**Team Members:**

Roshini Varada- 26

V H N V R S Sai Pavan -27

**Introduction:**

To perform CRUD operations of customers data using NODEJS and to perform the operations using Express Framework.

**Objective**

1.Creation of new CRUD API with fields like customer name, customer id and customer number. Creating command line functions to perform the CRUD operations using the express framework.

**Prerequisites:**

Node Js should be installed in the machine.

**Design and Implementation:**

**1.** Creating a new CRUD API for Customers data and creating methods for adding and deleting a customer:

1. Imported fs module to make use of utility functions for opening, reading and writing to a file.

**const fs =  require('fs');**

1. Created utility functions like fetchCustomers and saveCustomers which will be reused multiple times by the CRUD functions.

fetchCustomers used to read a file and load the data in JSON format.  
saveCustomers function saves the JSON variable to the file(writing to a file)

var fetchCustomers = () => {

    try {                          //if file won't exist

        var customersString = fs.readFileSync('customers-data.json')

        return JSON.parse(customersString);

    } catch(e){

        return [];

    }

};

var saveCustomers = (customers) => {

    fs.writeFileSync('customers-data.json',JSON.stringify(customers));

};

1. Implemented function to create a customer using id, name and number.

var addCustomer = (id, name, number) => {

    var customers = fetchCustomers();

    var customer = {id, name, number}

    var duplicateCustomers =  customers.filter((customer) => { // to check if customer already exists

        return customer.id === id;

    });

    if (duplicateCustomers.length === 0){

        customers.push(customer);

        saveCustomers(customers);

        return customer;

    }

};

4.Implemented function to get details about a customer using ID.

var getCustomer = (id) => {

    var customers = fetchCustomers();

    var getCustomers =  customers.filter((customer) => {  // to check if customer exists and returns the customer

        return customer.id === id;

    });

    return getCustomers[0];

};

5.Implemented function to update customer details using id.

var updateCustomer = (id, name, number) => {

    var customers = fetchCustomers();

    var customer = {id, name, number};

    var filteredCustomers = customers.filter((customer) => { // return all customers except the matched customer

        return customer.id !== id;

    });

    if ( customers.length !== filteredCustomers.length ) {

        filteredCustomers.push(customer);

        saveCustomers(filteredCustomers);

        return customer;

    }

};

6.Implemented function to remove the customer details from the list using customer ID.

var removeCustomer = (id) => {

    var customers = fetchCustomers(); // reusable func

    var filteredCustomers =  customers.filter((customer) => { // will return all other customers other than "note to be removed"

        return customer.id !== id;

    });

    saveCustomers(filteredCustomers); //save new notes array

    return customers.length !== filteredCustomers.length

};

**2: Creating command line functions using express framework:**

1. Imported fs for read write operations on a file.  
   imported yargs to enable the command line interface.  
   imported customers.js script file to make use of CRUD API functions.
2. const fs =  require('fs');
3. const \_ = require('lodash');
4. const yargs = require('yargs');

2. Created command line operations for adding updating, deleting a customer using the id, name, number fields.

const idOptions = {

describe: 'Customer ID',

demand : true,

alias : 'i'

}

const nameOptions = {

describe: 'Customer Name',

demand : true,

alias : 'n'

}

const numberOptions = {

describe: 'Customer Number',

demand : true,

alias : 'o'

}

3.Creating command line methods to add, update, list and delete customers.

const argv =  yargs

    .command('add','Add a new customer details',{

      id: idOptions,

      name: nameOptions,

      number: numberOptions

    })

    .command('list','List all customer details')

    .command('read','Get a customer details by ID',{

      id: idOptions

    })

    .command('update', "Update customer details", {

        id: idOptions,

        name:nameOptions,

        number: numberOptions

    })

    .command('remove','Remove customer by ID',{

      id:idOptions

    })

    .help()

    .argv;

4.Method For adding a customer to the list

var addCustomer = (id, name, number) => {

    var customers = fetchCustomers();

    var customer = {id, name, number}

    var duplicateCustomers =  customers.filter((customer) => { // to check if customer already exists

        return customer.id === id;

    });

    if (duplicateCustomers.length === 0){

        customers.push(customer);

        saveCustomers(customers);

        return customer;

    }

};

5.Method to list all the customers in the API.

var getAll = () => {

    return fetchCustomers();

};

6.Method to read a customer based on the id in the list.

var getCustomer = (id) => {

    var customers = fetchCustomers();

    var getCustomers =  customers.filter((customer) => {  // to check if customer exists and returns the customer

        return customer.id === id;

    });

    return getCustomers[0];

};

7.To update a customer

var updateCustomer = (id, name, number) => {

    var customers = fetchCustomers();

    var customer = {id, name, number};

    var filteredCustomers = customers.filter((customer) => { // return all customers except the matched customer

        return customer.id !== id;

    });

    if ( customers.length !== filteredCustomers.length ) {

        filteredCustomers.push(customer);

        saveCustomers(filteredCustomers);

        return customer;

    }

};

8.To delete a customer from the list

var removeCustomer = (id) => {

    var customers = fetchCustomers(); // reusable func

    var filteredCustomers =  customers.filter((customer) => { // will return all other customers other than "note to be removed"

        return customer.id !== id;

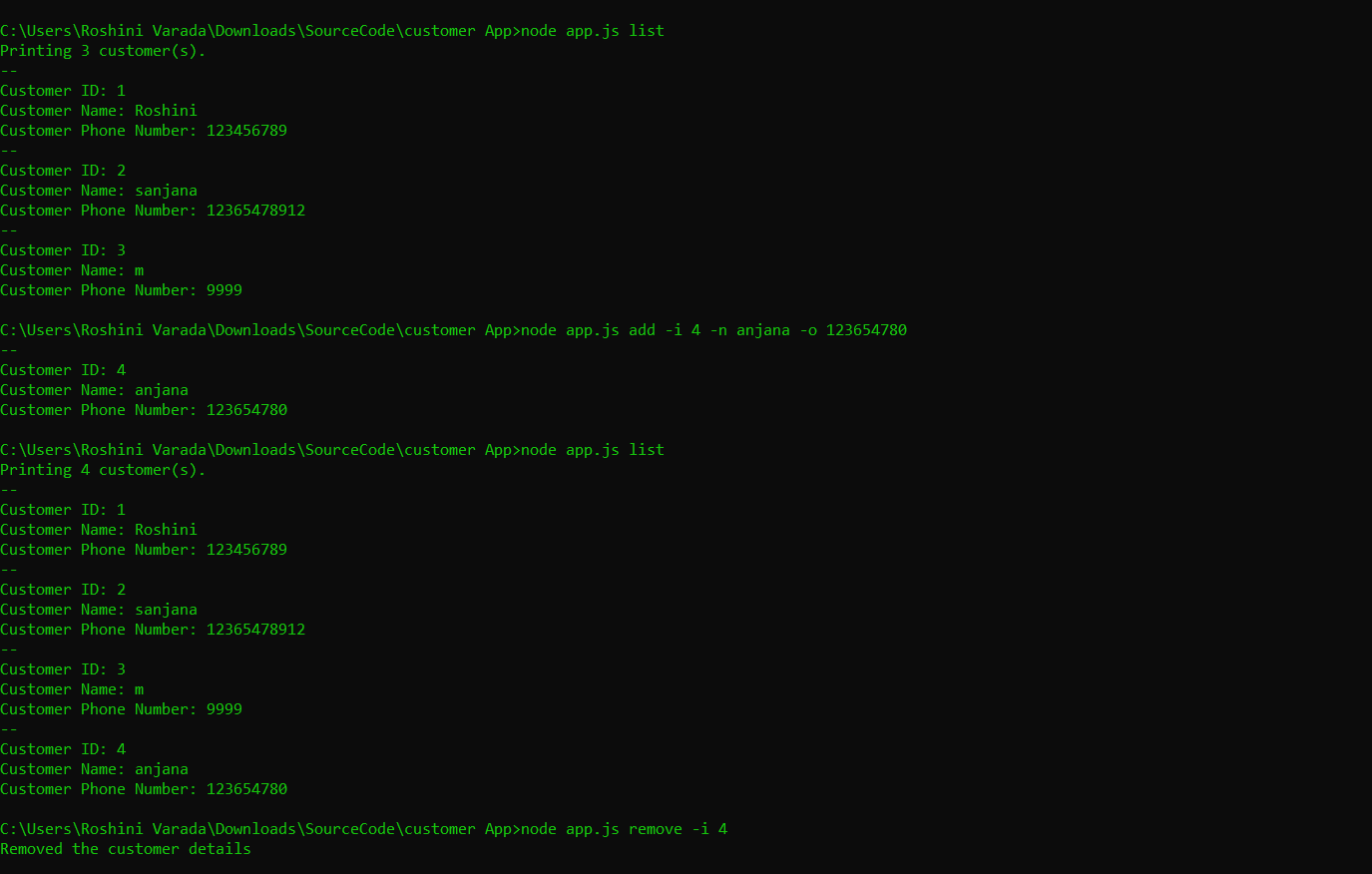
    });

    saveCustomers(filteredCustomers); //save new notes array

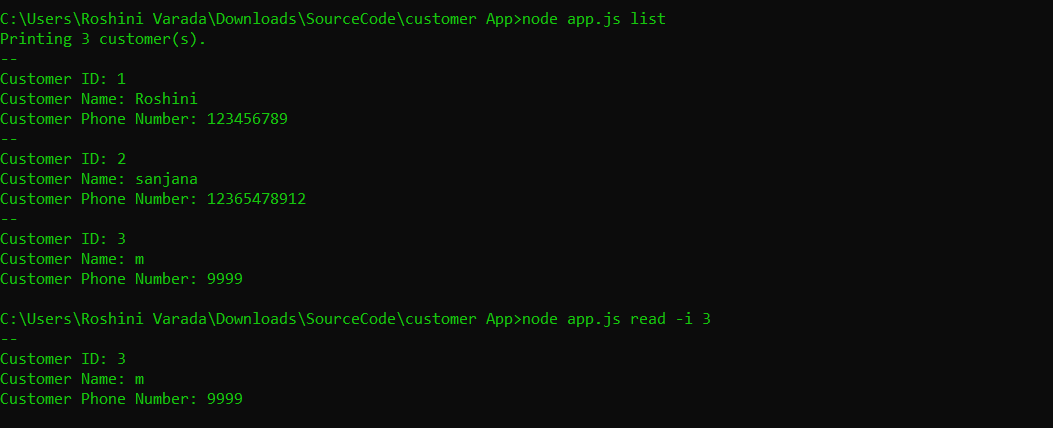
    return customers.length !== filteredCustomers.length

};

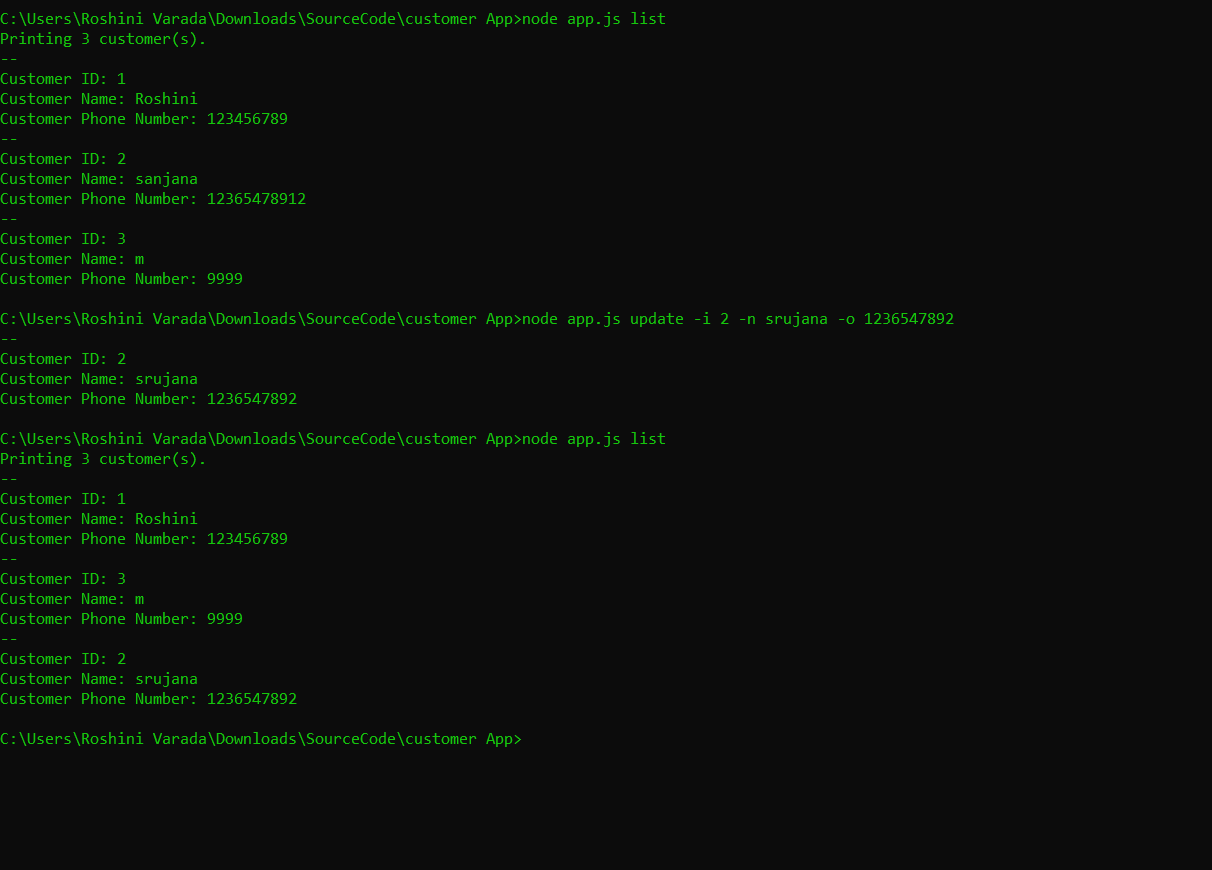
**Outputs:**



**Listing, adding and removing a customer**



**Reading a record from the API**



**Updating a record in the API**

**Conclusion:**

From this experiment we can create an API using Node js and designed the commands for adding, updating and deleting using command line API methods.

**YouTubeLink:**

**https://youtu.be/hjSTO8NOWVM**

**References:**

<https://codeburst.io/writing-a-crud-app-with-node-js-and-mongodb-e0827cbbdafb>