati per la richiesta post vengono presi dalla form nel codice html

```

<form method="POST" id="form">

<br>
<label>ID Palestra</label>
<div class="form-floating">
| <input id="id_palestra" name = "id_palestra" type="text" class="form-control">
</div>
<br>

<label>Palestra</label>
<div class="form-floating">
| <input id="name_palestra" name = "name_palestra" type="text" class="form-control">
</div>
<br>

<label>Phone</label>
<div class="form-floating">
| <input id="phone" name="phone" type="text" class="form-control">
</div>

<br>
| <label>Orario</label>
<div class="form-floating">
| <input id="orario_prenotazione" name="orario_prenotazione" type="text" placeholder="AAAA-MM-DD HH-MM-SS" class="form-control">
</div>

<br>
<br>
<button class="w-100 btn btn-lg btn-primary" type="submit" id="butsave">BOOK NOW!</button>
</form>

```

- **Segnalare un possibile contatto con un caso positivo in anonimato**

Anche qua viene fatta una richiesta POST alla determinata route del server.

```
$('#form').submit(function(e){
  e.preventDefault();
  $.ajax({
    type: 'POST',
    url: 'http://localhost:8080/fitness_club/cliente/segnalazione',
    crossDomain: true,
    dataType: 'json',
    headers:
    {
      Authorization : "Bearer " + jwt,
      id : id_utente
    },
    success: function(data) {
      $(".mypanelC").html("Segnalazione effettuata con successo e presa in carico dalla palestra");
    },
    error: function(data){
      console.log(error);
    }
  })
});
```

Una volta effettuata la richiesta post viene stampato un messaggio che segnala il successo della richiesta.

- **Vedere tutte le prenotazioni che ha fatto quell'utente**

L'ultima operazione che può effettuare l'utente è appunto andare a vedere tutte le prenotazioni che ha fatto. Per fare questo si fa una richiesta GET e si stampa il risultato.

```
$.ajax({
  type: 'GET',
  url: 'http://localhost:8080/fitness_club/cliente/prenotazioni',
  crossDomain: true,
  dataType: 'json',
  headers:
  {
    Authorization : "Bearer " + jwt,
    id : id_utente
  },
  success: function(data) {
    var text = '<tr><th>Palestra</th><th>DATA E ORA</th><th>N. PRENOTAZIONE</th></tr>'
    for (let i = 0; i < data.data.length; i++) {
      var text = text + `
      <tr><td>${data.data[i].name_palestra}</td>` +
      `<td>${data.data[i].orario_prenotazione}</td>` +
      `<td>${data.data[i]._id}</td></tr>`
    }
    $(".mypanelC").html(text);
  },
  error: function (data) {
    alert('GET failed.');
```

- **Account**



Oltre a queste operazioni l'utente può andare a modificare i propri dati di accesso o a eliminare il suo account.

Per fare ciò deve andare sulle impostazioni del proprio account dove è presente un bottone sulla nav bar.

Qua vengono fatte 3 richieste, una richiesta GET che va a stampare i suoi dati attuali.

```
102 $.ajax({
103   type: 'GET',
104   url: 'http://localhost:8080/fitness_club/cliente',
105   crossDomain: true,
106   dataType: 'json',
107   headers:
108   {
109     Authorization : "Bearer " + jwt,
110     id : id_utente
111   },
112   success: function(data) {
113     var text = ''
114     for (let i = 0; i < data.data.length; i++) {
115       var text = text + `
116       <p class="etichette">NOME: <span class="dato">${data.data[i].firstName}</span></p>` +
117       `<p class="etichette">COGNOME: <span class="dato">${data.data[i].lastName}</span></p>` +
118       `<p class="etichette">EMAIL: <span class="dato">${data.data[i].email}</span></p>`
119     }
120     $(".dati").html(text);
121
122   },
123   error: function (data) {
124     alert('GET failed.');
```

Una richiesta put per andare appunto a modificare i propri dati.

```
128 $('#form').submit(function(e){
129   e.preventDefault();
130   $.ajax({
131     url: 'http://localhost:8080/fitness_club/cliente',
132     method: 'PUT',
133     crossDomain: true,
134     dataType: 'json',
135     headers:
136     {
137       Authorization : "Bearer " + jwt,
138       id : id_utente
139     },
140     data: $(this).serialize(),
141     success: function(data){
142       $(".modifica").html("Dati correttamente modificati");
143       window.setTimeout(function(){
144         // Move to a new location or you can do something else
145         window.location = "account.html";
146       }, 3000);
147     },
148     error:function(data){
149       console.log(data);
150     }
151   })
152 })
153 });
154
```

Anche qua i dati della richiesta vengono presi dal codice html dove sono presenti delle form addette a l'inserimento dei dati.

Infine si può effettuare una richiesta DELETE per andare a cancellare l'account.

```

$( '#form2' ).submit(function(e){
    e.preventDefault();
    $.ajax({
        url: 'http://localhost:8080/fitness_club/cliente',
        method: 'DELETE',
        crossDomain: true,
        dataType: 'json',
        headers: {
            Authorization : "Bearer " + jwt,
            id : id_utente
        },
        data: $(this).serialize(),
        success: function(data){
            $(".eliminazione").html("Account eliminato correttamente!");
            window.setTimeout(function(){
                // Move to a new location or you can do something else
                window.location = "../index.html";
            }, 3000);
        },
        error: function(data){
            console.log(data);
        }
    })
});
</script>

```

Le richieste PUT e DELETE vengono attivate al click di un bottone.

## LATO PALESTRA

- **Registrazione palestra**

La prima cosa che andiamo a fare è registrare la palestra all'interno del web server, per fare questo andiamo a fare una richiesta POST alla route nella foto sottostante.

```

<script>
    $( '#form' ).submit(function(e){
        e.preventDefault();
        $.ajax({
            url: 'http://localhost:8080/fitness_club/palestra/signup',
            method: 'POST',
            data: $(this).serialize(),
            success: function(response){
                window.location = "login_palestra.html"
            },
            error: function(response){
                console.log(response);
            }
        })
    });
</script>

```

I dati all'interno della richiesta post vengono presi dalla form presente nel codice html seguente, e la richiesta viene attivata dopo la pressione del bottone.

L'eventuale successo/insuccesso è poi segnalato nella console, dove avremmo la risposta del server.



```

<form method="POST" id="form">
  <br>
  <div class="form-floating">
    <input id="name" name="name" type="text" class="form-control" placeholder="name">
  </div>
  <div id="result">
  </div>

  <br>
  <div class="form-floating">
    <input id="address" name="address" type="text" class="form-control" placeholder="address">
  </div>

  <br>
  <div class="form-floating">
    <input id="postal_code" name="postal_code" type="text" class="form-control" placeholder="postal-code">
  </div>

  <br>
  <div class="form-floating">
    <input id="email" name="email" type="email" class="form-control" placeholder="email">
  </div>

  <br>
  <div class="form-floating">
    <input id="password" name="password" type="password" class="form-control" placeholder="password">
  </div>

  <br>
  <label class="yellow"><b>ORARIO APERTURA</b></label>
  <div class="form-floating">
    <br>
    <input id="orario_apertura" name="orario_apertura" type="text" class="form-control">
  </div>

  <br>
  <label class="yellow"><b>ORARIO CHIUSURA</b></label>
  <div class="form-floating">
    <br>
    <input id="orario_chiusura" name="orario_chiusura" type="text" class="form-control">
  </div>

  <br>
  <button class="w-100 btn btn-lg btn-primary" type="submit" id="butsave">Register GYM</button>

</form>
</main>

```

Successivamente l'utente viene reindirizzato alla pagina per il log-in

## 🔗 Login Palestra

Dopo aver registrato la palestra si va ad effettuare il log-in.

```

46 <script>
47   $('#form').submit(function(e){
48     e.preventDefault();
49     $.ajax({
50       url: 'http://localhost:8080/fitness_club/palestra/login',
51       method: 'POST',
52       data: $(this).serialize(),
53       success: function(response){
54         console.log(response);
55         console.log(response.statusCode);
56         localStorage.setItem('user', JSON.stringify(response));
57         localStorage.setItem('id', JSON.stringify(response.id));
58         localStorage.setItem('jwt', JSON.stringify(response.accessToken));
59         console.log("user" + localStorage.getItem('user'));
60         console.log("id " + localStorage.getItem('id'));
61         console.log("jwt" + localStorage.getItem('jwt'));
62
63
64         cname = 'id';
65         cvalue = response.id;
66         cname1 = 'token'
67         ctoken = response.accessToken;
68         document.cookie = cname + "=" + cvalue + "; ";
69         document.cookie = cname1 + "=" + ctoken + "; ";
70
71         window.location = 'home_palestra.html';
72       },
73       error: function(response){
74         console.log(response);
75         var text = '<br><br><br><br><h4 class="yellow"> Incorrect USERNAME or PASSWORD </h4>';
76         var text2 = '<h4 class="yellow"> TRY AGAIN! </h4>';
77         $(".mypanel12").html(text);
78         $(".mypanel13").html(text2);
79       }
80     })
81   });
82 </script>

```

Anche nel log-in andiamo ad effettuare una richiesta POST alla route che si vede nella foto precedente, la quale è presente sul server.

Oltre alla richiesta POST nello script si va a creare anche un cookie che si occupa di salvare la sessione di log-in di quella determinata palestra, ovviamente una volta fatto il log-out, il cookie viene cancellato.

Sul cookie vengono salvati l'id della palestra e il codice di autenticazione, inoltre si va a controllare lo stato della risposta del server per verificare se l'utente ha immesso le giuste credenziali.

Se tutto è andato a buon fine si viene reindirizzati alla sua determinata home.

```
<h1 class="black"><b>LOG IN GYM</b></h1>
<main class="form-signin">
  <form id="form">
    <br>
    <div class="form-floating">
      <input id="email" type="email" name="email" class="form-control" placeholder="name@example.com">
    </div>

    <br>
    <div class="form-floating">
      <input id="password" type="password" name="password" class="form-control" placeholder="password">
    </div>
    <br>
    <br>
    <button class="w-100 btn btn-lg btn-primary" type="submit">Log in</button>
  </form>
</main>
<div class="mypanel2"></div>
<div class="mypanel3"></div>
```

I dati all'interno della richiesta post vengono presi dalla form presente nel codice html seguente, e la richiesta viene attivata dopo la pressione del bottone.

## Home Palestra

Nella home della palestra viene letto il cookie creato in precedenza e stampato in console.

Con la stessa funzione usata anche nel lato utente, andando solamente a cambiare l'id della palestra.

L'utente proprietario della palestra ha poi 3 scelte che stanno su 3 bottoni, dove puo':

- **Prenotare una sessione di allenamento per un cliente**

Qui viene fatta una richiesta post che si attiva come le altre al click del bottone.

```

$('#form').submit(function(e){
  e.preventDefault();
  $.ajax({
    type: 'POST',
    url: 'http://localhost:8080/fitness_club/palestra/prenotazioni',
    crossDomain: true,
    dataType: 'json',
    headers: {
      {
        Authorization : "Bearer " + jwt,
        id : id_utente
      },
    },
    data: $(this).serialize(),
    success: function(data){
      console.log(data)
    },
    error: function(data){
      console.log(data);
    }
  })
});
</script>

```

Anche in questo caso i parametri della richiesta vengono presi in input dalla form nel codice html.

```

<form method="POST" id="form">

  <label class="nero"><b>Cliente ID</b></label>
  <div class="form-floating">
    <br>
    <input id="id_cliente" name = "id_cliente" type="text" class="form-control">
  </div>
  <br>
  <label class="nero"><b>Phone</b></label>
  <div class="form-floating">
    <br>
    <input id="phone" name="phone" type="text" class="form-control">
  </div>
  <br>
  <label class="nero"><b>Orario</b></label>
  <div class="form-floating">
    <br>
    <input id="orario_prenotazione" name="orario_prenotazione" type="text" class="form-control">
  </div>

  <br>
  <br>
  <button class="w-100 btn btn-lg btn-primary" type="submit" id="butsave">BOOK NOW!</button>
</form>

```

- **Visualizzare tutte le segnalazioni degli utenti in quella data palestra**

Qui vengono fatte 2 richieste. La prima è una richiesta GET che stampa tutte le relative segnalazioni degli utenti nella palestra. La seconda è una POST dove si vanno a visualizzare tutti i dati dell'utente.

```
$.ajax({
  type: 'GET',
  url: 'http://localhost:8080/fitness_club/segnalazioni',
  crossDomain: true,
  dataType: 'json',
  headers:
  {
    Authorization : "Bearer " + jwt,
    id : id_utente
  },
  success: function(data) {
    var text = '<tr><th>ID SEGNALAZIONE</th><th>ID CLIENTE</th></tr>'
    for (let i = 0; i < data.data.length; i++) {
      text = text + `
      <tr><td>${data.data[i]._id}</td>` +
      `<td>${data.data[i].id_cliente}</td>`
    }
    $(".segnalazioni").html(text);
  },
  error: function (data) {
    alert('GET failed.');
```

La GET si attiva con l'apertura della pagina mentre la POST si attiva al click del bottone.

```
$('#form').submit(function(e){
  e.preventDefault();
  $.ajax({
    url: 'http://localhost:8080/fitness_club/palestra/prenotazioni/segnalazione',
    method: 'POST',
    data: $(this).serialize(),
    headers:
    {
      Authorization : "Bearer " + jwt,
      id : id_utente
    },
    success: function(data){
      console.log(data);
      console.log(data.data);
      var text = '<tr><th>ID SEGNALAZIONE</th><th>ID CLIENTE</th><th>PALESTRA</th><th>ORARIO PRENOTAZIONE</th><th>PHONE</th></tr>'
      for (let i = 0; i < data.data.length; i++) {
        var text = text + `
        <tr><td>${data.data[i]._id}</td>` +
        `<td>${data.data[i].id_cliente}</td>` +
        `<td>${data.data[i].name_palestra}</td>` +
        `<td>${data.data[i].orario_prenotazione}</td>` +
        `<td>${data.data[i].phone}</td>` +
        `</tr>`
      }
      $(".mypanelC").html(text);
    },
    error: function(data){
      console.log(data);
    }
  });
});
</script>
```

Anche qua i dati relativi alla POST vengono presi in input dalla form nel codice html.

```

<html lang="en">
<head>
  <title>REPORT GYM</title>
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <!-- Custom styles for this template -->
  <link href="../css/cover.css" rel="stylesheet">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
</head>
<nav class="home">
  <a href="home_palestra.html">Home</a>
</nav>
<body>
  <h2 class="titoloCovid">POSSIBLE CASES OF COVID19</h2>
  <br>
  
  <br>
  <br>
  <div>
    <table class="segnalazioni">
    </table>
    <br>
    <br>
  </div>
  <div>
    <table class="mypanelC">
    </table>
    <br>
  </div>
  <form id="form">
    <input id="id_cliente" name="id_cliente" type="text" placeholder="Search">
    <button class="w-100 btn btn-lg btn-primary" type="submit">Search</button>
  </form>
</body>
</script>

```

- **Visualizzare tutte le prenotazioni degli utenti in quella palestra**

L'ultima operazione che può effettuare il gestore della palestra è appunto andare a vedere tutte le prenotazioni che hanno fatto gli utenti. Per fare questo si fa una richiesta GET e si stampa il risultato.

```

$.ajax({
  type: 'GET',
  url: 'http://localhost:8080/fitness_club/palestra/prenotazioni',
  crossDomain: true,
  dataType: 'json',
  headers:
  {
    Authorization : "Bearer " + jwt,
    id : id_utente
  },
  success: function(data) {
    var text = '<tr><th>CLIENTE</th><th>DATA E ORA</th><th>PHONE</th><th>N. PRENOTAZIONE</th></tr>'
    for (let i = 0; i < data.data.length; i++) {
      text = text + `
      <tr><td>${data.data[i].id_cliente}</td>` +
      `<td>${data.data[i].orario_prenotazione}</td>` +
      `<td>${data.data[i].phone}</td>` +
      `<td>${data.data[i].id}</td></tr>`
    }
    $(".mypanelC").html(text);
  },
  error: function (data) {
    alert('GET failed.');
```

- **Account**

Oltre a queste operazioni il gestore della palestra può andare a modificare i propri dati di accesso o a eliminare il suo account.

Per fare ciò deve andare sulle impostazioni del proprio account dove è presente un bottone sulla nav bar.

Qua vengono fatte 3 richieste, una richiesta GET che va a stampare i suoi dati attuali.

```
$.ajax({
  type: 'GET',
  url: 'http://localhost:8080/fitness_club/palestra',
  crossDomain: true,
  dataType: 'json',
  headers:
  {
    Authorization : "Bearer " + jwt,
    id : id_utente
  },
  success: function(data) {
    console.log(data);
    var text = ''
    for (let i = 0; i < data.data.length; i++) {
      var text = text + `
      <p class="etichette">NOME: <span class="dato">${data.data[i].name}</span></p>` +
      `<p class="etichette">ADDRESS: <span class="dato">${data.data[i].address}</span></p>` +
      `<p class="etichette">POSTAL CODE: <span class="dato">${data.data[i].postal_code}</span></p>` +
      `<p class="etichette">EMAIL: <span class="dato">${data.data[i].email}</span></p>` +
      `<p class="etichette">ORARIO APERTURA: <span class="dato">${data.data[i].orario_apertura}</span></p>` +
      `<p class="etichette">ORARIO CHIUSURA: <span class="dato">${data.data[i].orario_chiusura}</span></p>`
    }
    $(".dati").html(text);
  },
});
```

Una richiesta put per andare appunto a modificare i propri dati.

```
$('#form').submit(function(e){
  e.preventDefault();
  $.ajax({
    url: 'http://localhost:8080/fitness_club/palestra',
    method: 'PUT',
    crossDomain: true,
    dataType: 'json',
    headers:
    {
      Authorization : "Bearer " + jwt,
      id : id_utente
    },
    data: $(this).serialize(),
    success: function(data){
      $(".modifica").html("Dati correttamente modificati");
      window.setTimeout(function(){
        // Move to a new location or you can do something else
        window.location = "account_palestra.html";
      }, 3000);
    },
    error: function(data){
      console.log(data);
    }
  })
});
```

Infine si può effettuare una richiesta DELETE per andare a cancellare l'account.

```

$('#form2').submit(function(e){
  e.preventDefault();
  $.ajax({
    url: 'http://localhost:8080/fitness_club/palestra',
    method: 'DELETE',
    crossDomain: true,
    dataType: 'json',
    headers:
    {
      Authorization : "Bearer " + jwt,
      id : id_utente
    },
    data: $(this).serialize(),
    success: function(data){
      $(".eliminazione").html("Account eliminato correttamente!");
      window.setTimeout(function(){
        // Move to a new location or you can do something else
        window.location = "../index.html";
      }, 3000);
    },
    error: function(data){
      console.log(data);
    }
  })
});

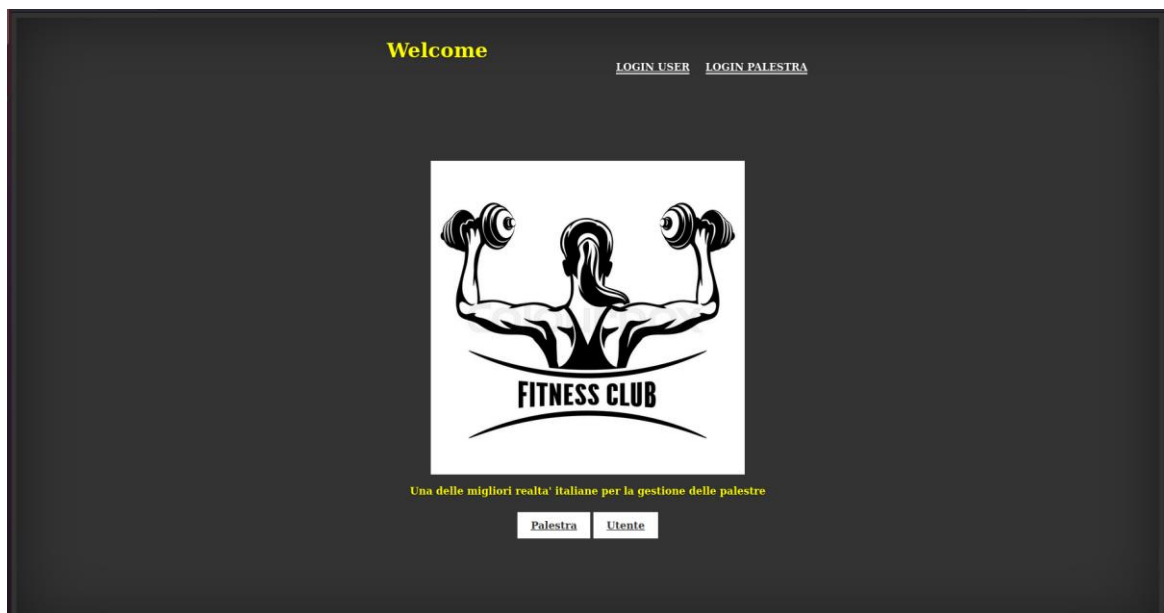
```

Le richieste PUT e DELETE vengono attivate al click di un bottone.

## 5.ESEMPIO DI UTILIZZO

Verranno qui riportate graficamente varie fasi di utilizzo del web service e del suo funzionamento.

### *HOMEPAGE PRINCIPALE*



Da qui l'utente può svolgere varie operazioni a seconda di cosa necessita di fare. Si può registrare una palestra, un cliente oppure loggarsi direttamente in uno dei due sistemi.

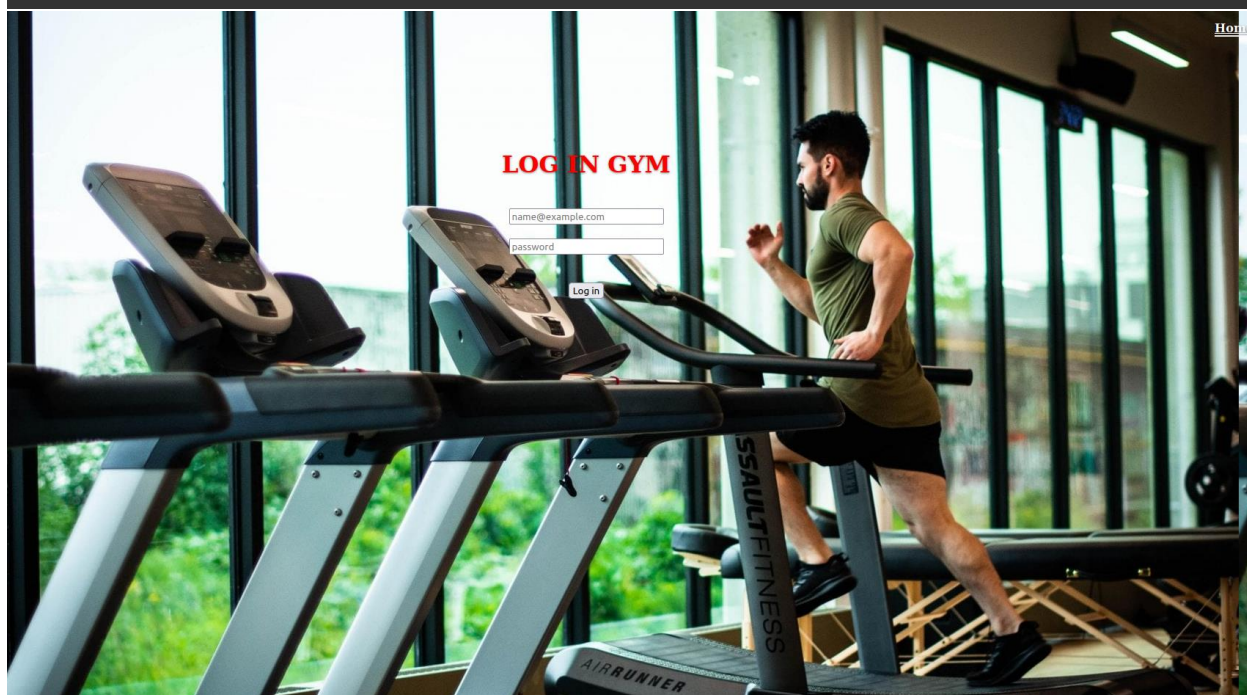
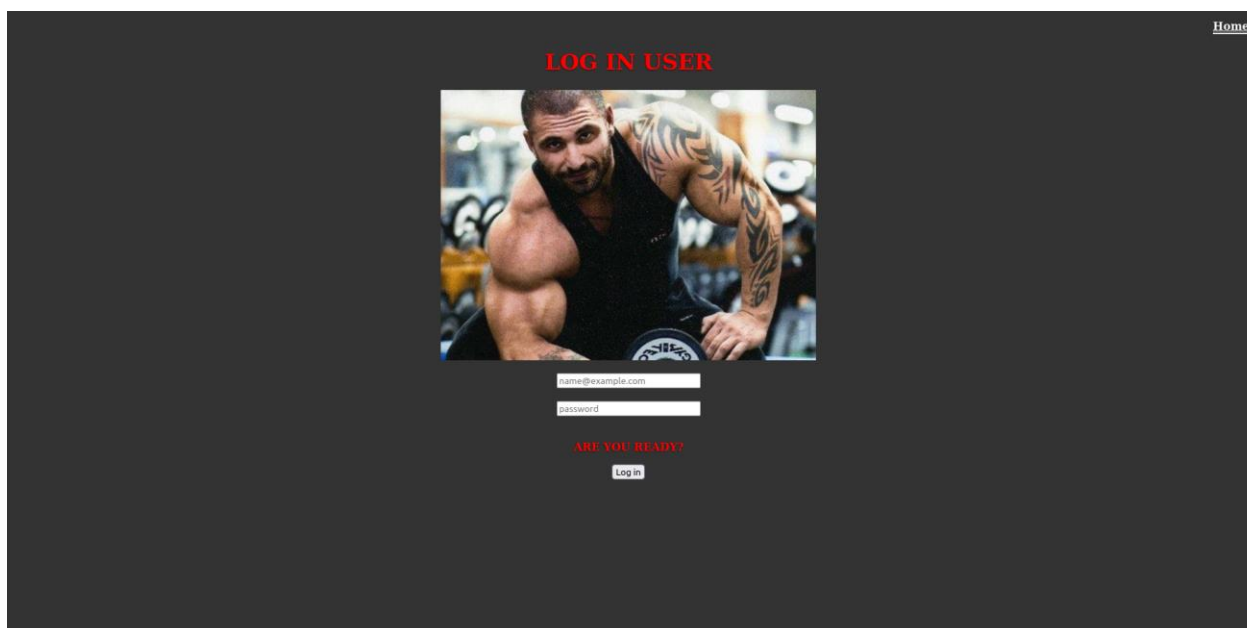
### *REGISTRAZIONE USER E PALESTRA*





**LOGIN USER E PALESTRA**





### ***EFFETTUARE PRENOTAZIONE E VISUALIZZARE PRENOTAZIONI LATO CLIENTE***

In queste due finestre l'utente ha possibilità di visionare le prenotazioni da lui effettuate e poter effettuare le prenotazioni verso una o più palestre, tra quelle disponibili.

## BOOK YOUR TRAINING SESSION NOW!

ID	Nome palestra	Indirizzo	ORARIO APERTURA	ORARIO CHIUSURA
60e9d34d356c3326fcb31fce	Egogym	Via Portella, Umbertide 06019	08:00	21:00
60e9d37a356c3326fcb31fcf	Gem	Via Rodolfo Morandi, Umbertide 06019	10:00	22:00
60e9d39f356c3326fcb31fd0	Mercagem	Via Pietro da Cortona, Mercatale 52044	09:00	18:00

ID Palestra

Palestra

Phone

Orario

BOOK NOW!

## THIS IS YOUR BOOKS

Palestra	DATA E ORA	N. PRENOTAZIONE
Mercagem	2021-06-21T14:00:00.000Z	60e9d556356c3326fcb31fd7
Egogym	2021-06-21T08:00:00.000Z	60e9d568356c3326fcb31fd8
Mercagem	2021-06-22T12:00:00.000Z	60e9d57d356c3326fcb31fd9
Egogym	2021-06-22T10:00:00.000Z	60ea011b591e120eced2c466

**EFFETTUARE UNA SEGNALAZIONE COVID-19**

[Home](#)

**ATTENTION!**

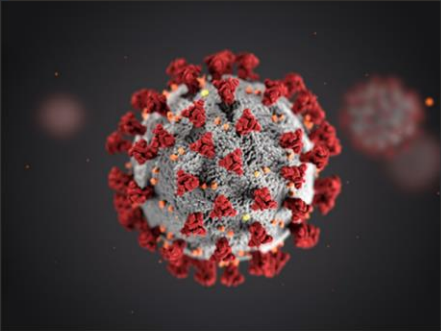
IF YOU HAVE BEEN IN CONTACT  
WITH A POSSIBLE CASE OF COVID19  
OR YOU ARE POSITIVE

**REPORT!**

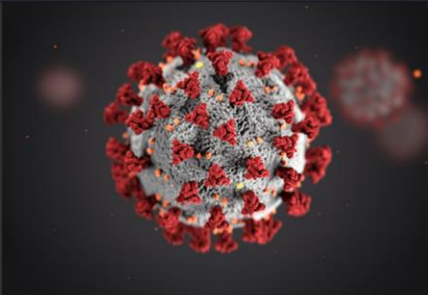
REPORT NOW!

***VISIONARE LE SEGNALAZIONI ED EFFETTUARE IL TRACCIAMENTO***

POSSIBLE CASES OF COVID19



ID SEGNALAZIONE	ID CLIENTE
60e9d616356c3326fcb31fda	60e9d2a0356c3326fcb31fcb
60e9d687356c3326fcb31fdb	60e9d2a0356c3326fcb31fcb



ID SEGNALAZIONE	ID CLIENTE
60e9d616356c3326fcb31fda	60e9d2a0356c3326fcb31fcb
60e9d687356c3326fcb31fdb	60e9d2a0356c3326fcb31fcb

ID SEGNALAZIONE	ID CLIENTE	PALESTRA	ORARIO PRENOTAZIONE	PHONE
60e9d568356c3326fcb31fd8	60e9d2a0356c3326fcb31fcb	Egogym	2021-06-21T08:00:00.000Z	3381064211
60ea011b591e120eced2c466	60e9d2a0356c3326fcb31fcb	Egogym	2021-06-22T10:00:00.000Z	3312386455

***VISUALIZZARE PRENOTAZIONI PRESSO LA PROPRIA SEDE***

## THIS IS YOUR BOOKS

CLIENTE	DATA E ORA	PHONE	N. PRENOTAZIONE
60e9d287356c3326fcb31fca	2021-06-21T08:00:00.000Z	3312386455	60e9d409356c3326fcb31fd1
60e9d287356c3326fcb31fca	2021-06-23T10:00:00.000Z	3312386455	60e9d506356c3326fcb31fd5
60e9d2a0356c3326fcb31fcb	2021-06-21T08:00:00.000Z	3381064211	60e9d568356c3326fcb31fd8
60e9d2a0356c3326fcb31fcb	2021-06-22T10:00:00.000Z	3312386455	60ea011b591e120eced2c466

## MODIFICA INFORMAZIONI O ELIMARE ACCOUNT

[LOGOUT](#) [HOME](#)

### YOUR DATA

**NOME:** Egogym

**ADDRESS:** Via Portella, Umbertide

**POSTAL CODE:** 06019

**EMAIL:** egogym@gmail.com

**ORARIO APERTURA:** 08:00

**ORARIO CHIUSURA:** 21:00

### UPDATE YOUR DATA

Orario Apertura

Orario Chiusura

UPDATE DATA

OR

DELETE ACCOUNT

Mostra applicazioni