Building Location Aware Games for Mobile Browsers

- This tutorial covers how to develop location aware games for mobile browsers step by step with server side solutions.
- During the detailed explanation and implementation of the developed game, Hidden Treasure of Pikachu, some encountered problems and solutions are also explained.
- The game is developed using HTML5+CSS3+jQuery, Bootstrap and Google Map API.
- To build RESTful web game, NodeJS,Express and MongoDB are used.

Building Location Aware Games for Mobile Browsers

What are the requirements?

- ➔ Basic knowledge of HTML,CSS and JavaScript will be helpful (not necessary)
- ➔ Google Chrome or any other modern web browser that supports geolocation

What is the target audience?

- ➔ Students who wish to learn how to develop location aware games/applications
- ➔ Developers who want to come up with location aware games/applications
- ➔ Anyone who wants to develop mobile browser based applications incorporating maps and geolocation

Outline

This tutorial consists of following parts:

1. Creating a simple desktop location aware treasure hunter game
2. Creating a simple mobile browser based location aware treasure hunter game
3. Creating RESTful mobile browser based location aware treasure hunter game

Google maps

1. Creating a simple desktop location aware treasure hunter game

Application Use Case

In this version of the game, we will build a simple desktop treasure hunter game to understand concept of the Google Map API and how to use it. In the game user walks around by help of Google Map Street View Service. By walking around the specified area user tries to collect the treasures. Google Map Street View provides the panorama view to walk around the street controlling an avatar with keyboards direction buttons.

1. Creating a simple desktop location aware treasure hunter game

Application Technology Stack

The game will be developed using the following technology stack:

- ***HTML5:*** The client side of the application is built using HTML5.
- ***Google Maps*** *:* The application will be using Google Maps to render the user and game information on it.

1. Creating a simple desktop location aware treasure hunter game

Let's Get Started to Hunt

➔ To create google map based location aware game, first step is to initialize google map and google map panorama view. Following JS code basically shows the how to manage this.

```
1  //center positin settings for current user
2  var center = new google.maps.LatLng(61.064802, 28.097180);
3  var mapOptions = {
4      center: center,
5      mapTypeId: google.maps.MapTypeId.ROADMAP,
6      zoom: 15
7  };
8  //create google map instance
9  var map = new google.maps.Map(document.getElementById('map-canvas'), mapOptions);
10 //panaroma settings
11 var panoramaOptions = {
12     position: center,
13     pov: {
14         heading: 34,
15         pitch: 20
16     }
17 };
18 //create panorama view to track user
19 var panorama = new google.maps.StreetViewPanorama(document.getElementById('pano'), panoramaOptions);
```

Explanation of the all code in github

1. Creating a simple desktop location aware treasure hunter game

Google Map Settings

➔ After the settings for the panorama view and google map instance, next step is to set panorama view on the google map instance. By doing that, user is enabled to see the current location both on panorama view and map.

```
1  //set panorama to google map instance
2  map.setStreetView(panorama);
```

➔ In the google map settings, zoom, center and mapType can be changed. More information about Google Map MapOptions can be found here. Next thing on the code is to put each treasure on the map. Following code shows the way how to manage it.

```
1  var pos = new google.maps.LatLng(treasure_locations[i][1], treasure_locations[i][2]);
2  marker = new google.maps.Marker({
3      icon: 'images/icon5.png',
4      position: pos,
5      map: map;
6  });
7  //store each marker(treasure) on the array list to track which treasure is picked
8  markers[i] = marker;
```

➔ The important thing on the above code is ti store those treasure informations also separate array to track which of them is picked or not.

Explanation of the all code in github

1. Creating a simple desktop location aware treasure hunter game

Google Map Settings

➔ We also need to put each treasure on the panorama view to enable the user to control the map and collect treasures.

```
1  //set treasures on panoramo map
2  marker = new google.maps.Marker({
3      icon: 'images/icon5.png',
4      position: pos,
5      map: panorama
6  });
7  //store each marker(treasure) on the array list to track which treasure is picked
8  markpano[i] = marker;
```

➔ After initializing all the treasure locations both the map and the panorama view, next step is to put information window for each treasure. This information window shows the name of the treasure by clicking on it.

```
1  // add listener to each marker to show infowindow on it
2  google.maps.event.addListener(marker, 'click', (function(marker, i) {
3      return function() {
4          infowindow.setContent(treasure_locationss[i][0]);
5          infowindow.open(map, marker);
6      } })(marker, i));
```

Explanation of the all code in github

1. Creating a simple desktop location aware treasure hunter game

Calculate Distance and remove collected treasure on the map

- As a rule of the game, if the user wants to collect the treasure (s)/he has to be close enough to the treasure. In this game design if the distance b/w the user and the treasure is less than 50 meters then this treasure is counted as collected. To calculate the distance b/w user and treasure Google Map Distance function is used. we call computeDistanceBetween() function by passing it two LatLng objects (the user and the treasure location information).

```
1  //calculates distance between two points in km's
2  function calcDistance(p1, p2) {
3      return (google.maps.geometry.spherical.computeDistanceBetween(p1, p2) / 1000).toFixed(3);
4  }
```

- *Explanation of the all code in github*

1. Creating a simple desktop location aware treasure hunter game

Calculate Distance and remove collected treasure on the map

➔ For calculating the distance b/w user and a treasure we need to get user current position on the panorama map. To do so we need to add position_changed event to our code as it is shown below code.

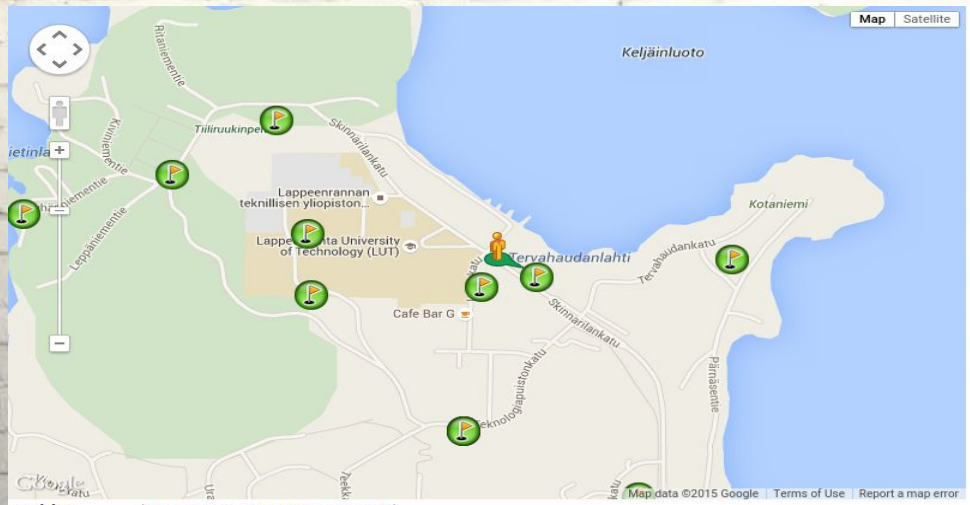
```
// add listener to panorama view to track user and treasures. The user needs to move around streets to hunt treasures
google.maps.event.addListener(panorama, 'position_changed', function() {
  var p2, p1;
  //get the user position from the panorama map
  p1 = new google.maps.LatLng(panorama.getPosition().lat(), panorama.getPosition().lng());
  for (var i = 0; i < markers.length; i++) {
      //get the user position from the panorama map
      p1 = new google.maps.LatLng(panorama.getPosition().lat(), panorama.getPosition().lng());
      //if the treasure is not picked before
      if (markers[i] != null) {
          p2 = new google.maps.LatLng(treasure_locations[i][1], treasure_locations[i][2]);
          if (calcDistance(p1, p2) < 0.50) {
              //remove marker from the map
              markers[i].setMap(null);
              markpano[i].setMap(null);
              //set marker i to null
              markers[i] = null;
              score = score + 1;
          }
      };
  };
}); //end of position_changed event
```

Explanation of the all code in github

1. Creating a simple desktop location aware treasure hunter game

Final Screen Shot

➔ All the detailed HTML5+CSS3+Javascript code can be found in github. Screen shot of the final design can be seen below.



Position	(61.064656, 28.09754099999998)
Score	1
Collected Balls	
Time	0:32

2. Creating a simple mobile browser based location aware treasure hunter game

Application Use Case

In this version of the game, we will build a simple mobile version of the treasure hunter game to understand concept of the W3C GeoLocation API and how to use it on Google Map. Also we will understand basic server side solution by sending and receiving data from the game server. In this version of the game user walks around the specified area by using GPS of the mobile device to hunt the treasures.

Google maps

2. Creating a simple mobile browser based location aware treasure hunter game

Application Technology Stack

The game will be developed using the following technology stack:

- ***HTML5:*** The client side of the application is built using HTML5.
- ***Google Maps API*** *:* The application will be using Google Maps API to render the user and game information on it.
- ***W3C Geolocation API:*** All modern browsers support the Geolocation API. Browser support information can be seen here.
- ***Node.js:*** is a JavaScript framework that runs outside the browser, in this case in the server. More up-to-date information about Node.js and how to download it can be seen here.
- ***MongoDB:*** is a NoSQL document oriented datastore. We will store the data of each collected treasure in MongoDB.
- ***Express.js:*** is a wrapper for the core Node.js HTTP module objects. We use Express.js for simple routing and support for Connect middleware. More benefits of Express.js can be found here.
- ***jQuery Mobile:*** is a framework for creating mobile web applications. jQuery Mobile works on all popular smartphones and tablets.
-

2. Creating a simple mobile browser based location aware treasure hunter game

Step 1 - Install Node.js and MongoDB

To build the application we need to install Node.js and MongoDB with required dependencies. Following links shows step by step how to download Node.js and MongoDB for your platform (Windows, Mac, Ubuntu)

- ➔ How to Install Node.js
- ➔ How to Install MongoDB

After installing the Node.js and MongoDB, create a folder for your application and name it "*Treasure_Hunter_Mobile*".

2. Creating a simple mobile browser based location aware treasure hunter game

Step 2 - Create Working Directory

➔ Once we create our application directory, "*Treasure_Hunter_Mobile", next step is to create following structure inside this folder.*

```
C:\Treasure_Hunter_Mobile
      - package.json
      - npm_modules
      - views
            - index.ejs
      - myapp.js
      - locationprovider.js
      - public
            - images
```

➔ Or you can install the project directory from github.

2. Creating a simple mobile browser based location aware treasure hunter game

Step 3 - Edit Dependencies

Now open *package.json* file and write following code. This code identifies our dependencies for our project.

More information about package.json file and why to use please refer here.

```
{
    "name": "application-name",
    "dependencies": {
        "express": "~4.x.x",
        "body-parser": "*",
        "mongodb": "*",
        "mongoose": "*",
        "ejs": "*"
    }
}
```

Google maps

2. Creating a simple mobile browser based location aware treasure hunter game

Step 3 - Edit Dependencies

Why do we need these dependencies ?

- ➔ ***body-parser:*** is used to pull POST content from our HTTP request. More information about body-parser middleware can be found here.
- ➔ ***Mongoose:*** is a Node.Js library that provides MongoDB object mapping similar to ORM with a familiar interface within Node.js. If you aren't familiar with Object Relational Mapping (ORM) or, Object Data Mapping (ODM) in the case of Mongoose, this means is that Mongoose translates data in the database to JavaScript objects for use in your application. We will use to communicate with our MongoDB database.

NOTE: *For more information about available Express.js and Node.js packages and dependencies please refer here.You can find detailed information for each package. Installation guide is also available.*

2. Creating a simple mobile browser based location aware treasure hunter game

Step 3 - Install Dependencies

Now we've defined our dependencies and we're ready to start developing our game. To install our dependencies we simply need to use npm command. To do so return to your command prompt, **cd to your Treasure_Hunter_Mobile directory**, and type this:

```
1   C:\Treasure_Hunter_Mobile >npm install
```

It's going to print out a ton of stuff. That's because it's reading the package.json file we just edited and installing all the stuff listed in the dependencies object (including Express). Once NPM has run its course, you should have a node_modules directory which contains all of our dependencies for this tutorial.

Note: After all those settings don't forget to test your Node.js server and MongoDB to make sure they are installed properly. To test them please refer here.

2. Creating a simple mobile browser based location aware treasure hunter game

Step 4 - Implementation- *myapp.js*

➔ First things, so let's define our dependencies in myapp.js:

```
1  var express = require("express");
2  var bodyParser = require("body-parser");
3  var Location = require('./locationprovider');
```

locationprovider is our mongoose instance to store game status data. We will explain it next chapters. for now just don't think about it much :)

➔ After the version 3.x, Express streamlines the instantiation of its app instance, in a way that this line will give us a server object:

```
4  var app = express();
```

This one's important. It instantiates Express and assigns our app variable to it. The next section uses this variable to configure a bunch of Express stuff.

2. Creating a simple mobile browser based location aware treasure hunter game

Step 4 - Implementation- *myapp.js*

➔ To extract params from the body of the requests we'll use bodyParser()middleware which looks more like a configuration statement:

```
5  app.use(bodyParser.urlencoded({ extended: false }));
```

➔ To set up the public directory to use static files we will use following line code:

```
6  app.use(express.static(__dirname + '/public'));
```

2. Creating a simple mobile browser based location aware treasure hunter game

Step 4 - Implementation- *myapp.js*

➔ The important thing to allow clients to send and receive the data from server, we need to add following line codes. These codes can be used any server side- clients solutions for Node.js applications.

```
9  app.use(function (req, res, next) {
10     // Website you wish to allow to connect
11     res.setHeader('Access-Control-Allow-Origin', 'http://ct100020vir6.pc.lut.fi:1072');
12     // Request methods you wish to allow
13     res.setHeader('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT, PATCH, DELETE');
14     // Request headers you wish to allow
15     res.setHeader('Access-Control-Allow-Headers', 'X-Requested-With,content-type');
16     // Set to true if you need the website to include cookies in the requests sent
17     // to the API (e.g. in case you use sessions)
18     res.setHeader('Access-Control-Allow-Credentials', true);
19     // Pass to next layer of middleware
20     next();
21 });
```

2. Creating a simple mobile browser based location aware treasure hunter game

Step 4 - Implementation- *myapp.js*

➔ After all the settings in our myapp.js, we need to handle our GET and POST requests from client side. The purpose of this code is: if you do an HTTP GET to /, this creates a **route handler**, which is a fancy name for a chain of request handlers for a given URL. Express matches the specified paths in the request and executes the callback appropriately. Your callback below tells Express to match the root "*/*" and return the given HTML in the response.

```
22  //routing is very easy with express, this will handle the request for root directory contents.
23  app.get("/", function(req, res) {
24      var locations = [⟺];
56      var lat = 61.065860;
57      var lon = 28.096065;
58      zoom = 15;
59      // now we use the templating capabilities of express and call our template to render the view,
60      //and pass a few parameters to it
61      res.render("index.ejs", {layout: false,loc: locations,lat: lat,lon: lon,zoom: zoom
62      });
63  });
```

2. Creating a simple mobile browser based location aware treasure hunter game

Step 4 - Implementation- *myapp.js*

➔ Following codes let us to send some map options and locations of the treasures to client side for displaying om the game.

```
res.render("index.ejs", {layout: false,loc: locations,lat: lat,lon: lon,zoom: zoom
});
```

2. Creating a simple mobile browser based location aware treasure hunter game

Step 4 - Implementation- *myapp.js*

➔ Now, we need to implement our solution for the HTTP POST requests from the client side. Following line codes take some parameters from client side and save it to our database.

```
64  app.post('/', function(req, res){
65        var name=req.body.name;
66        var lat=req.body.lat;
67        var lon=req.body.lon;
68        var Treasure = new Location({
69          name: name,
70          lat: lat,
71          lon: lon
72        });
73        Treasure.save(function(err) {
74          if (err) throw err;
75          console.log('data saved successfully!');
76        });
77        res.send("done");
78  });
```

2. Creating a simple mobile browser based location aware treasure hunter game

Step 4 - Implementation- *myapp.js*

➔ After all settings in our myapp.js, we need to set up the server to respond to the URL: *http://ct100020vir6.pc.lut.fi:1072*. The last line that actually starts the server on port 1072 in this case.

```
79  app.listen(1072);
```

2. Creating a simple mobile browser based location aware treasure hunter game

Step 4 - Implementation- *locationprovider.js*

➔ This section, we will go through the our database design and scheme to store treasure informations. The first thing we need to do is include mongoose in our project and open a connection to the test database on our locally running instance of MongoDB.

```
1  var mongoose = require('mongoose');
2  mongoose.connect('mongodb://157.24.188.41/mylocations');
```

➔ Note: *If you wish to connect to a remote database, substitute the string with your username, password, host and port values. Here is the format of the URI string: mongodb://[username:password@]host1[:port1][,host2[:port2],...[,hostN[:portN]]][/[database][?options]]*

2. Creating a simple mobile browser based location aware treasure hunter game

Step 4 - Implementation- *locationprovider.js*

➔ We now need to get notified if we connect successfully or if a connection error occurs:

```
1  var db = mongoose.connection;
2  db.on('error', function(err) {
3      console.log('connection error', err);
4  });
5  db.once('open', function() {
6      console.log('connected.');
7  });
```

Google maps

2. Creating a simple mobile browser based location aware treasure hunter game

Step 4 - Implementation- *locationprovider.js*

➔ We will create our schema design to keep our data in this format. The more details about the schema designs in mongoose can be seen here.

```
1  var Schema = mongoose.Schema;
2  var LocationSchema = new Schema({
3      name: String,
4      lat: Number,
5      lon: Number
6  });
```

2. Creating a simple mobile browser based location aware treasure hunter game

Step 4 - Implementation- *locationprovider.js*

➔ The schema is useless so far we need to create a model using it make this available to our users in our Node application.

```
1  module.exports = mongoose.model('Location', LocationSchema);
```

2. Creating a simple mobile browser based location aware treasure hunter game

Step 4 - Implementation- *index.ejs*

➔ When we finished our server side implementation for the game engine, now it is time to implement our client side code. First thing to do is to take the arguments and the treasure locations from server and put them on the google map. Following code shows how to do it.

```
1  var loc = <% -JSON.stringify(loc) %>
2  var myOptions = {
3      center: new google.maps.LatLng( <%= lat %> , <%= lon %> ),
4      zoom: <%= zoom %> ,
5      mapTypeId: google.maps.MapTypeId.ROADMAP
6  };
7  var map = new google.maps.Map(document.getElementById("map-canvas"),
8  myOptions);
```

Google maps

2. Creating a simple mobile browser based location aware treasure hunter game

Step 4 - Implementation- *index.ejs*

➔ Next step important thing in our implementation is to check if the user's browser supports the geolocation or not because all the initial codes are based on this geolocation support.

```
1        // Check to see if this browser supports geolocation.
2 ▸      if (navigator.geolocation) {⟷} // all the codes goes in this statement
222 ▾    else {
223        alert("your browser doesnot support Geolocation API");
224      }
```

2. Creating a simple mobile browser based location aware treasure hunter game

Step 4 - Implementation- *index.ejs*

➔ To initialize map correctly, we need to get the user current location by using *getCurrentPosition()* function that is provided by Geolocation API. This function takes two important arguments. one of them is *position* that indicates we have our GPS working and all the codes that depends on the user position goes inside this statement. Second one is *error* that indicates that GPS has some problems.

```
1   navigator.geolocation.getCurrentPosition(
2       function(position) {...},
13      function(error) {
14      console.log("GPS error: ", error);},
15      {
16      timeout: (5 * 1000),
17      maximumAge: (1000 * 60 * 15),
18      enableHighAccuracy: true;}
19  );
```

2. Creating a simple mobile browser based location aware treasure hunter game

Step 4 - Implementation- *index.ejs*

➔ After initializing map correctly according to the user current location, now we need to watch user current location to play the game. Geolocation API provides a method called *watchPosition()* that uses GPS to track the user. Our game logic is implemented inside this method because we need to get user current location and watch it to enable user to play the game. More details about the code can be seen with detailed explanation in github. The most important thing to remember here is to notify the player if there is a any GPS error occurred during the game because it is the GPS that makes the game playable.

```
1  var positionTimer = navigator.geolocation.watchPosition(
2      function(position) {...},
66     function(error) {
67     alert("GPSSSSS Problem! Please check your GPS connection: ", error);}
68 );
```

2. Creating a simple mobile browser based location aware treasure hunter game

Step 4 - Implementation- *index.ejs*

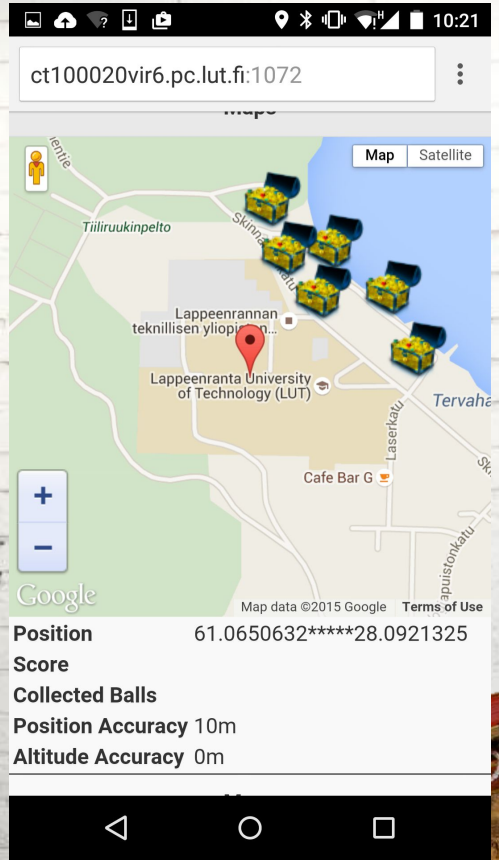
➔ Inside the *watchPosition(...),* if the player is close enough at least 40 meters to any treasure, we remove this treasure marker on the map and send the related data for the collected treasure to our game server (myapp.js) to store the data on database. Server side implementation and database model implementation already explained aforementioned sections. Following piece of code send the related data about any picked treasure to our server.

```
46              var name = loc[i][0];//name of the picked treasure
47              var lat = loc[i][1];//latitude of the picked treasure
48              var lon = loc[i][2];//longitude of the picked treasure
49              $.post("http://157.24.188.41:1072", {
50                  name: name,
51                  lat: lat,
52                  lon: lon;
53              }, function(res) {
54                  if (res === 'done') {
55                      console.log("login success");
56                      alert("success connection server");
57                  } else {
58                      alert("cannot connect server");
59                  }
60              });
```

2. Creating a simple mobile browser based location aware treasure hunter game

Step 5 - Final Design

➔ Now we finished first version of our mobile browser based location aware treasure hunter game. Next step is to take this concept forward and implement the game in details. For next version, we will create a database to store each user's information and game data with GPS related data to analyze. And also we will take advantage of the HTML5 Web Storage API to keep user data also inside the browser to make the game playable even a user loses data connectivity.



3. Creating RESTful mobile browser based location aware treasure hunter game

Application Use Case

In this version of the game, we will go one step further from our last version. We will build full RESTful location aware mobile treasure hunter game. We will have following logic in our game:

- ➔ First we need to identify the user via login page of our game. User needs to login to play the game. There is also an option to try the game as a guest.
- ➔ After the logging the game, user should be able to pick any tracks to play the game or can create his/her own track.
- ➔ Once the user choose a track to play the game, we need to design layout of the game according to the chosen/created track by the user.
- ➔ In the game logic, all the data for each user should be kept on the database.
- ➔ We should be able to overcome any data connectivity issue in our design.

3. Creating RESTful mobile browser based location aware treasure hunter game

Application Use Case

As we will build RESTful application, it is important to first understand *"what is Restful application?"* .

What is Restful application?

Simply, the RESTful API consists of two main concepts: **Resource**, and **Representation**. Resource can be any object associated with data, or identified with a URI (more than one URI can refer to the same resource), and can be operated using HTTP methods. Representation is the way you display the resource. For more information about the RESTful API please refer here.

Also there is a great tutorial that shows how to develop Node.js RESTful API step by step for any applications. It will be helpful to go through this tutorial to understand the concept of our game design.

3. Creating RESTful mobile browser based location aware treasure hunter game

Application Technology Stack

The game will be developed using the following technology stack:

- ➔ All technologies are explained in Slide 13.

- ➔ Bootstrap: is the most popular HTML, CSS, and JavaScript framework for developing responsive, mobile-first web sites. Bootstrap is used to develop our game layout for any screen resolution and also we aimed to solve our cross-browser compatibility issue by using Bootstrap..

3. Creating RESTful mobile browser based location aware treasure hunter game

Step 1- Install Node.js and MongoDB

Please go through the Slide #14

Google maps

3. Creating RESTful mobile browser based location aware treasure hunter game

Step 2 - Create Working Directory

➔ Once we create our application directory, "*Treasure_Hunter_Mobile_RESTful", next step is to create following structure inside this folder.*

working directory

```
- config
    - database.js
    - passport.js
- model
    - models.js
- node_modules
- public
    - css
    - js
    - images
- views
    - 404.ejs
    - create_tracks.ejs
    - demo.ejs
    - home.ejs
    - index.ejs
    - profile.ejs
    - tracks.ejs
- package.json
- server.js
```

➔ Or you can install the project directory from [github.](#)

3. Creating RESTful mobile browser based location aware treasure hunter game

Step 2 - Create Working Directory

Why we need these files ?

- ➔ ***config/database.js:*** This script file contains our database instance model to export for using in server. js script file to store data.
- ➔ ***config/passport.js:*** This script file enable us to register users when they log in our game.
- ➔ ***model/models.js:*** This script file contains our schema design for mongoose document and define our database design for each user.
- ➔ ***node_modules/..:*** under this file we will create all the dependencies for Node.js server.
- ➔ ***public/.. :*** This file contains all the static files to server for our game server like images and custom script and css files.
- ➔ ***package.json:*** We will define our dependencies for our game in this json file to install them under node_modules directory by npm command.

3. Creating RESTful mobile browser based location aware treasure hunter game

Step 2 - Create Working Directory

Why we need these files ?

- ➔ ***server.js:*** This is our Node.js server to handle all GET and POST requests, routing and data send/receive from client side.
- ➔ ***views/404.ejs:*** This our 404 not found web page template. If any error occurs we will render this page from client view.
- ➔ ***views/create_tracks.ejs:*** This template will be rendered if the user wants to create custom track.
- ➔ ***views/demo.ejs:*** This template will be rendered if the user wants to play the game w/o login.
- ➔ ***views/home.ejs:*** This template will be rendered to login the game.
- ➔ ***views/index.ejs:*** This is the index page of the game.
- ➔ ***views/profile.ejs:*** *This is the web page template for each individual users who log in the game to play.*
- ➔ ***views/tracks.ejs:*** Via this template the user can choose a track b/w available tracks or start to create custom track

3. Creating RESTful mobile browser based location aware treasure hunter game

Step 3 - Edit Dependencies

Now open *package.json* file and write following code.

This code identifies our dependencies for our project.

More information about package.json file and why to use please refer here.

```
{
    "name": "node-passport-ivb",
    "description": "demo of passport",
    "version": "0.0.3",
    "dependencies": {
        "express": "4.x.x",
        "passport": "*",
        "connect-flash": "*",
        "mongoose": "*",
        "ejs": "*",
        "passport-local": "*",
        "body-parser": "*",
        "morgan": "~1.0.0",
        "debug": "~0.7.4",
        "express-logger": "*",
        "cookie-parser": "*",
        "express-session": "*",
        "sql-injection": "*"
    },
    "scripts": {
        "start": "node server.js"
    },
    "main": "server.js",
    "devDependencies": {},
    "author": "",
    "license": "ISC"
}
```

3. Creating RESTful mobile browser based location aware treasure hunter game

Step 3 - Edit Dependencies

Why do we need these dependencies ?

- ➔ ***body-parser, ejs and mongoose:*** please see Slide #17
- ➔ **passport:** Passport is Express-compatible authentication middleware for Node.js. More details are here.
- ➔ **connect-flash:** The flash is a special area of the session used for storing messages and display it to the user when it is needed. More details are here.
- ➔ **passport-local:** It is a Passport strategy for authenticating with a username and password. more details are here.
- ➔ **morgan:** It is a HTTP request logger middleware for node.js. For more details please click here.
- ➔ **debug:** It is a debugging utility for the node.js. For more details please click here.

3. Creating RESTful mobile browser based location aware treasure hunter game

Step 3 - Edit Dependencies

Why do we need these dependencies ?

- ➔ **express-logger:** This middleware builds on the built-in Express logger. It will automatically rotate your logs and archive the old logs daily. More details are here.
- ➔ **cookie-parser:** Parse Cookie header and populate req.cookies with an object keyed by the cookie names. Optionally you may enable signed cookie support by passing a secret string, which assigns req.secret so it may be used by other middleware. For more details please click here.
- ➔ **express-session:** It create a session middleware for express. For more details please click here.
- ➔ **sql-injection:** This express module detects sql injection attacks and stops them by sending 403 as response. The module checks the query string, route parameters, and body for any sql injection related content. For more details please click here.

3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Install Dependencies

To install all dependencies in our package.json please refer Slide #18.

3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *server.js*

➔ First things, so let's define our dependencies in server.js:

```
1  var http = require("http");
2  var path = require("path");
3  var express = require("express");
4  var app = express();
5  var passport = require("passport");
6  var mongoose = require('mongoose');
7  var morgan = require('morgan');
8  var flash = require('connect-flash');
9  var bodyParser = require('body-parser');
10 var cookieParser = require('cookie-parser')
11 var logger = require("express-logger");
12 var session = require('express-session');
13 var database = require('./config/database.js');
14 var User = require('./model/models');
15 mongoose.connect(database.url);
16 require('./config/passport')(passport);
```

For more details about why to define dependencies please see Slide #19, #20 and #21.

3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *server.js*

➔ Before going through our implementation to handle GET and POST requests from the client side, it is important to understand from which pages we get those requests and for which purpose. The table below shows the way how we handle those requests.

URL	HTTP Verb	POST body	Results
/	GET	empty	render index.ejs page
/*	GET	empty	render 404.ejs page
/home	GET	empty	render home.ejs page and return signup error if occurs
/demo	GET	empty	render demo.ejs page
/track	GET	empty	render create_track.ejs page
/login	POST	JSON String	if login is successful direct to /track url, if it not direct to /home url and show message
/tracks	GET	empty	render tracks.ejs page
/profile	GET	empty	render profile.ejs page
/profile	POST	JSON string	save user game status data to database

3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *server.js*

➔ The important thing to allow clients to send and receive the data from server, we need to add following line codes. These codes can be used any server side- clients solutions for Node.js applications.

```
app.use(function (req, res, next) {
    // Website you wish to allow to connect
    res.setHeader('Access-Control-Allow-Origin', 'http://ct100020vir6.pc.lut.fi:1072');
    // Request methods you wish to allow
    res.setHeader('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT, PATCH, DELETE');
    // Request headers you wish to allow
    res.setHeader('Access-Control-Allow-Headers', 'X-Requested-With,content-type');
    // Set to true if you need the website to include cookies in the requests sent
    // to the API (e.g. in case you use sessions)
    res.setHeader('Access-Control-Allow-Credentials', true);
    // Pass to next layer of middleware
    next();
});
```

3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *server.js*

- After all the settings in our server.js, we need to handle our GET and POST requests from client side. As it is mentioned at Slide #22. We are not going to go through all the GET/POST requests implementation from the client side because in the previous version of the game all those basic implementations are explained in details. We will only go through the different implementations that we did not use during previous version of the game.
- All the codes can be found in github.

3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *server.js*

➔ One of the important implementation in our server.js that we need be careful about is that when the player wants to login the game to play, we need to authenticate the user and also check it in our database to see if the user already registered or not. Following code line uses *passport* middleware of Express to authenticate user and then if the user is not registered before, we will route the user to /tracks url to pick a track or create a track to play the game. If the user already registered we will route the user to /home url.

```
app.post("/login", passport.authenticate('login', {
    successRedirect: '/tracks',
    failureRedirect: '/home',
    failureFlash: true
}));
```

3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *server.js*

➔ *How to handle the user's game status data and save it to DB??*

Another important thing to remember during the implementation of the game is to handle game status data that is sent from the client side. In our game we store game status data in the format that is explained at Slide#XX. For any post request from the profile page, which is unique for each player, first we check if the player session exists or not by looking his/her session id in the DB. If the player is not found in the DB, we simply reset the session info and redirect the player to login page. However, if we found the player in the DB, we get the data from the client side and save it to the related field of the DB for each player.

The code on the next page shows the way how to handle this situation on the code level in server.js.

3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *server.js*

➔ *How to handle the user's game status data and save it to DB??*

```
app.post("/profile", function(req, res) {
    if (req.user) { // Check if session exists lookup the user in the DB by pulling their id from the session
        User.findOne({
            "_id": req.user.id
        }, function(err, user) {
            if (!user) {
                // if the user isn't found in the DB, reset the session info and redirect the user to the login page
                req.session.reset();
                res.redirect('/login');
                console.log("user not found");
            } else {
                console.log("user found");
                var name = req.body.name;
                var lat = req.body.lat;
                var lon = req.body.lon;
                var pos_accuracy = req.body.pos_accuracy;
                var pos_speed = req.body.pos_speed;
                var timestamp = req.body.pos_timestamp;
                var ball = {
                    name: name,
                    lat: lat,
                    lon: lon,
                    pos_accuracy: pos_accuracy,
                    pos_speed: pos_speed,
                    timestamp: timestamp
                };
                user.users.collected_balls.push(ball);
                user.users.save(function(err) {
                    if (err) throw err;
                    console.log('User saved successfully!');
                });
            }
        });
    } else { console.log("cannot found session");}
    res.sendStatus(200);
});
```

3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *config/passport.js*

- After all the implementation on server.js script file, now we need to configure our server for passport authentication and database schema design. We will start with passport.js file. In this file we **c**onfigure passport for local authentication, and use our configured passport to process our login form
- First we will load our dependencies for passport.js file.

```
1  var LocalStrategy = require('passport-local').Strategy;
```

- Secondly, we need to load up the user model that we are going to create to store each user data.

```
1  var User = require('../model/models');
```

3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *config/passport.js*

➔ Now we need persistent login sessions because passport needs ability to serialize and unserialize users out of session. To make it happen we will use following line codes in our passport.js file.

```
1   // used to serialize the user for the session
2   passport.serializeUser(function(user, done) {
3       done(null, user.id);
4   });
5   // used to deserialize the user
6   passport.deserializeUser(function(id, done) {
7       User.findById(id, function(err, user) {
8           done(err, user);
9       });
10  });
```

3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *config/passport.js*

➔ Next step in this file is that to get user data from the login form and open a session for this user by using session middleware of Express. In the game design we are using named strategies since we have one for *login*. When we get the login named form we will use below code to open a session for each individual user.

```
passport.use('login', new LocalStrategy({
    // by default, local strategy uses username and password, we will override with email
    usernameField: 'email',
    passwordField: 'username',
    passReqToCallback: true // allows us to pass back the entire request to the callback
},
```

➔

3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *config/passport.js*

➔ Once we get the username and email from the login form, we need to find a user whose email is the same as the forms email and we will be checking to see if the user trying to login already exists.

```
function(req, email, username, done) {
    User.findOne({
        'users.email': email
    }, function(err, user) {
        if (err) // if there are any errors, return the error
        return done(err);
        // check to see if theres already a user with that email
        if (user) {
            return done(null, false, req.flash('signupError', 'Email is already taken.'));
        } else {
            // if there is no user with that email create the user
            var newUser = new User();
            // set the user's local credentials
            newUser.users.email = email;
            newUser.users.username = username;
            // save the user
            newUser.save(function(err) {
                if (err) throw err;
                return done(null, newUser);
            });
        }
    });
}));
```

3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *config/database.js*

➔ When we finished our passport.js configurations for logging the game, the next step is configure our database. As we already are calling this file in server.js. Now we just have to set it up our DB url.

```
module.exports = {

    'url': 'mongodb://157.24.188.41/users'
};
```

3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *model/models.js*

➔ Create User Model

We will create our user model for this tutorial series. Our user will have the ability to be linked to a local account. For local accounts, we will be keeping **email, username, score** of the each played game and each collected treasure detail that includes **name** of the treasure, l**atitude** and **longitude** of the treasure, **GPS accuracy** and **speed** of the collected treasure location and **timestamp** that show when the treasure picked. We will keep the details of the each collected treasure on JSON array. We will use mongoose schemas to make our design much more simpler and understandable.

The code on the next page shows the implementation of the above mentioned user model design in models.js file.

3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *model/models.js*

➔ Create User Model

```
// load the things we need
var mongoose = require('mongoose');
// define the schema for our user model
var user = mongoose.Schema({
    users: {
        email: String,
        username: String,
        score: Number,
        collected_balls: [{
            name: String,
            lat: Number,
            lon: Number,
            pos_accuracy: String,
            pos_speed: String,
            timestamp: String
        }]
    }
});
// create the model for users and expose it to our app
module.exports = mongoose.model('User', user);
```

3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *views/…*

We now finished our server side implementation and it is time to move forward and implement client side code. Client side of the game is coded with HTML5, jQuery Mobile and Bootstrap. By using different JS libraries we design each page for different screen resolutions(responsive design).

All the codes with detailed explanation can be found in github. We will only go through important part of the codes that play an important role on game logic.

3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *views/index.ejs*

This is our index page that we access when we enter the our game url on any browser. This page contains information about the game. And when the player press the “Start Game” button that will direct the player to “home.ejs” page via our server.js script file. Inside the <a>.. tag in html code, we specify our route page as a value of href.

```
1
2  <a href="home" class="btn btn-dark btn-lg">
3          Start Game
4  </a>
```

3. Creating RESTful mobile browser based location aware treasure hunter game

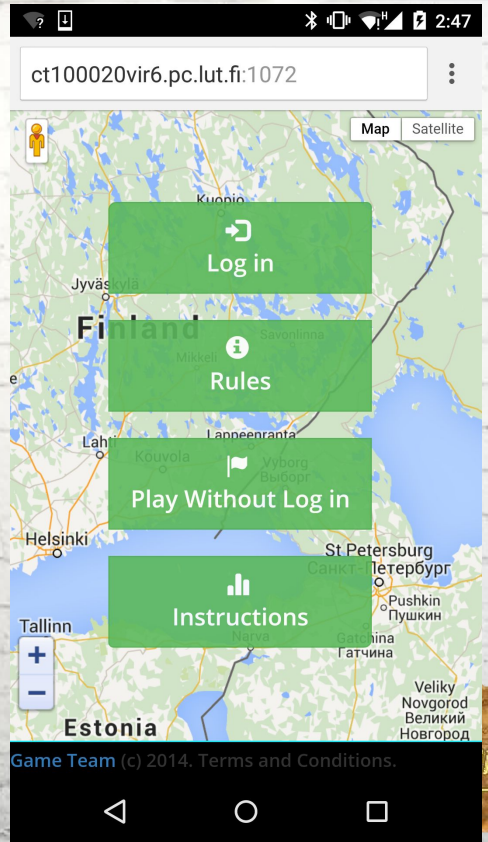
Step 4 - Implementation- *views/home.ejs*

In this page provides different options to player as it can be seen in the screenshot of the page. We put the Google Map as a background of the page by simply initializing the z index to 0 in css.

```
<style>
    #map-canvas {
        /* Set z-index to 0 as it will be on a layer below the contact form */
        z-index: 0;
    }
</style>
```

There are four buttons that lead the user different pages and provide different information about the game. Those are as follows.

- ➔ *Log in: to login the game*
- ➔ *Rules: to read the game rules*
- ➔ *Play Without Log in: to play the game without login*
- ➔ *Instructions: to read instructions about the game*



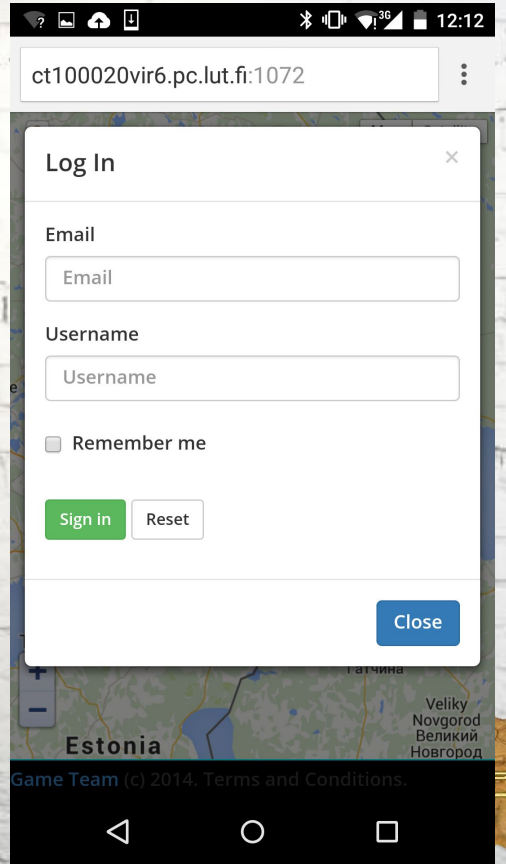
3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *views/home.ejs*

➔ *Log in*

When the user clicks the "Log in" button, that opens a login form to sig in the game as on the screenshot. To login the game user has to provide email and username to open an account on game server. If the user already exists on the DB. The following code display error message on the login form. This code also provides the concept to get data from server. If the player logins successfully via clicking "Sign in" button,

```
<div class="form-group">
    <div class="col-sm-12"><% if (message.length >0) { %>
        <script>
            $('#myModalNorm').modal('show');
        </script>
        <div class="alert alert-danger"><%= message %></div><% } %>
    </div>
</div>
```

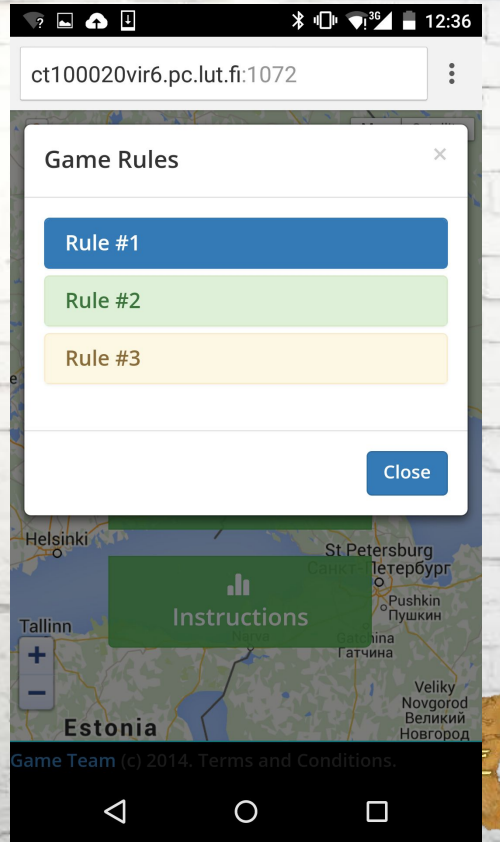


3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *views/home.ejs*

➔ Rules

This modal displays the game rules for the player.

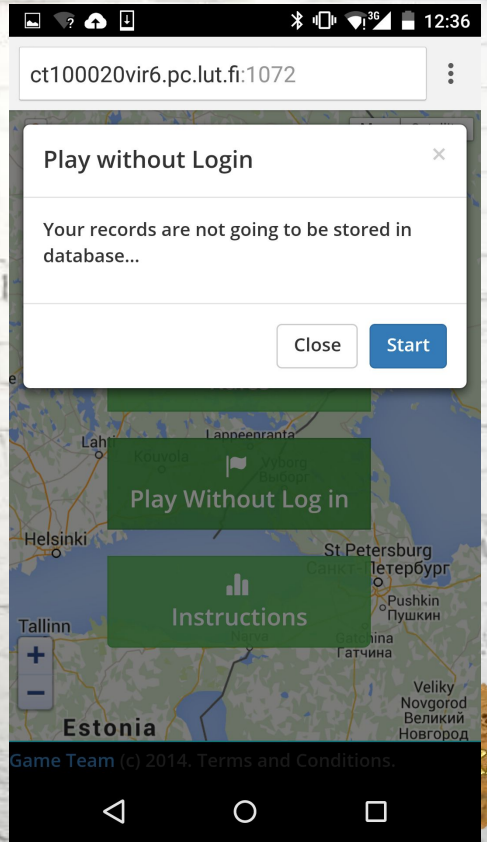


3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *views/home.ejs*

➔ Play Without Log in

if the player wants to play the game as a guest, it can be done simply by clicking this button. After clicking this button a modal is going to be opened and display a message as on the screenshot. When the user clicks "Start" button, this button directs the player to demo.ejs page.

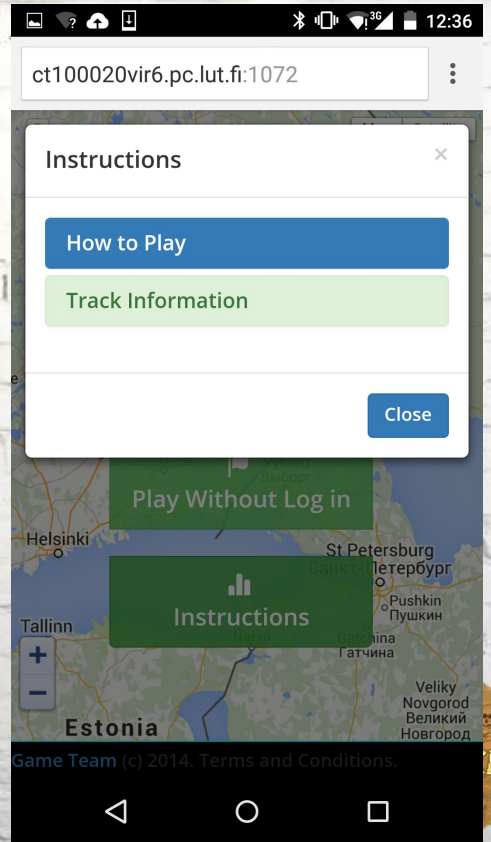


3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *views/home.ejs*

➔ Instructions

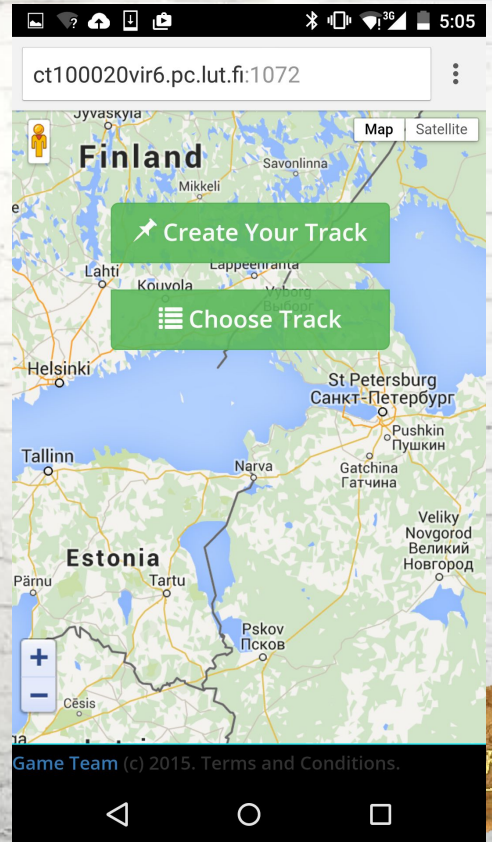
By clicking this button, user can access the information about "How To Play" the game and available "Track Information" as it can be seen on screenshot.



3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *views/tracks.ejs*

When the user logins successfully, this page opens and asks the user to choose a track or create his/her own custom track to play the game.

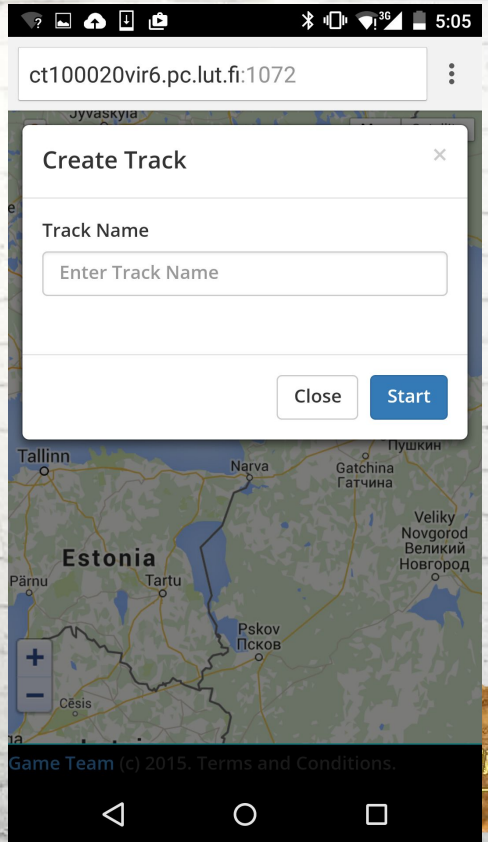


3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *views/tracks.ejs*

➔ *Create Your Track*

When the user clicks this button to create a custom track, user sees this modal to fill in. To create a custom track user has to enter track name.

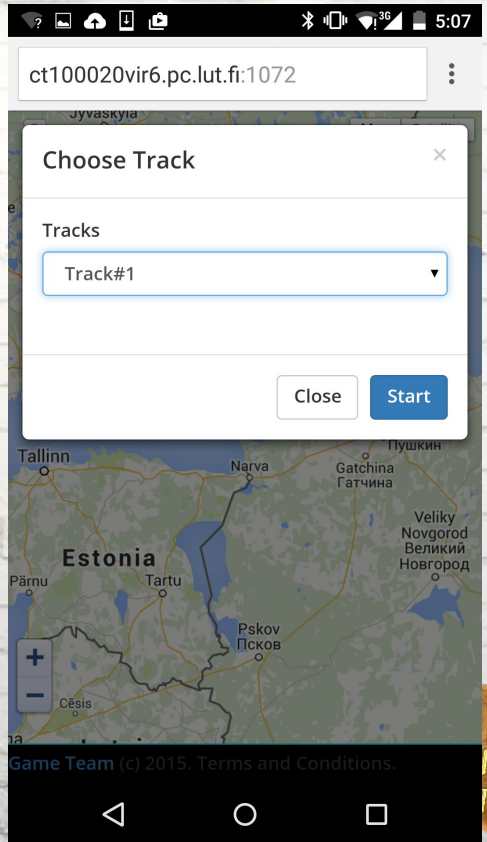


3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *views/tracks.ejs*

➔ *Choose Track*

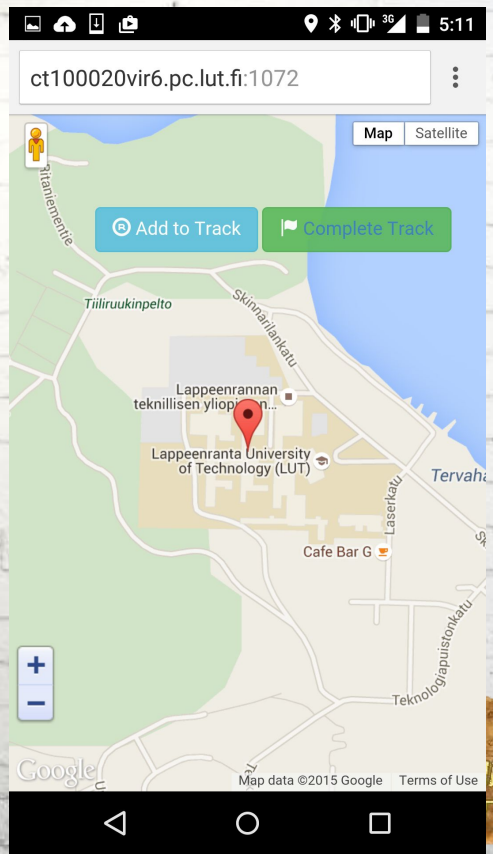
When the user clicks this button to choose a track inside all available tracks, user sees this modal to pick a track among them.



3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *views/create_tracks.ejs*

This page enables user to create a custom track by walking around the route in which user wants to play. On the layout of the page there is two buttons, which are "Add to Track" and "Complete Track". Functionality of these buttons are explained on the next slides



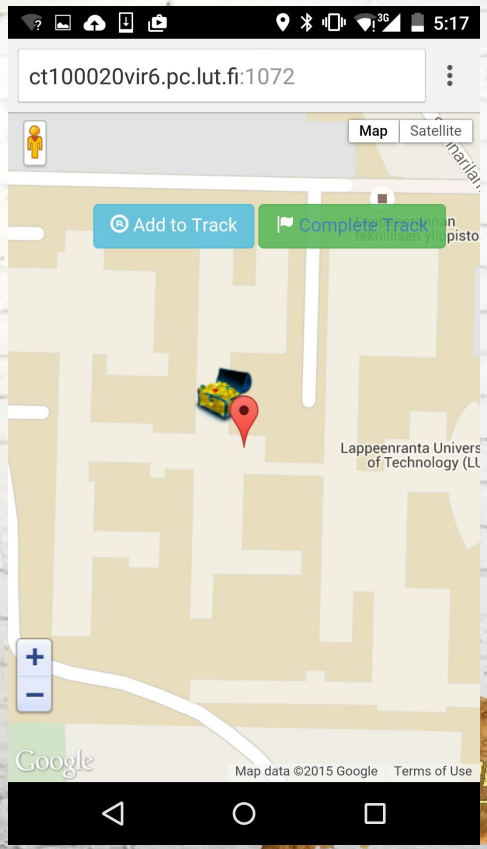
3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *views/create_tracks.ejs*

➔ *Add to Track*

By clicking this button, user can mark the current location as a location of the treasure on the track. When this button is clicked, we put the location information on the array to store in localstorage. The code below simply provides this concept to us on the code level. By taking advantage of localstorage, we also don't lose marked positions even we closed the browser.

```
function(position) {
    locMarker = addTreasureMarker(
    position.coords.latitude,
    position.coords.longitude);
    var point = {
        id: locMarker.id,
        lan: position.coords.latitude,
        lon: position.coords.longitude
    };
    track_locations.push(point);
    localStorage.setItem("loc_store_tracks", JSON.stringify(track_locations));
},
```



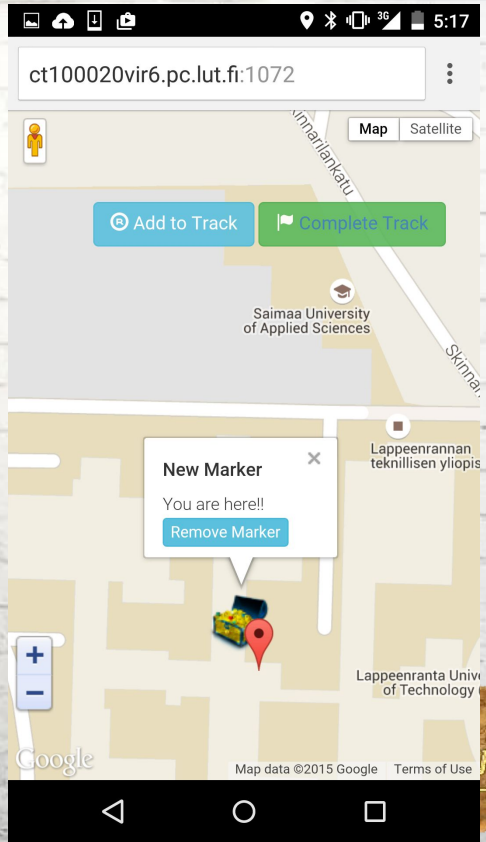
3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *views/create_tracks.ejs*

➔ *Add to Track*

We also enable user to remove markers basically clicking on them. In this way, user can edit track as he/she wishes. To do so, user needs to click on the treasure marker that user wants to remove from track. After clicking on it, information window of the marker is going to be opened and user can remove it by simply clicking on “Remove Marker” button. The code for this functionality is below.

```
var removeBtn = contentString.find('button.remove-marker')[0];
google.maps.event.addDomListener(removeBtn, "click", function(event) {
    for (var i = 0; i < track_locations.length; i++) {
        if (track_locations[i].id == marker.id) {
            marker.setMap(null);
            track_locations.splice(i, 1);
            localStorage.removeItem("loc_store_tracks");
            localStorage.setItem("loc_store_tracks", JSON.stringify(track_locations));
            return;
        }
    }
});
```



3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *views/create_tracks.ejs*

➔ *Complete Track*

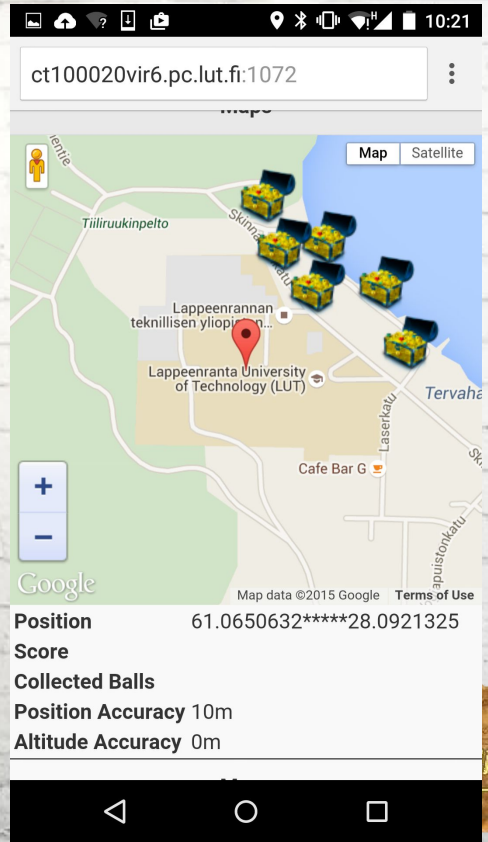
When clicking this button, user finishes to create the track and after initial settings on the server side, profile.ejs page is rendered to play the game.

3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *views/profile.ejs*

In this page, user plays the game based on the picked or created track. All the GPS and localstorage codes are located inside this page. We will go through important implementations in this file which are as follows:

- ➔ *How to get track data from the server and put on the map*
- ➔ *How to use local storage*
- ➔ *How to use geolocation in HTML5*
- ➔ *How to send and receive data from the server*



3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *views/profile.ejs*

➔ *How to get track data from the server and put on the map*

When the user choose a track to play the game, we need take track data from the server. Following code takes treasure locations on the track and put them on the google map.

```
var loc = <% -JSON.stringify(loc) %>
var myOptions = {
  center: new google.maps.LatLng( <%= lat %> , <%= lon %> ),
  zoom: <%= zoom %> ,
  mapTypeId: google.maps.MapTypeId.ROADMAP
};
var map = new google.maps.Map(document.getElementById("map-canvas"), myOptions);
```

3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *views/profile.ejs*

➔ *How to use local storage*

In this part we will go through usage of local storage in our code. During the game we store data of each collected treasure on the localstorage of the browser to prevent any data losses when data connection is gone or the player closes the mobile browser. Following code creates our local storage and saves collected treasure information on localstorage.

```
//create local storage for locations
function createAndSaveLocStorage(name, lat, lon, pos_accuracy, pos_speed, pos_timestamp, p) {
    var data = {
        name: name,//name of the picked treasure
        lan: lat, //latitude of the picked treasure
        lon: lon, // longitude of the picked treasue
        pos_accuracy: pos_accuracy, //GPS position accuracy
        pos_speed: pos_speed, // GPS position speed
        pos_timestamp: pos_timestamp, // timestamp
        collected: 1,//that indicates the treasure is picked.
        sendserver: p //if p=1 that means we sent this treasure info to server but if p=0 we could not sent it.
    };
    ArrayObject.push(data);
    localStorage.setItem("locations", JSON.stringify(ArrayObject));
}
```

3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *views/profile.ejs*

➔ *How to use local storage*

At the beginning of the game, we check if the player has previous session on this track via localstorage. As we store all the collected treasure data on localstorage, we check if there is still some unpicked treasure is available or not. Following line codes in our application checks if any locations data available on localstorage or not.

```
1        // check local storage for any previous itemss
2        if (localStorage.getItem("locations") !== null) {
3            alert("welcome back  the game");
4            //get the values from previous session
5            document.getElementById("score_cell").innerHTML = localStorage.getItem("score");
6            ArrayObject = JSON.parse(localStorage.getItem("locations"));
7            var myArrayObject = JSON.parse(localStorage.getItem("locations"));
8            //for debugging
9            for (var i = 0; i < myArrayObject.length; i++) {...}
13           for (i = 0; i < loc.length; i++) {...}
49       } //end if statement for checking localstogare
50       else { //if there is nothing on localstorage
51           alert("you are new in the game");
52           ///ad markers on the map
53           for (i = 0; i < loc.length; i++) {...}
69       }
```

3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *views/profile.ejs*

➔ *How to use geolocation in HTML5*

To check how to use Geolocation functions please refer Slide #31, #32, #33.

3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *views/profile.ejs*

➔ *How to send and receive data from the server*

We send each collected treasure data to server via ajax POST. Whenever the player collects a treasure we just try to send related data for the treasure to the server to store in DB.

```
var pos_accuracy = Math.round(position.coords.accuracy, 1) + "m";//get position accuracy
var pos_speed = position.coords.speed + "m/s";// //get position speed
var pos_timestamp = (new Date(position.timestamp)).toLocaleString();//get the timestamp
//get the collected treasure information that includes name,latitude and longitude
var name = loc[i][0];
var lat = loc[i][1];
var lon = loc[i][2];
//send data to nodejs server to store
$.ajax({
    url: "http://ct100020vir6.pc.lut.fi:1072/profile",
    method: 'post',
    data: {
        name: name,
        lat: lat,
        lon: lon,
        pos_accuracy: pos_accuracy,
        pos_speed: pos_speed,
        pos_timestamp: pos_timestamp;
    },
    success: function(data, textStatus, jqXHR) {...},
    error: function(data, textStatus, jqXHR) {...}
});
```

3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *views/profile.ejs*

➔ *How to send and receive data from the server*

In the success callback inside ajax post function, we put the data also on localstorage via aforementioned *createAndSaveLocStorage()* function. We mark *sendserver* field as true. After that we also check our localstorage if there is any data which has not been sent to server. Please see *updateServer()* *function implementation. In the error callback, which indicates we could not send any data to server, we put data on localstorage also but we mark sendserver field as false.*

```
success: function(data, textStatus, jqXHR) {
    alert("Data Added!!!");
    createAndSaveLocStorage(name, lat, lon, pos_accuracy, pos_speed, pos_timestamp, true);
    updateServer();
},
error: function(data, textStatus, jqXHR) {
    createAndSaveLocStorage(name, lat, lon, pos_accuracy, pos_speed, pos_timestamp, false);
    alert("Data could not be added!");
}
```

3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *views/profile.ejs*

➔ *How to send and receive data from the server- updateServer() function*

This function checks if there is any data which has not been sent to server due to the data connectivity issues. If the value of sendserver flag in the array, which contains picked treasures data, is false then we send this data again to server. After we send the data to server we also need to update the localstorage.

```
function updateServer() {
    for (var i = 0; i < ArrayObject.length; i++) {
        if (ArrayObject[i].sendserver === false) {
            $.ajax({
                url: "http://ct100020vir6.pc.lut.fi:1072/profile",
                method: 'post',
                data: {
                    name: ArrayObject[i].name,
                    lat: ArrayObject[i].lan,
                    lon: ArrayObject[i].lon,
                    pos_accuracy: ArrayObject[i].pos_accuracy,
                    pos_speed: ArrayObject[i].pos_speed,
                    pos_timestamp: ArrayObject[i].pos_timestamp;
                },
                async: false,
                success: function(data, textStatus, jqXHR) {
                    ArrayObject[i].sendserver = true;
                    alert("Data updated successfullyy!!!");
                },
                error: function(data, textStatus, jqXHR) {
                    alert("Couldnot updated!");
                }
            });
        }
    }
    localStorage.setItem("locations", JSON.stringify(ArrayObject));
}
```

3. Creating RESTful mobile browser based location aware treasure hunter game

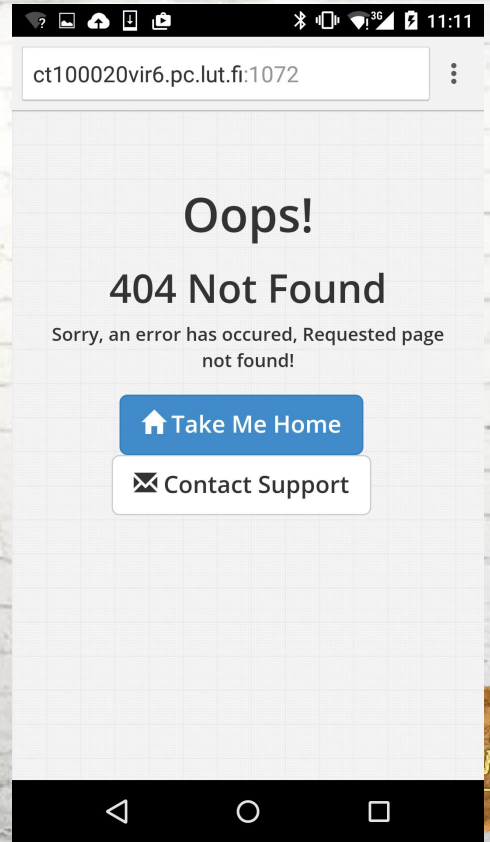
Step 4 - Implementation- *views/demo.ejs*

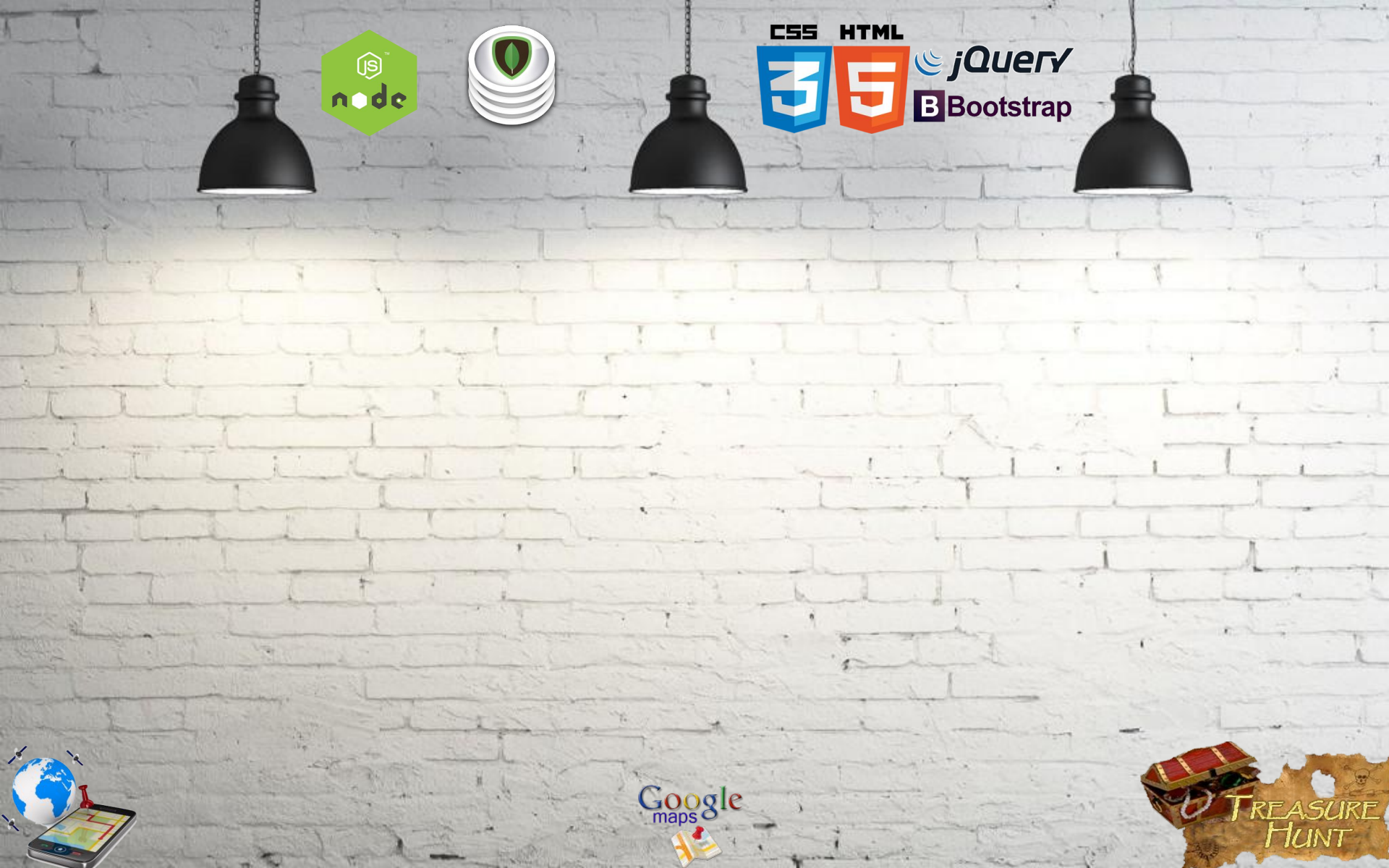
This page is rendered when the user wants to play the game without login. All the codes in this page are as same as in the profile.ejs page. The only difference is that we don't store user information like email and username on the DB.

3. Creating RESTful mobile browser based location aware treasure hunter game

Step 4 - Implementation- *views/404.ejs*

This page is rendered if any error occurs during the gameplay.




node
CSS
HTML
jQuery
Bootstrap
Google
maps
Treasure
Hunt