

Spring Boot and MongoDB

- MongoDB is the most popular NoSQL database because of the ease with which data can be stored and retrieved.
- Combining Spring Boot and MongoDB results in applications that are fast, secure, reliable, and require minimum development time.

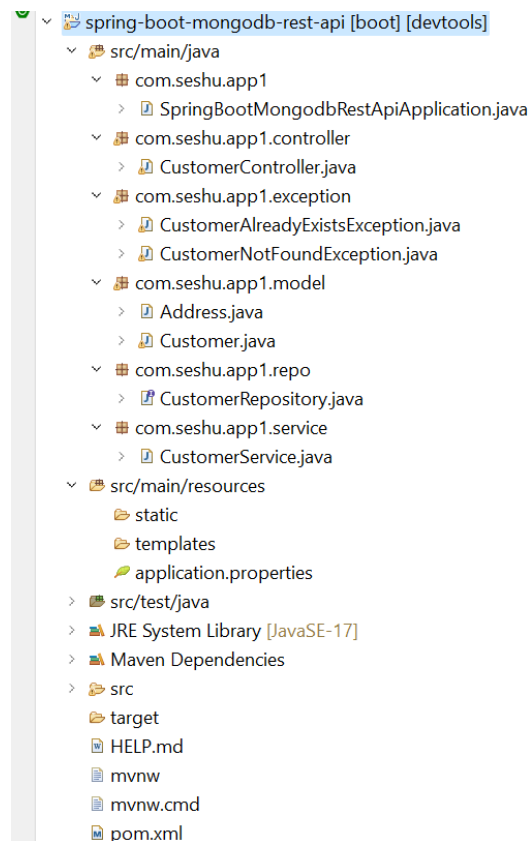
MongoRepository:

MongoRepository is used for basic queries that involve all or many fields of the document.

Examples;

Data creation, viewing documents, and more

Example:



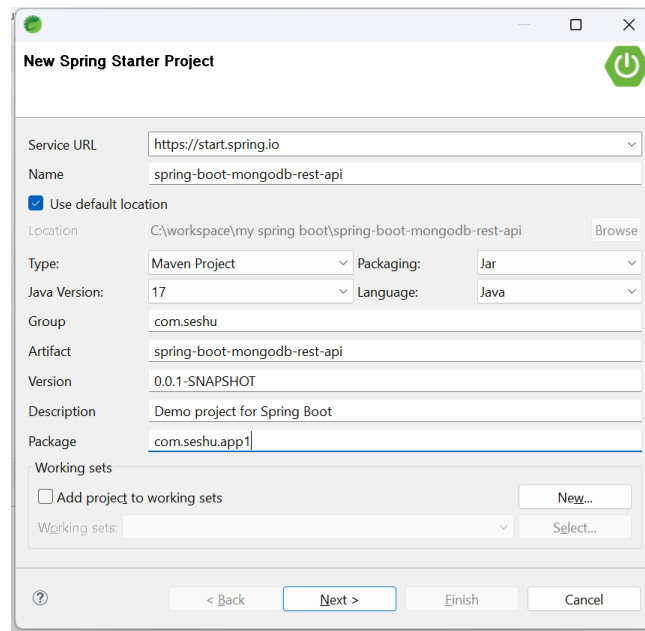
Create a new spring boot project *spring-boot-mongodb-rest-api* with following dependencies

Spring Boot DevTools

Spring Data MongoDB

Spring Web

Lombok



src/main/resources/application.properties

```
server.port=8181
spring.data.mongodb.database=customerdb
spring.data.mongodb.uri=mongodb://localhost:27017/
server.error.include-message=always
```

Address.java

```
package com.seshu.app1.model;

import org.springframework.data.mongodb.core.mapping.Document;

import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.NonNull;
import lombok.RequiredArgsConstructor;

@Data
@Document
@NoArgsConstructor
@RequiredArgsConstructor
public class Address {
    @NonNull
    private String city;

    @NonNull
    private String state;

    @NonNull
    private String country;
}
```

Customer.java

```
package com.seshu.app1.model;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.NonNull;
import lombok.RequiredArgsConstructor;

@Data
@Document
@NoArgsConstructor
@RequiredArgsConstructor
public class Customer {
    @Id
    @NonNull
    private int customerId;
}
```

```
@NonNull
private String customerName;

@NonNull
private String customerEmail;

@NonNull
private Address customerAddress;
}
```

CustomerRepository.java

```
package com.seshu.app1.repo;

import java.util.List;

import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.data.mongodb.repository.Query;

import com.seshu.app1.model.Customer;

public interface CustomerRepository extends MongoRepository<Customer,Integer> {
    @Query("{ 'customerAddress.city' : { $in : [?0] } }")
    List<Customer> findAllCustomerFromCity(String city);
}
```

CustomerAlreadyExistsException.java

```
package com.seshu.app1.exception;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(code = HttpStatus.CONFLICT, reason = "Customer already exists")
public class CustomerAlreadyExistsException extends Exception{
}
}
```

CustomerNotFoundException.java

```
package com.seshu.app1.exception;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(code= HttpStatus.NOT_FOUND , reason = "Customer with specified id is not found")
public class CustomerNotFoundException extends Exception{

}
```

CustomerService.java

```
package com.seshu.app1.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.seshu.app1.exception.CustomerAlreadyExistsException;
import com.seshu.app1.exception.CustomerNotFoundException;
import com.seshu.app1.model.Customer;
import com.seshu.app1.repo.CustomerRepository;

@Service
public class CustomerService {
    @Autowired
    private CustomerRepository customerRepository;

    public Customer saveCustomerDetails(Customer customer) throws CustomerAlreadyExistsException
    {
        if (customerRepository.findById(customer.getCustomerId()).isPresent()) {
            throw new CustomerAlreadyExistsException();
        }
        return customerRepository.save(customer);
    }

    public boolean deleteCustomer(int customerCode) throws CustomerNotFoundException {
        boolean flag = false;
        if (customerRepository.findById(customerCode).isEmpty()) {
            throw new CustomerNotFoundException();
        }
    }
}
```

```
        } else {
            customerRepository.deleteById(customerCode);
            flag = true;
        }
        return flag;
    }

    public List<Customer> getAllCustomerDetails() {
        return customerRepository.findAll();
    }

    public List<Customer> getAllCustomersByCity(String city) {
        return customerRepository.findAllCustomerFromCity(city);
    }

    public Customer updateCustomerDetails(Customer customer) throws CustomerNotFoundException
    {
        if (!customerRepository.findById(customer.getId()).isPresent()) {
            throw new CustomerNotFoundException();
        }
        return customerRepository.save(customer);
    }
}
```

CustomerController.java

```
package com.seshu.app1.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.seshu.app1.exception.CustomerAlreadyExistsException;
import com.seshu.app1.exception.CustomerNotFoundException;
import com.seshu.app1.model.Customer;
```

```
import com.seshu.app1.service.CustomerService;

@RestController
@RequestMapping("/api/v1/customerservice/")
public class CustomerController {

    @Autowired
    private CustomerService customerService;

    private ResponseEntity<?> responseEntity;

    @PostMapping("customer")
    public ResponseEntity<?> saveCustomer(@RequestBody Customer customer) throws
CustomerAlreadyExistsException {
        try {
            customerService.saveCustomerDetails(customer);
            responseEntity = new ResponseEntity<>(customer, HttpStatus.CREATED);
        } catch (CustomerAlreadyExistsException e) {
            throw new CustomerAlreadyExistsException();
        } catch (Exception e) {
            responseEntity = new ResponseEntity<>("Error !!!Try after sometime",
HttpStatus.INTERNAL_SERVER_ERROR);
        }
        return responseEntity;
    }

    @GetMapping("customer")
    public ResponseEntity<?> getAllCustomer() {
        try {
            responseEntity = new ResponseEntity<>(customerService.getAllCustomerDetails(),
HttpStatus.OK);
        } catch (Exception e) {
            responseEntity = new ResponseEntity<>("Error !!! Try after sometime.",
HttpStatus.INTERNAL_SERVER_ERROR);
        }
        return responseEntity;
    }

    @GetMapping("customer/{city}")
    public ResponseEntity<?> getAllCustomerByCity(@PathVariable String city) {
        try {
            responseEntity = new
ResponseEntity<>(customerService.getAllCustomersByCity(city), HttpStatus.OK);
        } catch (Exception e) {
```

```
        responseEntity = new ResponseEntity<>("Error !!! Try after sometime.",
HttpStatus.INTERNAL_SERVER_ERROR);
    }
    return responseEntity;
}

@DeleteMapping("customer/{customerId}")
public ResponseEntity<?> deleteCustomer(@PathVariable("customerId") int customerId)
    throws CustomerNotFoundException {

    try {
        customerService.deleteCustomer(customerId);
        responseEntity = new ResponseEntity<>("Successfully deleted !!!", HttpStatus.OK);
    } catch (CustomerNotFoundException e) {
        throw new CustomerNotFoundException();
    } catch (Exception exception) {
        responseEntity = new ResponseEntity<>("Error !!! Try after sometime.",
HttpStatus.INTERNAL_SERVER_ERROR);
    }
    return responseEntity;
}

@PutMapping("customer")
public ResponseEntity<?> updateCustomer(@RequestBody Customer customer) throws
CustomerNotFoundException {
    try {
        customerService.updateCustomerDetails(customer);
        responseEntity = new ResponseEntity<>(customer, HttpStatus.CREATED);
    } catch (CustomerNotFoundException e) {
        throw new CustomerNotFoundException();
    } catch (Exception e) {
        responseEntity = new ResponseEntity<>("Error !!!Try after sometime",
HttpStatus.INTERNAL_SERVER_ERROR);
    }
    return responseEntity;
}
}
```


CustomerServiceApplication.java

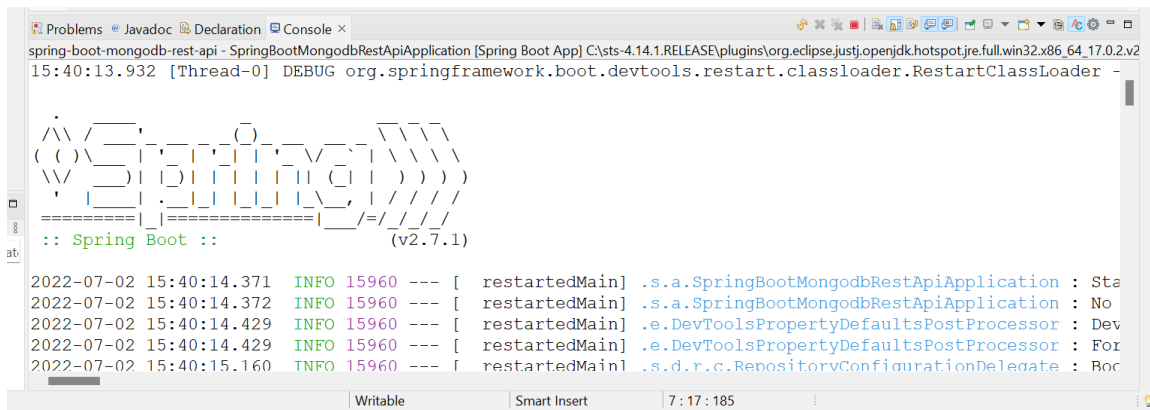
```
package com.seshu.app1;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringBootMongodbRestApiApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringBootMongodbRestApiApplication.class, args);
    }
}
```

Run the starter class.



```
Problems  Javadoc  Declaration  Console x
spring-boot-mongodb-rest-api - SpringBootMongodbRestApiApplication [Spring Boot App] C:\sts-4.14.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v2
15:40:13.932 [Thread-0] DEBUG org.springframework.boot.devtools.restart.classloader.RestartClassLoader -
:: Spring Boot :: (v2.7.1)
2022-07-02 15:40:14.371 INFO 15960 --- [ restartedMain] .s.a.SpringBootMongodbRestApiApplication : Sta
2022-07-02 15:40:14.372 INFO 15960 --- [ restartedMain] .s.a.SpringBootMongodbRestApiApplication : No
2022-07-02 15:40:14.429 INFO 15960 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Dev
2022-07-02 15:40:14.429 INFO 15960 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For
2022-07-02 15:40:15.160 INFO 15960 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Boc
Writable Smart Insert 7 : 17 : 185
```

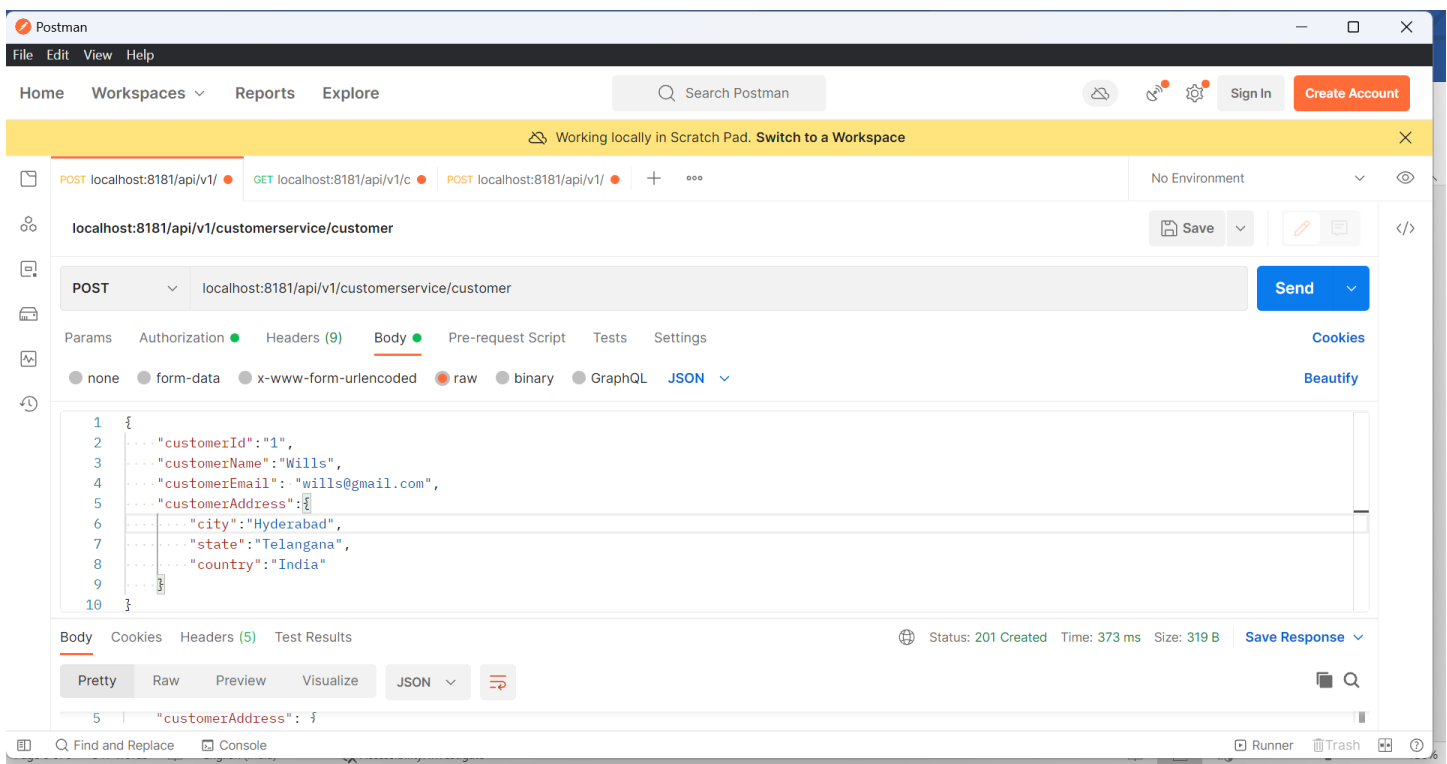
Test using Postman

Post Request

localhost:8181/api/v1/customerservice/customer

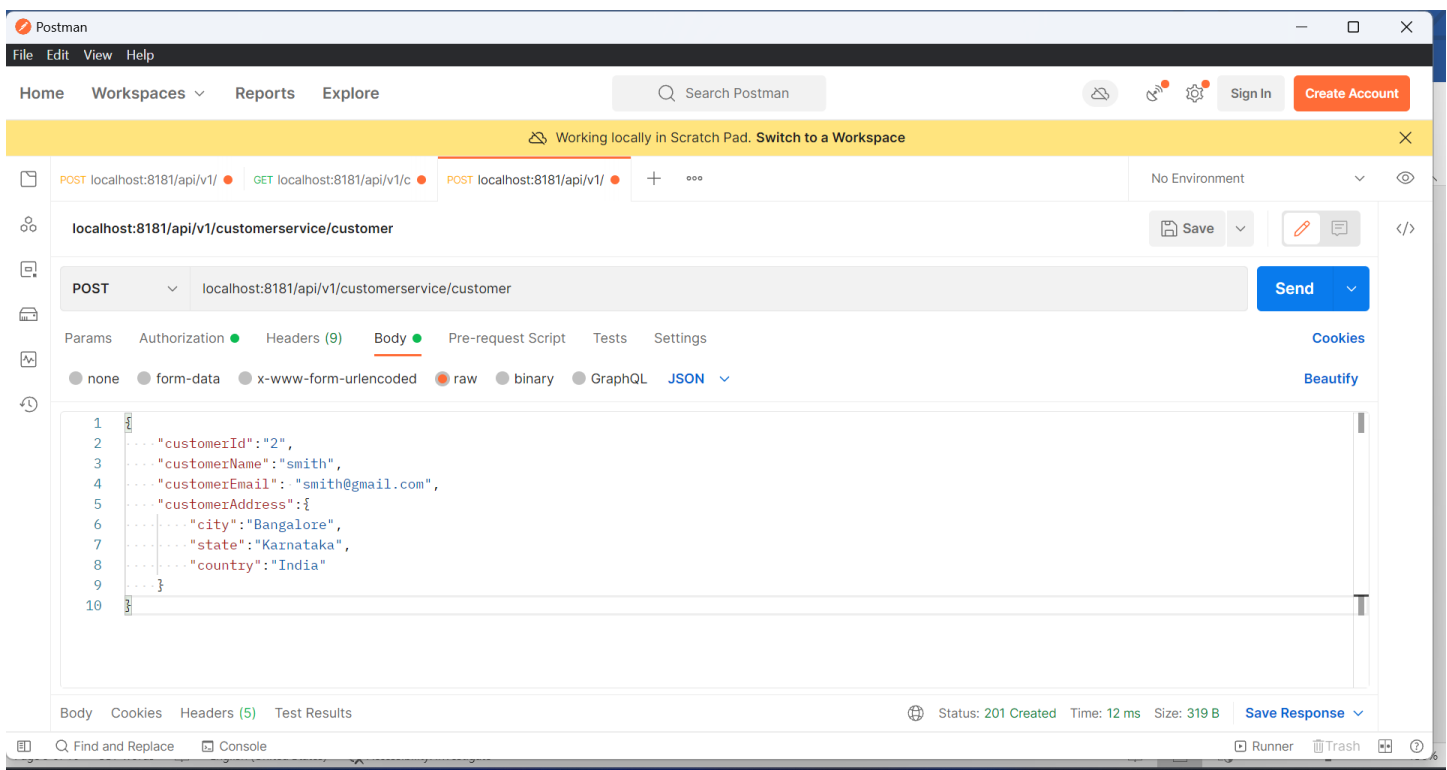
Adding Customer 1 details (add data in Body)

```
{
  "customerId": "1",
  "customerName": "Wills",
  "customerEmail": "wills@gmail.com",
  "customerAddress": {
    "city": "Hyderabad",
    "state": "Telangana",
    "country": "India"
  }
}
```



Adding customer 2 details (add data in Body)

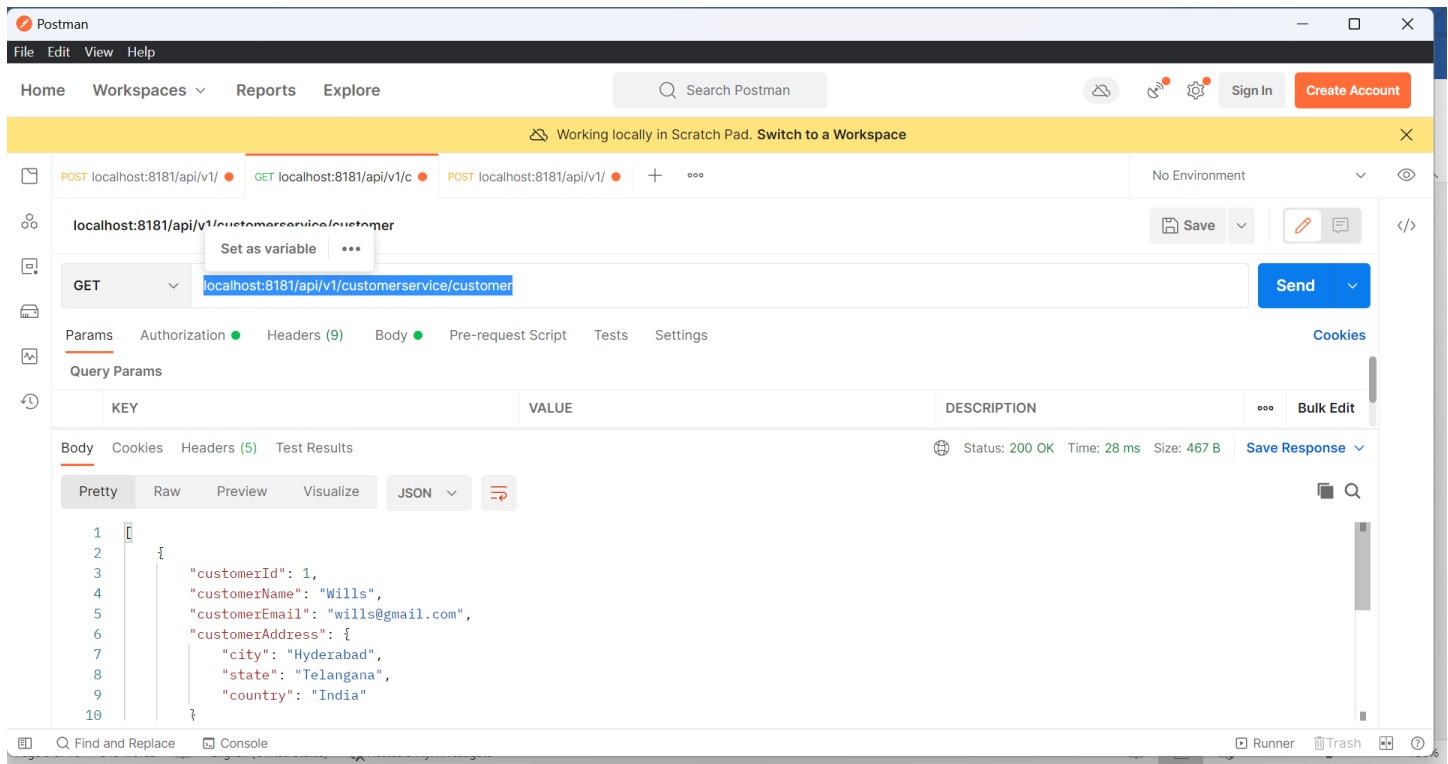
```
{
  "customerId": "2",
  "customerName": "smith",
  "customerEmail": "smith@gmail.com",
  "customerAddress": {
    "city": "Bangalore",
    "state": "Karnataka",
    "country": "India"
  }
}
```



Handling Get Request;

(Get all customer details)

localhost:8181/api/v1/customerservice/customer



Check in MongoDB Server;

Execute these commands in CMD

```
>mongo
```

```
> use customerdb
```

```
switched to db customerdb
```

```
> show collections
```

```
customer
```

```
> db.customer.find().pretty()
```

Note: To drop existing database

```
> use customerdb
```

```
> db.dropDatabase()
```

Handling Get Request; (Get Customer details based on city)

<http://localhost:8181/api/v1/customerservice/customer/Hyderabad>

The screenshot shows the Postman application interface. At the top, there's a navigation bar with 'Home', 'Workspaces', 'Reports', and 'Explore'. Below this is a search bar and buttons for 'Sign In' and 'Create Account'. A yellow banner indicates 'Working locally in Scratch Pad. Switch to a Workspace'. The main workspace shows a list of requests, with the selected one being a GET request to 'localhost:8181/api/v1/customerservice/customer/Hyderabad'. The request details are visible, including the method 'GET' and the URL. The 'Params' tab is active, showing a table with 'KEY' and 'VALUE' columns. The 'Body' tab is also active, showing a JSON response. The response status is '200 OK', with a time of '77 ms' and a size of '316 B'. The response body is displayed in a 'Pretty' JSON format.

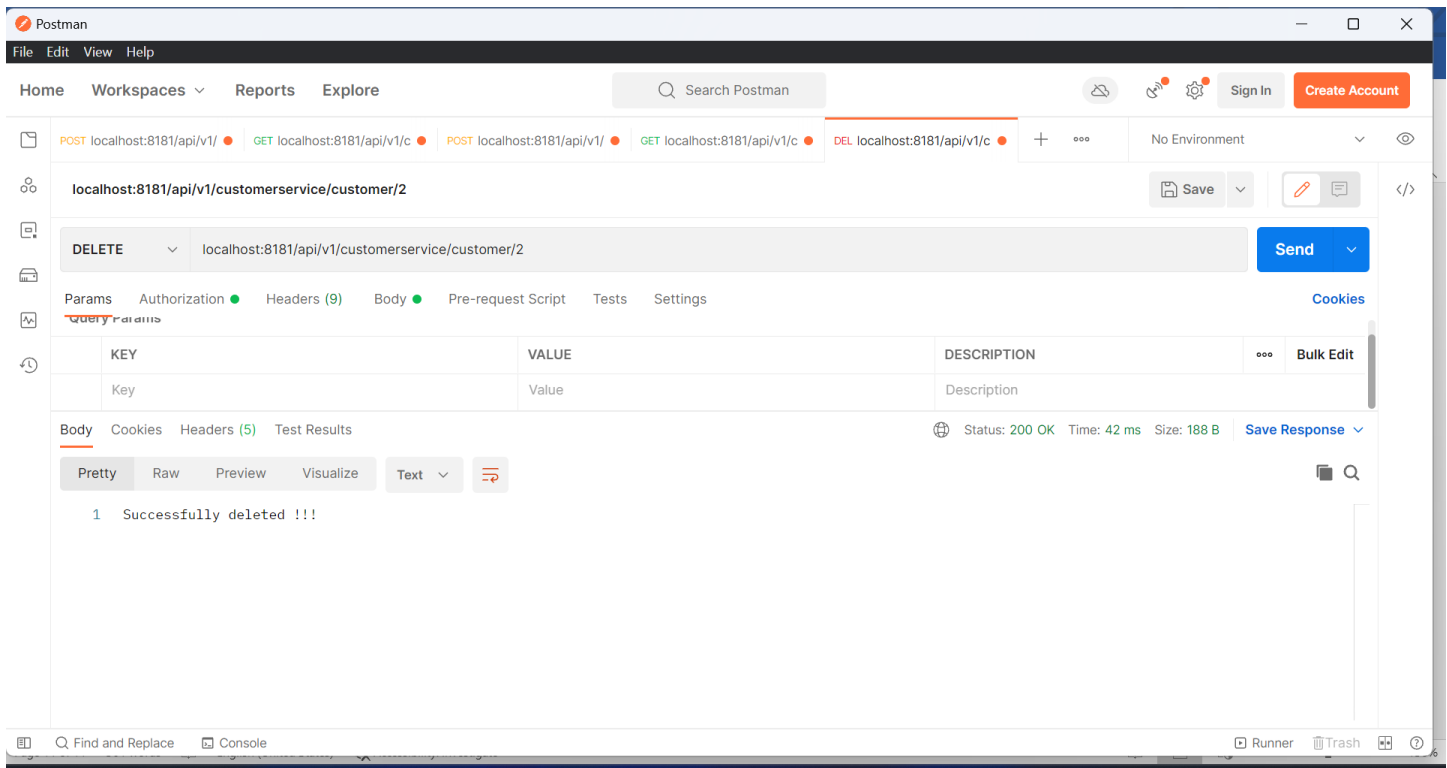
KEY	VALUE	DESCRIPTION
Key	Value	Description

```
1 {
2   "customerId": 1,
3   "customerName": "Wills",
4   "customerEmail": "wills@gmail.com",
5   "customerAddress": {
6     "city": "Hyderabad",
7     "state": "Telangana",
8     "country": "India"
9   }
10 }
```

Handling Delete Request;

(Delete customer by customer id)

localhost:8181/api/v1/customerservice/customer/2

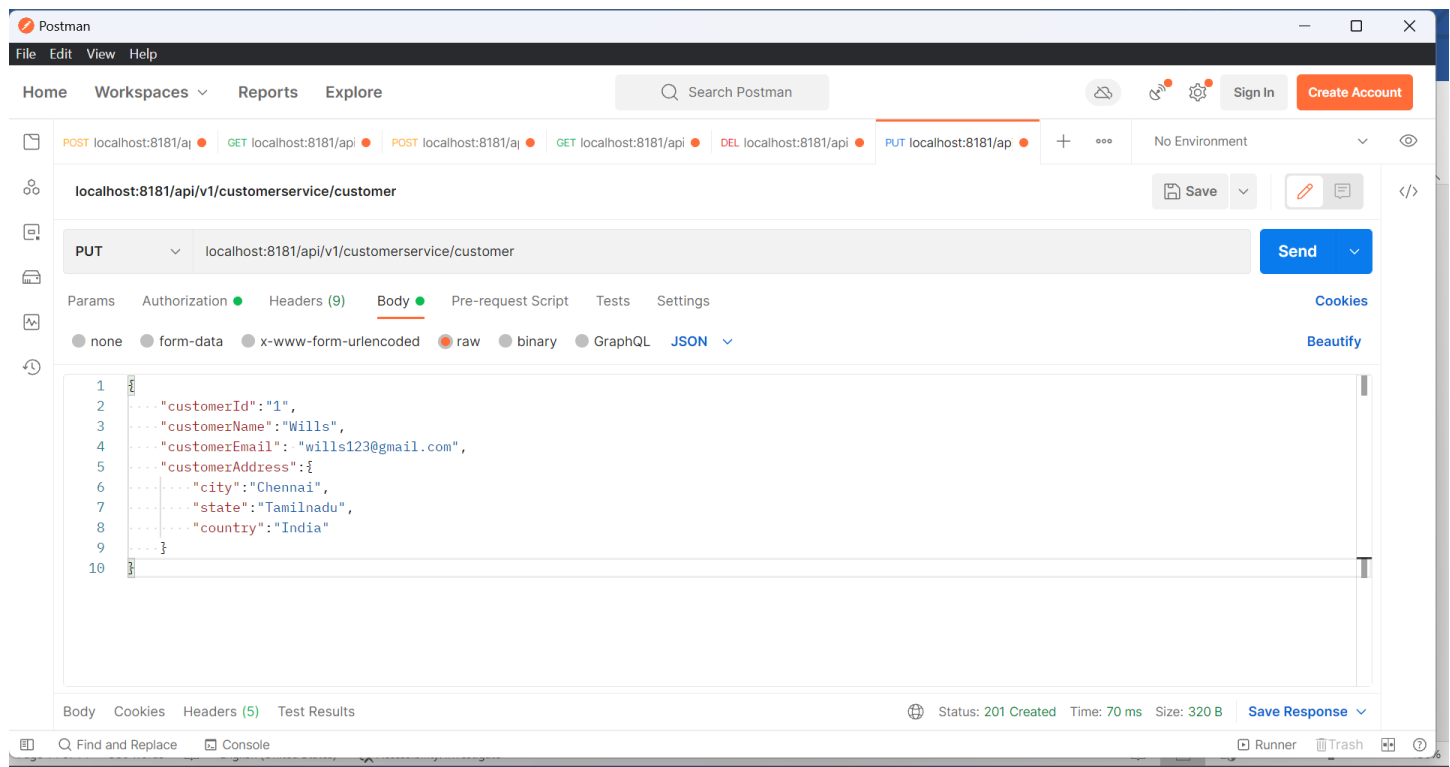


Handling Put Request

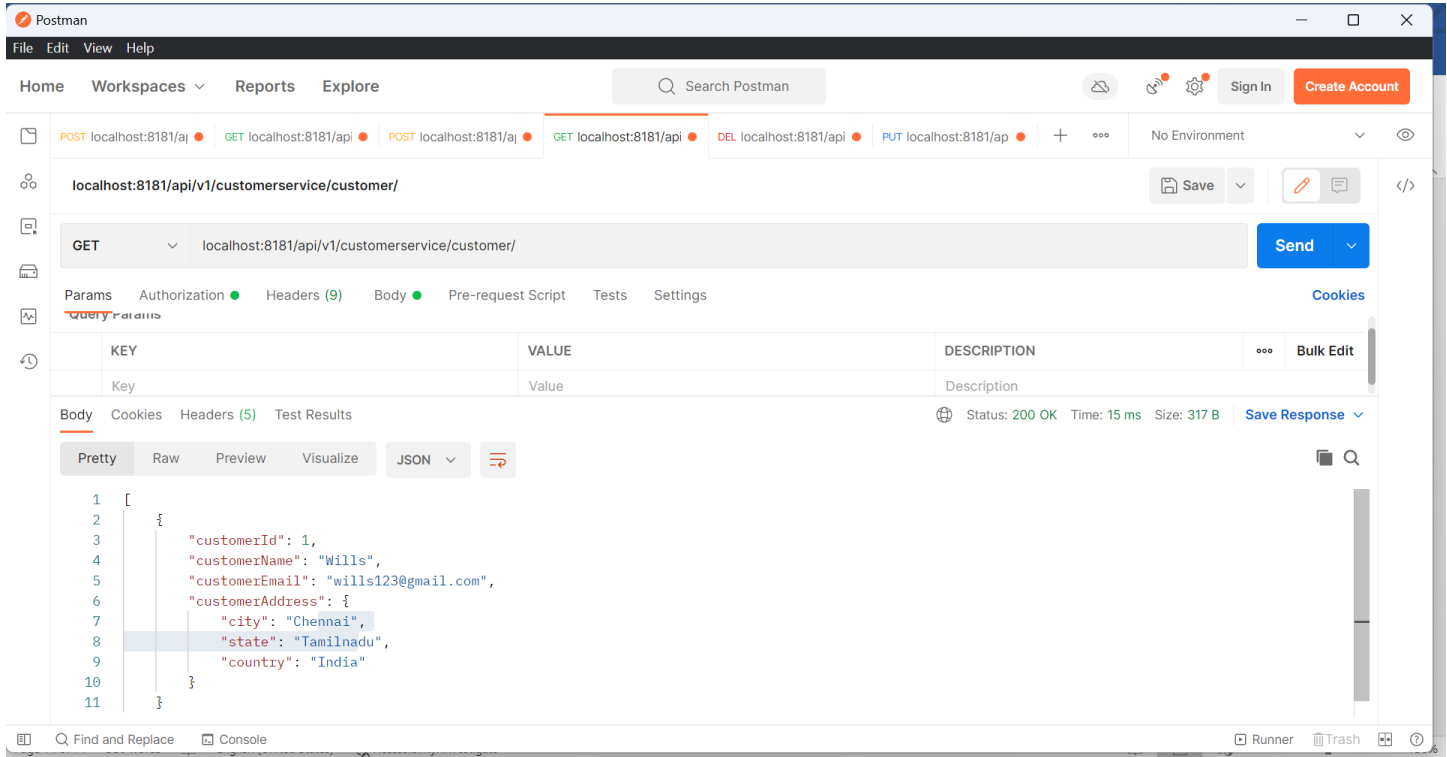
(Update customer details)

localhost:8181/api/v1/customerservice/customer

```
{
  "customerId": "1",
  "customerName": "Wills",
  "customerEmail": "wills123@gmail.com",
  "customerAddress": {
    "city": "Chennai",
    "state": "Tamilnadu",
    "country": "India"
  }
}
```



Updated information;



The screenshot shows the Postman application interface. The top bar includes the Postman logo, menu items (File, Edit, View, Help), a search bar, and user options (Sign In, Create Account). The main workspace displays a collection of requests for the endpoint `localhost:8181/api/v1/customerservice/customer/`. The selected request is a GET method. The response is displayed in the 'Body' tab, showing a 200 OK status, 15 ms time, and 317 B size. The response body is a JSON object with the following structure:

```
1 [
2   {
3     "customerId": 1,
4     "customerName": "Wills",
5     "customerEmail": "wills123@gmail.com",
6     "customerAddress": {
7       "city": "Chennai",
8       "state": "Tamilnadu",
9       "country": "India"
10    }
11  }
```

The bottom of the interface includes a 'Find and Replace' search bar, a 'Console' tab, and a 'Runner' button.