Backend Questions Solutions

1) **Create a basic RESTful web service for a book management system that allows the API user to add new books, delete existing books, update existing books, and list registered books. The book schema can have only two properties name and ISBN to keep things simple.**

⇨ The Restful web Service for book management system is in the books folder of the directory. Follow the given Readme file for running the application.

2) **Generate an API documentation of the RESTful web service you created in question #1 using any API documentation template of your choice. Hint: Swagger, RestDoc, etc.**

⇨ Documentation using swagger has been done for the above web service. You can navigate to http://localhost:5000/doc for the required documentation or follow the Read me for more details

3) **How would you explain semantic versioning to an eight-year-old.**

⇨ Imagine your teacher gave you multiple vacation homework's for a same subject. You'd have a hard time to keep track of each and every home work at a single glance. So, you ask your teacher to generalize the home works in an understandable way. Thus, teacher would most probably give you a note or explain it to you.

Sematic Versioning works in a similar way. It provides a generalized way to understand and keep track of the multiple dependencies and packages of your project. It has 3 phases: Major, Minor and Patch.
To understand it better, a v2.1.3 semantic version would mean the "2nd major version" with "one minor update" and with "three updates for fixing bugs or minor changes"

4) **Below is a simple code for stack operation. However, there is a major issue with how it has been implemented. Can you find it out and fix it?**

⇨ The given Stack class has a crucial fault i.e., during the ensure capacity operation on every push to the stack. Acc to the defined values, the static final value of elements array of stack is 16 i.e., DEFAULT_INITAL_CAPACITY = 16. For a new stack, on every push, size of the stack will be increased by 1 whereas on every pop, size will be

decreased by 1. But, when the size of the stack reaches the default threshold of elements length i.e. 16 on our case, ensure Capacity () will try to copy the elements of our stack and add a new length to our array of 2 * size +1 resulting in several null elements and faulty data.

⇨

Test Example:

```
public static void main(String[] args) {
        Stack stack = new Stack();
        stack.push(1);
        stack.push(2);
        stack.push(3);
        stack.push(4);
        stack.push(5);
        stack.push(6);
        stack.push(7);
        stack.push(8);
        stack.push(9);
        stack.push(10);
        stack.push(11);
        stack.push(12);
        stack.push(13);
        stack.push(14);
        stack.push(15);
        stack.push(16);
        stack.push(17);
        System.out.println(Arrays.toString(stack.elements));


        System.out.println(stack.size);
    }
```

the resulted stack elements will be -> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null]

To prevent this, we might have to either throw an Stack Exception to remind the user that the capacity has been exceeded or ensure that the stack size is increased simultaneously on every push which can be done by:

```
private void ensureCapacity() {
    if (elements.length == size)
        elements = Arrays.copyOf(elements, size + 1);
}
```

So, the stack of the previous input would be -> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17] and the stack will be able to continue as one wishes.

5) Why does SELECT * FROM table WHERE field = null not match records with field column having null set in SQL?

⇨ NULL indicates the absence of a field value or an unknown value that is set when a column data is not given. It'd never make sense to compare two values of NULL or NULL= NULL. Example -> User Table has two rows Ram and Hari.
Ram's age is NULL
Hari's age is NULL
SQL won't be able to assume that Ram's age is equal to Hari's age on the basis of NULL value since both of the ages of Ram and Hari are unknown, and might be any random age. Ram's age might be 30 while Hari's age might be 50 or any random age. So, SQL doesn't match records for such case.

6) Below is a pseudocode of a poorly designed function. How can you refactor the code below to make it more optimized and robust?

⇨ The given pseudocode can be greatly simplified using Switch Statement as it contains multiple nested if else.
Using JavaScript, it can be done in the following way: (Simple Example)

```
function test(operation) {
  let error = null;
  const operations = operation;
  switch (operations) {
    case 1:
      error = "operation1failed";
      break;
    case 2:
      error = "operation2failed";
      break;
    case 3:
      error = "operation3failed";
      break;
    case 4:
      error = "operation4failed";
      break;
  }
  return error;
}
```

console.log("Output is:", test(1))

7) How do you suggest we handle race conditions? Code an example of race condition along with the solution you devise for the fix.

⇨ A race condition occurs when two or more threads can access shared data and they try to change it at the same time. Because the thread scheduling algorithm can swap between threads at any time, you don't know the order in which the threads will attempt to access the shared data. Therefore, the result of the change in data is dependent on the thread scheduling algorithm, i.e., both threads are "racing" to access/change the data.

Java Example:

Example:

```
public class Sum {

   private long count = 0;

   public void add(long value){
      this.count = this.count + value;
   }
}
```

Imagine if two threads, A and B, are executing the add method on the same instance of the Sum class. There is no way to know when the operating system switches between the two threads. The code in the add () method is not executed as a single atomic instruction by the Java virtual machine. Rather it is executed as a set of smaller instructions.

To prevent this, we can use:

```
public class Sum {

   private int num1 = 0;
   private int num2 = 0;

   public void add (int num1, int num2){
      synchronized(this){
         this.num1 += num1;
         this.num2 += num2;
      }
   }
}
```

Herer, the add () method adds values to two different sum member variables. To prevent race conditions the summing is executed inside a Java synchronized block. With this implementation only a single thread can ever execute the summing at the same time.

8) Consider two tables named customer and order in any RDBMS system with the following records.

What would be the output of the following SQL query and why?

```
SELECT
o.customernumber,
ordernumber,
o.status,
customerName FROM orders o
RIGHT JOIN customers c
ON o.customernumber = c.customernumber
9
```

The output would be the following table:

This is because the initial select operation from orders table selects the column customer number, order number status and customer name from orders table, but customer name column is not present on orders table for which right join has been used on customers table for same columns customer number on both tables, such that all the rows of the right side of the join i.e., customers table and matching rows of left table i.e., orders table is shown. In short, establishes a Foreign Key relationship between the two tables i.e., Orders has the FK of Customers Table and the output will include the combined table of customer number, order number, status and its respective customer name. Tina's other fields are left null because Tina is the only customer to be not associated with the Order's table

customernumber	ordernumber	status	customername
1	2	shipped	Ram
1	1	shipped	Ram
2	3	processing	Hari
3	4	open	Rita
			Tina

9) What is the difference between encryption and hashing? Explain it in a way you are teaching to an eight-year-old kid. Program a very simple code in any language of your preference that shows hash operation with the use of Salt. Hint: Salt is any random string of characters added to the original message for enhancing the

security. E.g. message= “suvrat”, salt= “a1sd”, new message= “suvrata1sd” which will now generate a different hash protecting from potential reverse lookups.

⇨ Hashing refers to permanent data conversion into message digest while encryption works in two ways, which can encode and decode the data.

Example of password hashing and password check using bcrypt and salt in Javascript

```
const bcrypt = require('bcrypt')

async function run(password) {
   const salt = await bcrypt.genSalt(10);
   const hashed = await bcrypt.hash(password, salt);
   return hashed;
}

async function check(reqPassword, dbPassword) {
   const validPassword = await bcrypt.compare(reqPassword, dbPassword)
   return validPassword
}
```

10) Given below is a sample pseudocode that computes the perimeter of a rectangle taking length and breadth as input. Create necessary unit tests (pseudocode will do or you can choose any programming language of your choice) to have maximum coverage so that each line inside the function body is executed at least once.

⇨ Using JavaScript:

```
// function to calculate the perimeter of a rectangle
function perimeter(length, breadth) {
   if (!length || !breadth) {
      throw "Undefined";
   }
   if (!Number.isInteger(length)) {
      throw "Not a number";
   }
   if (!Number.isInteger(breadth)) {
      throw "Not a number";

   }
   if (length < 0 || breadth < 0) {
      throw ("dimension cannot be negative");
   }
   return 2 * (length + breadth);
```

```
}


// test udnefined
describe('test undefined', () => {
    it('should have both length and breadth as valid parameters', () => {
        expect(perimeter(1).toThrow("Undefined"))
    })
})

// test length number
describe('test length', () => {
    it('should be an integer datatype', () => {
        expect(perimeter('A', 2).toThrow("Not a number"))
    })
})

// test breadth number
describe('test breadth', () => {
    it('should be an integer datatype', () => {
        expect(perimeter(2, 'B').toThrow("Not a number"))
    })
})

// test if less than zero
describe('test length and breadth', () => {
    it('should not be less than zero', () => {
        expect(perimeter(-1, 2).toThrow("dimension cannot be negative"))
    })
})
```