CRUD functions with Node.js and AWS dynamoDB

Maeluenie Jul 22, 2020 · 4 min read ★

In this post, I will be showing how I setup local AWS DynamoDB instances along with the implementation of some basic CRUD functions using Node.js. I used MERN-Boilerplate code on the *master-w-dynamodb* as the project template and changed it to how I want to implement my services. You can also view my finished code for CRUD-with-dynamodb here. For this project, you'll need to have Node.js installed and AWS CLI by typing brew install awscli onto terminal for MacOSx users.

Implementation

Setup

First, clone the repository, either MERN-Boilerplate code or CRUD-with-dynamodb onto your local machine. Extract the file and use the command line to access to the directory. Then run:

> *npm install*

Then check if there are a file called config.js inside. If you cloned MERN-Boilerplate code remember to change *config.example.js* to *config.js* inside the config folder.

AWS DynamoDB

For this, you'll need to download the local app for DynamoDB based on your prefered region that are closest to you. Load the zip file and extract it, then move it where you want to place. Use your command line and go into the directory where you place it, then run the following command to start it.

> *java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -sharedDb*

Once done, open a new terminal command line window and configure your local machine. Run the following command and enter the following values:

> *aws configure*
>
> *AWS Access Key ID [None]: foo*
> *AWS Secret Access Key [None]: bar*

> *Default region name [None]: local*
> *Default output format [None]: json*

After you're done with the configuration on your local machine, test to see if there are any tables on your local machine by run the following command.

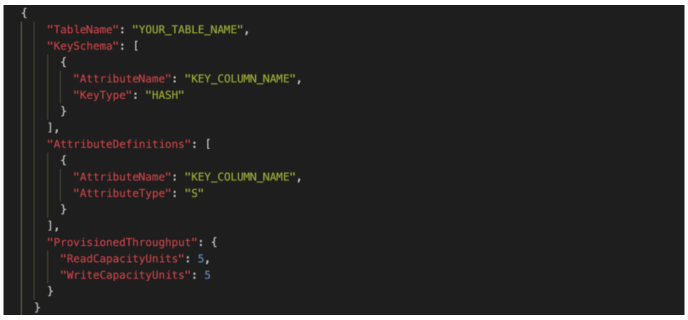
> *aws dynamodb list-tables — endpoint-url http://localhost:8000*

Expected result:

> *{*
> *"TableNames": []*
> *}*

Create a table

To create a table you'll need to create a file called "YOUR_TABLE_NAME.json" in the path `config/tables` and create a table from the following template.



```
{
  "TableName": "YOUR_TABLE_NAME",
  "KeySchema": [
    {
      "AttributeName": "KEY_COLUMN_NAME",
      "KeyType": "HASH"
    }
  ],
  "AttributeDefinitions": [
    {
      "AttributeName": "KEY_COLUMN_NAME",
      "AttributeType": "S"
    }
  ],
  "ProvisionedThroughput": {
    "ReadCapacityUnits": 5,
    "WriteCapacityUnits": 5
  }
}
```

Then you'll need to change the `YOUR_TABLE_NAME` to your own table name and the `KEY_COLUMN_NAME` into your primary key that you are going to use. For this table that I'm creating about the client's information my primary key is the clients ID or just ID which is a string.

Now that you've set everything, we will need to create the table and connect it to your local DyanmoDB that you've previously download and run it on your command line. Open a new terminal window and run the following command on your command line.

> *aws dynamodb create-table — cli-input-json file://YOUR_FULL_PATH/config/tables/YOUR_TABLE_NAME.json — endpoint-url http://localhost:8000*

Now run the command to view the table lists.

> *aws dynamodb list-tables — endpoint-url http://localhost:8000*

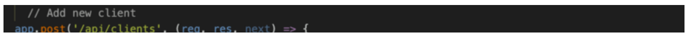
And you should see the expected result from the table that you created.

> *{*
> *"TableNames": [*
> *"YOUR_TABLE_NAME"*
> *]*
> *}*

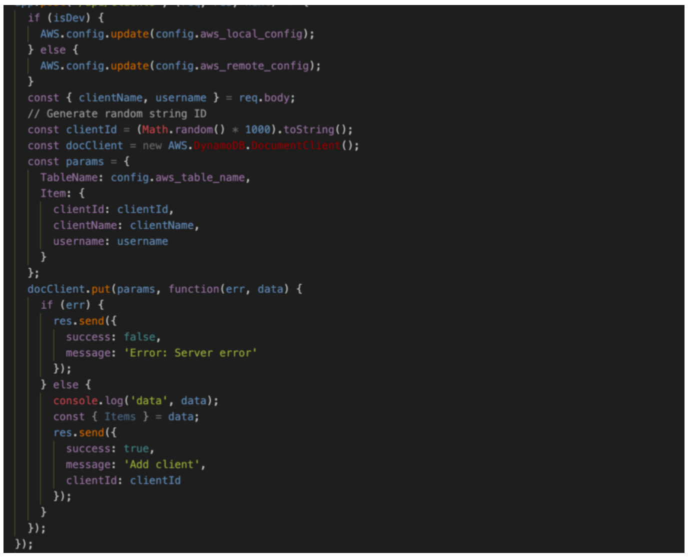
Code CRUD functions

First, update the config file by changing table name into your own table name. Then code for CRUD functions in the `server/routes/api/clients.js`

Create a new client

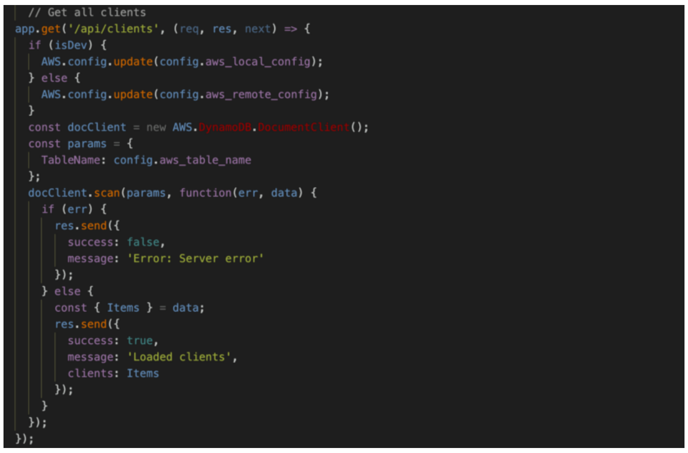


```
  // Add new client
app.post('/api/clients', (req, res, next) => {
```



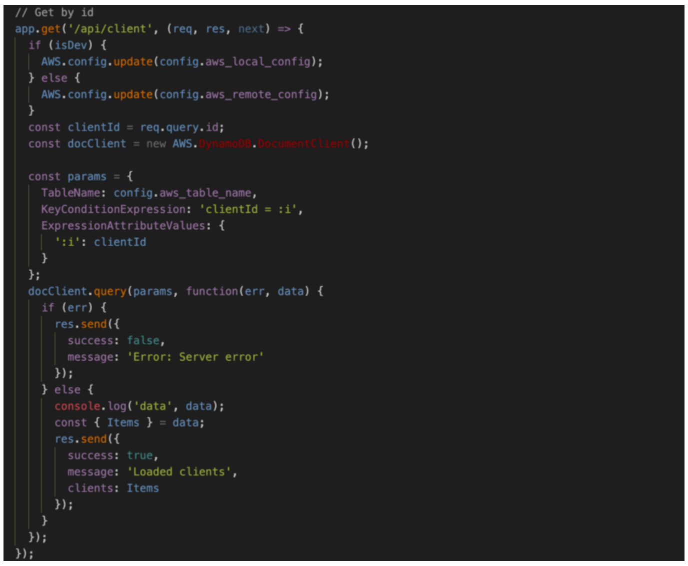
```
  if (isDev) {
    AWS.config.update(config.aws_local_config);
  } else {
    AWS.config.update(config.aws_remote_config);
  }
  const { clientName, username } = req.body;
  // Generate random string ID
  const clientId = (Math.random() * 1000).toString();
  const docClient = new AWS.DynamoDB.DocumentClient();
  const params = {
    TableName: config.aws_table_name,
    Item: {
      clientId: clientId,
      clientName: clientName,
      username: username
    }
  };
  docClient.put(params, function(err, data) {
    if (err) {
      res.send({
        success: false,
        message: 'Error: Server error'
      });
    } else {
      console.log('data', data);
      const { Items } = data;
      res.send({
        success: true,
        message: 'Add client',
        clientId: clientId
      });
    }
  });
});
```

Get all clients informations from the table



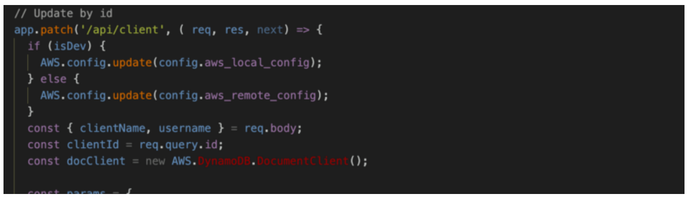
```
  // Get all clients
app.get('/api/clients', (req, res, next) => {
  if (isDev) {
    AWS.config.update(config.aws_local_config);
  } else {
    AWS.config.update(config.aws_remote_config);
  }
  const docClient = new AWS.DynamoDB.DocumentClient();
  const params = {
    TableName: config.aws_table_name
  };
  docClient.scan(params, function(err, data) {
    if (err) {
      res.send({
        success: false,
        message: 'Error: Server error'
      });
    } else {
      const { Items } = data;
      res.send({
        success: true,
        message: 'Loaded clients',
        clients: Items
      });
    }
  });
});
```

Get client's information from the table by ID

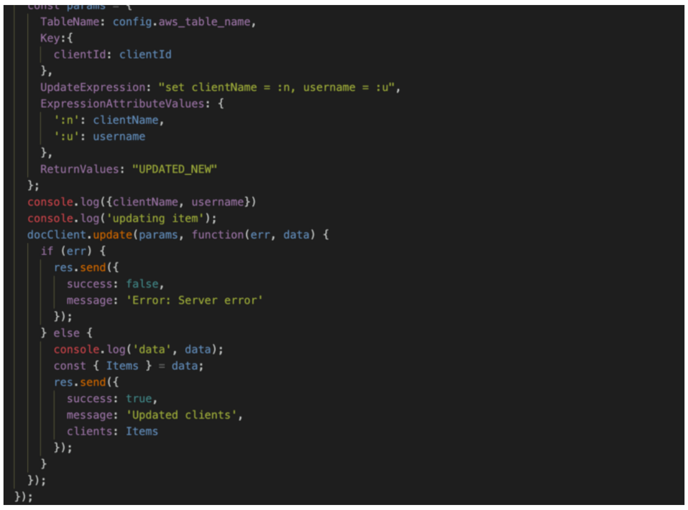


```
// Get by id
app.get('/api/client', (req, res, next) => {
  if (isDev) {
    AWS.config.update(config.aws_local_config);
  } else {
    AWS.config.update(config.aws_remote_config);
  }
  const clientId = req.query.id;
  const docClient = new AWS.DynamoDB.DocumentClient();

  const params = {
    TableName: config.aws_table_name,
    KeyConditionExpression: 'clientId = :i',
    ExpressionAttributeValues: {
      ':i': clientId
    }
  };
  docClient.query(params, function(err, data) {
    if (err) {
      res.send({
        success: false,
        message: 'Error: Server error'
      });
    } else {
      console.log('data', data);
      const { Items } = data;
      res.send({
        success: true,
        message: 'Loaded clients',
        clients: Items
      });
    }
  });
});
```

Update client's information to the table by ID

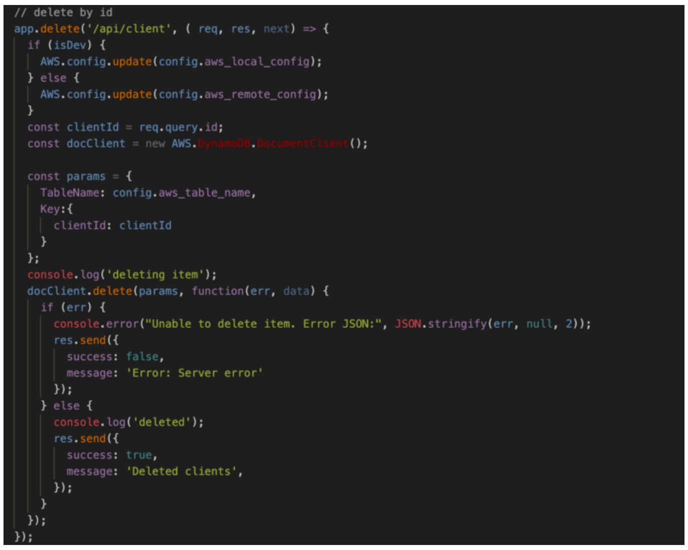


```
// Update by id
app.patch('/api/client', ( req, res, next) => {
  if (isDev) {
    AWS.config.update(config.aws_local_config);
  } else {
    AWS.config.update(config.aws_remote_config);
  }
  const { clientName, username } = req.body;
  const clientId = req.query.id;
  const docClient = new AWS.DynamoDB.DocumentClient();

```



```
  const params = {
    TableName: config.aws_table_name,
    Key:{
      clientId: clientId
    },
    UpdateExpression: "set clientName = :n, username = :u",
    ExpressionAttributeValues: {
      ':n': clientName,
      ':u': username
    },
    ReturnValues: "UPDATED_NEW"
  };
  console.log({clientName, username})
  console.log('updating item');
  docClient.update(params, function(err, data) {
    if (err) {
      res.send({
        success: false,
        message: 'Error: Server error'
      });
    } else {
      console.log('data', data);
      const { Items } = data;
      res.send({
        success: true,
        message: 'Updated clients',
        clients: Items
      });
    }
  });
});
```

Delete client’s information from the table by ID

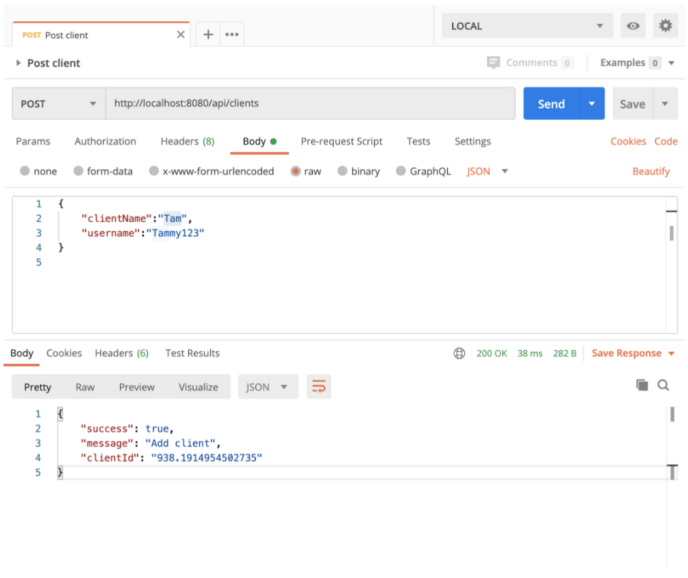


```
// delete by id
app.delete('/api/client', ( req, res, next) => {
  if (isDev) {
    AWS.config.update(config.aws_local_config);
  } else {
    AWS.config.update(config.aws_remote_config);
  }
  const clientId = req.query.id;
  const docClient = new AWS.DynamoDB.DocumentClient();

  const params = {
    TableName: config.aws_table_name,
    Key:{
      clientId: clientId
    }
  };
  console.log('deleting item');
  docClient.delete(params, function(err, data) {
    if (err) {
      console.error("Unable to delete item. Error JSON:", JSON.stringify(err, null, 2));
      res.send({
        success: false,
        message: 'Error: Server error'
      });
    } else {
      console.log('deleted');
      res.send({
        success: true,
        message: 'Deleted clients',
      });
    }
  });
});
```

Tests

For testing if our functions work, use Postman or any REST client to send requests to those endpoints and see the response whether it matches with our expected responses.

POST [/api/clients] : add client into the table



POST Post client

LOCAL

Post client

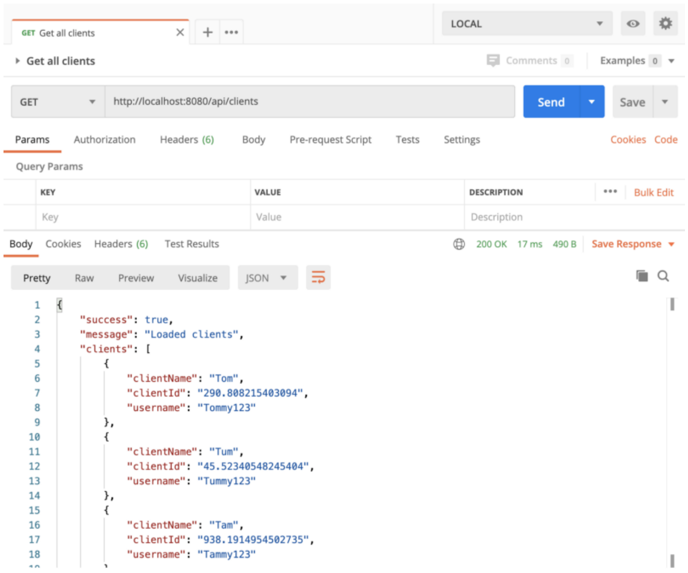
POST http://localhost:8080/api/clients

```
{
    "clientName":"Tam",
    "username":"Tammy123"
}
```

200 OK 38 ms 282 B

```
{
    "success": true,
    "message": "Add client",
    "clientId": "938.1914954502735"
}
```

GET ALL[/api/clients] : get all the clients info from the table and return in list of JSON objects template.

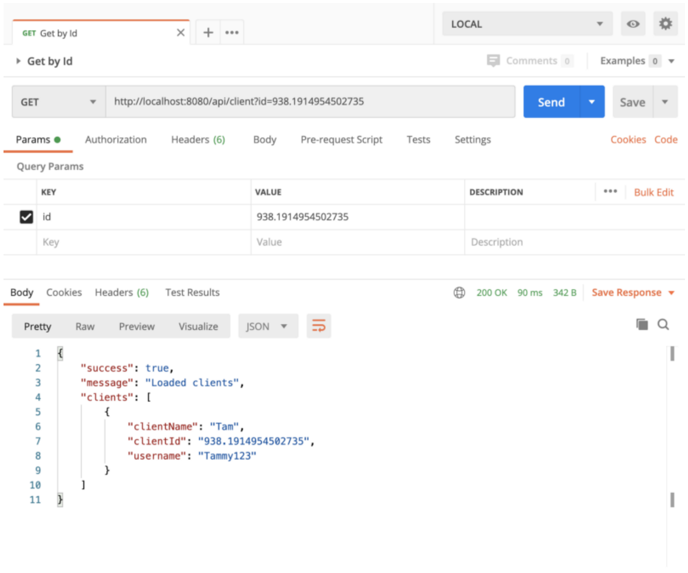


GET http://localhost:8080/api/clients

200 OK 17 ms 490 B

```
{
    "success": true,
    "message": "Loaded clients",
    "clients": [
        {
            "clientName": "Tom",
            "clientId": "290.808215403094",
            "username": "Tommy123"
        },
        {
            "clientName": "Tum",
            "clientId": "45.52340548245404",
            "username": "Tummy123"
        },
        {
            "clientName": "Tam",
            "clientId": "938.1914954502735",
            "username": "Tammy123"
```

GET BY ID [/api/client?id=:id] : find a client and return the client's information from a specific ID input.



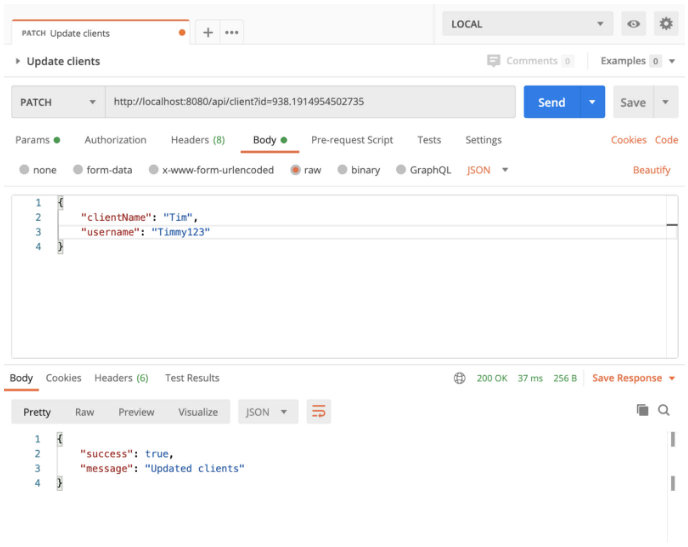
GET http://localhost:8080/api/client?id=938.1914954502735

	KEY	VALUE	DESCRIPTION
✓	id	938.1914954502735	

200 OK 90 ms 342 B

```
{
    "success": true,
    "message": "Loaded clients",
    "clients": [
        {
            "clientName": "Tam",
            "clientId": "938.1914954502735",
            "username": "Tammy123"
        }
    ]
}
```

UPDATE BY ID [/api/client?id=:id] : update client's information from a given JSON body input.



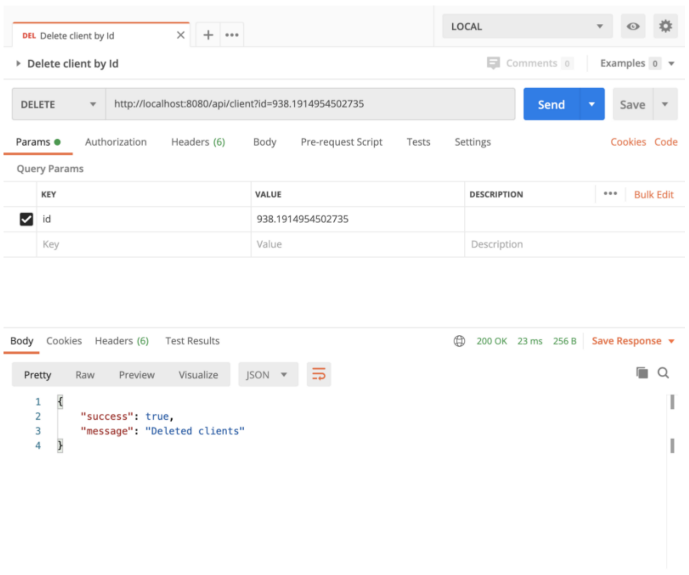
PATCH http://localhost:8080/api/client?id=938.1914954502735

```
{
    "clientName": "Tim",
    "username": "Timmy123"
}
```

200 OK 37 ms 256 B

```
{
    "success": true,
    "message": "Updated clients"
}
```

DELETE [/api/client?id=:id] : delete a client's information from a specific ID input.



```
http://localhost:8080/api/client?id=938.1914954502735
```

KEY	VALUE	DESCRIPTION
✓ id	938.1914954502735	
Key	Value	Description

```
{
    "success": true,
    "message": "Deleted clients"
}
```

References

Getting Started with JavaScript and DynamoDB
In this tutorial, you use JavaScript to write simple programs to perform the following Amazon DynamoDB operations...
docs.aws.amazon.com

Getting Started with Node.js and DynamoDB
In this tutorial, you use the AWS SDK for JavaScript to write simple applications to perform the following Amazon...
docs.aws.amazon.com

Dynamodb Local Nodejs JavaScript AWS