Après quelques articles techniques sur **node.js**, on va replacer les choses dans le contexte du **jeu vidéo** par le biais d'un cas simple de sauvegarde de données relatives aux joueurs en appliquant ce qui a été expliqué dans les articles relatifs à mysql, les librairies utilisateurs **node.js**.

Prérequis

– Avoir un serveur **mysql** installé;
– savoir ce qu'est **node.js** (voir l'article Comprendre **node.js**
– avoir installé **node.js** (voir l'article Installer **node.js** sous Windows);
– avoir lu l'article Accès mysql avec node.js;
– avoir lu l'article Créer une librairie **node.js**;
– avoir lu l'article **node.js socket.io** par l'exemple.

Le principe

Imaginez que vous ayez créé un jeu vidéo (peu importe le jeu vidéo) dont vous souhaitez sauvegarder les meilleurs scores de chaque joueur.
Le principe est d'invoquer une url à laquelle on joint des données au format **json**, notamment le joueur et le score. Cette url appelle un script **node.js**.
Ce script vérifie si le joueur donné paramètre existe en base de données.
S'il existe, il compare le score stocké en base de données au score reçu. Si le score reçu est supérieur au score stockée, le script met à jour le score du joueur sinon il ne fait rien.
S'il n'existe pas, il le crée avec un score égal au score reçu.
Ne seront stockés que le meilleur score du joueur, son adresse email et la date du score.
Pour l'exemple et pour poster l'email et le score, vous allez utiliser une ihm html dédiée comprenant un bouton et deux zones de saisie : la première pour l'email du joueur, la seconde pour son score.
Je vous propose d'utiliser :
– une base de données mysql pour sauvegarder les score et donc la librairie mysql; initialisée dans l'article Créer une librairie **node.js**;
– la librairie **socket.io** pour invoquer le script **node.js** chargé de la sauvegarde ou de la mise à jour des scores;
– enfin la librairie **node.js express** pour instancier simplement un serveur http permettant d'exposer l'url d'accès à l'ihm permettant de poster l'email et le score.

La base de données mysql

Vous allez devoir créer une base de données contenant une table dédiée aux scores des joueurs.
En ligne de commande, connectez vous à votre serveur **mysql** en remplaçant serveur, user et password par les valeurs correspondantes à votre configuration :

```
mysql -hserveur -uuser -ppassword
```

Puis créez une nouvelle base de données nommée ***game*** :

```
create database game;
```

Avant de créer la table chargée de stocker les scores des joueurs, positionnez vous sur la base de données créées :

```
use game;
```

Ensuite créez la table de la base de données :

```
CREATE TABLE IF NOT EXISTS `scores` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `email` varchar(128) NOT NULL,
  `score` int(11) NOT NULL,
  `date` date NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Cette table contient un identifiant unique, l'email du joueur, son score et la date du score.

Initialisation du projet score

Créez, à l'endroit où vous avez l'habitude de créer vos projets, un dossier *score* dédié au projet. Depuis la ligne de commande, placez vous dans le dossier de votre projet score et exécutez la commande suivante pour installer la librairie **mysql node.js** :

```
npm install mysql
```

Vous devriez voir se faire l'installation comme suit :

```
npm http GET https://registry.npmjs.org/mysql
npm http 304 https://registry.npmjs.org/mysql
npm http GET https://registry.npmjs.org/require-all/0.0.3
npm http GET https://registry.npmjs.org/bignumber.js/1.0.1
npm http GET https://registry.npmjs.org/readable-stream
npm http 304 https://registry.npmjs.org/require-all/0.0.3
npm http 304 https://registry.npmjs.org/bignumber.js/1.0.1
```

```
npm http 304 https://registry.npmjs.org/readable-stream
npm http GET https://registry.npmjs.org/string_decoder
npm http GET https://registry.npmjs.org/core-util-is
npm http GET https://registry.npmjs.org/debuglog/0.0.2
npm http 304 https://registry.npmjs.org/core-util-is
npm http 304 https://registry.npmjs.org/string_decoder
npm http 304 https://registry.npmjs.org/debuglog/0.0.2
mysql@2.1.0 node_modules\mysql
├── require-all@0.0.3
├── readable-stream@1.1.11 (debuglog@0.0.2, string_decoder@0.10.25-1, core-util-is@1.0.1)
└── bignumber.js@1.0.1
```

Toujours depuis le dossier de votre projet score et exécutez la commande suivante pour installer la librairie **socket.io** :

```
npm install socket.io
```

Vous devriez voir se faire l'installation comme suit :

```
npm http GET https://registry.npmjs.org/socket.io
npm http 200 https://registry.npmjs.org/socket.io
npm http GET https://registry.npmjs.org/socket.io/-/socket.io-1.0.6.tgz
npm http 200 https://registry.npmjs.org/socket.io/-/socket.io-1.0.6.tgz
npm http GET https://registry.npmjs.org/has-binary-data
npm http GET https://registry.npmjs.org/socket.io-adapter
npm http GET https://registry.npmjs.org/debug
npm http GET https://registry.npmjs.org/socket.io-client
npm http GET https://registry.npmjs.org/engine.io
npm http GET https://registry.npmjs.org/socket.io-parser
npm http 200 https://registry.npmjs.org/debug
npm http GET https://registry.npmjs.org/debug/-/debug-0.7.4.tgz
....
socket.io@1.0.6 node_modules\socket.io
├── debug@0.7.4
├── has-binary-data@0.1.1 (isarray@0.0.1)
├── socket.io-adapter@0.2.0 (socket.io-parser@2.1.2)
├── socket.io-parser@2.2.0 (isarray@0.0.1, emitter@1.0.1, json3@3.2.6)
├── engine.io@1.3.1 (base64id@0.1.0, debug@0.6.0, engine.io-parser@1.0.6, ws@0.4.31)
```

```
└── socket.io-client@1.0.6 (to-array@0.1.3, indexof@0.0.1, component-bind@1.0.0,
 object-component@0.0.3, component-emitter@1.1.2, parseuri@0.0.2, engine.io-client@1.3.1)
```

Toujours depuis le dossier de votre projet score et exécutez la commande suivante pour installer la librairie **express** :

```
npm install express
```

Vous devriez voir se faire l'installation comme suit :

```
npm http GET https://registry.npmjs.org/express
npm http 200 https://registry.npmjs.org/express
npm http GET https://registry.npmjs.org/express/-/express-4.4.5.tgz
npm http 200 https://registry.npmjs.org/express/-/express-4.4.5.tgz
npm http GET https://registry.npmjs.org/methods
npm http GET https://registry.npmjs.org/accepts
npm http GET https://registry.npmjs.org/range-parser
npm http GET https://registry.npmjs.org/escape-html
npm http GET https://registry.npmjs.org/serve-static
npm http GET https://registry.npmjs.org/fresh
npm http GET https://registry.npmjs.org/cookie-signature
npm http GET https://registry.npmjs.org/vary
npm http GET https://registry.npmjs.org/proxy-addr
npm http GET https://registry.npmjs.org/parseurl
npm http GET https://registry.npmjs.org/buffer-crc32
npm http GET https://registry.npmjs.org/utils-merge
npm http GET https://registry.npmjs.org/qs
npm http GET https://registry.npmjs.org/merge-descriptors
...
express@4.4.5 node_modules\express
├── parseurl@1.0.1
├── utils-merge@1.0.0
├── cookie@0.1.2
├── merge-descriptors@0.0.2
├── cookie-signature@1.0.4
├── vary@0.1.0
├── fresh@0.2.2
├── qs@0.6.6
```

```
├── escape-html@1.0.1
├── range-parser@1.0.0
├── methods@1.0.1
├── serve-static@1.2.3
├── buffer-crc32@0.2.3
├── path-to-regexp@0.1.2
├── debug@1.0.2 (ms@0.6.2)
├── proxy-addr@1.0.1 (ipaddr.js@0.1.2)
├── accepts@1.0.6 (negotiator@0.4.7, mime-types
├── type-is@1.2.1 (mime-types@1.0.0)
└── send@0.4.3 (mime@1.2.11, finished@1.2.2)
```

L'ihm pour poster l'email et le score

Dans le projet ***scoreGame***, créez un nouveau fichier nommé ***scoreGame.html***. Le code à y intégrer html est trivial :

```
<html>
 <body>
  SAUVEGARDE DE SCORE AVEC NODE.JS ET SOCKET.IO<br>
  <input type="text" id="email"><br>
  <input type="text" id="score"><br>
  <input type="button" id="boutonSaveScore" value="Sauvegarder Score">
 </body>
</html>
```

Il reste à intégrer le code javascript pour soumettre l'email et le score.
Vous avez donc besoin de créer 3 variables qui se rapporteront aux objets de l'ihm : les 2 zones de saisie et le bouton.
Celles-ci sont à initialiser au chargement de la page par le biais de l'**événement javascript load** :

```
  var email;
  var score;
  var boutonSaveScore;

  window.addEventListener("load", function() {
    email = document.getElementById("email");
    score = document.getElementById("score");
```

```
    boutonSaveScore = document.getElementById("boutonSaveScore");
  });
```

Ceci fait, il reste à associer au bouton une action sur le clic par le biais de l'**événement click javascript**.
L'action associée, quant à elle, envoie les données (l'email et le score) par le biais d'un message sur le socket grâce à la librairie **node.js socket.io**.
Dans l'exemple, le message est nommé ***saveScore***. Ce nom a son importance puisqu'il est utilisé pour la réception et la capture des données envoyées.
Le tout est encapsulé dans une fonction autoexécutée pour isoler le code de tout autre code **javascript** existant dans la page html.

```
( function() {
  var email;
  var score;
  var boutonSaveScore;

  var envoyerMessage = function() {
    var socket = io.connect('http://localhost:9090');
    socket.emit('saveScore',{'email':email.value,'score':score.value});
  };

  window.addEventListener("load", function() {
    email = document.getElementById("email");
    score = document.getElementById("score");
    boutonSaveScore = document.getElementById("boutonSaveScore");
    boutonSaveScore.addEventListener("click", envoyerMessage);
  });
})();
```

Ceci n'est toutefois pas suffisant pour avoir l'ihm fonctionnelle : il manque l'import de la librairie **socket.io**.

```
<script src="http:/adresse_de_votre_serveur_node_js:9090/socket.io/socket.io.js"></script>
```

Le port indiqué (9090) dans l'url correspond au port choisi sur lequel seront émises les requêtes. Le choix de ce port est purement arbitraire et il se configure côté serveur.

Le script de sauvegarde des scores

Vous allez reprendre la librairie ***db.js*** créée dans l'article Créer une librairie **node.js** pour y ajouter deux nouvelles méthodes :
– une dédiée à l'insertion de données cas où le joueur n'existe pas);
– la seconde dédiée à la mise à jour de données (cas où le joueur existe).
Tout ça, bien entendu, en utilisant la librairie **node.js mysql**.
La méthode pour l'insertion :

```
executeInsertQuery : function( insertQuery ) {
  this.mySqlClient.query(
    insertQuery,
    function result(error, info) {
      if (error) {
        db.close();
        return error;
      }
      return info.insertId;
    }
  );
}
```

La méthode pour la mise à jour :

```
executeUpdateQuery : function( updateQuery ) {
  this.mySqlClient.query(
    updateQuery,
    function result(error) {
      if (error) {
        db.close();
        return error;
      }
      return;
    }
  );
}
```

Le script de sauvegarde réalise 2 tâches :
– l'exposition de l'ihm par le biais de la librairie **nodes.js express**;

– l'écoute sur le port 9090 en attente de messages.

L'exposition de l'ihm

Il suffit ici d'instancier un objet express auquel on indique :
– d'écouter les requêtes http sur le port 80;
– de servir le fichier de l'ihm : ***scoreGame.html*** sur ce même port.

```
var express = require('express');


// serveur html
var server= express();
server.listen(80);
server.get('/scoreGame.html', function(request, response) {
 response.sendfile('./scoreGame.html');
});
```

Ainsi lorsqu'est demandée ***/scoreGame.html***, le fichier ***scoreGame.html*** est servi.
Il est tout à fait possible de servir un autre fichier sur ***/scoreGame.html*** : c'est la méthode ***sendfile*** qui indique le fichier renvoyé sur cet appel à laquelle il est possible d'indiquer le fichier de votre choix.

La réception des messages socket.io

Le cœur du problème de la sauvegarde est abordé ici : ajouter le score si l'email n'existe pas et dans le cas contraire le mettre à jour.
Sur réception du message, vous devez vérifier si l'email existe et selon le cas ajouter ou mettre à jour le score.
La première chose à faire est d'instancier un objet socket.io, puis créer un socket en écoute sur le port 9090 :

```
var io = require("socket.io");
var sockets = io.listen(9090);
```

La deuxième chose à réaliser est de rendre le serveur réactif au message ***saveScore*** envoyé par l'ihm :

```
// serveur socket.io
sockets.on('connection', function (socket) {
  socket.on('saveScore', function (data) {
    // actions exécutées à la réception du message
```

```
  });
});
```

Pour plus de détails sur la librairie **node.js socket.io**, référez vous à l'article **node.js socket.io** par l'exemple.
Il ne reste plus qu'à faire la vérification de l'existence de l'email en base de données par le biais de la librairie ***db.js*** : vous devez donc instancier un objet db comme suit :

```
var db = require('./db.js');
```

Et à l'aide de cet objet, vous vous connectez à la base de données, exécutez la requête de vérification de l'email et en dernier exécutez la requête d'ajout du score ou la requête de mise à jour du score.
La requête **sql** de recherche par l'email en base de données :

```
select id,email,score from scores where email='email réceptionné par le socket'
```

La requête **sql** d'ajout du score en base de données :

```
insert into scores (email,score,date) values('email réceptionné par le socket','score
réceptionné par le socket',NOW())
```

La requête **sql** de modification du score en base de données :

```
update scores set score=score_réceptionné_par_le_socket,date=NOW() where email='email
réceptionné par le socket'
```

A la réception du message, vous devez :
– vous connecter à la base de données;
– exécuter la **requête sql** de recherche;
– exécuter selon les cas, la **requête sql** d'ajout ou de mise à jour.
Comme l'ajout ou la mise à jour se font en fonction du résultat de la requête de recherche, l'ajout ou la mise à jour doivent se faire par le biais d'une fonction ***callback*** de la méthode ***executeSelectQuery*** de l'objet ***db***.
Appelez cette fonction ***enregistrerScore*** :

```
// serveur socket.io
sockets.on('connection', function (socket) {
  socket.on('saveScore', function (data) {
db.connect('adresse_de_votre_serveur_mysql','votre_utilisateur_mysql','votre_mot_de_passe_mys
```

```
ql','game');
    var sqlSelect = "select id,email,score from scores where email='" + data.email + "'";
    db.executeSelectQuery(sqlSelect, enregistrerScore);
  });
});
```

Il ne reste plus qu'à implémenter cette fonction : insérer ou mettre à jour le score selon le cas.
Cette fonction doit :
– insérer les données du joueur si l'email n'existe pas;
– mettre à jour les données du joueur si le score reçu est supérieur au score stocké.
Cette fonction doit donc avoir à disposition :
– les données existantes (results) ;
– les données reçues (data).
Cette fonction prend donc deux paramètres :
– les données existantes (results) ;
– les données reçues (data).

```
var enregistrerScore = function(results, data) {
  if ( empty(results)  ) {
    var sqlInsert = "insert into scores (email,score,date) values('" + data.email + "'," +
data.score + ",NOW()) ";
    db.executeInsertQuery(sqlInsert);
  } else {
    if ( results[0].score < data.score ) {
      var sqlUpdate = "update scores set score=" + data.score + ",date=NOW() where email='" +
data.email + "' ";
      db.executeUpdateQuery(sqlUpdate);
    }
  }
  db.close();
}
```

Notez l'utilisation de la fonction ***empty*** servant à indiquer si une structure de données est vide et dont voici l'implémentation :

```
var empty = function empty(object) {
  for (var i in object)
    if (object.hasOwnProperty(i))
      return false;
```

```
  return true;
}
```

L'appel à la fonction ***enregistrerScore*** nécessite de disposer des données existantes et des données reçues.
Ce qui oblige à modifier l'appel à la fonction ***executeSelectQuery***, passant de :

```
db.executeSelectQuery(sqlSelect, enregistrerScore);
```

à

```
db.executeSelectQuery(sqlSelect, enregistrerScore, data);
```

Et qui nécessite de modifier aussi l'implémentation de ***executeSelectQuery*** dans la librairie ***db.js*** :

```
executeSelectQuery : function( selectQuery, callbackResultFunction, data ) {
  this.mySqlClient.query(
    selectQuery, function( error, results ) {
      if ( error ) {
        db.close();
        return error;
      } else {
          callbackResultFunction(results, data);
      }
    }
  );
},
```

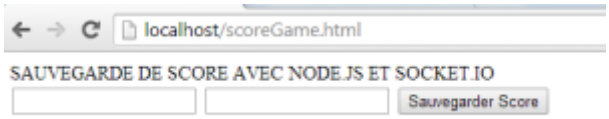
Il ne reste plus qu'à tester.

Le test

En ligne de commande, lancez le script ***scoreGame.js*** depuis le dossier du projet :

```
node scoreGame.js
```

Puis rendez vous à l'adresse http://localhost/scoreGame.html qui donne à l'affichage :



Les sources du projet sont téléchargeables ici : scoreGame.zip.
[important]Des remarques, des améliorations, des idées, des coquilles : faites le savoir. Faites savoir si cet article vous a été utile par un commentaire ou les réseaux sociaux.[/important]