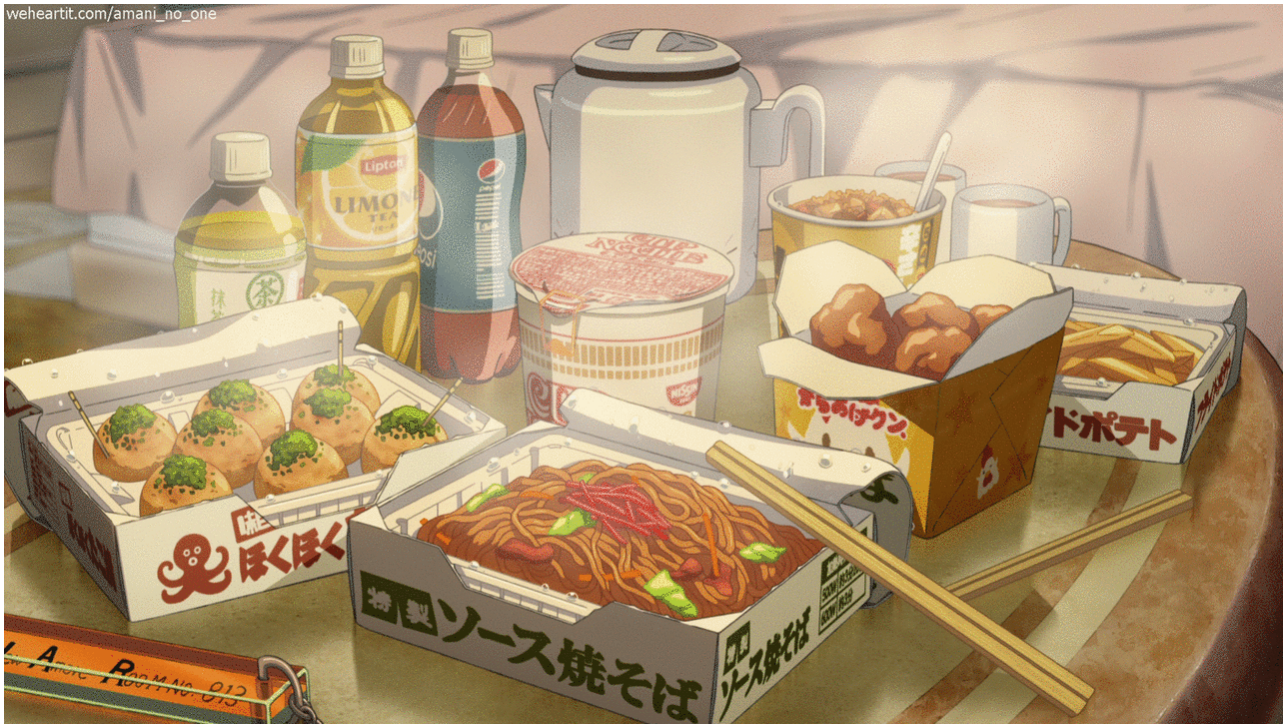
Berlin Restaurants

Application with user friendly interface based on Apache Cordova framework

Belfort, France - January 2, 2021



Project Structure	
Used Plugins	cordova-sqlite-storage, cordova-plugin-sqlserver
SQLite	Local database, used to save favorite restaurants
Remote Database	Database on XAMPP server, used to locate restaurants on the map
Map	Google Maps API, shows restaurants' location and description

Project Structure

```
+– node_modules/
|   +–– …
+– platforms/
|     +– android/…
|     +– browser/…
|     +– ios/…
+–– plugins/
|     +– cordova-plugin-dialogs/…
|     +– cordova-plugin-sqlserver/…
|     +– cordova-plugin-whitelist/…
|     +– cordova-sqlite-storage/…
+–– www/
|     +–– deps/
|     |     +– bootstrap-3.3.7/
|     |     +– jquery-3.3.1.min.js
|     +–– imgs/desc.gif
|     +–– themes/
|     +–– jquery.mobile.icons.min.css
|     +–  mytheme.css
|     +–  mytheme.min.css
|     +–  app.js
|     +–  evothings.json
|     +– index.html
|     +– loc.html
+–  .gitignore
+–  config.xml
+–  package.json
+–  README.md
```

Main files used in development: app.js, index.html, loc.html.
Loc.html - used for displaying the map and implementing it's functionality, connects to the remote database
Index.html - main html file, responsible for navigation in the app pages, check-in, contact us and favorite pages implementation
App.js - local database implementation

User Plugins / SQLite

- **cordova-sqlite- storage** - https://github.com/storesafe/cordova-sqlite-storage

```
function initDatabase() {
  database = window.sqlitePlugin.openDatabase({ name: 'test.db', location: 'default' });

  database.transaction(function (transaction) {
    transaction.executeSql('CREATE TABLE TestTable (id, date, address, number, mail)');
  });
```

This plugin was used in app.js file to create SQLite database.
The database is called test.db, the table is TestTable, which has 5 columns: id, date, address, number, mail inserted as strings.

```
database.transaction(function (transaction) {
  transaction.executeSql('INSERT INTO TestTable VALUES (?,?,?,?,?)', ['Restaurant: ' + nextUser, date, address, numb, mail]);
}, function (error) {
  showMessage('INSERT error: ' + error.message);
}, function () {
  showMessage('INSERT OK');
  ++nextUser;
});
```

Here is shown how is a check-in data inserted in TestTable as a row.

```
database.transaction(function (transaction) {
  transaction.executeSql('SELECT *  FROM TestTable', [], function (tx, results) {
    var len = results.rows.length, i;
    for (i = 0; i < len; i++) {
      insertString(table, [results.rows.item(i).date, results.rows.item(i).address, results.rows.item(i).number, results.rows.item(i).mail]);
    }
  });
}, function (error) {
  showMessage('SELECT count error: ' + error.message);
});
```

This transaction is used to get all rows from the table and display in the favorite restaurants' table.
- **cordova-plugin-sqlserver** - https://github.com/SergioDosSantos/cordova-plugin-sqlserver
It is a cordova plugin to connect to SQL Server without services. The purpose of this plugin is to avoid using services to access data directly. We tried to use this plugin, which could initialize plugin with XAMPP server, but got "Connection refused" message while connecting to the database. That's why was decided to use **Ajax** instead.

```
  SqlServer.init("192.168.43.175", "localhost", "root", "berlin", "berlindb", function(event) {
    alert(JSON.stringify(event));
    SqlServer.testConnection(function(event) {
    alert(JSON.stringify(event));
 }, function(error) {
   alert("Error : " + JSON.stringify(error));
 });
    SqlServer.executeQuery("SELECT count(*) FROM mytable", function(event) {
    alert(JSON.stringify(event));
 }, function(error) {
   alert("Error : " + JSON.stringify(error));
 });
}, function(error) {
 alert(JSON.stringify(error));
});
```

User Plugins

Here you can see the attempt to init the SqlServer and in case of success to test connection and get the count of rows of the remote sql.

Ajax

```
$.ajax({
    type: "GET",
    url: "http://192.168.43.175/config.php",
    headers: {
        'Content-Type': 'application/x-www-form-urlencoded'
    },
    dataType: "text",
    success: function (response) {
        var res = response.split("Restaurant:");
        res.forEach(function (element) {
            var indLat = element.indexOf("Lat:");
            var indLon = element.indexOf("Lon:");
            var indNumber = element.indexOf("Number:");
            var indMail = element.indexOf("Email:");
            var indSite = element.indexOf("Website:");
            var addStr = element.substring(0, indLat);
            var latStr = element.substring(indLat + "Lat:".length + 1, indLon);
            var lonStr = element.substring(indLon + "Lon:".length);
            markerPoints.push({ lat: parseFloat(latStr), lng: parseFloat(lonStr) });
            markerTitle.push(addStr);
            markerNumber.push(element.substring(indNumber + "Number:".length, indMail));
            markerMail.push(element.substring(indMail + "Email:".length, indSite));
            markerSite.push(element.substring(indSite + "Website:".length, element.length));
        });
        navigator.geolocation.getCurrentPosition(onMapSuccess, onMapError);
    },
    error: function (error) {
        console.log("FFF" + error);
        alert(error);
    }
});
```

The GET request was made to the file http://192.168.43.175/config.php, which is placed on a XAMPP server. The response is sliced and displayed as an info window on the map. 192.168.43.175 - ip of the server connected to the local network

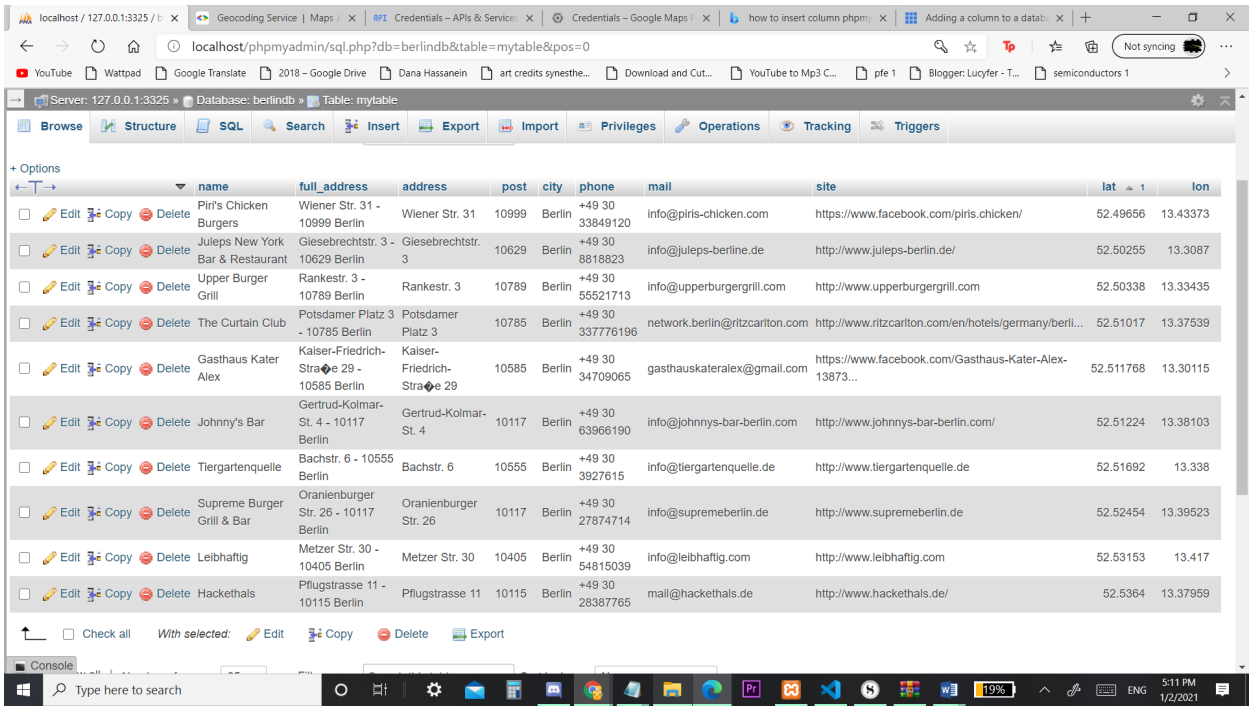
Tools

While testing the application in the browser, an “Access-Control-Allow-Origin” access error was encountered, that’s why was decided to use specific extension on Chrome https://mybrowseraddon.com/access-control-allow-origin.html

Remote Database

PhpMyAdmin is a software tool written in PHP, intended to handle the administration of MySQL over the Web. It supports operations on MySQL and MariaDB (managing databases, tables, columns, relations, indexes, users, permissions, etc) that can be performed via the user interface.

Berlin restaurants database in localhost/PHPMyAdmin :



XAMPP creates the username for it to be accessed, password is empty. First, to set up a connection to the database, we create a ***config.php*** file inside xampp/htdocs folder because XAMPP uses folders in htdocs to execute and run your PHP sites. Then we use the function mysql_connect. This function returns a pointer (also known as a database handle) to the database connection.

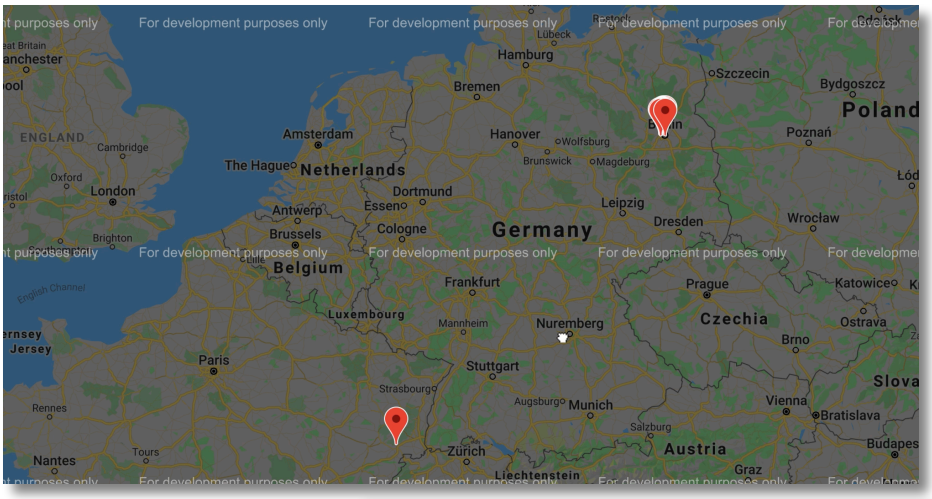
- 'localhost' is the location where the server is located
- 'root' is the global username of server
- 'berlin' is the password
- 'berlindb' is the database with which we want to connect
- '3325' is the port

Config.php :

```
<?php
    $con = mysqli_connect('localhost','root','berlin','berlindb', '3325') or
 die ('unable to connect');

    if ($con->connect_error) {
        echo("Connection failed: " . $conn->connect_error);
      }
    else {
        echo "Connection successful\n";
    }

// Query Select columns from database table
$sql = "SELECT lat, lon, name, phone, mail, site FROM mytable";
$result = $con->query($sql);

if ($result->num_rows > 0) {

  while($row = $result->fetch_assoc()) {
    echo "Restaurant: " . $row["name"]. "Lat: " . $row["lat"]. "Lon: " . $ro
w["lon"]. "Number: " . $row["phone"]. "Email: " . $row["mail"]. "Website: "
. $row["site"]. "\n";
  }
} else {
  echo "no results";
}
$con->close();
?>
```

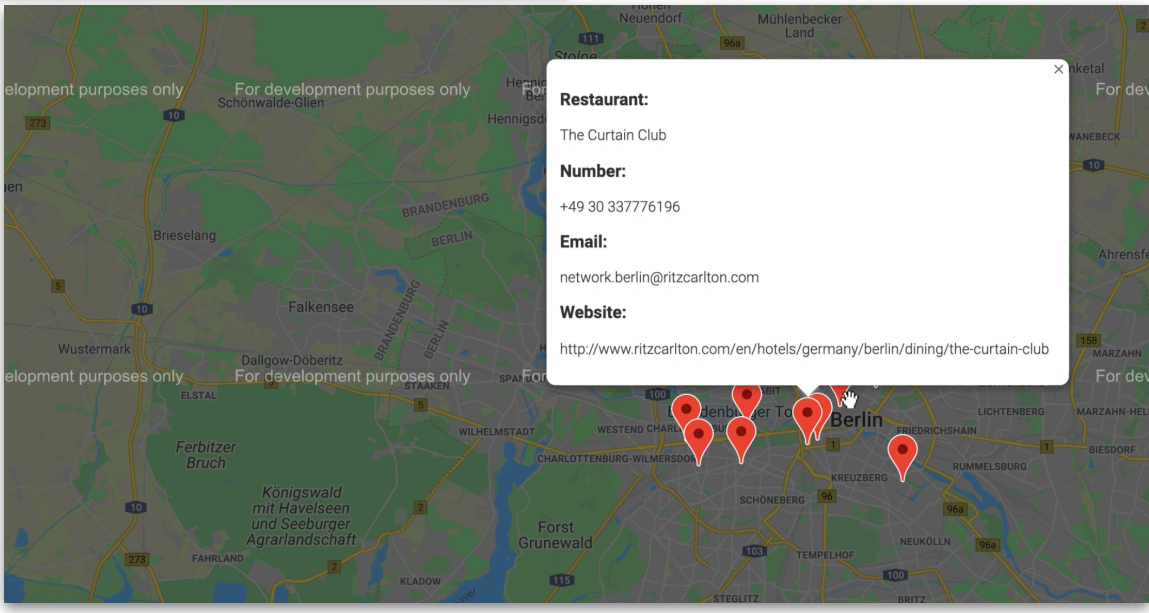
In case of success a string with required data is echoed, so we can receive it as response in the ajax GET request and then be sliced based on the specific keys (“Lat:”, “Email:”, etc.) and used as a coordinate for marker and description of an info window.

Map



Shows current location and restaurants fetched from the remote database.

Clickable markers with the information about each restaurant.




```
markerPoints.forEach(function (element) {
    var marker = new google.maps.Marker({
        position: element,
        map,
        title: markerTitle[markerPoints.indexOf(element)],
    });
```

Creating markers

Creating info window

```
var infowindow = new google.maps.InfoWindow({
    content: contentString,
});
marker.addListener("click", () => {
    infowindow.open(map, marker);
});
```