

## Location Based Quiz – Application Document

<b>Github and Webpage Details .....</b>	<b>1</b>
<b>1. Overall System Aims .....</b>	<b>2</b>
<b>2. Technical Documentation .....</b>	<b>2</b>
2.1 Mobile Application .....	2
2.2 Web Application .....	7
2.3 Testing.....	10
<b>Appendix A: Mobile Application Code (Quiz).....</b>	<b>12</b>
File: appActivity.js .....	12
File: index.html.....	19
<b>Appendix B: Web Application Code (Questions).....</b>	<b>23</b>
File: appActivity.js .....	23
File: uploadData.js .....	25
File: index.html.....	26
<b>Appendix C: Server Code .....</b>	<b>30</b>
File: httpServer.js .....	30
<b>Appendix D: SQL Table Creation Code .....</b>	<b>34</b>

### Github and Webpage Details

The code for the three parts of the system can be found in the following Github account:

User: nikscrits → <https://github.com/nikscrits>

The three repositories that should be used for testing each component are as follows:

Quiz (Mobile application) → <https://github.com/nikscrits/quiz>

Questions (Web application) → <https://github.com/nikscrits/questions>

Server → <https://github.com/nikscrits/server>

The URL for the web application is → <http://developer.cege.ucl.ac.uk:31288/>

The user guides for each can be found in the README files of the corresponding repository as both a html page and PDF file.

The technical guide for the whole system can be found in the server repository given above.

The question and answers information can be found in the following database:

User → user11

Schema → public

Tables → quizquestions, quizanswers

## 1. Overall System Aims

This system comprises of two applications: a web application, and a mobile application. Together, these applications aim to create a functional location-based quiz. The overall purpose of this application is to encourage users to explore new places, whilst also learning something about the locations they visit through the answering of multi choice trivia questions.

The web application is responsible for administrator tasks, such as the creation and then viewing of quiz questions. These questions are each associated with a specific geographic location, therefore they are set by selecting a point on an interactive map, and then consequently viewed as markers on the same map. All of the questions that have been created by the administrator are stored in a database.

The mobile application is the user interface portion of the system and can be used on an android device. The questions that have been created by the administrator are shown as markers on a map, in addition to the user's current location. Using GPS, the user's location is tracked, thus allowing only questions within 20m of their current location to be answered. After answering a question, the user is informed as to whether they answered correctly or not, and the answer data is sent to a database.

## 2. Technical Documentation

### 2.1 Mobile Application

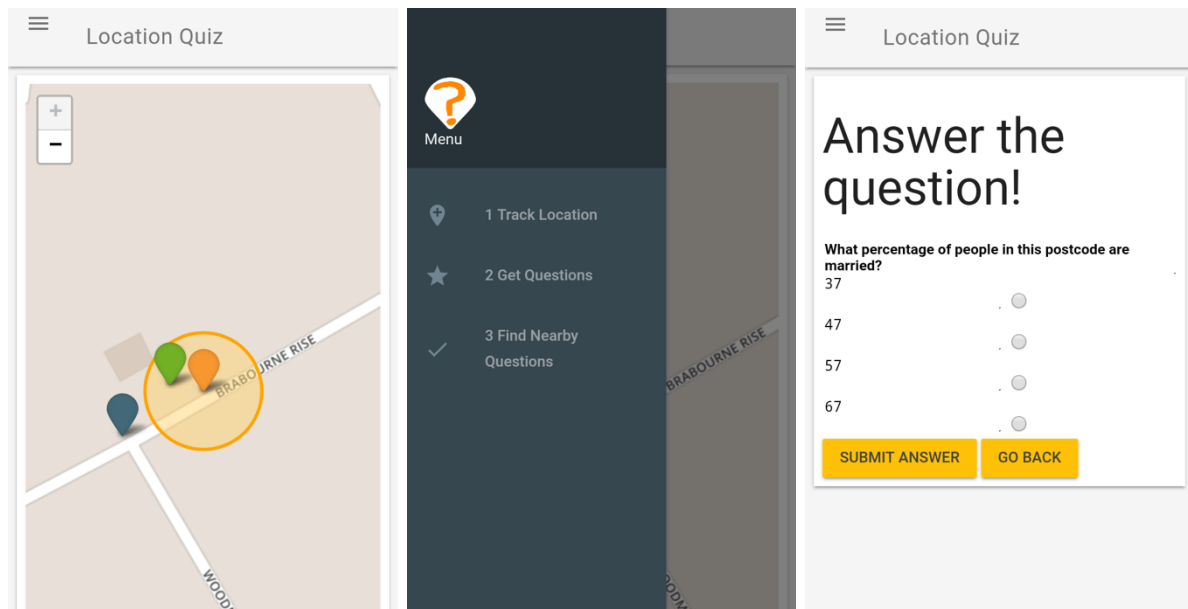
#### Requirements / Task Specification

The mobile application carries out the following tasks:

- Uses a Material Design Lite template to display a simple and user-friendly interface containing a map implemented using the Leaflet API.
- Has three menu options that when pressed have the following functions:
  - Track the user's current location
  - Get the questions from the database
  - Work out which questions are within 20m of the user and allow the user to click those markers and answer a question
- Displays and then tracks the user's current location using the phone's geolocation capabilities. This location is then shown as an orange marker on the map and the map zooms into this location.
- Downloads the questions from the `httpServer.js` as a GeoJSON file and then plots their locations using blue markers on the Leaflet map.
- Checks through each of the question markers and colours those within 20m of the user's current location to purple.
- Allows the user to click on only the purple markers (those within 20m) and then displays the question associated with that marker.
- Ensures the user selects one of the potential answers before checking to see if they got the answer correct and then informing them of the result.
- Uploads the user's answer and the question information to the database using the `httpServer.js`

## User Interface

The mobile application interface uses a Material Design Lite template. This means that the design and styling is consistent throughout and aims to be user friendly. There are three main components to the user interface: The map, the menu, and the question answering form (Figure 1).



*Figure 1: The three main components of the mobile application's user interface: The map (left), the menu (centre), and the question answering form (right).*

These three components are responsible for allowing the user to complete all the tasks associated with the application.

### **1. The map**

- a. Displays a leaflet map that zooms in and out depending on what is being shown.
- b. Shows an orange marker of the user's current location, which is tracked and updated depending upon the user's movement.
- c. Shows markers displaying the questions from the database, coloured initially blue, and then purple if they are within a 20m proximity to the user.

### **2. The menu**

- a. Displays three menu options, which when clicked carry out the associated task.

### **3. The question answering form**

- a. Presents the question from the marker that has been clicked and the four potential answers.
- b. Ensures the user has selected an answer before submitting.
- c. Shows an alert informing the user whether they got the answer correct.

## Error Handling

The following error handling measures have been taken within the mobile application:

- If the app is opened on a web browser or on a device without GPS tracking the geolocation functionality will not supported. An alert will be displayed stating 'Geolocation is not supported' to inform the user of this error.
- If the user attempts to submit a question without selecting an answer an alert will show saying 'Please select an answer.'. This prevents the user from encountering an error as this data would not successfully be able to be added to the database.

## Future Updates

There are a number of updates that would be made in the future to improve this mobile application and increase functionality. Some improvements include:

- Allowing the user to create an account and then log in so they can track their score.
- Preventing the user from answering questions they have previously answered and got the answer correct.
- Automatically updating the question markers that can be answered when the user changes location. Currently, the user is required to click the '3 Find Nearby Questions' menu option when they change location.

## Code Overview

The overview of the code for the mobile application will be presented according to the two key files: Index.html, and appActivity.js.

### **File name: Index.html**

As this file is a .html file and is responsible for the layout of the application, the sections in the following table refer to specific classes or divs within the code.

Code Section	Purpose
<code>&lt;div class="demo-drawer mdl-layout__drawer mdl-color--blue-grey-900 mdl-color-text--blue-grey-50"&gt;</code>	- Display a title for the menu bar and the application icon.
<code>&lt;nav class="demo-navigation mdl-navigation mdl-color--blue-grey-800"&gt;</code>	- To hold the three menu options available within this application. - Gives each of the menu options a graphical icon and allow them to be clicked on.

	<ul style="list-style-type: none"> <li>- When clicking each of the menu options, the relevant function in the appActivity.js file will be called to execute the task.</li> </ul>
<pre>&lt;div id="mapid" style="width: 100%; height: 100%;"&gt;&lt;/div&gt;</pre>	<ul style="list-style-type: none"> <li>- To hold the Leaflet map.</li> </ul>
<pre>&lt;div id="questionsection" style ="display:none"&gt;</pre>	<ul style="list-style-type: none"> <li>- To hold the question answering form, which is initially hidden when the application is opened.</li> <li>- To show the question, four potential answers, radio buttons for selected answer, and two buttons.</li> </ul>

### File name: appActivity.js

The app activity file is a javascript file responsible for carrying out each of the application tasks. The following table will detail the purpose of each function in this file.

Function(s)	Purpose
trackLocation()	<ul style="list-style-type: none"> <li>- Finds the user's current location using the geolocation function of the phone.</li> <li>- Pans the leaflet map to display the current location of the user.</li> <li>- Keeps tracking the location of the user as they move position.</li> </ul>
showPosition()	<ul style="list-style-type: none"> <li>- Creates an orange marker to display the user's current location on the map.</li> <li>- Creates a circle marker to represent the 20m distance from the user's location where questions can be answered.</li> <li>- Pans the leaflet map to display the current location marker.</li> </ul>
getQuestions() questionResponse()	<ul style="list-style-type: none"> <li>- Uses a XMLHttpRequest to get the questions set by the administrator from the database.</li> <li>- When the question data has been downloaded, the loadQuestionLayer() function is called with the question data passed in.</li> </ul>
loadQuestionLayer()	<ul style="list-style-type: none"> <li>- Takes the question data that has been downloaded from the database and converts it into JSON format.</li> <li>- Creates a layer formed of blue markers that represent the location of each of the questions.</li> <li>- Fits the map to the show all of the available questions.</li> <li>- Adds all of the markers to a global array so their information can be accessed globally.</li> </ul>

checkQuestion()	<ul style="list-style-type: none"> <li>- Iterates around each question point in the global array and calls the getDistance() function to check if each is located within 20m of the user's current location.</li> <li>- If it is within 20m, the colour of the marker is changed to purple and it can be clicked on.</li> <li>- If it is not within 20m, the colour remains blue, and a pop-up is shown when clicked saying 'Can't Answer! This question is too far away.'</li> </ul>
getDistance()	<ul style="list-style-type: none"> <li>- The user's current longitude and latitude, along with the marker's longitude and latitude are passed in and the distance between the two locations is calculated and returned.</li> </ul>
onClick()	<ul style="list-style-type: none"> <li>- Calls the showClickedQuestion() function passing in the information about the marker that was clicked.</li> </ul>
showClickedQuestion()	<ul style="list-style-type: none"> <li>- Changes the interface by showing the div created in the index.html file that holds the question answering form, whilst hiding the div that holds the leaflet map.</li> <li>- Populates the text areas with the relevant question and answer information for the marker that was clicked.</li> </ul>
validateAnswer()	<ul style="list-style-type: none"> <li>- Checks that the user has selected an answer and if not shows an alert reminding them to do so.</li> <li>- Calls the answerResponse() function passing in the number of the answer that has been given and its value.</li> </ul>
answerResponse()	<ul style="list-style-type: none"> <li>- Checks the user submitted answer against the correct answer and informs the user whether they got the answer correct or incorrect.</li> <li>- Calls the submitAnswer() function padding in the answer, its value and a Boolean specifying whether the user got the answer correct or not.</li> </ul>
submitAnswer()	<ul style="list-style-type: none"> <li>- Creates a string for the data that will be uploaded to the database and passes this into the processData() function.</li> </ul>
processData() dataUploaded()	<ul style="list-style-type: none"> <li>- Uses a XMLHttpRequest to send the question answer from the user to the database.</li> <li>- After this is done, an alert is shown informing the user that the answer has been successfully uploaded.</li> <li>- Then the returnToMap() function is called so the hidden map div can be made visible, and the marker is changed to either green or red depending on if the user was correct.</li> </ul>
returnToMap()	<ul style="list-style-type: none"> <li>- Changes the interface back to its starting view by showing the div created in the index.html file that holds the map, whilst hiding the div that holds the question answering form.</li> </ul>

## 2.2 Web Application

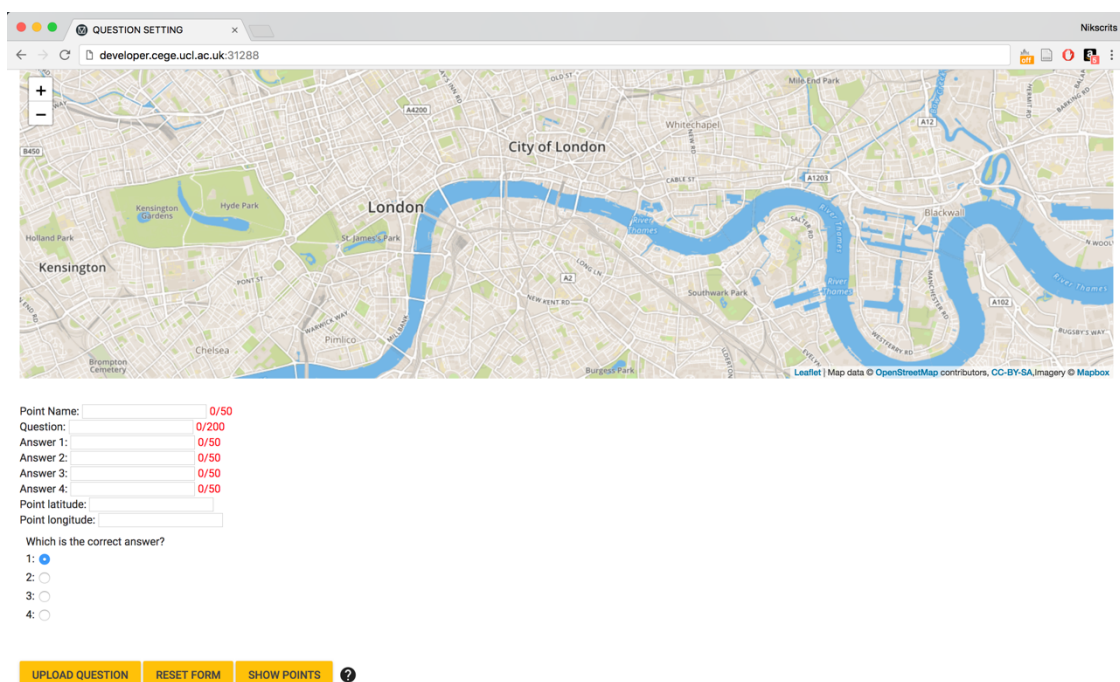
### Requirements / Task Specification

The web application carries out the following tasks:

- Uses a Material Design Lite template to display a simple and user-friendly interface containing a map implemented using Leaflet API and a form that the user can fill out with the question they wish to add.
- Displays three buttons that have the following functions:
  - Submit the question to the database
  - Reset the form so the text boxes are blank
  - Show the previously set questions as markers on the map
- Downloads the questions already set from the httpServer.js as a GeoJSON file and then plots their locations using orange markers on the Leaflet map.
- Allows the user to click on the question markers and see a pop-up showing the point name and the question that has been set.
- Counts the characters in the text boxes so the user knows how many are left.
- Ensures the user fills in each of the text boxes with the relevant question data before pressing submit so that blank information is not sent to the database.
- Uploads the new question to the database using the httpServer.js

### User Interface

The web application interface uses a Material Design Lite template. This means that the design and styling is consistent. There are two main components to this interface: The map and the question details form (Figure 2).



The screenshot displays a web browser window titled "QUESTION SETTING" with the URL "developer.cege.ucl.ac.uk:31288". The interface is split into two main sections. The top section features a map of London, showing the River Thames and surrounding areas like Kensington, Chelsea, and the City of London. The bottom section contains a form for setting a question. The form includes text input fields for "Point Name:", "Question:", "Answer 1:", "Answer 2:", "Answer 3:", "Answer 4:", "Point latitude:", and "Point longitude:". Each text field has a character count indicator on the right (e.g., "0/50" for Point Name, "0/200" for Question, and "0/50" for each answer). Below the form fields is a radio button group labeled "Which is the correct answer?" with options 1, 2, 3, and 4. At the bottom of the form are three buttons: "UPLOAD QUESTION", "RESET FORM", and "SHOW POINTS", followed by a help icon.

*Figure 2: The interface for the web application. The two main components – the map and the question details form – can be seen in the top and bottom sections, respectively.*



These two components are responsible for allowing the user to complete all the tasks needed to upload a new question to the database.

**1. The map**

- a. Displays a leaflet map that zooms in and out depending on what is being shown and to provide a closer look at the question locations.
- b. Shows orange markers for the questions already set once the 'Show Points' button has been clicked.

**2. The question details form**

- a. Displays a form made up of text boxes and radio buttons that can be filled out by the user.
- b. Allows the user to click on any point on the map to populate the 'point latitude' and 'point longitude' boxes.
- c. Counts the characters in each text box so the user knows how many they have left.
- d. Displays three buttons that can be clicked on to upload the question to the database, reset the form, or show the questions already set.
- e. Shows a question mark icon that takes the user to the user help guide.

Error Handling

The following error handling measures have been taken within the web application:

- When the button 'Submit Question' has been clicked, the text box values are checked to ensure a value has been entered and blank information is not sent to the database.
- The text boxes each have a character limit, so the user cannot enter a value too long for the database to handle. Additionally, next to each text box there is a character counter letting the user know how many characters they have left.
- The longitude and latitude text boxes are populated by clicking on the map to avoid the user from typing in an incorrect value or mixing up the two values.
- The question ID value is created by the database rather than being typed in by the user. This prevents primary key errors arising from records not being unique.

Future Updates

There are a number of updates that could be made in order to increase the functionality of the web application. For example:

- The ability to edit the questions once they are downloaded from the database.
- The ability to delete a question once they are all downloaded from the database.
- An option to cancel the question being sent to the database if the user realises something was incorrect when the final alert is shown.

## Code Overview

### **File name: Index.html**

As this file is a .html file and is responsible for the layout of the webpage, the sections in the following table refer to specific classes or divs within the code.

Code Section	Purpose
<pre>&lt;div id="mapid" style="width: 100%; height: 400px;"&gt;&lt;/div&gt;</pre>	<ul style="list-style-type: none"><li>- To hold the Leaflet map.</li></ul>
<pre>&lt;div class = "mdl-grid" id = "form"&gt;</pre>	<ul style="list-style-type: none"><li>- To hold each of the text boxes and their labels where the user enters the question information.</li><li>- Each of the text boxes call the function countChars() from the appActivity.js file when text is entered. This counts the characters entered by the user.</li></ul>
<pre>&lt;div class = "mdl-grid" id = "buttons"&gt;</pre>	<ul style="list-style-type: none"><li>- To hold each of the necessary buttons.</li><li>- When clicking each of the buttons, the relevant function in the appActivity.js file will be called to execute the task.</li></ul>

### **File name: appActivity.js**

The app activity file is a javascript file responsible for carrying out each of the web page tasks. The following table will detail the purpose of each function in this file.

Function(s)	Purpose
loadMap()	<ul style="list-style-type: none"><li>- Loads the tiles to display a leaflet map.</li></ul>
mymap.on()	<ul style="list-style-type: none"><li>- Gets the longitude and latitude for where the user clicked the leaflet map and populates the text boxes for the question location.</li></ul>
resetForm()	<ul style="list-style-type: none"><li>- Makes each of the text areas blank so the question information can be entered from scratch.</li></ul>
getQuestions() questionResponse()	<ul style="list-style-type: none"><li>- Uses a XMLHttpRequest to get the questions already set by the administrator from the database.</li><li>- When the question data has been downloaded, the loadQuestionLayer() function is called with the question data passed in.</li></ul>

loadQuestionLayer()	<ul style="list-style-type: none"> <li>- Takes the question data that has been downloaded from the database and converts it into JSON format.</li> <li>- Creates a layer formed of orange markers that represent the location of each of the questions.</li> <li>- Fits the map to the show all of the available questions.</li> <li>- Adds a pop-up to each marker displaying its point name and question.</li> </ul>
countChars()	<ul style="list-style-type: none"> <li>- Counts the characters that have been entered into each of the question text boxes and prints the number next to the text box so the user knows how many characters they have left.</li> </ul>

### File name: uploadData.js

The upload data file is a javascript file responsible for upload the question data entered by the user into the database. The following table will detail the purpose of each function in this file.

Function(s)	Purpose
validateData()	<ul style="list-style-type: none"> <li>- Ensures there is a value entered into each field.</li> <li>- If any field has been left blank, an alert will show telling the user 'Please fill in all fields.'</li> </ul>
startDataUpload()	<ul style="list-style-type: none"> <li>- To get the values from each of the question text boxes and create a string to send to the database.</li> <li>- Call the processData() function passing in the string of question data.</li> </ul>
processData() dataUploaded()	<ul style="list-style-type: none"> <li>- Uses a XMLHttpRequest to send the question data from the user to the database.</li> <li>- After this is done, an alert is shown informing the user that the data has been successfully uploaded.</li> </ul>

## 2.3 Testing

Both of the applications have been tested multiple times by the system creator, along with testing the situations where error handling is used. Additionally, both applications were then tested by two users who had never seen the interface before to ensure they were easy to use. These tests were done after the servers had been set up.

For the mobile application, questions were tested in two locations: the UCL campus and in Park Langley, Beckenham. It was ensured that the tracking of where the user was currently located worked correctly, along with locating the question markers within

a 20m radius, amongst other things. The error handling where an alert shows if the user has not selected an answer was also checked to ensure it worked correctly.

For the web application, the main testing was to ensure the questions uploaded to the database correctly, which they did. Each of the 15+ questions currently stored in the database were each added using the web application to ensure it worked as expected. This included attempting to add values of the wrong type and length, in which they were not successfully added. Initially, this was not the case, so the code needed to be modified to stop the text boxes from accepting longer values. The error handling of ensuring each of the values were entered before uploading was also tested.

The server was tested to ensure it communicated with the database correctly. This included checking the questions were loaded from the database to be shown on the maps in both applications and then checking the questions were uploading correctly from the form in the web application.

## Appendix A: Mobile Application Code (Quiz)

File: appActivity.js

```
//Load the map
var mymap = L.map('mapid').fitWorld();

//Load the tiles
L.tileLayer("https://api.tiles.mapbox.com/v4/{id}/{z}/{x}/{y}.png?access_to
ken=pk.eyJ1IjoibWFwYm94IiwiYSI6ImNpejY4NXVycTA2emYycXBndHRqcmZ3N3gifQ.rJcFI
G214AriISLbB6B5aw", {
  attribution: 'Map data &copy; <a
href="http://openstreetmap.org">OpenStreetMap</a> contributors, <a
href="http://creativecommons.org/licenses/by-sa/2.0/">CC-BY-SA</a>, Imagery
© <a href="http://mapbox.com">Mapbox</a>',
  maxZoom: 18,
  id: 'mapbox.streets'}) .addTo (mymap);

mymap.locate({setView: true, maxZoom: 18});

//Create the custom marker styles

//Will represent the user's lcoation
var markerOrange = L.AwesomeMarkers.icon({
  icon: 'play',
  markerColor: 'orange'
});

//Will represent the question point if the user gets the answer correct
var markerGreen = L.AwesomeMarkers.icon({
  icon: 'play',
  markerColor: 'green'
});

//Will represent the question point if the user gets the answer incorrect
var markerRed = L.AwesomeMarkers.icon({
  icon: 'play',
  markerColor: 'red'
});

//Will represent all question points that are close enough to be answered
var markerPurple = L.AwesomeMarkers.icon({
  icon: 'play',
  markerColor: 'purple'
});

//Will represent all the question points when they are loaded and if they
are not close enough to be answered
var markerBlue = L.AwesomeMarkers.icon({
  icon: 'play',
  markerColor: 'cadetblue'
});

//The following code tracks the user's location as they move
//adapted from: https://www.w3schools.com/html/html5_geolocation.asp
//adapted from: https://gis.stackexchange.com/questions/182068/getting-
current-user-location-automatically-every-x-seconds-to-put-on-leaflet
var initialTracking = true;
var userLocation;
```

```

var userLocationRad;
var autoPan = false;

function trackLocation() {
    if (!initialTracking){
        //Zoom to center
        mymap.fitBounds(userLocation.getLatLng().toBounds(250));
        autoPan = true;
    } else {
        if (navigator.geolocation) {
            alert("Finding your position!");
            navigator.geolocation.watchPosition(showPosition);

            //Error handling for if it cannot geolocate
        } else {
            alert("Geolocation is not supported.");
        }
    }
}

//Shows the user's current position as an orange marker on the map and pans
the map to that location
function showPosition(position) {

    if(!initialTracking){
        mymap.removeLayer(userLocation);
        mymap.removeLayer(userLocationRad);
    }

    //Adds a circle around the marker to show the radius where questions
    can be answered (20m)
    var setRad = 20;

    userLocationRad =
    L.circle([position.coords.latitude,position.coords.longitude], {
        color: 'orange',
        fillColor: '#FFD36E',
        fillOpacity: 0.5,
        radius: setRad
    }).addTo(mymap);

    userLocation =
    L.marker([position.coords.latitude,position.coords.longitude],
    {icon:markerOrange}).addTo(mymap);

    if(initialTracking){
        initialTracking = false;
        mymap.fitBounds(userLocation.getLatLng().toBounds(250));
        autoPan = true;
    }else if (autoPan) {
        mymap.panTo(userLocation.getLatLng());
    }
}

//Create a variable that will hold the XMLHttpRequest()
var client;

//Create a variable that will hold the layer itself

```

```

var questionsLayer;

//A function to get the questions from the database using an XMLHttpRequest
function getQuestions() {

    client = new XMLHttpRequest();
    client.open('GET', 'http://developer.cege.ucl.ac.uk:30288/getquestions');
    client.onreadystatechange = questionResponse;
    client.send();
}

//A function that will wait for the response from the data server, and
process the response once it is received
function questionResponse() {

    //This listens out for the server to say that the data is ready -
    i.e. has state 4
    if (client.readyState == 4) {
        //Once the data is ready, process the data by moving to the
        'loadQuestionLayer' function
        var questionData = client.responseText;
        loadQuestionLayer(questionData);
    }
}

//Create an empty array that will hold all of the question markers once
they are created
markers = [];

//A function to convert the received data - which is text - to JSON format
and add it as a marker to the map
function loadQuestionLayer(questionData) {

    //Convert the text to JSON
    var questionJSON = JSON.parse(questionData);

    //Load the geoJSON layer
    var questionsLayer = L.geoJson(questionJSON,
    {
        //Use point to layer to create the points
        pointToLayer: function (feature, latlng)
        {
            //Create a marker for each of the questions and set it to blue
            //Add a pop-up stating that there is a question there
            layer_marker = L.marker(latlng, {icon:markerBlue});

            layer_marker.bindPopup("<b>There's a question here!</b>");

            //Add the marker to the 'markers' array
            markers.push(layer_marker);

            return layer_marker;
        },
    }).addTo(mymap);

    //Change the map zoom so that all the question markers are shown
    mymap.fitBounds(questionsLayer.getBounds());
}

```

```

//A function to check if each question point is within 20m of the user's
current location
function checkQuestions() {

    //Assign the latitude and longitude of the user's current location
    var latlng = userLocation.getLatLng();
    alert("Checking for nearby questions");

    //Zoom into the user's current location
    mymap.fitBounds(latlng.toBounds(500));

    //Iterate around each of the markers and check if they are within 20m
of the user's current location
    for(var i=0; i<markers.length; i++) {

        //Assign the latitude and longitude of the marker currently
being checked
        current_point = markers[i];
        currentpoint_latlng = current_point.getLatLng();

        //Find the distance between the current question marker, and the
user's current location using the 'getDistance' function
        //A distance between the markers in meters is returned
        var distance = getDistance(currentpoint_latlng.lat,
currentpoint_latlng.lng, latlng.lat, latlng.lng);

        //If the distance is under 20m then make the marker purple and
allow click events to be processed
        if (distance <= 20) {
            markers[i].setIcon(markerPurple);
            markers[i].on('click', onClick);

        } else {
            //If the distance is over 20m then make the marker blue and
show a pop up when the marker is clicked
            markers[i].setIcon(markerBlue);
            markers[i].bindPopup("<b>Can't Answer!</b><br>This question is
too far away.");
        }
    }
}

//Function to work out the distance between the latitudes and longitudes of
two points
//Code from: https://stackoverflow.com/questions/27928/calculate-distance-between-two-latitude-longitude-points-haversine-formula
function getDistance(lat1,lon1,lat2,lon2) {

    var R = 6371; // Radius of the earth in km
    var dLat = deg2rad(lat2-lat1); // deg2rad below
    var dLon = deg2rad(lon2-lon1);
    var a =
        Math.sin(dLat/2) * Math.sin(dLat/2) +
        Math.cos(deg2rad(lat1)) * Math.cos(deg2rad(lat2)) *
        Math.sin(dLon/2) * Math.sin(dLon/2)
        ;
    var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
    var d = R * c; //Distance in km
    var d2 = d * 1000; //Distance in m
    return d2;
}

```



```

}

function deg2rad(deg) {
    return deg * (Math.PI/180)
}

//Create a variable for the marker that has been clicked
var clickedQuestion;

function onClick(e) {

    //When a marker is clicked go to the 'showClickedQuestion' function
    showClickedQuestion(this);

    //Assign the marker that has been clicked to the 'clickedQuestion'
    variable
    clickedQuestion = this;
}

//A function to show the div that holds the question form where the user
can submit their answer
function showClickedQuestion(clickedQuestion){

    //Hide the div that holds the leaflet map and show the div that holds
the question form
    document.getElementById('questionsection').style.display = 'block';
    document.getElementById('mapid').style.display = 'none';

    //Populate the textareas with the question data for the marker that
has been clicked
    document.getElementById("question").value =
clickedQuestion.feature.properties.question;
    document.getElementById("answer1").value =
clickedQuestion.feature.properties.answer1;
    document.getElementById("answer2").value =
clickedQuestion.feature.properties.answer2;
    document.getElementById("answer3").value =
clickedQuestion.feature.properties.answer3;
    document.getElementById("answer4").value =
clickedQuestion.feature.properties.answer4;

    //Make the radio buttons all unchecked for the user to make their
choice
    document.getElementById("check1").checked = false;
    document.getElementById("check2").checked = false;
    document.getElementById("check3").checked = false;
    document.getElementById("check3").checked = false;
}

//A function to validate and process the answer given by the user
function validateAnswer(){

    //Get the radio button values and ensure an answer has been given
    if ( (document.getElementById("check1").checked == false) &&
        (document.getElementById("check2").checked == false) &&
        (document.getElementById("check3").checked == false) &&
        (document.getElementById("check4").checked == false) ) {

        //If an answer hasn't been selected the user will be warned to
give an answer
        alert("Please select an answer");
    }
}

```

```

    } else {

        //If an answer has been given
        //Create two variables for the answer that has been given and its
value
        var givenAnswer;
        var answerValue;

        //Assign the created variables based upon the answer given by the
user
        if (document.getElementById("check1").checked) {
            givenAnswer = 1;
            answerValue = clickedQuestion.feature.properties.answer1;
        }
        if (document.getElementById("check2").checked) {
            givenAnswer = 2;
            answerValue = clickedQuestion.feature.properties.answer2;
        }
        if (document.getElementById("check3").checked) {
            givenAnswer = 3;
            answerValue = clickedQuestion.feature.properties.answer3;
        }
        if (document.getElementById("check4").checked) {
            givenAnswer = 4;
            answerValue = clickedQuestion.feature.properties.answer4;
        }

        //Send the answer given and its value to the 'answerResponse'
function
        answerResponse(givenAnswer, answerValue);
    }
}

//Create a variable to hold a boolean for whether the user gets the answer
right or wrong
var answer_correct;

function answerResponse(answer, answerValue){

    //Assign the correct answer from the clicked marker
    var correctAnswer =
clickedQuestion.feature.properties.correct_answer;
    var correctAnswerValue;

    //Assign the value of the answer depending on the number
    if (correctAnswer == 1) {
        correctAnswerValue = clickedQuestion.feature.properties.answer1;
    }
    if (correctAnswer == 2) {
        correctAnswerValue = clickedQuestion.feature.properties.answer2;
    }
    if (correctAnswer == 3) {
        correctAnswerValue =
clickedQuestion.feature.properties.answer3;
    }
    if (correctAnswer == 4) {
        correctAnswerValue =
clickedQuestion.feature.properties.answer4;
    }
}

```

```

    }

    //If the answer the user gives is the same as the correct answer
    if (answer == correctAnswer) {

        //Alert them that they have got the answer correct
        alert("That is the correct answer: " + correctAnswer + "\nWell
done!");

        //Make variable true
        answer_correct = true;
        //Submit the answer by sending it to the 'submitAnswer'
        function
            submitAnswer(answer, answerValue, answer_correct);

    } else {
        //If the answer the user gives is incorrect
        //Alert them that they have got the answer wrong
        alert("That is the wrong answer.\nThe correct answer is: " +
correctAnswer + " - " + correctAnswerValue);
        //Make variable false
        answer_correct = false;
        //Submit the answer by sending it to the 'submitAnswer'
        function
            submitAnswer(answer, answerValue, answer_correct);

    }
}

//A function to create a string with all the question and answer details to
send to the database
function submitAnswer(answer, answer_value, answer_correct){

    var question = clickedQuestion.feature.properties.question;
    //Create and assign a variable with all of the details about the
clicked marker, the answer given by the user, and if they were correct
    var postString = "&question="+question+"&answer="+answer
+"&answer_value="+answer_value+"&answer_correct="+answer_correct;
    processData(postString);
}

//Create a variable that will hold the XMLHttpRequest()
var client2;

//A function that will send the data to the database using an
XMLHttpRequest
function processData(postString) {

    client2 = new XMLHttpRequest();

    client2.open('POST', 'http://developer.cege.ucl.ac.uk:30288/uploadAnswerData
', true);
    client2.setRequestHeader("Content-type", "application/x-www-form-
urlencoded");
    client2.onreadystatechange = dataUploaded;
    client2.send(postString);
}

//A function to wait for the response from the data server, and process the
response once it is received
function dataUploaded() {
    //Listens out for the server to say that the data is ready - i.e. has
state 4

```

```

    if (client2.readyState == 4) {

        //Show an alert with the response text
        alert(client2.responseText);

        //Go to the 'returnToMap' function to show the div holding the map
again
        returnToMap();

        //Change the colour of the question marker depending on if they
were right or wrong
        if (answer_correct) {
            //If they were correct = green
            clickedQuestion.setIcon(markerGreen);
        } else {
            //If they were wrong = red
            clickedQuestion.setIcon(markerRed);
        }
    }
}

//A function to return the map to its first state
//This is with the div that holds map showing, and the div that holds the
question form hidden
function returnToMap() {
    document.getElementById('questionsection').style.display = 'none';
    document.getElementById('mapid').style.display = 'block';
}

```

## File: index.html

```

<doctype html>
<!--
    Material Design Lite
    Copyright 2015 Google Inc. All rights reserved.

    Licensed under the Apache License, Version 2.0 (the "License");
    you may not use this file except in compliance with the License.
    You may obtain a copy of the License at

        https://www.apache.org/licenses/LICENSE-2.0

    Unless required by applicable law or agreed to in writing, software
    distributed under the License is distributed on an "AS IS" BASIS,
    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    See the License for the specific language governing permissions and
    limitations under the License
-->
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="description" content="A front-end template that helps you
build fast, modern mobile web apps.">
    <meta name="viewport" content="width=device-width, initial-scale=1.0,
minimum-scale=1.0">
    <title>Material Design Lite</title>

    <!-- Add to homescreen for Chrome on Android -->

```

```

<meta name="mobile-web-app-capable" content="yes">
<link rel="icon" sizes="192x192" href="images/android-desktop.png">

<!-- Add to homescreen for Safari on iOS -->
<meta name="apple-mobile-web-app-capable" content="yes">
<meta name="apple-mobile-web-app-status-bar-style" content="black">
<meta name="apple-mobile-web-app-title" content="Material Design Lite">
<link rel="apple-touch-icon-precomposed" href="images/ios-desktop.png">

<!-- Tile icon for Win8 (144x144 + tile color) -->
<meta name="msapplication-TileImage" content="images/touch/ms-touch-
icon-144x144-precomposed.png">
<meta name="msapplication-TileColor" content="#3372DF">

<link rel="shortcut icon" href="images/favicon.png">

<!-- SEO: If your mobile URL is different from the desktop URL, add a
canonical link to the desktop page
https://developers.google.com/webmasters/smartphone-sites/feature-phones -
->
<!--
<link rel="canonical" href="http://www.example.com/">
-->

<link rel="stylesheet"
href="https://fonts.googleapis.com/css?family=Roboto:regular,bold,italic,thin,light,bolditalic,black,medium&lang=en">
<link rel="stylesheet"
href="https://fonts.googleapis.com/icon?family=Material+Icons">
<link rel="stylesheet"
href="https://code.getmdl.io/1.3.0/material.amber-orange.min.css" />
<link rel="stylesheet" href="styles.css">

<!-- the following links add the CSS and Javascript required for the
Leaflet Map -->
<link rel="stylesheet"
href="https://unpkg.com/leaflet@1.1.0/dist/leaflet.css"
integrity="sha512-
w6w6ts8Anuw10Mzh9Y9Y4pylW8+NAD4ch3lq99lZAsTxxG0GFeJgoAtxuCLREZSC5lUXdVyo/7yf
sqFjQ4S+aKw=="
crossorigin=""/>
<script src="https://unpkg.com/leaflet@1.1.0/dist/leaflet.js"
integrity="sha512-
mNqn2Wg7tSToJhvHcqfzLMU6J4mkOImSPTxVZAdo+lcPlk+GhZmYgACEe0x35K7YzW1zJ7XyJV/
TT1MrdXvMcA=="
crossorigin=""></script>

<!-- the following links incorporate the CSS required for custom icon
creation -->
<link rel="stylesheet" href="ionicons.min.css">
<link rel="stylesheet" href="leaflet.awesome-markers.css">
<script src="leaflet.awesome-markers.js"></script>

<!-- the following CSS is used to set the size of the Map -->
<style type="text/css">
#mapid { height: 100%;
width: 100%; }
</style>

</head>
</head>

```

```

<body>
  <div class="demo-layout mdl-layout mdl-js-layout mdl-layout--fixed-
drawer mdl-layout--fixed-header">
    <header class="demo-header mdl-layout__header mdl-color--grey-100
mdl-color-text--grey-600">
      <div class="mdl-layout__header-row">
        <span class="mdl-layout-title">Location Quiz</span>
      </div>
    </header>

    <div class="demo-drawer mdl-layout__drawer mdl-color--blue-grey-900
mdl-color-text--blue-grey-50">
      <header class="demo-drawer-header">
        
        <div class="demo-avatar-dropdown">
          <span>Menu</span>
        </div>
      </header>

      <nav class="demo-navigation mdl-navigation mdl-color--blue-grey-
800">
        <a class="mdl-navigation__link"
href=""onclick='trackLocation();return false;'><i class="mdl-color-text--
blue-grey-400 material-icons" role="presentation">add_location</i>1 Track
Location</a>
        <a class="mdl-navigation__link"
href=""onclick='getQuestions();return false;'><i class="mdl-color-text--
blue-grey-400 material-icons" role="presentation">star</i> 2 Get
Questions</a>
        <a class="mdl-navigation__link"
href=""onclick='checkQuestions();return false;'><i class="mdl-color-text--
blue-grey-400 material-icons" role="presentation">check</i>3 Find Nearby
Questions</a>
      </nav>
    </div>

    <main class="mdl-layout__content mdl-color--grey-100">
      <div class="demo-charts mdl-color--white mdl-shadow--2dp mdl-cell
mdl-cell--12-col mdl-grid">
        <div id="mapid" style="width: 100%; height: 100%;"></div>

        <div id="questionsection" style="display:none">
          <h2 id="header">Answer the question!</h2>

          <textarea id="question" readonly></textarea>

          <textarea id="answer1" style="border: none" readonly></textarea>
          <input type="radio" name="answer" id="check1" value="1"/><br />

          <textarea id="answer2" style="border: none" readonly></textarea>
          <input type="radio" name="answer" id="check2" value="2"/><br />

          <textarea id="answer3" style="border: none" readonly></textarea>
          <input type="radio" name="answer" id="check3" value="3"/><br />

          <textarea id="answer4" style="border: none" readonly></textarea>
          <input type="radio" name="answer" id="check4" value="4"/><br />

          <button class="mdl-button mdl-button--raised mdl-button--colored"
id="submitAnswer" onclick="validateAnswer()">Submit Answer</button>

```

```
        <button class="mdl-button mdl-button--raised mdl-button--colored"
id="goBack" onclick="returnToMap()" >Go Back</button>

    </div>
</div>
</main>

<script src="https://code.getmdl.io/1.3.0/material.min.js"></script>
<script src="js/appActivity.js"></script>

</body>
</html>
```

## Appendix B: Web Application Code (Questions)

File: appActivity.js

```
//A variable for the leaflet map
var mymap = L.map('mapid').setView([51.505, -0.09], 13);

//Will represent all the question points when they are loaded
var markerOrange = L.AwesomeMarkers.icon({
    icon: 'play',
    markerColor: 'orange'
});

//Loads the leaflet map tiles
function loadMap() {
    L.tileLayer('https://api.tiles.mapbox.com/v4/{id}/{z}/{x}/{y}.png?access_token=pk.eyJ1IjoibWFwYm94IiwiYSI6ImNpejY4NXVycTA2emYycXBndHRqcmZ3N3gifQ.rJcFIG214AriISLb6B5aw',{
        maxZoom: 18,
        attribution: 'Map data &copy; <a href="http://openstreetmap.org">OpenStreetMap</a> contributors, ' +
            '<a href="http://creativecommons.org/licenses/by-sa/2.0/">CC-BY-SA</a>, ' +
            'Imagery &copy; <a href="http://mapbox.com">Mapbox</a>',
        id: 'mapbox.streets'
    }).addTo(mymap);
}

//Gets the longitude and latitude of the location where the user clicks on the map
mymap.on('click', function(e) {
    //Populates the text boxes for the question's location
    document.getElementById("lat").value = e.latlng.lat;
    document.getElementById("long").value = e.latlng.lng;
});

//Makes each of the text areas blank so the question information can be entered from scratch
function resetForm() {
    document.getElementById("pointname").value = "";
    document.getElementById("question").value = "";
    document.getElementById("answer1").value = "";
    document.getElementById("answer2").value = "";
    document.getElementById("answer3").value = "";
    document.getElementById("answer4").value = "";
    document.getElementById("lat").value = "";
    document.getElementById("long").value = "";

    document.getElementById("namecharcount").innerHTML = "0";
    document.getElementById("questcharcount").innerHTML = "0";
    document.getElementById("answer1charcount").innerHTML = "0";
    document.getElementById("answer2charcount").innerHTML = "0";
    document.getElementById("answer3charcount").innerHTML = "0";
    document.getElementById("answer4charcount").innerHTML = "0";
}

//Create a variable that will hold the XMLHttpRequest()
var client;
//Create a variable that will hold the layer itself
var questionsLayer;
```



```

//A function that will get the questions data using an XMLHttpRequest
function getQuestions() {
    client = new XMLHttpRequest();
    client.open('GET', 'http://developer.cege.ucl.ac.uk:30288/getquestions');
    client.onreadystatechange = questionResponse; // note don't use
    earthquakeResponse() with brackets as that doesn't work
    client.send();
}

//A function to wait for the response from the data server, and process the
response once it is received
function questionResponse() {

    //This listens out for the server to say that the data is ready -
    i.e. has state 4
    if (client.readyState == 4) {
        // once the data is ready, process the data
        var questionData = client.responseText;
        loadQuestionLayer(questionData);
    }
}

//Convert the received data - which is text - to JSON format and add it to
the map
function loadQuestionLayer(questionData) {

    //Convert the text to JSON
    var questionJSON = JSON.parse(questionData);

    //Load the geoJSON layer
    var questionsLayer = L.geoJson(questionJSON,
{
    //Use point to layer to create the points
    pointToLayer: function (feature, latlng)
{

        //Create a marker for each of the questions and set it to orange
        //Add a pop-up stating the marker name and question
        return L.marker(latlng,
{icon:markerOrange}).bindPopup("<b>"+feature.properties.point_name + "</b>"
+ "<p>" + feature.properties.question + "</b>");

    },
}).addTo(mymap);

    //Change the map zoom so that all the data is shown
    mymap.fitBounds(questionsLayer.getBounds());
}

//Counts the characters that have been entered into the text box
//This aims to prevent the user from entering a value too long for the
database
//Adapted from: https://stackoverflow.com/questions/9767521/count-and-
display-number-of-characters-in-a-textbox-using-javascript
function countChars(countTextBox, printTextBox) {
    var charLen = document.getElementById(countTextBox).value.length;
    document.getElementById(printTextBox).innerHTML = charLen;
}

```

## File: uploadData.js

```
//This function checks that the fields have not been left empty
function validateData() {
    var a=document.getElementById("pointname").value;
    var b=document.getElementById("question").value;
    var c=document.getElementById("answer1").value;
    var d=document.getElementById("answer2").value;
    var e=document.getElementById("answer3").value;
    var f=document.getElementById("answer4").value;

    if (a==" " || b==" " || c==" " || d==" " || e==" " || f == " ")
    {
        alert("Please fill in all fields.");
        return false;
    }
    else
    {
        startDataUpload();
    }
}

//This function gets the values from the question form text boxes and
creates a string to be sent to the database
function startDataUpload() {

    //Gets the values from the textboxes
    var pointname = document.getElementById("pointname").value;
    var question = document.getElementById("question").value;
    var answer1 = document.getElementById("answer1").value;
    var answer2 = document.getElementById("answer2").value;
    var answer3 = document.getElementById("answer3").value;
    var answer4 = document.getElementById("answer4").value;
    var lat = document.getElementById("lat").value;
    var long = document.getElementById("long").value;

    //Shows an alert telling the user what data is being sent to the database
    alert("Submitting: " + pointname + "\n" + question + " \n" + answer1 +
    " " + answer2 + " " + answer3 + " " + answer4 + " \n" + lat + " " + long);

    //Creates a string containing the question data
    var postString = "pointname="+pointname+"&question="+question
   +"&answer1="+answer1+"&answer2="+answer2+"&answer3="+answer3+
    "&answer4="+answer4;

    // now get the radio button values
    if (document.getElementById("check1").checked) {
        postString=postString+"&correctanswer=1";
    }
    if (document.getElementById("check2").checked) {
        postString=postString+"&correctanswer=2";
    }
    if (document.getElementById("check3").checked) {
        postString=postString+"&correctanswer=3";
    }
    if (document.getElementById("check4").checked) {
        postString=postString+"&correctanswer=4";
    }

    postString = postString + "&lat=" + lat + "&long=" + long;
}
```

```

    //Calls the processData() function passing in the string of question data
    processData(postString);
}

//Create a variable that will hold the XMLHttpRequest()
var client;

//A function that will send the data to the database using an
XMLHttpRequest
function processData(postString) {
    client = new XMLHttpRequest();

    client.open('POST', 'http://developer.cege.ucl.ac.uk:30288/uploadData', true)
    ;
    client.setRequestHeader("Content-type", "application/x-www-form-
urlencoded");
    client.onreadystatechange = dataUploaded;
    client.send(postString);
}

//A function to wait for the response from the data server, and process the
response once it is received
function dataUploaded() {
    //Listens out for the server to say that the data is ready - i.e. has
state 4
    if (client.readyState == 4) {
        //Show an alert with the response text
        alert(client.responseText);
    }
}

```

## File: index.html

```

<!doctype html>
<!--
Material Design Lite
Copyright 2015 Google Inc. All rights reserved.

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    https://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License
-->

<html lang="en">

  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="description" content="A front-end template that helps you
build fast, modern mobile web apps.">
    <meta name="viewport" content="width=device-width, initial-scale=1.0,
minimum-scale=1.0">

```

```

<title>QUESTION SETTING</title>

<!-- Add to homescreen for Chrome on Android -->
<meta name="mobile-web-app-capable" content="yes">
<link rel="icon" sizes="192x192" href="images/android-desktop.png">

<!-- Add to homescreen for Safari on iOS -->
<meta name="apple-mobile-web-app-capable" content="yes">
<meta name="apple-mobile-web-app-status-bar-style" content="black">
<meta name="apple-mobile-web-app-title" content="Material Design Lite">
<link rel="apple-touch-icon-precomposed" href="images/ios-desktop.png">

<!-- Tile icon for Win8 (144x144 + tile color) -->
<meta name="msapplication-TileImage" content="images/touch/ms-touch-
icon-144x144-precomposed.png">
<meta name="msapplication-TileColor" content="#3372DF">

<link rel="shortcut icon" href="images/favicon.png">

<!-- SEO: If your mobile URL is different from the desktop URL, add a
canonical link to the desktop page
https://developers.google.com/webmasters/smartphone-sites/feature-phones -
->
<!--
<link rel="canonical" href="http://www.example.com/">
-->

<link rel="stylesheet"
href="https://fonts.googleapis.com/css?family=Roboto:regular,bold,italic,th
in,light,bolditalic,black,medium&lang=en">
<link rel="stylesheet"
href="https://fonts.googleapis.com/icon?family=Material+Icons">
<link rel="stylesheet"
href="https://code.getmdl.io/1.3.0/material.amber-orange.min.css" />
<link rel="stylesheet" href="styles.css">

<!-- the following links add the CSS and Javascript required for the
Leaflet Map -->
<link rel="stylesheet"
href="https://unpkg.com/leaflet@1.1.0/dist/leaflet.css" integrity="sha512-
wCw6ts8Anuw10Mzh9Ytw4pylW8+NAD4ch3lqm9lZAsTxg0GfFeJgoAtxuCLREZSC5lUXdVyo/7yf
sqFjQ4S+aKw==" crossorigin="" />
<script src="https://unpkg.com/leaflet@1.1.0/dist/leaflet.js"
integrity="sha512-
mNqn2Wg7tStoJhvHcqfzLMU6J4mkOImSPTxVZAdo+lcPlk+GhZmYgACEe0x35K7YzW1zJ7XyJV/
TT1MrdXvMcA==" crossorigin=""></script>

<!-- the following links add the CSS and Javascript required for the
custom icons -->
<link rel="stylesheet" href="ionicons.min.css">
<link rel="stylesheet" href="leaflet.awesome-markers.css">
<script src="js/leaflet.awesome-markers.js"></script>

<!-- the following CSS is used to set the size of the Map -->
<style type="text/css">
#mapid { height: 500px; }
</style>

</head>

<body>

```

```

<main>
  <div class = "mdl-grid mdl-cell--stretch">

    <div class = "mdl-cell mdl-cell--12-col mdl-cell--stretch
graybox" div align="center">
      <div id="mapid" style="width: 100%; height: 400px;"></div>
    </div>

    <div class = "mdl-grid" id = "form">

      <div class = "mdl-cell mdl-cell--6-col graybox">
        <label for="pointname">Point Name: </label><input
type="text" size="25" id="pointname" maxlength="50"
onkeyup="countChars('pointname', 'namecharcount');"
onkeydown="countChars('pointname', 'namecharcount');"
onmouseout="countChars('pointname', 'namecharcount');"/>
        <span id="namecharcount" style="color:red">0</span><span
id="outof" style="color:red">/50</span><br/>

        <label for="question">Question: </label><input type="text"
size="25" id="question" maxlength="200"
onkeyup="countChars('question', 'questcharcount');"
onkeydown="countChars('question', 'questcharcount');"
onmouseout="countChars('question', 'questcharcount');"/>
        <span id="questcharcount" style="color:red">0</span><span
id="outof" style="color:red">/200</span><br/>

        <label for="answer1">Answer 1: </label><input type="text"
size="25" id="answer1" maxlength="50"
onkeyup="countChars('answer1', 'answer1charcount');"
onkeydown="countChars('answer1', 'answer1charcount');"
onmouseout="countChars('answer1', 'answer1charcount');"/>
        <span id="answer1charcount" style="color:red">0</span><span
id="outof" style="color:red">/50</span><br/>

        <label for="answer2">Answer 2: </label><input type="text"
size="25" id="answer2" maxlength="50"
onkeyup="countChars('answer2', 'answer2charcount');"
onkeydown="countChars('answer2', 'answer2charcount');"
onmouseout="countChars('answer2', 'answer2charcount');"/>
        <span id="answer2charcount" style="color:red">0</span><span
id="outof" style="color:red">/50</span><br/>

        <label for="answer3">Answer 3: </label><input type="text"
size="25" id="answer3" maxlength="50"
onkeyup="countChars('answer3', 'answer3charcount');"
onkeydown="countChars('answer3', 'answer3charcount');"
onmouseout="countChars('answer3', 'answer3charcount');"/>
        <span id="answer3charcount" style="color:red">0</span><span
id="outof" style="color:red">/50</span><br/>

        <label for="answer4">Answer 4: </label><input type="text"
size="25" id="answer4" maxlength="50"
onkeyup="countChars('answer4', 'answer4charcount');"
onkeydown="countChars('answer4', 'answer4charcount');"
onmouseout="countChars('answer4', 'answer4charcount');"/>
        <span id="answer4charcount" style="color:red">0</span><span
id="outof" style="color:red">/50</span><br/>

```

```

        <label for="lat">Point latitude: </label><input
type="text" size="25" id="lat"/><br/>
        <label for="long">Point longitude: </label><input
type="text" size="25" id="long"/><br/>

        <div class = "mdl-cell mdl-cell--6-col graybox">
            Which is the correct answer?
            <p>1: <input type="radio" name="answer" id = check1
value="1" checked="yes" /><br />
                2: <input type="radio" name="answer" id = check2
value="2"/><br />
                3: <input type="radio" name="answer" id = check3
value="3"/><br />
                4: <input type="radio" name="answer" id = check4
value="4"/><br />
            </div>

        </div>
    </div>

    <div class = "mdl-grid" id = "buttons">
        <div class = "mdl-cell mdl-cell--6-col graybox">
            <button class="mdl-button mdl-button--raised mdl-button--
colored" id="startUpload" onclick="validateData()">Upload Question</button>
            <button class="mdl-button mdl-button--raised mdl-button--
colored" id="reset" onclick="resetForm()">Reset Form</button>
            <button class="mdl-button mdl-button--raised mdl-button--
colored" id="reset" onclick="getQuestions()">Show Points</button>

        </div>
    </div>

    </main>
</div>

<script src="https://code.getmdl.io/1.3.0/material.min.js"></script>
<script src="js/appActivity.js"></script>
<script src="js/uploadData.js"></script>
<script>loadMap()</script>

</body>
</html>

```

## Appendix C: Server Code

### File: httpServer.js

```
//Express is the server that forms part of the nodejs program
var express = require('express');
var path = require("path");
var app = express();
var bodyParser = require('body-parser');

app.use(bodyParser.urlencoded({
  extended: true
}));
app.use(bodyParser.json());

//Adding functionality to allow cross-domain queries when PhoneGap is
running a server
app.use(function(req, res, next) {
  res.setHeader("Access-Control-Allow-Origin", "*");
  res.setHeader("Access-Control-Allow-Headers", "X-Requested-
With");
  res.setHeader('Access-Control-Allow-Methods',
'GET,PUT,POST,DELETE');
  next();
});

//This function uploads the question set by the user to the database
app.post('/uploadData',function(req,res){
  //Using POST here as we are uploading data
  console.dir(req.body);
  pool.connect(function(err,client,done) {
    if(err){
      console.log("not able to get connection "+ err);
      res.status(400).send(err);
    }

    //Pull the geometry (longitude/latitude) component together
    var geometrysting = "st_geomfromtext('POINT(" + req.body.long + " " +
req.body.lat + ")'";

    //Creates the string with the SQL insert statement
    var querystring = "INSERT into quizquestions
(point_name,question,answer1,answer2,answer3,answer4,correct_answer,coordin
ates) values ('";
    querystring = querystring + req.body.pointname + "',' " +
req.body.question + "',' " + req.body.answer1+"',' " + req.body.answer2+"',' "
+ req.body.answer3+"',' " + req.body.answer4+"',' " +
req.body.correctanswer+"',' " + geometrysting +"))";
    console.log(querystring);

    client.query( querystring,function(err,result) {
      done();

      if(err){
        console.log(err);
        res.status(400).send(err);
      }

      res.status(200).send("Question sent to database!");
    });
  });
});
```

```

    });
});

// adding functionality to log the requests
app.use(function (req, res, next) {
    var filename = path.basename(req.url);
    var extension = path.extname(filename);
    console.log("The file " + filename + " was requested.");
    next();
});

//Add a http server to serve files to the Edge browser
//Due to certificate issues it rejects the https files if they are not
directly called in a typed URL
var http = require('http');
var httpServer = http.createServer(app);
httpServer.listen(4480);

app.get('/', function (req, res) {
    res.send("hello world from the HTTP server");
});

//This is needed for the connection to the database to work
var fs = require('fs');
var configtext =
    ""+fs.readFileSync("/home/studentuser/certs/postGISConnection.js");

//Now convert the configuration file into the correct format -i.e. a
name/value pair array
var configarray = configtext.split(",");
var config = {};
for (var i = 0; i < configarray.length; i++) {
    var split = configarray[i].split(':');
    config[split[0].trim()] = split[1].trim();
}

var pg = require('pg');
var pool = new pg.Pool(config);
console.log(config);

//This function gets the questions from the database
app.get('/getQuestions', function (req, res) {
    pool.connect(function (err, client, done) {
        if (err) {
            console.log("not able to get connection "+ err);
            res.status(400).send(err);
        }

        // use the inbuilt geoJSON functionality
        // and create the required geoJSON format using a query adapted from
        here: http://www.postgresonline.com/journal/archives/267-Creating-GeoJSON-Feature-Collections-with-JSON-and-PostGIS-functions.html, accessed 4th
        January 2018
        // note that query needs to be a single string with no line breaks so
        built it up bit by bit

        var querystring = " SELECT 'FeatureCollection' As type,
        array_to_json(array_agg(f)) As features FROM ";

```



```

        querystring = querystring + "(SELECT 'Feature' As type      ,
ST_AsGeoJSON(lg.coordinates)::json As geometry, ";
        querystring = querystring + "row_to_json((SELECT 1 FROM (SELECT
point_name, question, answer1, answer2, answer3, answer4, correct_answer)
As 1      )) As properties";
        querystring = querystring + "      FROM quizquestions  As lg limit 100  )
As f ";
        console.log(querystring);
        client.query(querystring, function(err, result) {
            //call `done()` to release the client back to the pool
            done();

            if(err){
                console.log(err);
                res.status(400).send(err);
            }

            res.status(200).send(result.rows);
        });
    });
});

app.post('/uploadAnswerData', function(req, res) {
    // note that we are using POST here as we are uploading data
    // so the parameters form part of the BODY of the request rather than the
    RESTful API
    console.dir(req.body);
    pool.connect(function(err, client, done) {
        if(err){
            console.log("not able to get connection "+ err);
            res.status(400).send(err);
        }

        //Creates a SQL string for the SQL statement to insert the answer into
        the database
        var querystring = "INSERT into quizanswers
(question,answer,answer_value,answer_correct) values ('";
        querystring = querystring + req.body.question + "',' " +
req.body.answer+"',' " + req.body.answer_value+"',' " +
req.body.answer_correct+"')";
        console.log(querystring);
        client.query( querystring, function(err, result) {
            done();
            if(err){
                console.log(err);
                res.status(400).send(err);
            }

            res.status(200).send("Answer sent to database");
        });
    });
});

// the / indicates the path that you type into the server - in this case,
what happens when you type in:
http://developer.cege.ucl.ac.uk:32560/xxxxxx/xxxxxx
app.get('/:name1', function (req, res) {
    // run some server-side code
    // the console is the command line of your server - you will see the
    console.log values in the terminal window

```

```

console.log('request '+req.params.name1);

// the res is the response that the server sends back to the browser - you
// will see this text in your browser window
res.sendFile(__dirname + '/' + req.params.name1);
});

// the / indicates the path that you type into the server - in this case,
// what happens when you type in:
// http://developer.cege.ucl.ac.uk:32560/xxxxx/xxxxx
app.get('/:name1/:name2', function (req, res) {
  // run some server-side code
  // the console is the command line of your server - you will see the
  // console.log values in the terminal window
  console.log('request '+req.params.name1+"/"+req.params.name2);

  // the res is the response that the server sends back to the browser - you
  // will see this text in your browser window
  res.sendFile(__dirname + '/' + req.params.name1+"/"+req.params.name2);
});

// the / indicates the path that you type into the server - in this case,
// what happens when you type in:
// http://developer.cege.ucl.ac.uk:32560/xxxxx/xxxxx/xxxx
app.get('/:name1/:name2/:name3', function (req, res) {
  // run some server-side code
  // the console is the command line of your server - you will see the
  // console.log values in the terminal window
  console.log('request '+req.params.name1+"/"+req.params.name2+"/"+req.params.name3);
  // send the response
  res.sendFile(__dirname + '/' + req.params.name1+"/"+req.params.name2+
    '/' + req.params.name3);
});

// the / indicates the path that you type into the server - in this case,
// what happens when you type in:
// http://developer.cege.ucl.ac.uk:32560/xxxxx/xxxxx/xxxx
app.get('/:name1/:name2/:name3/:name4', function (req, res) {
  // run some server-side code
  // the console is the command line of your server - you will see the
  // console.log values in the terminal window
  console.log('request '+req.params.name1+"/"+req.params.name2+"/"+req.params.name3+"/"+req.params
    .name4);
  // send the response
  res.sendFile(__dirname + '/' + req.params.name1+"/"+req.params.name2+
    '/' + req.params.name3+"/"+req.params.name4);
});

```

## Appendix D: SQL Table Creation Code

```
create table public.quizquestions (  
    question_id serial not null primary key,  
    point_name character varying (50) not null,  
    question character varying (200) not null,  
    answer1 integer not null,  
    answer2 integer not null,  
    answer3 integer not null,  
    answer4 integer not null,  
    answer_correct integer not null,  
    coordinates geometry not null  
);  
  
ALTER TABLE public.quizquestions ADD CONSTRAINT question_unique UNIQUE  
(question, coordinates);  
  
ALTER TABLE public.quizquestions ADD CONSTRAINT correct_answer_check  
CHECK (correct_answer > 0 AND correct_answer <= 4);  
  
create table public.quizanswers (  
    answer_id serial not null primary key,  
    question character varying (200) not null,  
    answer integer not null,  
    answer_value character varying (50) not null,  
    answer_correct boolean not null,  
);
```