BRIDGE BATCH TEST

Kunal Bandooni
kunal.bandooni@wissen.com

QUESTION:

For Student: Perform the CRUD operations on the given domain. Also add functionality to provide login and register the user. You need to have User table, once the user registers, this information should go to User Table. When the user provides a Username and Password for Login, the system should provide as Valid user or Invalid user.

1. Controller - register, login
2. Pojo - extra
3. Repository. - extra
4. Application.properties

OUTPUT AFTER THE CODE

CODE:

CONTROLLER

UserRepository.java | Controller.java | User.java

```java
package com.example.demo.Controller;

import java.util.List;

@RestController
public class Controller {

    // USER METHODS HERE...

    // Use this object to use methods of UserRepository
    @Autowired
    UserRepository userObject;

    // Method to add new User
    @PostMapping(value="/register")
    public String register(@RequestBody User newUser) {
        userObject.save(newUser);
        return "Added New User!! :D";
    }

    // Method to login with registered-user only
    @GetMapping(value="/login")
    public String login(@RequestBody User user) {
        Optional<User> userOptional = userObject.findById(user.getUsername(), user.getPassword());

        if(userOptional.isPresent())
            return "Welcome User!\nGood to see you back!! =D";
        else
            return "Not a valid user! :(";
    }


    // STUDENT METHODS HERE...

    // Use this object to use methods of StudentRepository
    @Autowired
    StudentRepository studentObject;
```

```java

    // GET ALL STUDENTS
    @GetMapping(value="/viewAll")
    public Object getAllStudents() {
        List<Student> list = studentObject.getAllStudents();
        System.out.print("Student Details: ");
        for(Student p : list)
            System.out.print(p);
        return list;
    }

    // GET STUDENT BY ID
    @GetMapping(value="/view/{id}")
    public Student getStudentById(@PathVariable int id) {
        System.out.print("Student Detail with id as " + id + " : ");
        Student p = studentObject.getStudentById(id);
        return p;
    }

    // ADDING NEW STUDENT RECORDS
    @PostMapping(value="/add")
    public String addStudent(@RequestBody Student newStudent) {
        studentObject.save(newStudent);
        return "Added Student!! :D";
    }

    // UPDATING VALUES FOR EXSISTING STUDENT
    @PutMapping(value="/update")
    public String updateRecord(@RequestBody Student student) {
        Optional<Student> studentOptional = studentObject.findById(student.getId());

        if(studentOptional.isPresent()) {
            Student toUpdate = studentOptional.get();

            toUpdate.setName(student.getName());
            toUpdate.setStandard(student.getStandard());
            toUpdate.setDateOfBirth(student.getDateOfBirth());
            toUpdate.setBloodGroup(student.getBloodGroup());
```

```
91
92              studentObject.save(toUpdate);
93
94              return "Updated Record for student! =D";
95          }
96
97          return "No such student exsists found! :(";
98      }
99
100     // DELETE STUDENT WITH GIVEN ID
101     @DeleteMapping("/delete/{id}")
102     public String delete(@PathVariable int id) {
103         System.out.println("Given id : " + id);
104
105         studentObject.deleteById(id);
106
107         return "Record Deleted Successfully!! :D";
108     }
109 }
```

POJO

USER

User.java

```
3  import jakarta.persistence.Entity;
7
8  @Entity
9  @Table(name="user_details")
10 public class User {
11
12     @Id
13     @Column(name="username")
14     private String username;
15
16     @Column(name="password")
17     private String password;
18
19     // Constructors
20     public User() { }
21
22     public User(String username, String password) {
23         this.username = username;
24         this.password = password;
25     }
26
27     // Get Methods
28     public String getUsername() {
29         return this.username;
30     }
31
32     public String getPassword() {
33         return this.password;
34     }
35
36     // Set Methods
37     public void setUsername(String username) {
38         this.username = username;
39     }
40
41     public void setPassword(String password) {
42         this.password = password;
43     }
44 }
```

STUDENT

Student.java | User.java | application.properties | UserRepository.java | StudentRepository.java

```java
package com.example.demo.Student;

import jakarta.persistence.Entity;

@Entity
@Table(name="student_details")
public class Student {

    @Id
    @Column(name="id")
    private int studentId;

    @Column(name="name")
    private String name;

    @Column(name="standard")
    private int standard;

    @Column(name="date_of_birth")
    String dateOfBirth;

    @Column(name="blood_group")
    private String bloodGroup;

    // Constructors
    public Student() {  }

    public Student(int id, String name, int standard, String dob, String bloodGroup) {
        this.studentId = id;
        this.name = name;
        this.standard = standard;
        this.dateOfBirth = dob;
        this.bloodGroup = bloodGroup;
    }

   // Get methods
   public int getId() {
       return this.studentId;
   }

   public String getName() {
       return this.name;
   }

   public int getStandard() {
       return this.standard;
   }

   public String getDateOfBirth() {
       return this.dateOfBirth;
   }

   public String getBloodGroup() {
       return this.bloodGroup;
   }

   // Set Methods
   public void setId(int id) {
       this.studentId = id;
   }

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

   public void setStandard(int standard) {
       this.standard = standard;
   }

    public void setDateOfBirth(String dob) {
        this.dateOfBirth = dob;
    }

    public void setBloodGroup(String bloodGroup) {
        this.bloodGroup = bloodGroup;
    }
}
```


```java
    public void setDateOfBirth(String dob) {
        this.dateOfBirth = dob;
    }

    public void setBloodGroup(String bloodGroup) {
        this.bloodGroup = bloodGroup;
    }
}
```

REPOSITORIES

USER REPOSITORY

UserRepository.java

```java
package com.example.demo.Repository;

import java.util.Optional;

public interface UserRepository extends JpaRepository<User, String>{
    @Query(value="select * from user_details where username = ?1 and password = ?2", nativeQuery=true)
    public Optional<User> findById(String username, String Password);
}
```

STUDENT REPOSITORY

StudentRepository.java

```java
package com.example.demo.Repository;

import java.util.List;

public interface StudentRepository extends JpaRepository<Student, Integer>{

    // GET ALL STUDENTS
    @Query(value="select * from student_details", nativeQuery=true)
    public List<Student> getAllStudents();

    // GET STUDENTS BY ID
    @Query(value="select * from student_details where id = ?1", nativeQuery=true)
    public Student getStudentById(int id);
}
```

APPLICATION.PROPERTIES

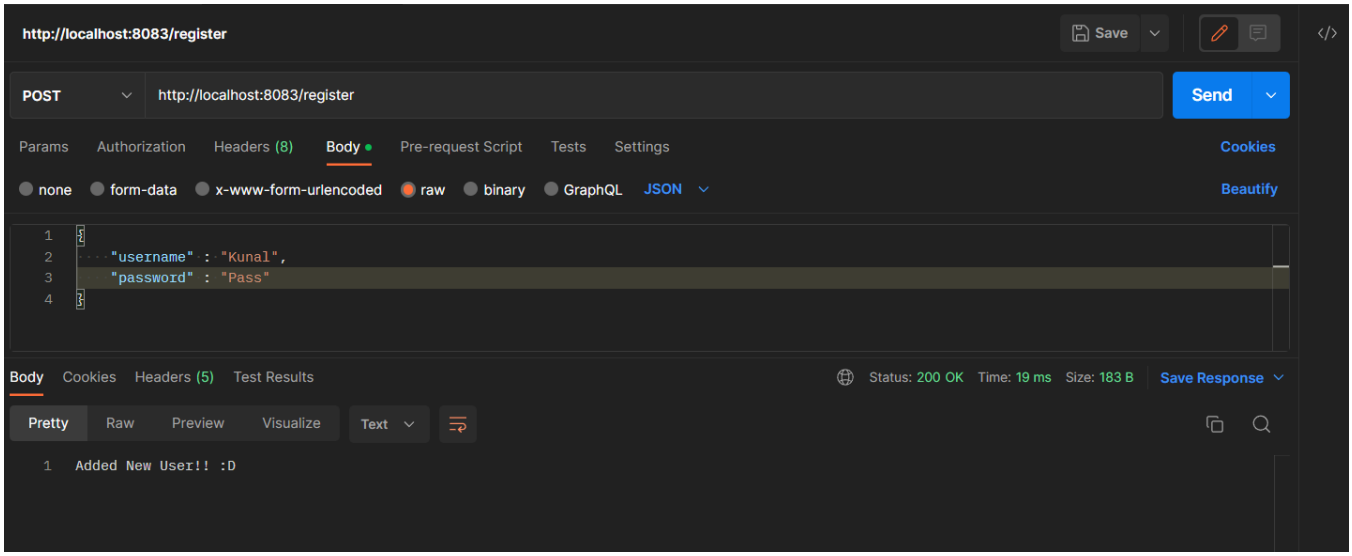
application.properties

```
server.port=8083

spring.datasource.url=jdbc:mysql://127.0.0.1:3306/db

spring.datasource.username=root
spring.datasource.password=tiger

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto=update
```

OUTPUT:

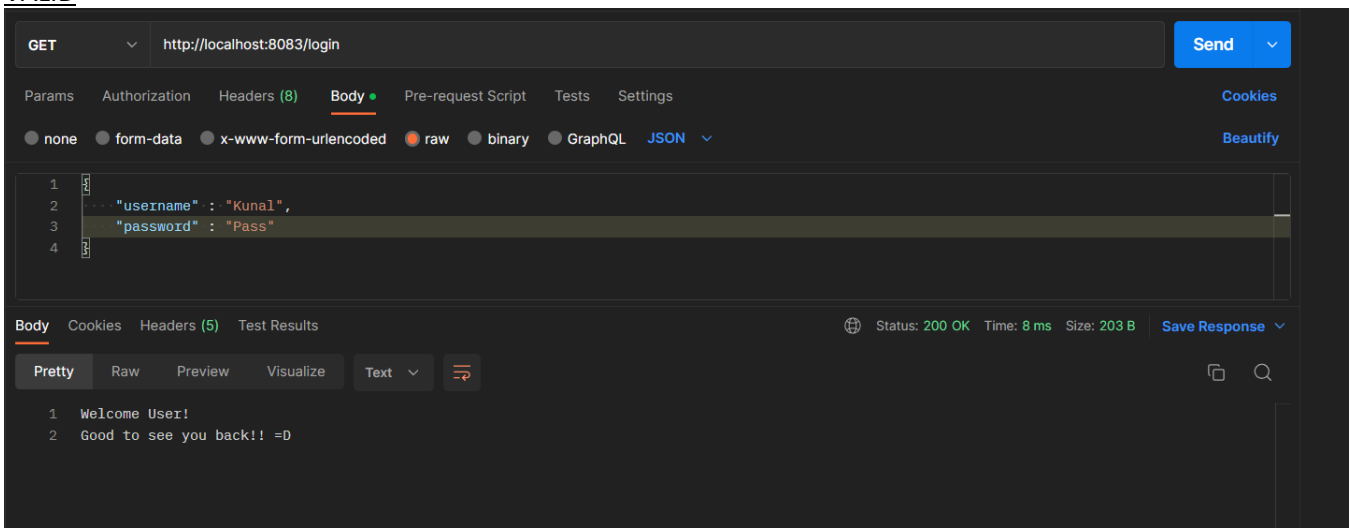
USER

REGISTER

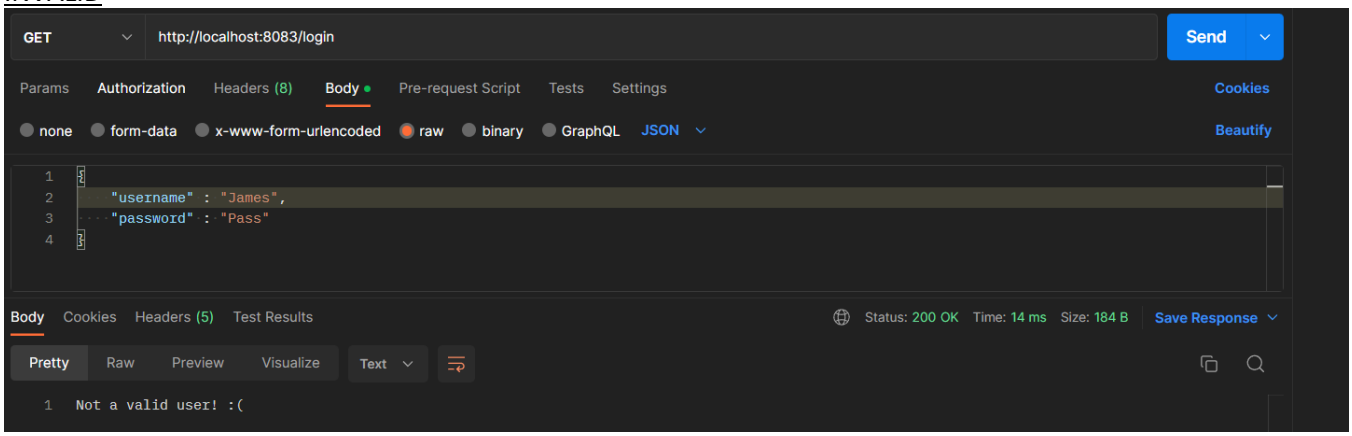


LOGIN

VALID

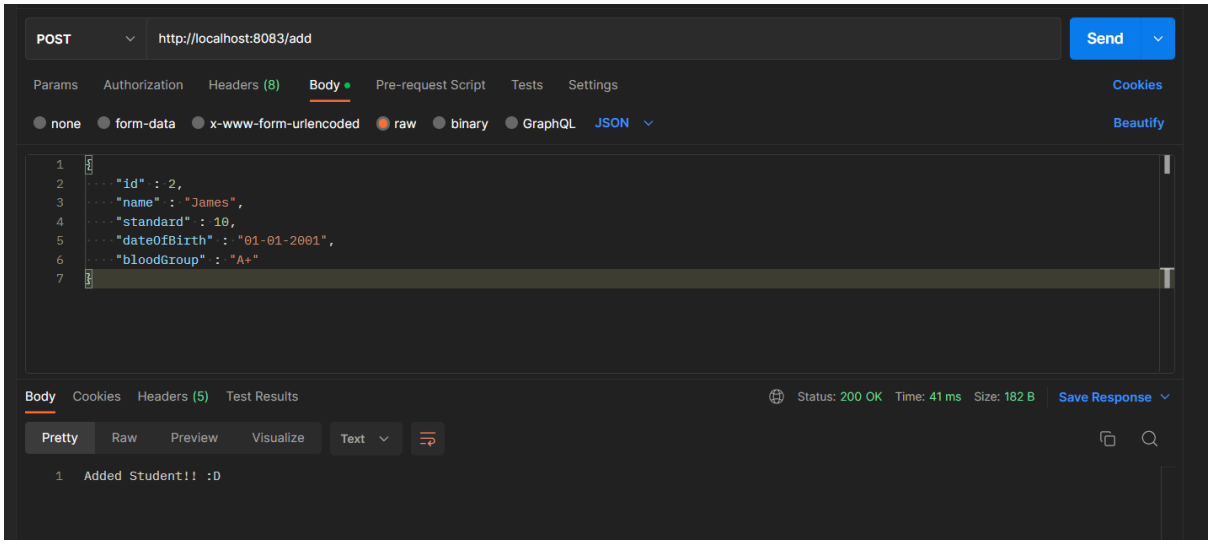


INVALID

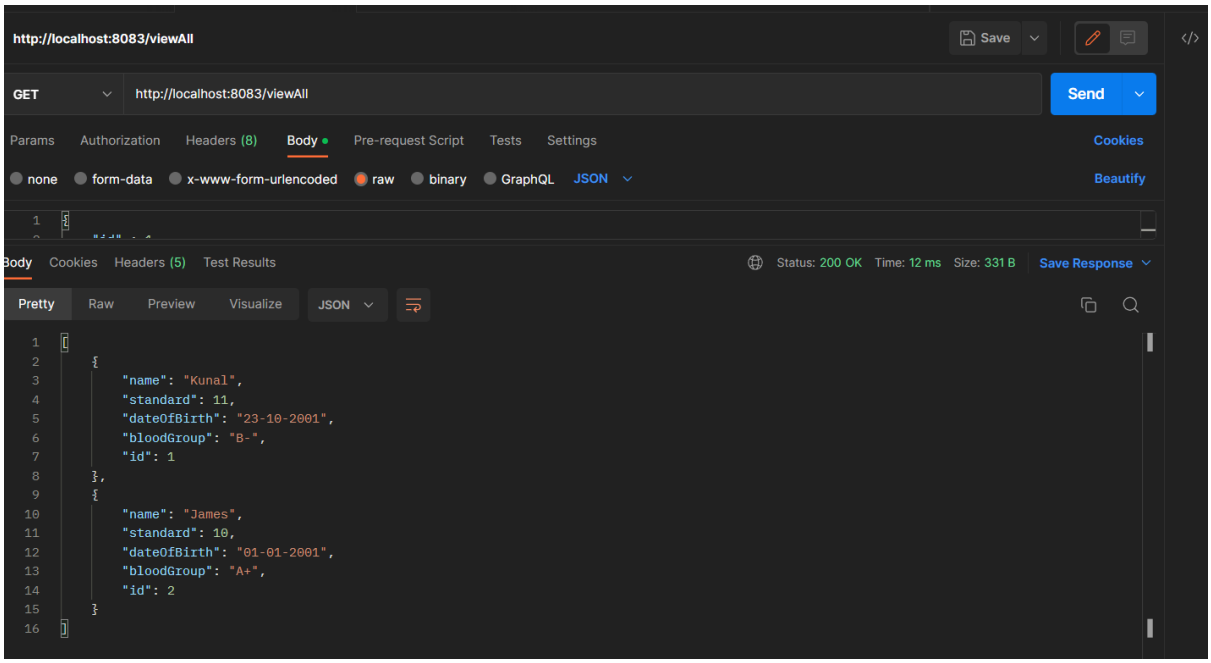


STUDENT

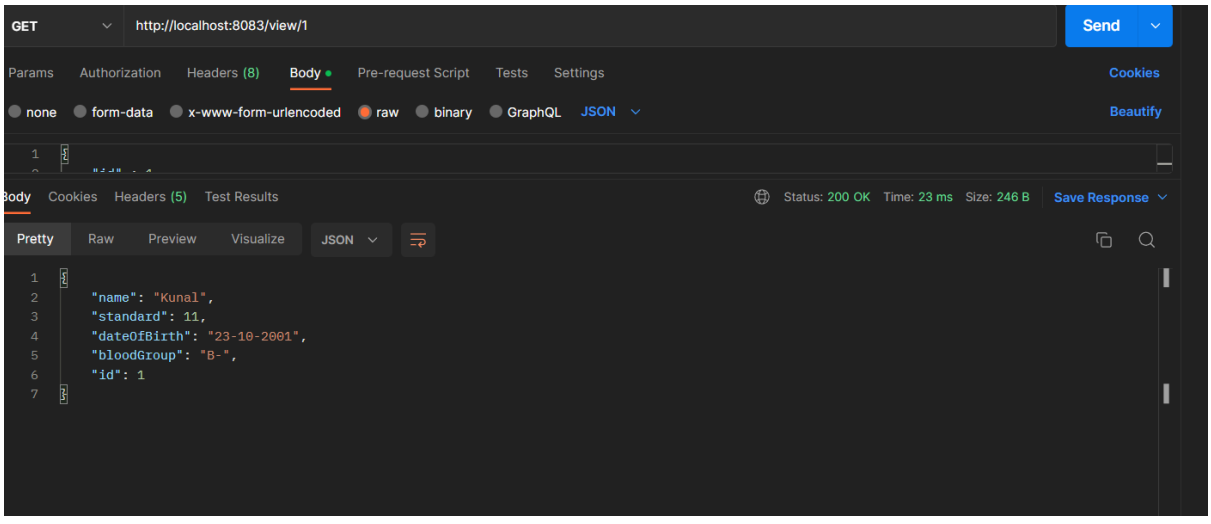
POST



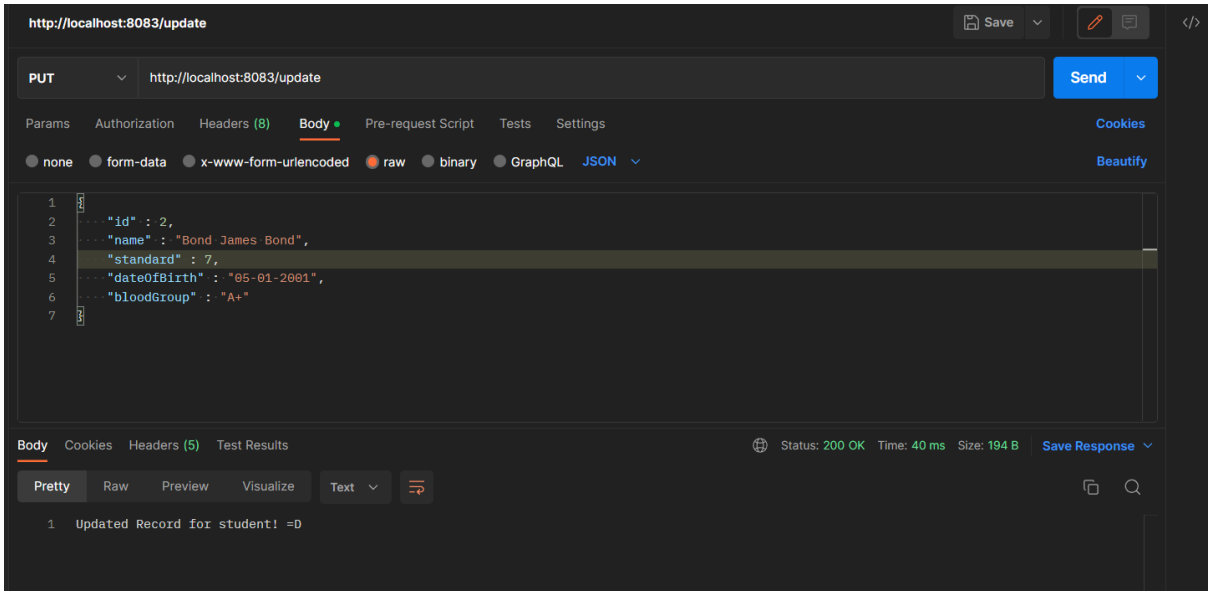
GET



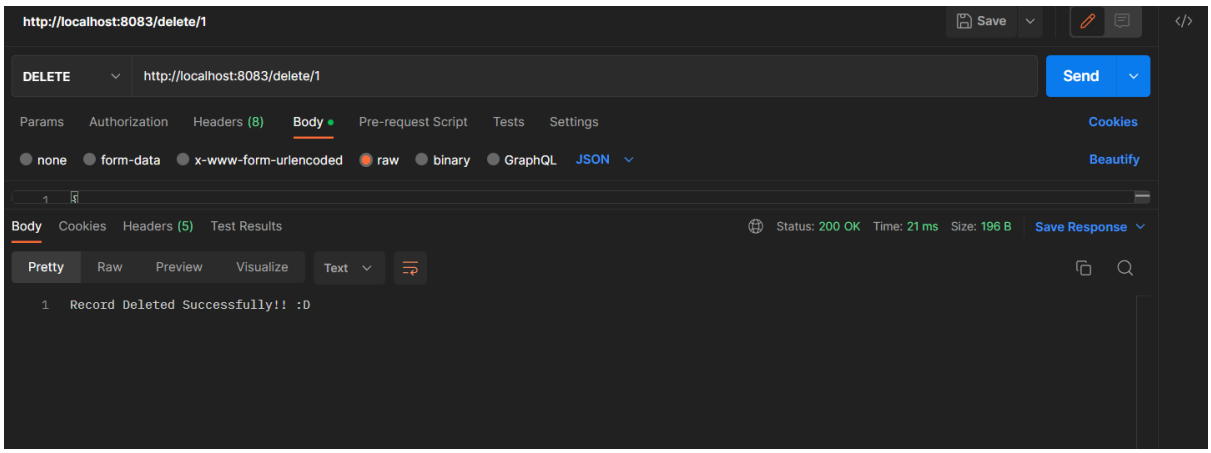
GET BY ID



PUT



DELETE



OUTPUT AFTER DELETE: