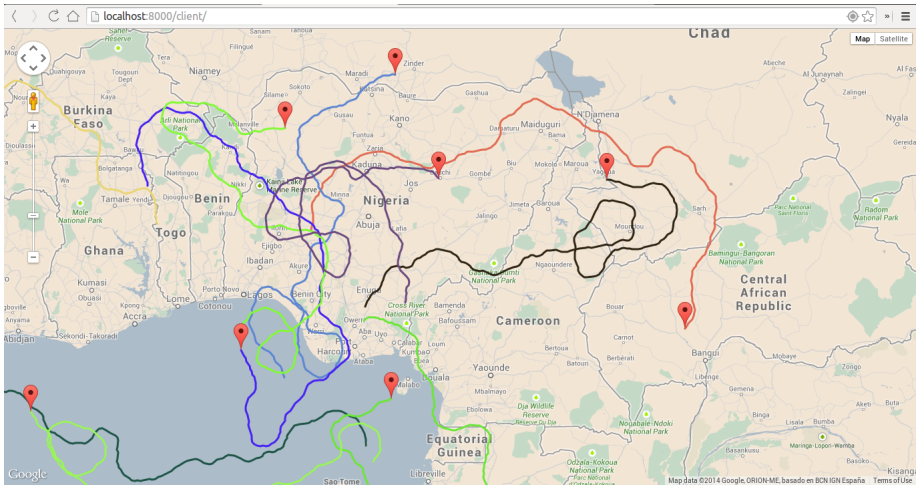
1 Introduction

Le but de ce project et de réaliser un application de tracking GPS, qui permet à un ensemble d'utilisateur de partager leur trajets en temps réel.



Afin de réliser cette application nous avons utiliser les outils suivant :

2 Outils utilisés

2.1 Node.js

Node.js est une plateforme logicielle libre et événementielle en JavaScript orientée vers les applications réseau.

Elle utilise la machine virtuelle V8 et implémente sous licence MIT les spécifications CommonJS.

Node.js contient une bibliothèque de serveur HTTP intégrée, ce qui rend possible de faire tourner un serveur web sans avoir besoin d'un logiciel externe comme Apache ou Lighttpd, et permettant de mieux contrôler la façon dont le serveur web fonctionne.

2.2 Socket.io

Socket.io est un framework pour les application temps-réel. Il est composé de deux partie une partie client qui tourne sur un navigateur et une autre serveur qui tourne sur Node.js. Le deux parties ont une API identique.

La technologie principale que Socket.io est le protocol WebSockets, mais il peut utiliser d'autre méthodes comme Flash, Long-polling, JSONP.

2.3 Angular.js

Angular.js est un framework Javascript coté client dévélopé par Google qui à pour but d'accélerer le développement d'application web monopages. Il propose d'étendre la syntax de HTML avec des balise personnlisé, organiser l'application sous une architecture MVC, communiquer avec le serveur, routing, data-binding et autres fonctionnalités.

2.4 Phantom.js et Casper.js

Phantom.js et un headless browser c'est un dire une navigateur qui sans interface graphique mais plutot une API qui permet d'écrire des tests pour les applications web coté client.

Casper.js est framework de Phantom.js qui facilite l'écriture de ces scripts.

3 Réalisation

3.1 Serveur HTTP

Le code suivant utilise le framework express pour servir les fichier html, css et Javascript qui se trouve dans le dossier client et lance le serveur HTTP sur le port 'port' :

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

// create http server
var app = express()
```

```
  , server = http.Server(app);

// serve client's static files
app.use('/client', express.static(__dirname + '/../client'));

// run the server
var port = process.argv[2] || 8000;
server.listen(port, function() {
  console.log('Server started  : http ://localhost :'
  + port + '/client');
});
```

3.2 Transmition de données en temps réel

3.2.1 Le modèle :

On declare une class Users qui contient la listes des utilisateurs et leur données géographiques.

```
function Users() {
  this._users = {}; // the list of users
}
```

On ajoute des méthodes qui manipule la liste _users tout en envoyant des signal vers les clients pour les prévenir des changements de données

Ajout d'un utilisateur :

```
Users.prototype.addUser = function(socket) {
  this._users[socket.id] = [];

  // tell all other users
  socket.broadcast.emit('add user', socket.id);

  // send the list of users to the new one
  socket.emit('list', this._users);
};
```

Suppression d'un utilisateur :

```
Users.prototype.removeUser = function(socket) {
  delete this._users[socket.id];

  // inform other users
  socket.broadcast.emit('remove user', socket.id);
}
```

Ajout d'un mouvement de l'utilisateur :

```
Users.prototype.addStep = function(socket, pos) {
  this._users[socket.id].push(pos);
```

```
  // tell everyone else this user moved
  socket.broadcast.emit('add step', {id : socket.id, pos  : pos});
}
```

3.2.2 Le controlleur :

Dans le controlleur on repond au signux de l'utilsateur et on manipule le modèle.

```
var socketio = require('socket.io')
  , Users = require('../models/users');

// start socket.io
var io = socketio.listen(server)
  , users = new Users();

// The user connects
io.sockets.on('connection', function(socket) {
   users.addUser(socket);

   // he moves
   socket.on('moved', function(pos) {
     users.addStep(socket, pos);
   });

   // or quits
   socket.on('disconnect', function() {
     users.removeUser(socket);
   });
});
```

3.3 Le Client

Le client est écrit en utilisant Angular.js et le module angular-google-maps ce dernier permet d'utiliser l'API Google maps avec des directives.

3.3.1 Les modèles

Options Google maps :

```
 var app = angular.module('gps-tracking');

 app.value('map', {
   center : {
     latitude : 0,
     longitude : 0
   },
   zoom : 8
   // ...
 });
```

Liste des utilisateurs

```
// socket.io
app.factory('socket', function() {
  return io.connect('/');
});

// users' positions
app.value('users', {});
```

3.3.2 Le controlleur

Dans le controlleur on répond aux signaux du serveur qui sont :
— **list** : Envoie la liste de tous les utilisateurs qui sont déja connéctés.
— **add user** : Ajout d'un nouvel uilisateur.
— **remove user** : Suppression d'un utilisateur.

Ensuite on apelle watchPosition qui nous permet de mettre à jour les coordonées de l'utilisateur.

```
app.controller('MapController', function (
   $scope, map, socket, users, helpers, STROKE_WIDTH) {
  // THE LIST OF USERS
  $scope.users = users;

  // GOOGLE MAPS SETTINGS
  $scope.map = map;

  // SOCKET.IO SERVER
  socket.on('list', function(list) {
    $scope.$apply(function() {
      // Add previousely connected users
      for(var id in list) {
        users[id] = {
          stroke  : {
            color : helpers.getRandomColor(),
            stroke : STROKE_WIDTH
          },
          path : list[id]
        };
      }
    });
  });

  socket.on('add user', function(id) {
    $scope.$apply(function() {
      users[id] = {
        stroke  : {
          color : helpers.getRandomColor(),
          stroke : STROKE_WIDTH
```

```
      },
      path  : []
    };
  });
});

socket.on('remove user', function(id) {
  $scope.$apply(function() {
    delete users[id];
  });
});

socket.on('add step', function(data) {
  $scope.$apply(function() {
    users[data.id].path.push(data.pos);
  });
});

// GEOLOCATION API
navigator.geolocation.watchPosition(
  // Sucess callback
  function (pos) {
    var p = {
      timestamp : pos.timestamp,
      longitude : pos.coords.longitude,
      latitude : pos.coords.latitude
    };

    // send signal to the server and update local data
    if(!users[socket.id]) {
      users[socket.id] = {
        stroke : {
          color : helpers.getRandomColor(),
          weight : STROKE_WIDTH
        },
        path : []
      };
    }

    // store locally
    users[socket.id].path.push(p);

    // send to server
    socket.emit('moved', p);
  },
  // Error callback
  function (err) {
    alert(err.message);
  }
);
```

```
});
```

3.4 La vue

A l'aide du module angualar-google-maps on peut afficher les données stockées dans le tableau users dans Google maps on utilisant les derictives suivantes :

```
<google-map center="map.center" zoom="map.zoom"
  draggable="true" options="map.options">
    <polyline ng-repeat="user in users" path="user.path"
      stroke="user.stroke">
    </polyline>
  <marker ng-repeat="user in users"
    ng-show="user.path.length > 0" coords="user.path | last">
  </marker>
</google-map>
```