Junit framework for unit testing.pdf

SUBMISSION DATE / POSTED DATE

07-04-2020 / 13-04-2020

CITATION

Venkatesan, Praveen Kumar; Gade Rozario, Rikhil; Fiaidhi, Jinan (2020): Junit framework for unit testing.pdf. TechRxiv. Preprint. https://doi.org/10.36227/techrxiv.12092259.v1

DOI

10.36227/techrxiv.12092259.v1

JUnit Framework For Unit Testing

Praveen Kumar Venkatesan
Department of Computer Science
Lakehead University
Thunder Bay, Canada
pvenkat1@lakeheadu.ca

Rikhil Gade Rozario
Department of Computer Science
Lakehead University
Thunder Bay, Canada
gaderozarior@lakeheadu.ca

Dr. Jinan Fiaidhi
Department of Computer Science
Lakehead University
Thunder Bay, Canada
jfiaidhi@lakeheadu.ca

***Abstract*—Testing software before deploying is a mandatory task in SDLC. Various types of testing tools are used to test the software. This research focuses on JUnit framework to perform unit testing for Java applications. We have developed a Banking Inventory application using spring framework by connecting the application to the MongoDB. The application contains operations such as Create, Update, Delete and find for the customers and Unit test cases has been developed for all the modules using JUnit framework and the test cases are discussed and validated.**

***Index Terms*—SDLC (Software Development Lifecycle), JUnit, test case, MongoDB (Database)**

I. Introduction

In world of Information Technology, software development to satisfy the needs of the stakeholders has been a keen importance to all the professionals ensuring that the stakeholder's requirements are met. Hence software after development will be tested with specific parameters to verify the software.

Software testing is the process of validating the software in order to check whether the functionality of the developed software is intact and to detect errors and rectify them to ensure the quality of the software. Software validation also includes evaluation of the application and working tests out under various settings and situations and evaluating the features of the system.

The software may be used in as a whole or in components. If a product is to be appropriate for usage, each test must be passed. The program would be updated after each set of tests. The research department then completes the next set of experiments until they have been overcome these errors. This process persists until the optimal output is reached. The process of testing a particular software is stopped when Testing Deadlines, completion of test case execution, completion of functional and code coverage to a certain point, bug rate falls below a certain level and no high-priority bugs are identified, management decision.

The advantages of testing lies in the fact that the time and cost can be reduced, bugs can be fixed in at an early stage. Software testing helps in determining following set of properties of any software such as Functionality,Reliability,Usability,Efficiency,Maintainability and re-usability.

Testing can be manual as well as automated. Manual Testing is the process of writing the test cases on our own and validating the software. Automated Testing is process of testing that compared the actual result with the expected result using an automation tool.

Moreover, there are various levels of testing such as Unit testing, Integration testing, System tests and Acceptance Testing. Unit testing is dependent on units. Unit testing is a type of functionality testing. Each unit/part of the application code is tested individually. A unit can be a function, module, object etc. In unit testing multiple units can be tested simultaneously increasing the speed of the detecting the bugs in the code. Application code can be refactored based on the actual output compared to the expected output. As unit testing is the primary base for testing, it helps in better understanding of the code. Unit testing is also known as module testing or component testing. This research primarily focuses on Unit testing and the tool that has been used to write unit test cases.

This paper is organized as follows, Section II focus on brief details and explanation of JUnit framework tool and the operations that can be done it,Section III speaks about the environmental setup for developing a application.Section IV and VI emphasis the application development procedure as well the testing the application.

II. Literature Review

JUnit is a simple unit testing framework used for testing java language. This is an essential technology powered by experiments and a set of constructs known collectively as xUnit. JUnit encourages "first testing then coding," which stresses the creation of test details for the initial and then application part of the code has to be evaluated. This improves programmer efficiency

and system code reliability, which in effect decreases programmer frustration and debug time. This enhances the developer's efficiency. There are mainly two types of unit testing which are mainly manual testing and automated testing. In manual testing we can execute these tests cases manually without any tool support and this method is time consuming and less reliable. In automated testing we can implement the test cases by support tools and it is fast and more reliable.

A. Features of JUnit

JUnit gives an insight of its key test features, which are Fixtures, Test suites, Test runners and JUnit classes. Fixtures is basically setting fixed states for objects and running tests using baseline. Its objective of test fixture is to provide that there is a well-known and setup environment in the console to run test, so that it can replicate the results. Test suites combines some unit test cases and compiles it together.to run this test suite we use @RunWith and @Suite annotations before testing a code. Test runners are used for test case execution purposes. JUnit classes are used for testing and writing the JUnits and few important classes are assert, testcase and testresult. Whereas it contains assert functions, it can define fixtures for running multiple tests and it's a collection of answers from test cases.

B. Annotations

In JUnit we use annotations before writing the test cases. Annotations are the tags which define which part of the test needs to executed first and last while running the java test file. Some of the examples of annotations are as follows.

1) @Test : it defines the part of code under test is which needs to be executed to the public void method.

2) @Before: it instructs the public void method that this annotation needs to be executed before running the test methods according to the priority.

3) @After: it instructs the public void method that this annotation needs to be executed after running the test methods to ensure that if we allocate some resources in before method it need to be released after execution of the test case.

4) BeforeClass: Explicitly states that just once before all the experiments commence the process must be invoked.

5) @AfterClass: Specifically states the implemented method just once, after all of the tests have been performed.

C. Assertions

Another useful methods for writing the test cases is by using assertions method, over here only failed methods or tests will be recorded. Some of the few assert class methods are as follows.

1) void assertEquals(boolean expected,boolean actual) : Monitors the equality of two specific objects. It is overloaded.

2) void assertTrue(boolean condition): Verifies whether it's condition is true or not.

3) void assertFalse(boolean condition) : Verifies whether it's condition is false or not.

4) void assertNull(Object obj) : verifies its subject tests are zero.

5) void assertNotNull(Object obj): verifies whether its object is null or not.

D. Types of testing in JUnit

1) Ignore test: It occurs often that our application is not prepared for the test case. The test case therefore does not operate. The tag @Ignore aims to make that possible. Using @Ignore annotation that test case will not be executed and when we annotate a test class with @Ignore it can never run either of the test methods.

2) Time test: JUnit delivers a convenient timeout function. When a test case takes much longer than that of the defined millisecond number, JUnit immediately marks that it is not effective. The timeout parameter @Test annotation can be used.

3) Exception test : JUnit may track the exception and even verify if the code does or does not cause an intended error. While testing exception, we need to make sure that the exception class we provide in that automatic @test annotation variable is the same. This is when we are anticipating exception to the test method we are using Unit Testing, otherwise our JUnit test would fail. We can specify the exception name that our test will deliver by using "expected" parameter.

4) Parameterized test: Parameterized testing enables the developer to perform the same test for different values over and over again. To build a parametric test there are five measures you will follow.

- Test object annotation using @RunWith(Parameterized.xml).
- Create a static public function that returns an Object Collections (as Array) as a check data set using @Parameters.
- Develop a public construct that integrates a set of test data equal to one row.
- For each column we need to create an instance variable for test data.
- We need to develop the test cases of instance variable as source of test data.

III. Environment Setup

A. Java Installation

JUnit is a framework for Java, so the very first requirement is to have JDK installed in your machine. Set the environment variable JAVA_HOME to JDK file path. In the system variable 'Path', append the file path of bin in JDK.

B. JUnit and Hamcrest jar

The primary requirement for JUnit is Java.The latest versions of Junit requires jdk 8 or higher.

1) Download junit jar: https://github.com/junit-team/junit4/wiki/Download-and-Install

2) Hamcrest-core jar: https://search.maven.org/artifact/org.hamcrest/hamcrest-core/1.3/jar

Environmental variable set to the the location folder of JUnit. Download the jar files and add to the test class path.

C. IntelliJ IDEA Ultimate

IntelliJ IDEA can be downloaded from www.jetbrains.com/student To add the JUnit jar to a new project, create a new project from file project structure and add the dependencies, JUnit jar and hamcrest.

D. MongoDB

MongoDB is a cross-platform, document oriented database that provides, high performance, high availability, and easy scalability. MongoDB works on concept of collection and document. Mongo DB is a Nosql Database. Unlike traditional databases, it is not a relational database. MongoDB can manage unstructured data also called as a document database. MongoDB is easy to scale out and doesn't have complex joins and eases faster access of data. It is used I the fields of Big Data, Mobile ad Social Infrastructure etc.install the setup file from the website, https://www.mongodb.com/download-center/community

1) Collections: A collection is a group of documents ad is equal to RDMS table. Collections are present in single database. Collections does not enforce schema.

2) Document: Each document in the database is a record equals to a row in relational DB. Each record is a set of key-value pairs. Documents in the same schema need not have same data structure and can be of different data types.

E. Postman

Postman is an interactive and automatic tool for verifying the APIs of your project. It works on the backend, and makes sure that each API is working as intended. Postman can be downloaded from the following link: https://www.postman.com/downloads/

IV. Application Development

A Banking application that governs the customer's bank details using spring boot framework using mongo Database. Operations performed in the applications:

- Creation of Customer account - Inserting a record of customer in the database that has the following details such as name, email, contact number, id and the initial balance.
- Updating of customer details - Modify the customer details and store them in the database
- Display the records of the customers - This module returns all the details and records of the customers from the repository
- Deletion of a particular customer details - This module deletes the particular entry of the customer based on the id of the document specified.

A. Project Setup

Create a new project using spring initializer. create a package "com.bank.spring.api". Give in the details of the group name, package name, inside the "com.bank.spring.api " ,create three packages model,repository and resource.

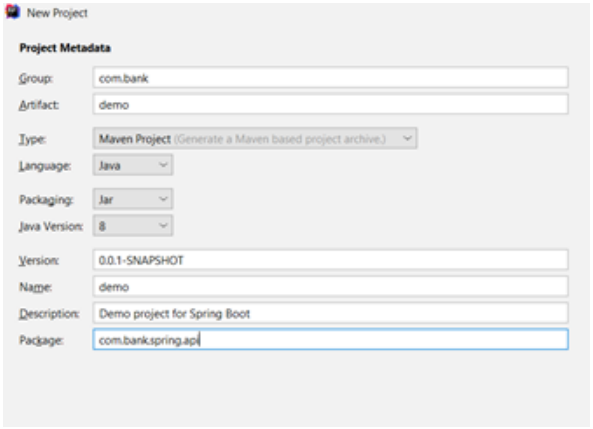

Figure 1: Create the project.

B. Database Connectivity

To connect to the database we have to create a new database using command prompt.Move to the location where mongo db is present.To start up the mongo database server ,we should type in the command "mongod" to start the server.

To open the mongo shell,we use "mongo" command to start the shell and perform the operations of creating the database and collections to store the records.

we create a database called as "bankappln".To store the records of the customers we create a collection

called as "Customers".Figure 2 shows the database created in mongo shell.

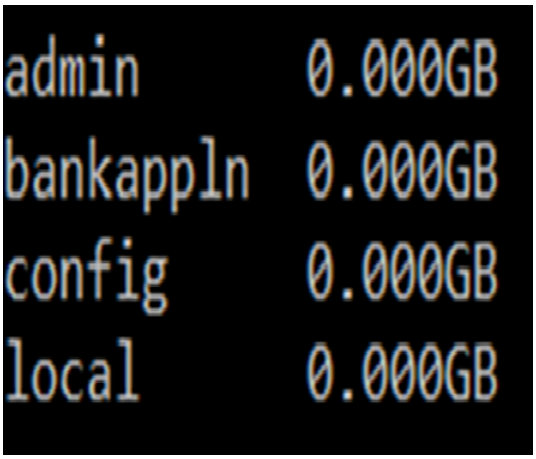

Figure 2: Database creation.

C. *Create the application properties file*

Connection attributes should be specified in the applications.properties file.To connect to the database ,configuration details such as localhost and port number of the database server has to be specified in the file in order to use the database in the application.The username and password is also mentioned to access the database.

application.properties

```
spring.data.mongodb.host=localhost
spring.data.mongodb.port=27017
spring.data.mongodb.database=bankappln
```

.

D. *Create Model*

- MongoDB stores data in collections. Spring Data MongoDB maps the Customer class into a collection called customer. If you want to change the name of the collection, you can use Spring Data MongoDB's @Document annotation on the class.
- @ToString is used to print the details of the customer.
- Getter and Setter methods are used to read the values from the variable and the setter method is used to set the values to the variable.
- @Id – used to get document from the database

Student.java

```
package com.bank.spring.api.model;

import lombok.Getter;
import lombok.Setter;
import lombok.ToString;
import org.springframework.data.
   annotation.Id;
import org.springframework.data.
   mongodb.core.mapping.Document;

@Getter
@Setter
@ToString
@Document(collection="customers")
public class Customer {


   @Id
   private int id;
   private String name;
   private String email;

   public int getId() {
      return id;
   }

   public void setId(int id) {
      this.id = id;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public String getEmail() {
      return email;
   }

   public void setEmail(String email) {
      this.email = email;
   }

   public long getContact() {
      return contact;
   }

   public void setContact(long contact)
       {
      this.contact = contact;
   }

   public int getBalance() {
      return balance;
   }

   public void setBalance(int balance) {
      this.balance = balance;
   }

   private long contact;
```

```
    private int balance;
}
```

E. Create Repository

We extend the database repository to the mongo repository.The customer repository is use to store the records of the customers.

CustomerRepository.java

```
package com.bank.spring.api.repository;

import
    com.bank.spring.api.model.Customer;
import org.springframework.data.mongodb.
                repository.MongoRepository;

public interface CustomerRepository
    extends MongoRepository<Customer,
    Integer> { }
```

F. Controller

1) @Autowired: Autowiring feature of spring framework enables you to inject the object dependency implicitly. It requires the less code because we don't need to write the code to inject the dependency explicitly.

2) @RestController: This annotation is applied to a class to mark it as a request handler.RestController takes care of mapping request data to the defined request handler method.

3) @GetMapping: annotation maps HTTP GET requests onto specific handler methods. It is a composed annotation that acts as a shortcut for @RequestMapping (method = RequestMethod.GET).They handle the HTTP GET requests matched with given URI expression.

4) @PostMapping: is specialized version of @RequestMapping annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.POST).

5) @PostMapping: annotated methods handle the HTTP POST requests matched with given URI expression.

6) @DeleteMapping: @DeleteMapping annotation maps HTTP DELETE requests onto specific handler methods. It is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.DELETE).

CustomerController.java

```
package com.bank.spring.api.resource;

import
    com.bank.spring.api.model.Customer;
import com.bank.spring.api.repository.
                CustomerRepository;
import org.springframework.
      beans.factory.annotation.Autowired;
import org.springframework.
                web.bind.annotation.*;

import java.util.List;
import java.util.Optional;

@RestController
public class CustomerController {
    @Autowired
    private CustomerRepository
        repository;
    @PostMapping("/addCustomer")
    public String
        saveCustomer(@RequestBody
        Customer customer){
        repository.save(customer);
        return "Added Customer with ID "+
            customer.getId();


    }
    @GetMapping("/findallCustomers")
    public List<Customer> getCustomers()
    {
        return repository.findAll();
    }
    @GetMapping("/findCustomer/{id}")
    public Optional<Customer>
        getCustomer(@PathVariable int id)
    {
        return repository.findById(id);
    }
    @DeleteMapping("/delete/{id}")
    public String
        deleteCustomer(@PathVariable int
        id){
        repository.deleteById(id);
        return "Customer deleted with
            Id"+ id;
    }
    @DeleteMapping("/delete/")
    public String deleteCustomer(){
        repository.deleteAll();
        return "Customers are deleted";
    }

}
```

G. Create an Application Class

The main()method in this MainApplication class uses SpringApplication.run()method of the spring boot to launch the application.Spring Initializer creates a simple class called 'MainApplication' for our Inventory application.

MainApplication.java

```
package com.bank.spring.api;

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

@SpringBootApplication
public class DemoApplication {

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

}
```

V. POSTMAN

A. *Add a Customer*

POST method /addCustomer is used to create a new customer with details such as name,email,phone,balance. A new customer record with JSON object shown in Fig.3 is added to the database. Post method is tested using postman as shown in Fig.3.

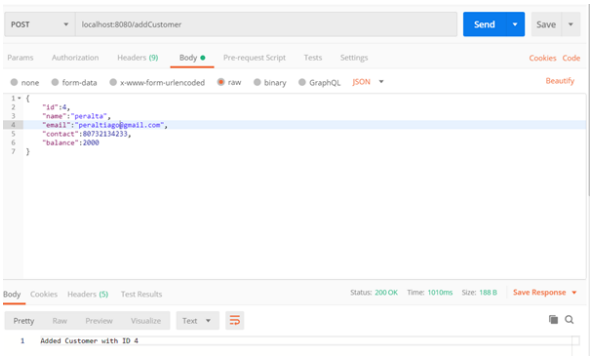

Figure 3: Add Customer

B. *Get all Customers*

GET method /findAllCustomers retrieves all the records from the database and it is tested using postman as shown in Fig. 4.

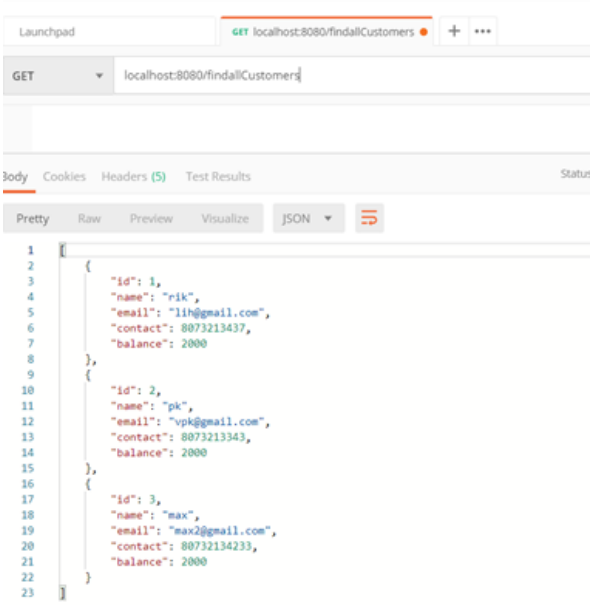

Figure 4: retrieve all the customers

C. *Get a customer based on id*

GET method /findCustomer/id retrieves the record based on id.

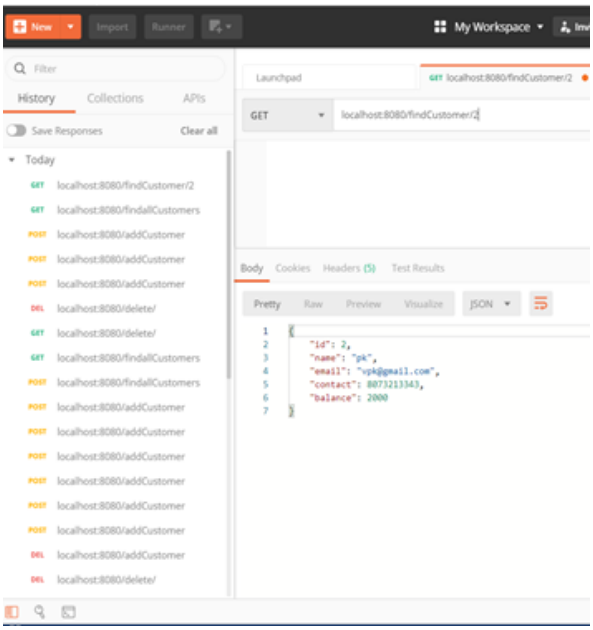

Figure 5: Customer record based on id

D. *Delete a Customer based on id*

DELETE method /delete/id deletes the record based on id. It is shown in Fig. 6.

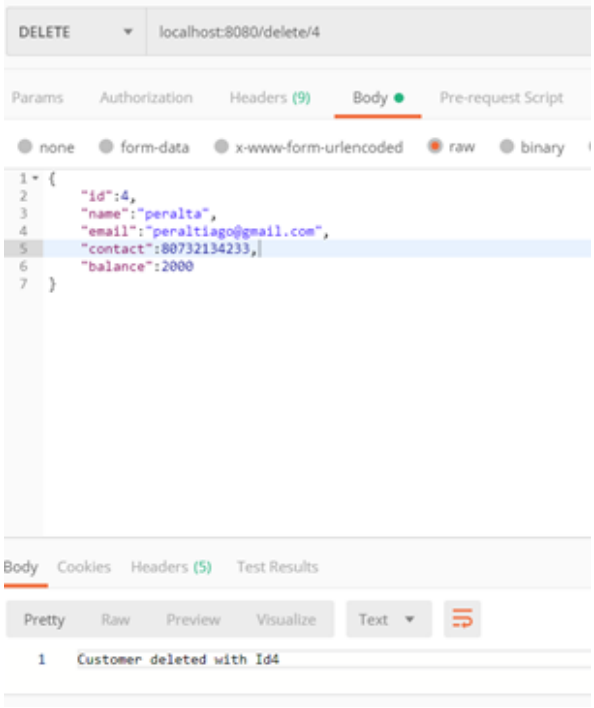

Figure 6: Deletion of a record based on id

VI. JUnit Test Cases

JSONAssert is used to compare the actual and expected result. We can see the test cases for all the API's our application.

Mockito is used to mock a method by returning a specific JSON when it is invoked.
@RunWith(SpringRunner.class) is used to launch the Spring TestContext Framework.
@WebMvcTest(value = CustomerController.class): WebMvcTest annotation is used for unit testing Spring MVC application.

A. Find all customers

In this test case, we are checking whether the API to find all the customers in the repository has been retrieving the details of all the customers. We create a mock object and store the values of the attributes. We mock the API ad we are invoking the API to check the status. If the status returns 200, then the API executes correctly and we get the details of all the customers.

```
//method to test find all customer
    records
@Test
   public void getCustomers() throws
       Exception {
      Customer c1 = new Customer();
      c1.setName("emp1");
      c1.setEmail("pk@gmail.com");
      c1.setId(1);
      c1.setBalance(3000);
     Mockito.when(
   customerRepository.findAll()).then
       Return
   (Collections.singletonList(c1));
   this.mockMvc.
   perform(get("/findallCustomers"))
      .andExpect(status().isOk())
      .andDo(print());
```

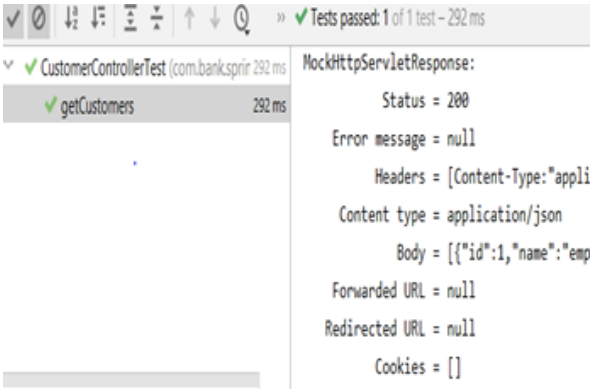

Figure 7: Test Case- findall().

B. Customer record based on id

Here the test case checks whether a single record of the customer can be retrieved based on the id. The test case gets passed and a single record corresponding to id number 5 is retrieved. The HTTP method used here is GET method.We use JSONAssert to check whether the json returned matches with the expected json.

```
//method to test Customer by by id
@Test
   public void findCustomersbyId()
       throws Exception {
      Customer c1 = new Customer();
      c1.setName("emp1");
      c1.setEmail("pk@gmail.com");
      c1.setId(5);
      c1.setBalance(3000);
      int id=5;
      Mockito.when(
      customerRepository.findById(id)).
      then Return
      (java.util.Optional.of(c1));
   this.mockMvc.
   perform(get("/findCustomer/"+id))
       .andExpect(status().isOk())
       .andDo(print());
   }
```

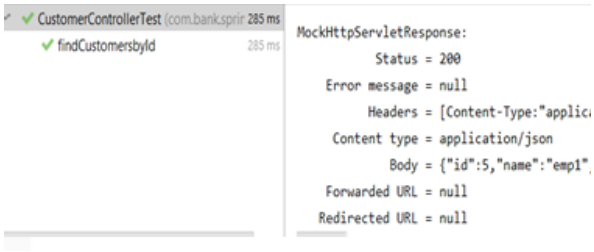

Figure 8: Test case for-customer by id

C. Delete Customers

This test case checks whether the API called will actually delete all the customer details from the repository. All the test cases are run in the mock environment making sure that the actual details doesn't get deleted. The test retrieves the status and the test case is passed and API has been checked.

```
//method to test the deletion of all
    customers

 @Test
 public void deleteCustomer() throws
     Exception
 { mockMvc.perform(
     MockMvcRequestBuilders.
       delete("/delete/") )
       .andExpect(status().isOk());
   }
```

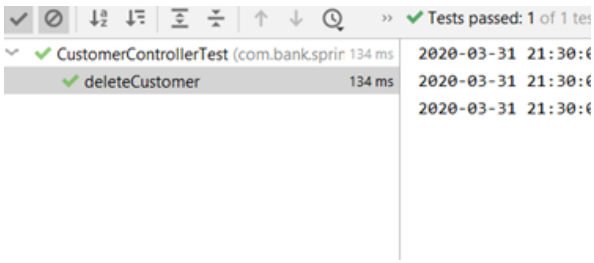

Figure 9: Test case-delete all Customers

D. Delete Customer by id

Similar to the previous test case, this API is also checked and the API is invoked with the mock object ad the validation result is retrieved.

```
//method to test the deletion of a
    customer based on id

@Test
   public void deleteCustomerbyId()
       throws Exception
   {
       mockMvc.perform(
           MockMvcRequestBuilders.
       delete("/delete/{id}",1) )
              .andExpect(status().isOk());
   }
```

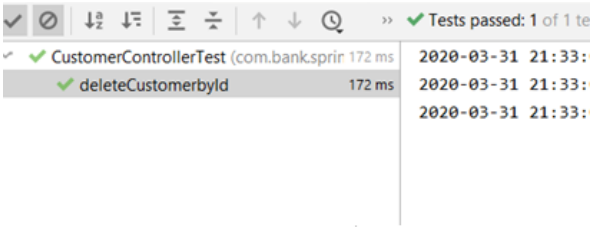

Figure 10: Test case-Delete Customer by Id

VII. Conclusion

In this paper, we gave a brief description of the JUnit Framework tool for Unit testing.we developed a Banking Inventory appication with MongoDB and wrote various test cases for diferent API and validated the results.Through this research we came to know about the JUnit Tool in detail and the tasks that can be performed with the tool.

References

[1] Guru99.com. 2020. Junit Tutorial For Beginners: Learn In 3 Days. [online] Available at: <https://www.guru99.com/junit-tutorial.html> [Accessed 3 February 2020].

[2] Tutorialspoint.com. 2020. Junit- API-Tutorialspoint. [online] Available at: https://www.tutorialspoint.com/junit/junit_api.htm [Accessed 3 March 2020].

[3] Accessing Data with MongoDB. (n.d.). Retrieved 1, 2020, from https://spring.io/guides/gs/accessing-data-mongodb/

[4] GitHub. 2020. Junit-Team/Junit4. [online] Available at: <https://github.com/junit-team/junit4/wiki> [Accessed 3 March 2020].