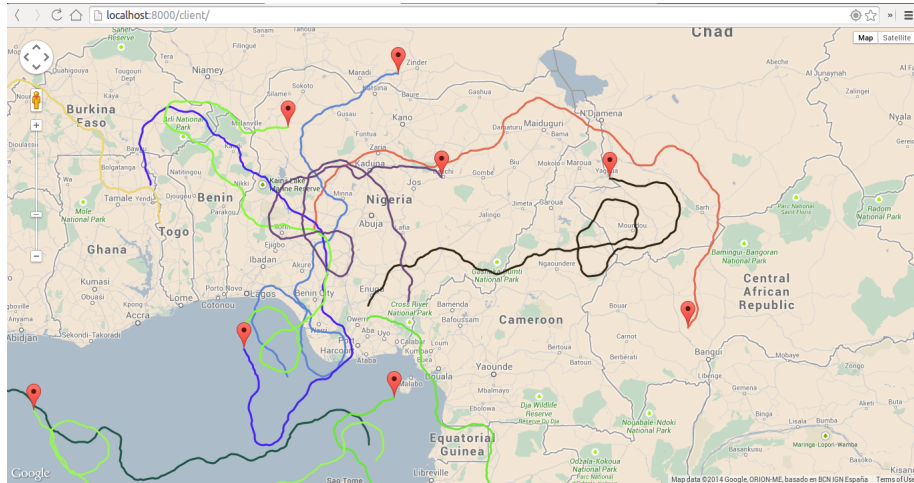


1 Introduction

Le but de ce projet est de réaliser une application de tracking GPS, qui permet à un ensemble d'utilisateurs de partager leurs trajets en temps réel.



Afin de réaliser cette application nous avons utilisé les outils suivants :

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 développé par Google qui à pour but d'accélérer le développement d'application web monopages. Il propose d'étendre la syntax de HTML avec des balise personnalisé, 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 est un headless browser c'est à dire un navigateur qui sans interface graphique mais plutôt 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 Transmission 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'utilisateur 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);
    });
});
```