# CauseCode Challenge

Date: Oct 25, 2016

API Document for Store Finder Application

Author: Sagarsingh Jadhav

**Introduction**

This application is developed in MEAN stack platform. This application enable user to save, edit, update and delete Store information in mongodb. Also, user can find list of store from the saved stores in database by giving the zip code and the distance to find the stores in that range.

|  |  |
| --- | --- |
| **Title** | **Details** |
| Technologies Used | * NodeJS, * ExpressJS, * MongoDB, * AngularJS |
| NPM modules used | * promises, * zipcodes. * mongoose, * morgan, * body-parser |
| Link used for project setup, code reference and database configuration. | * <https://scotch.io/tutorials/creating-a-single-page-todo-app-with-node-and-angular> * <http://mongoosejs.com/docs/index.html> |
| MongoDB Database | * Create “test” database in mongodb (Collection would be “stores”) |
| Requirement | * Save store information in database. * Edit and update store information. * Delete a store. * To find stores within a range of a US zip code along with the distance in miles. |

**Requirement Analysis**

Actual problem statement:

1. Create/Update/Delete a store

2. Find a store within "x" miles of a USA zipcode

According to the problem statement user will add, delete and update stores in the database. And in the second point it is specified that on a US zip-code and distance provided in miles the application should return the stores within that radius range with the zip code as epicenter.

So, in the first case user should enter the save the store with the following details to support the second point in the requirement.

* Store Name
* Latitude
* Longitude

In the point number 2, the user will provide the zip-code and distance in miles. On the basis of this zip-code we can get the latitude and longitude of the zip-code location. This location coordinates and the distance can be used to construct the MongoDB geospatial query and fetch the desired data from the database.

**Adding Store API logic:**

* User Input: - Store name, latitude and longitude.
* Using the above data create a insert query and insert the data in mongodb

**Updating Store API Logic:**

* User Input: - \_id and update fields.
* Using the above information find the specific store using \_id and update the document from the collections.

**Deleting Store API Logic:**

* User Input: - \_id
* Using the \_id delete the specific document from the database.

**Store finder API logic:**

* User Input: - Zip-Code and Distance.
* Use Zip-code to get zip-code location information using zipcode npm module.

Link for module: <https://www.npmjs.com/package/zipcodes>

**Eg**: var locationDetails = zipcodes.lookup(90210);

**Outpout**: {

“zip”:”90210”,

“latitude”: 34.08888,

“longitude”: -118.40125,

“city”: “Beverly Hills”,

“state”:”CA”

}

* We have the distance given by the user:

Distance cannot be used directly in as miles. We have to do some conversions before using this value. MongoDB query takes the radius value in radian. We need to divide this value with the approximate equatorial radius of the earth’s sphere i.e. 3963.2 miles.

actualDistance = userDistance / 3963.2

* Using the above data of given zip-code and distance following mongodb query is being constructed:

**Query**:

db.collection.find({'position': { '$geoWithin': { '$centerSphere': [[parseFloat(longitude), parseFloat(latitude)] , actualDistance] } } })

The above query will return the desired data from the database.

**REST API Details**

|  |  |
| --- | --- |
| **URL** | http://localhost:8080/api/getStores |
| **Method** | GET |
| **Header** | N/A |
| **Content-type** | application/json |
| **Response Body** | [{  "\_id": "580f3b9633b133208ce8ef49",  "storeName": "Store 1",  "position": [38.98076, -76.50346]  }, {  "\_id": "580f3bb533b133208ce8ef4a",  "storeName": "Store 2",  "position": [-76.53059, 38.99624]  }, {  "\_id": "580f3bd433b133208ce8ef4b",  "storeName": "Store 3",  "position": [-76.55239, 39.02011]  }] |

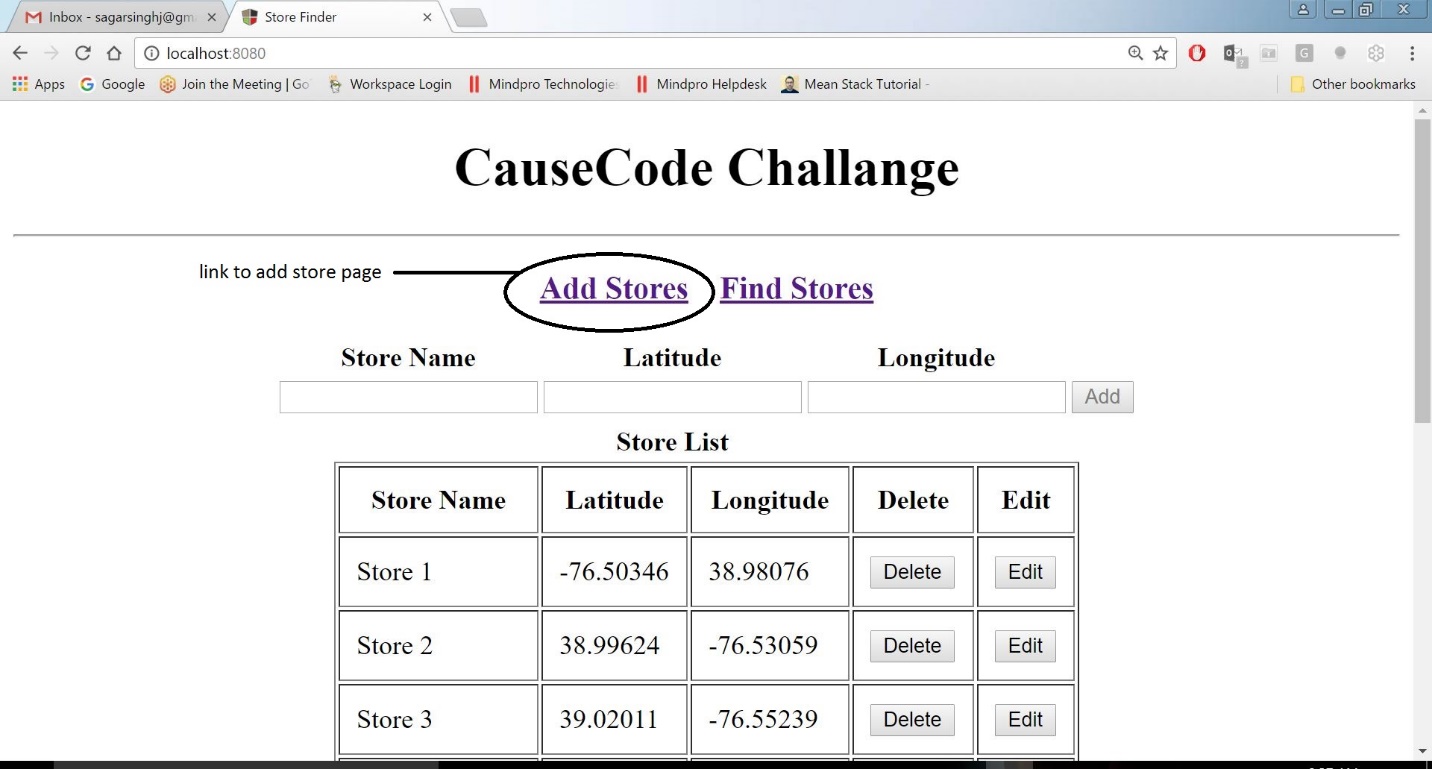
|  |  |
| --- | --- |
| **URL** | http://localhost:8080/api/saveStores |
| **Method** | POST |
| **Header** | N/A |
| **Content-type** | application/json |
| **Request Body** | {  "storeName": "Mango Store",  "lat": "24.5644",  "lon": "-98.34565"  } |
| **Response Body** | [{  "\_id": "580f3b9633b133208ce8ef49",  "storeName": "Store 1",  "position": [38.98076, -76.50346]  }, {  "\_id": "580f3bb533b133208ce8ef4a",  "storeName": "Store 2",  "position": [-76.53059, 38.99624]  }, {  "\_id": "580faca7c1cfd72890fae6ec",  "storeName": "Mango Store",  "position": [-98.34565, 24.5644]  }] |

|  |  |
| --- | --- |
| **URL** | http://localhost:8080/api/updateStore/580faca7c1cfd72890fae6ec |
| **Method** | PUT |
| **Header** | N/A |
| **Content-type** | application/json |
| **Request Body** | {  "storeName": "Mango Store 2",  "lat": -98.34565,  "lon": 24.5644,  "id": "580faca7c1cfd72890fae6ec"  } |
| **Response Body** | [{  "\_id": "580f3b9633b133208ce8ef49",  "storeName": "Store 1",  "position": [38.98076, -76.50346]  }, {  "\_id": "580f3bb533b133208ce8ef4a",  "storeName": "Store 2",  "position": [-76.53059, 38.99624]  }, {  "\_id": "580faca7c1cfd72890fae6ec",  "storeName": "Mango Store 2",  "position": [-98.34565, 24.5644]  }] |

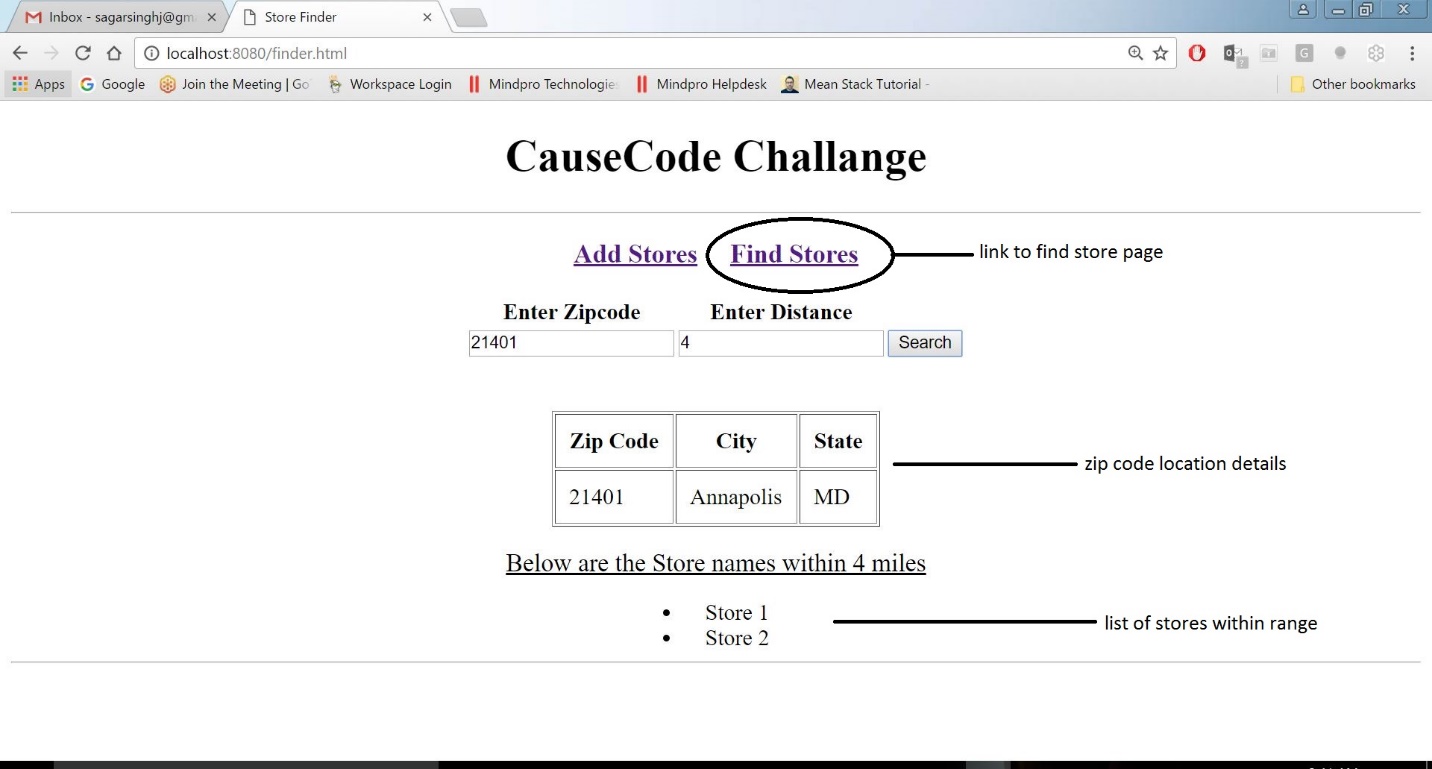
|  |  |
| --- | --- |
| **URL** | http://localhost:8080/api/deleteStore/580faca7c1cfd72890fae6ec |
| **Method** | DELETE |
| **Header** | N/A |
| **Content-type** | application/json |
| **Request Body** | N/A |
| **Response Body** | [{  "\_id": "580f3b9633b133208ce8ef49",  "storeName": "Store 1",  "position": [38.98076, -76.50346]  }, {  "\_id": "580f3bb533b133208ce8ef4a",  "storeName": "Store 2",  "position": [-76.53059, 38.99624]  }] |

|  |  |
| --- | --- |
| **URL** | http://localhost:8080/api/findStore |
| **Method** | POST |
| **Header** | N/A |
| **Content-type** | application/json |
| **Request Body** | {  "zipcode": "21401",  "distance": "4"  } |
| **Response Body** | {  "stores": [{  "\_id": "580f3bb533b133208ce8ef4a",  "storeName": "Store 2",  "position": [-76.53059, 38.99624]  }],  "zipcodeCity": "Annapolis",  "zipcodeState": "MD",  "zipcode": "21401",  "distance": "4"  } |

* **UI for Adding Store information in database**



* **UI for finding stores**



**How to execute the Application**

* **Clone the project from the bitbucket repository**
* **Open the command prompt**
* **Browse to the project folder having package.json file**
* **Run command – “npm install” and let the installation complete**
* **Open another command prompt and start the mongo server with command “mongod”**
* **In the first command prompt window make sure you have browsed till the folder containing server.js file**
* **Now type command “node server.js” and hit enter to run application. You will see the following line printed**

**“App listening on port 8080”**

* **Now open the browser and enter the following address in the address bar and hit enter**

<http://localhost:8080>

* **Application is started.**

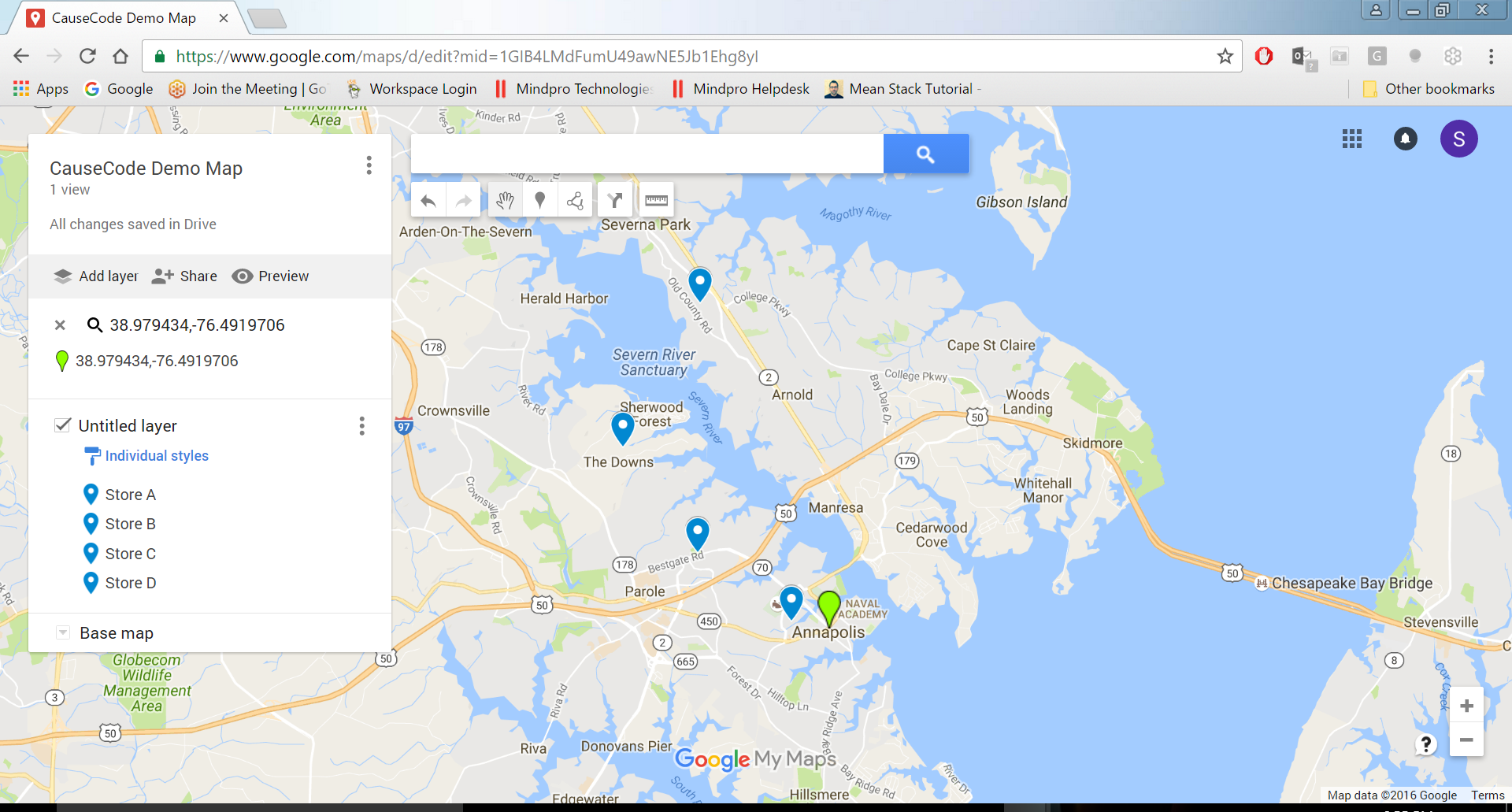
**Use Cases**

In the following use cases I have took the real-time map for testing and for taking the input values for Store locations. The yellow pin on the map shows the center point of the given zip code.

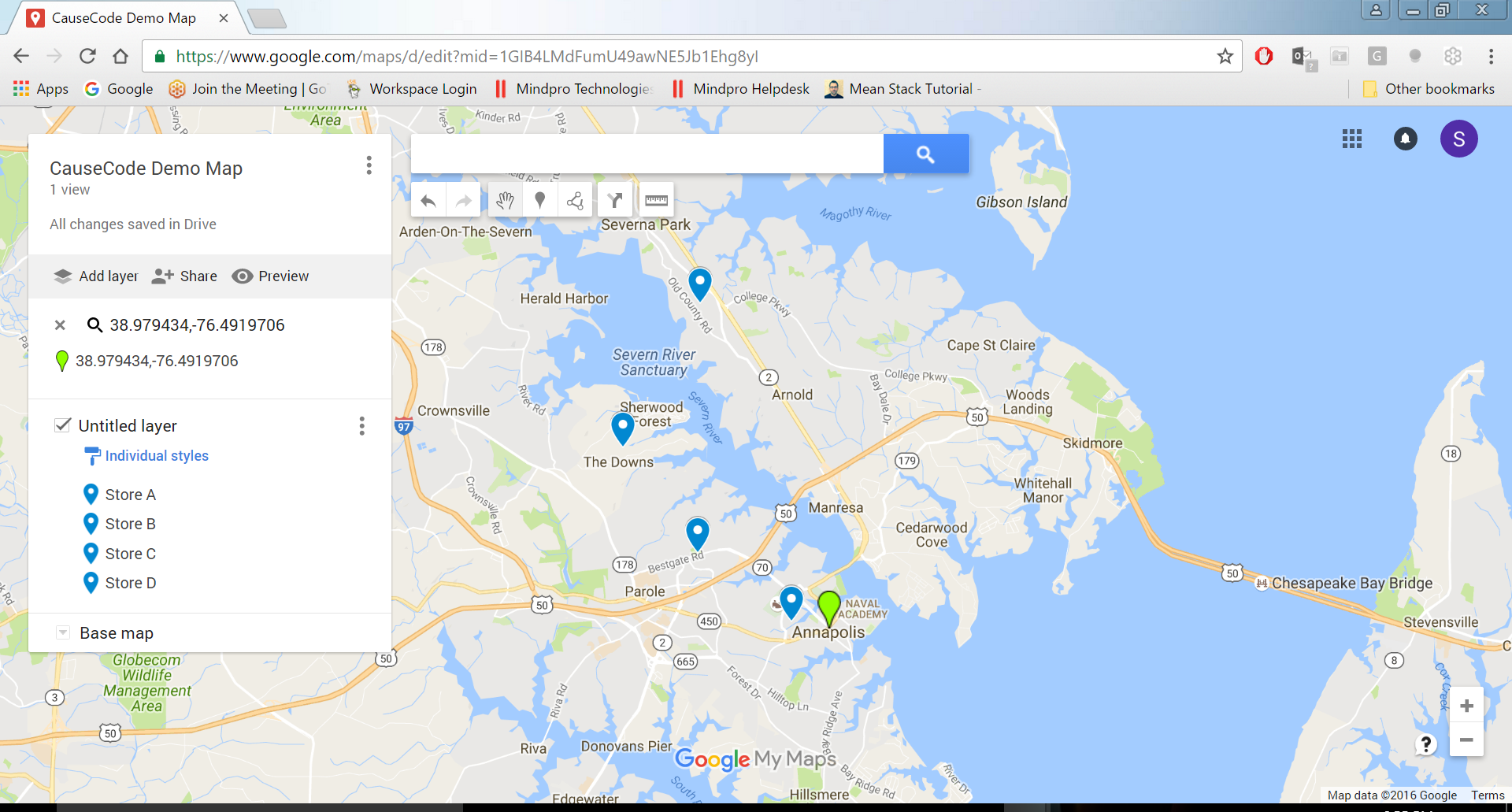
e.g. Foe zip code 21401 Annapolis is the yellow pin which is the center point for this zip-code

All other blue color pins are the actual locations of stores on the map. The name given for this stores can be seen on the left-hand side panel.

**Note: You can use the following values which are given in the table under use cases to test the application and see the expected output**



* **Use Case for Zip code 21401**



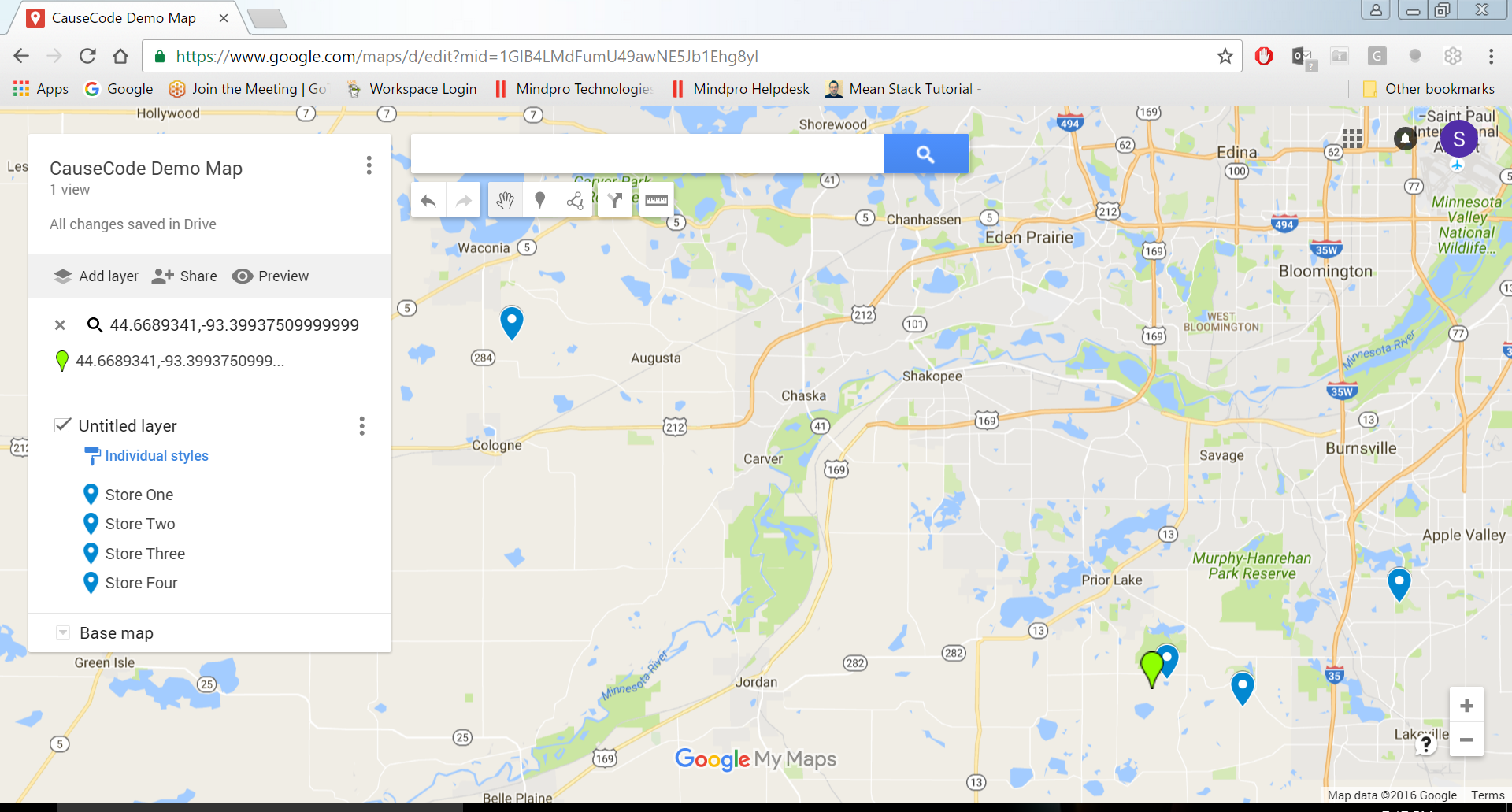
* **Data to Add Stores**

|  |  |  |
| --- | --- | --- |
| **Name** | **Latitude** | **Longitude** |
| Store A | 38.98076 | -76.50346 |
| Store B | 38.99624 | -76.53059 |
| Store C | 39.02011 | -76.55239 |
| Store D | 39.05265 | -76.5299 |

* **Data to Find Stores**

|  |  |  |
| --- | --- | --- |
| **Zip Code** | **Miles** | **Output** |
| 21401 | 1 | Store A |
| 21401 | 2 | Store A |
| 21401 | 3 | Store A  Store B |
| 21401 | 4 | Store A  Store B |
| 21401 | 5 | Store A  Store B  Store C |
| 21401 | 6 | Store A  Store B  Store C  Store D |
| 21401 | 7 | Store A  Store B  Store C  Store D |

* **Use case for zip code 55372**



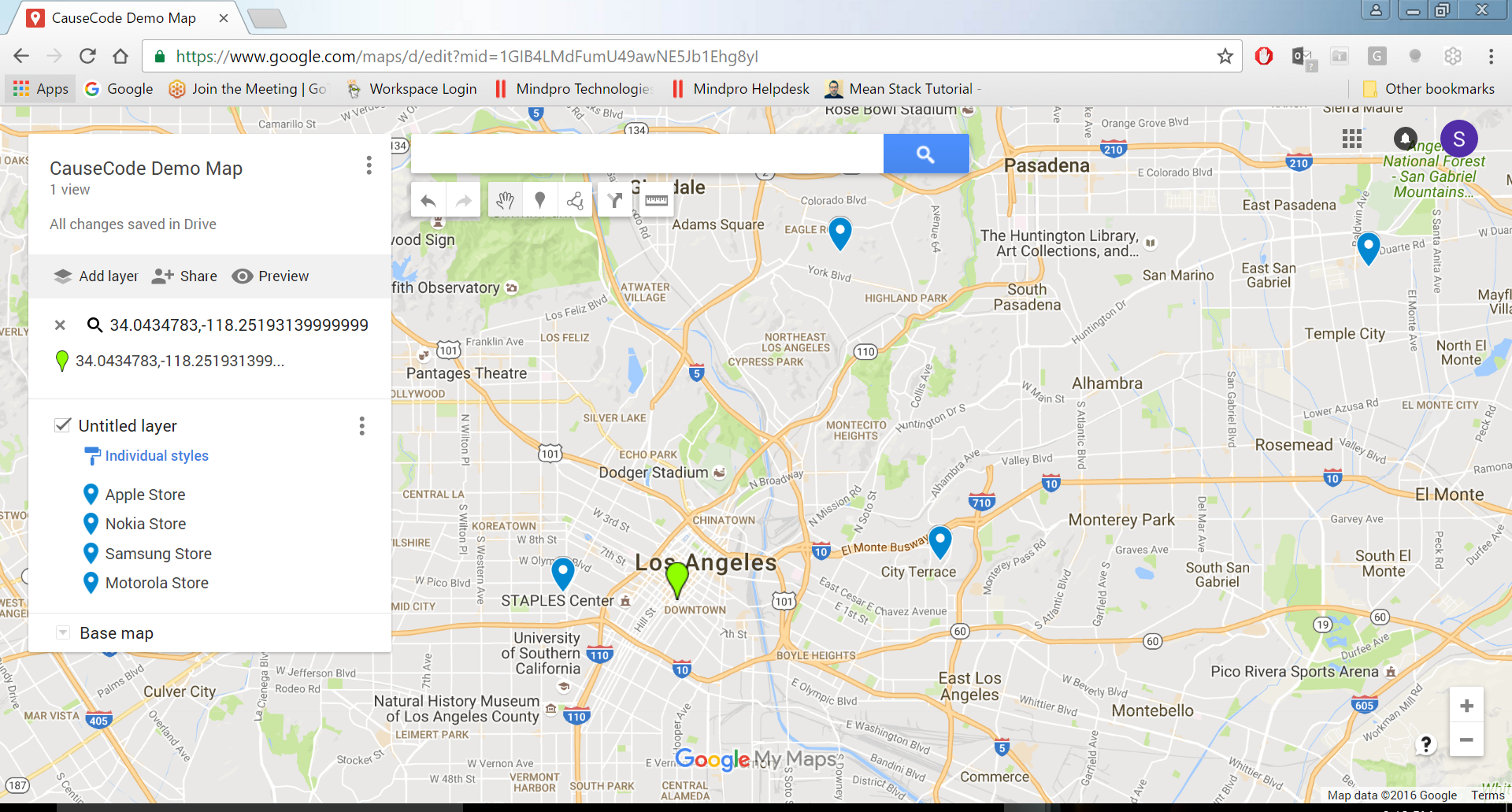
* **Data to Add Stores**

|  |  |  |
| --- | --- | --- |
| **Name** | **Latitude** | **Longitude** |
| Store One | 44.67243 | -93.39151 |
| Store Two | 44.66108 | -93.34722 |
| Store Three | 44.70404 | -93.25641 |
| Store Four | 44.81204 | -93.77243 |

* **Data to Find Stores**

|  |  |  |
| --- | --- | --- |
| **Zip Code** | **Miles** | **Output** |
| 55372 | 4 | Store One |
| 55372 | 6 | Store One  Store Two |
| 55372 | 9 | Store One  Store Two  Store Three |
| 55372 | 18 | Store One  Store Two  Store Three  Store Four |

* **Use Case for zip code 90014**



* **Data to Add Stores**

|  |  |  |
| --- | --- | --- |
| **Name** | **Latitude** | **Longitude** |
| Apple Store | 44.67243 | -93.39151 |
| Nokia Store | 44.66108 | -93.34722 |
| Samsung Store | 44.70404 | -93.25641 |
| Motorola Store | 44.81204 | -93.77243 |

* **Data to find stores**

|  |  |  |
| --- | --- | --- |
| **Zip Code** | **Miles** | **Output** |
| 90014 | 3 | Apple Store |
| 90014 | 5 | Apple Store  Nokia Store |
| 90014 | 7 | Apple Store  Nokia Store  Samsung Store |
| 90014 | 13 | Apple Store  Nokia Store  Samsung Store  Motorola Store |