SOFTWARE ENGINEERING

Name: Yashvi Lathiya

Student ID: 202201220

LAB: Program Inspection and Debugging

Q1) Armstrong

```
//Armstrong Number
class Armstrong{
        public static void main(String args[]){
                int num = Integer.parseInt(args[0]);
                int n = num; //use to check at last time
                int check=0,remainder;
                while(num > 0){
                        remainder = num / 10;
                        check = check + (int)Math.pow(remainder,3);
                        num = num % 10;
                }
```

```
            if(check == n)
                    System.out.println(n+" is an Armstrong Number");
            else
                    System.out.println(n+" is not a Armstrong Number");
    }
```

Input: 153

Output: 153 is an armstrong Number.

Program Inspection:

1. Errors:

- Error 1: In the while loop, you are calculating the remainder incorrectly. It should be `remainder = num % 10;` instead of `remainder = num / 10;`
- Error 2: Missing closing bracket for the class Armstrong '}'

2. Effective Category:

- Category B (Data Declaration Errors) and Category E (Control Flow Errors).

3. Unidentified Error Types:

- The program inspection did not identify potential issues like integer overflow.

4. Applicability:

- Program inspection helps catch syntax and some semantic errors, but it doesn't verify the correctness of the logic in the program.

Debugging:

1. Errors:

- Error 1: In the while loop, change the calculation of `remainder` to `remainder = num %10;` to correctly calculate the remainder.

2. Breakpoints Needed:

- At least one breakpoints are needed to address these errors.

Corrected Code:

```
class Armstrong {
   public static void main(String args[]) {
      int num = Integer.parseInt(args[0]);
      int n = num; // original number to check at last
      int check = 0, remainder;
      while (num > 0) {
         remainder = num % 10;
         check = check + (int)Math.pow(remainder, 3);
         num = num / 10;
      }
      if (check == n)
         System.out.println(n + " is an Armstrong Number");
      else
         System.out.println(n + " is not an Armstrong Number");
   }
}
```

Q2) GCD_LCM

```
//program to calculate the GCD and LCM of two given numbers
import java.util.Scanner;
public class GCD_LCM
```

```
{
   static int gcd(int x, int y)
   {
      int r=0, a, b;
      a = (x > y) ? y : x; // a is greater number
      b = (x < y) ? x : y; // b is smaller number
      r = b;
      while(a % b == 0) //Error replace it with while(a % b != 0)
      {
         r = a % b;
         a = b;
         b = r;
      }
      return r;
   }
   static int lcm(int x, int y)
   {
      int a;
      a = (x > y) ? x : y; // a is greater number
      while(true)
      {
         if(a % x != 0 && a % y != 0)
            return a;
         ++a;
      }
  }
```

```
    public static void main(String args[])
    {
        Scanner input = new Scanner(System.in);
        System.out.println("Enter the two numbers: ");
        int x = input.nextInt();
        int y = input.nextInt();
        System.out.println("The GCD of two numbers is: " + gcd(x, y));
        System.out.println("The LCM of two numbers is: " + lcm(x, y));
        input.close();
    }
}
```

```
Input: 4 5
Output: The GCD of two numbers is 1
        The GCD of two numbers is 20
```

Program Inspection:

1. Errors:

- Error 1: In the `gcd` method, the while loop condition is incorrect. It should be `while(a %b!= 0)` instead of `while(a % b == 0)`.

2. Effective Category:

- Category B (Semantic Errors).

3. Unidentified Error Types:

- The program inspection did not identify potential issues like integer overflow or incorrect logic in the `lcm` method.

4. Applicability:

- Program inspection helps catch syntax and some semantic errors but does not verify the correctness of the mathematical logic in the `gcd` and `lcm` methods.

Debugging:

1. Errors:

- Error 1: In the `gcd` method, change the while loop condition to `while(a % b != 0)` to find the greatest common divisor correctly.

2. Breakpoints Needed:

- At least one breakpoints are needed to address these errors. import java.util.Scanner;

Corrected Code:

```
import java.util.Scanner;


public class GCD_LCM {

  static int gcd(int x, int y) {

    int r = 0, a, b;

    a = (x > y) ? y : x; // a is the smaller number

    b = (x < y) ? x : y; // b is the greater number

    r = b;

    while (a % b != 0) {  // Corrected loop condition

      r = a % b;

      a = b;

      b = r;

    }

    return r;

  }
```

```
    static int lcm(int x, int y) {
        int a;
        a = (x > y) ? x : y;  // a is the greater number
        while (true) {
            if (a % x == 0 && a % y == 0) {  // Condition corrected to find LCM
                return a;
            }
++a;
        }
    }
    public static void main(String args[]) {
        Scanner input = new Scanner(System.in);
        System.out.println("Enter the two numbers: ");
        int x = input.nextInt();
        int y = input.nextInt();
        System.out.println("The GCD of the two numbers is: " + gcd(x, y));
        System.out.println("The LCM of the two numbers is: " + lcm(x, y));
        input.close();
    }
}
```

Q3) Knapsack

```
//Knapsack
public class Knapsack {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);   // number of items
        int W = Integer.parseInt(args[1]);   // maximum weight of knapsack
        int[] profit = new int[N+1];
        int[] weight = new int[N+1];
        // generate random instance, items 1..N
        for (int n = 1; n <= N; n++) {
            profit[n] = (int) (Math.random() * 1000);
            weight[n] = (int) (Math.random() * W);
        }
        // opt[n][w] = max profit of packing items 1..n with weight limit w
     // sol[n][w] = does opt solution to pack items 1..n with weight limit w
include item n?
        int[][] opt = new int[N+1][W+1];
        boolean[][] sol = new boolean[N+1][W+1];
        for (int n = 1; n <= N; n++) {
            for (int w = 1; w <= W; w++) {
                // don't take item n
                int option1 = opt[n++][w];
                // take item n
                int option2 = Integer.MIN_VALUE;
                if (weight[n] > w) option2 = profit[n-2] + opt[n-1][w-weight[n]];

                // select better of two options
                opt[n][w] = Math.max(option1, option2);
                sol[n][w] = (option2 > option1);
            }
        }
        // determine which items to take
        boolean[] take = new boolean[N+1];
        for (int n = N, w = W; n > 0; n--) {
            if (sol[n][w]) { take[n] = true;  w = w - weight[n]; }
            else           { take[n] = false;                    }
        }
        // print results
        System.out.println("item" + "\t" + "profit" + "\t" + "weight" + "\t" + "take");
        for (int n = 1; n <= N; n++) {
            System.out.println(n + "\t" + profit[n] + "\t" + weight[n] + "\t" + take[n]);
        }
    }
}
```

Input: 6, 2000

Output:

```
Item   Profit  Weight  Take
1      336     784     false
2      674     1583    false
3      763     392     true
4      544     1136    true
5      14      1258    false
6      738     306     true
```

<u>Program Inspection:</u>

1. Errors:

- Error 1: In the for loop header, there is a post-increment operator (`n++`) used instead of just incrementing `n` by one (`n++` should be `n+1`).
- Error 2: The indexing of arrays should be from `0` to `N`, but it starts from `1` to `N`. In Java, arrays are zero-indexed.
- Error 3: In the option2 calculation, the code is using the item's profit and weight at index `n 2`, which is likely incorrect. It should be using `n-1`.
- Error 4: The code calculates the maximum profit value using `opt[N][W]`, but this should be `opt[N][W]` for the actual result.

2. Effective Category:

- Category B (Data Declaration Errors) and Category C (Computational Errors).

3. Unidentified Error Types:

- The program inspection did not identify potential logical errors, such as the correctness of the knapsack algorithm's implementation.

4. Applicability:

- Program inspection helps catch syntax and some semantic errors but does not verify the correctness of the algorithm's implementation.

Debugging:

1. Errors:

- Error 1: Change `n++` to `n+1` in the for loop header.
- Error 2: Adjust array indexing to start from `0`.
- Error 3: Use `n-1` instead of `n-2` for item profit and weight in option2 calculation.
- Error 4: Change `opt[N][W]` to `opt[N][W]` for the actual result.

2. Breakpoints Needed:

- At least four breakpoints are needed to address these errors.

Corrected Code:

```
public class Knapsack {

   public static void main(String[] args) {
```

```
int N = Integer.parseInt(args[0]);
int W = Integer.parseInt(args[1]);
int[] profit = new int[N];
int[] weight = new int[N];
for (int n = 0; n < N; n++) {
   profit[n] = (int) (Math.random() * 1000);
   weight[n] = (int) (Math.random() * W);
}
int[][] opt = new int[N + 1][W + 1];
boolean[][] sol = new boolean[N + 1][W + 1];
for (int n = 1; n <= N; n++) {
   for (int w = 1; w <= W; w++) {
      int option1 = opt[n - 1][w];
      int option2 = Integer.MIN_VALUE;
      if (weight[n - 1] <= w) {
         option2 = profit[n - 1] + opt[n - 1][w - weight[n - 1]];
      }
      opt[n][w] = Math.max(option1, option2);
      sol[n][w] = (option2 > option1);
   }
}
boolean[] take = new boolean[N];
for (int n = N, w = W; n > 0; n--) {
   if (sol[n][w]) {
      take[n - 1] = true;
      w = w - weight[n - 1];
```

```
            } else {
                take[n - 1] = false;
            }
        }
        System.out.println("item" + "\t" + "profit" + "\t" + "weight" + "\t" + "take");
        for (int n = 0; n < N; n++) {
            System.out.println((n + 1) + "\t" + profit[n] + "\t" + weight[n] + "\t" +
take[n]);
        }
    }
}
```

Q4) MagicNumber

```
// Program to check if number is Magic number in JAVA
import java.util.*;
public class MagicNumberCheck
{
    public static void main(String args[])
    {
        Scanner ob=new Scanner(System.in);
        System.out.println("Enter the number to be checked.");
        int n=ob.nextInt();
        int sum=0,num=n;
        while(num>9)
        {
            sum=num;int s=0;
            while(sum==0)
            {
                s=s*(sum/10);
                sum=sum%10
            }
            num=s;
        }
        if(num==1)
        {
            System.out.println(n+" is a Magic Number.");
        }
        else
        {
            System.out.println(n+" is not a Magic Number.");
        }
    }
}
```

Input: Enter the number to be checked 119

Output 119 is a Magic Number.

Input: Enter the number to be checked 199

Output 199 is not a Magic Number.

<u>Program Inspection:</u>

1. Errors:

- Error 1: In the inner while loop, the loop condition is `while (sum == 0)`, which means the loop will only execute if `sum` is initially 0, causing an infinite loop. The loop condition should likely be changed.
- Error 2: Missing semicolons at the end of lines with `sum=sum%10` and `s=s*(sum/10)`.

2. Effective Category:

- Category A (Data Reference Errors Errors) and Category C (Computation Errors).

3. Unidentified Error Types:

- Program inspection identified lexical and computation errors. However, it might not identify potential logical errors, such as the loop condition and the computation within the loops.

4. Applicability:

- Program inspection is useful for catching syntax and computation errors. To identify and fix logical errors in the code, additional testing and debugging are required.

Debugging:

1. Errors:

- Error 1: Change the inner while loop's condition from `while (sum == 0)` to `while (sum > 0)` to avoid an infinite loop.
- Error 2: Add semicolons at the end of lines with `sum=sum%10` and `s=s*(sum/10)` to fix the syntax errors.

2. Breakpoints Needed:

- At least two breakpoints are needed to address these errors.

Corrected Codes:

```
import java.util.*;


public class MagicNumberCheck {
    public static void main(String args[]) {
        Scanner ob = new Scanner(System.in);
        System.out.println("Enter the number to be checked.");
        int n = ob.nextInt();
        int sum = 0, num = n;
        while (num > 9) {
            sum = num;
            int s = 0;
            while (sum > 0) {
                s = s + (sum % 10);
                sum = sum / 10;
            }
            num = s;
        }
        if (num == 1) {
            System.out.println(n + " is a Magic Number.");
        } else {
            System.out.println(n + " is not a Magic Number.");
        }
    }}
```

Q5) Merge Sort

```
// This program implements the merge sort algorithm for
// arrays of integers.
import java.util.*;
public class MergeSort {
    public static void main(String[] args) {
```

```
    int[] list = {14, 32, 67, 76, 23, 41, 58, 85};
    System.out.println("before: " + Arrays.toString(list));
    mergeSort(list);
    System.out.println("after:  " + Arrays.toString(list));
}
// Places the elements of the given array into sorted order
// using the merge sort algorithm.
// post: array is in sorted (nondecreasing) order
public static void mergeSort(int[] array) {
    if (array.length > 1) {
        // split array into two halves
        int[] left = leftHalf(array+1);
        int[] right = rightHalf(array-1);
        // recursively sort the two halves
        mergeSort(left);
        mergeSort(right);
        // merge the sorted halves into a sorted whole
        merge(array, left++, right--);
    }
}
// Returns the first half of the given array.
public static int[] leftHalf(int[] array) {
    int size1 = array.length / 2;
    int[] left = new int[size1];
    for (int i = 0; i < size1; i++) {
        left[i] = array[i];
    }
    return left;
}
// Returns the second half of the given array.
public static int[] rightHalf(int[] array) {
    int size1 = array.length / 2;
    int size2 = array.length - size1;
    int[] right = new int[size2];
    for (int i = 0; i < size2; i++) {
        right[i] = array[i + size1];
    }
    return right;
}
// Merges the given left and right arrays into the given
// result array.  Second, working version.
// pre : result is empty; left/right are sorted
// post: result contains result of merging sorted lists;
public static void merge(int[] result,
                         int[] left, int[] right) {
    int i1 = 0;   // index into left array
    int i2 = 0;   // index into right array

    for (int i = 0; i < result.length; i++) {
        if (i2 >= right.length || (i1 < left.length &&
                left[i1] <= right[i2])) {
            result[i] = left[i1];    // take from left
```

```
                i1++;
            } else {
                result[i] = right[i2];   // take from right
                i2++;
            }
        }
    }
}
```

Input: before 14 32 67 76 23 41 58 85

after 14 23 32 41 58 67 76 85

Program Inspection:

1. Errors:

- Error 1: In the `mergeSort` method, the function calls `leftHalf(array+1)` and `rightHalf(array 1)`. Instead, it should call `leftHalf(array)` and `rightHalf(array)`.
- Error 2: The `merge` method is missing, which is called in the `mergeSort` method.

2. Effective Category:

- Category C (Computation Errors).

3. Unidentified Error Types:

- Program inspection identified computation errors, but it might not catch potential logical errors in the sorting algorithm.

4. Applicability:

- Program inspection is useful for catching syntax and computation errors. For more comprehensive testing and fixing logical errors, additional testing techniques (e.g., debugging and test cases) are required.

Debugging:

1. Errors:

- Error 1: In the `mergeSort` method, replace `int[] left = leftHalf(array+1);` with `int[] left = leftHalf(array);` and `int[] right = rightHalf(array-1);` with `int[] right = rightHalf(array);`.
- Error 2: The `merge` method is called in the code but not provided. A correct implementation of the `merge` method is needed for the code to work.

2. Breakpoints Needed:

- At least two breakpoints are needed to address these errors.

Corrected Code:

```
import java.util.*;

public class MergeSort {

  public static void main(String[] args) {

    int[] list = {14, 32, 67, 76, 23, 41, 58, 85};

    System.out.println("before: " + Arrays.toString(list));

    mergeSort(list);

    System.out.println("after: " + Arrays.toString(list));

  }

  public static void mergeSort(int[] array) {

    if (array.length > 1) {

      int[] left = leftHalf(array);

      int[] right = rightHalf(array);

      mergeSort(left);

      mergeSort(right);

      merge(array, left, right);
    }
}
public static int[] leftHalf(int[] array) {
    int size1 = array.length / 2;
    int[] left = new int[size1];
    for (int i = 0; i < size1; i++) {
        left[i] = array[i];
    }
    return left;
}
public static int[] rightHalf(int[] array) {
    int size1 = array.length / 2;
    int size2 = array.length - size1;
    int[] right = new int[size2];
    for (int i = 0; i < size2; i++) {
        right[i] = array[i + size1];
    }
    return right;
}
public static void merge(int[] result, int[] left, int[] right) {
    int i1 = 0;
    int i2 = 0;
    for (int i = 0; i < result.length; i++) {
        if (i2 >= right.length || (i1 < left.length && left[i1] <= right[i2])) {
            result[i] = left[i1];
            i1++;
            } else {
                result[i] = right[i2];
                i2++;
            }
        }
    }
}
```

Q6) Multiply Matrix

```
//Java program to multiply two matrices
import java.util.Scanner;

class MatrixMultiplication
{
  public static void main(String args[])
  {
    int m, n, p, q, sum = 0, c, d, k;
    Scanner in = new Scanner(System.in);
    System.out.println("Enter the number of rows and columns of first matrix");
    m = in.nextInt();
    n = in.nextInt();
    int first[][] = new int[m][n];
    System.out.println("Enter the elements of first matrix");
    for ( c = 0 ; c < m ; c++ )
      for ( d = 0 ; d < n ; d++ )
```

```
        first[c][d] = in.nextInt();
    System.out.println("Enter the number of rows and columns of second
matrix");
    p = in.nextInt();
    q = in.nextInt();
    if ( n != p )
      System.out.println("Matrices with entered orders can't be multiplied with
each other.");
    else
    {
      int second[][] = new int[p][q];
      int multiply[][] = new int[m][q];
      System.out.println("Enter the elements of second matrix");
      for ( c = 0 ; c < p ; c++ )
        for ( d = 0 ; d < q ; d++ )
          second[c][d] = in.nextInt();
      for ( c = 0 ; c < m ; c++ )
      {
        for ( d = 0 ; d < q ; d++ )
        {
          for ( k = 0 ; k < p ; k++ )
          {
            sum = sum + first[c-1][c-k]*second[k-1][k-d];
          }
          multiply[c][d] = sum;
          sum = 0;
        }
      }
      System.out.println("Product of entered matrices:-");

      for ( c = 0 ; c < m ; c++ )
      {
        for ( d = 0 ; d < q ; d++ )
          System.out.print(multiply[c][d]+"\t");
        System.out.print("\n");
      }
    }
  }
}
```

Input: Enter the number of rows and columns of first matrix

```
     2 2
     Enter the elements of first matrix
     1 2 3 4
     Enter the number of rows and columns of first matrix
     2 2
     Enter the elements of first matrix
     1 0 1 0
```

Output: Product of entered matrices:

```
      3 0
      7 0
```

<u>**Program Inspection:**</u>

1. Errors:

- Error 1: In the nested loop that calculates the product of matrices, change `sum = sum + first[c-1][c-k]*second[k-1][k-d];` to `sum = sum + first[c][k] * second[k][d];`. The indices should start from 0.
- Error 2: In the nested loops, the variables `c`, `d`, and `k` should be properly initialized within the for loops.

2. Effective Category:

- Category C (Computation Errors) Category B (Data declaration Errors).

3. Unidentified Error Types:

- Program inspection identified computation errors, but it might not catch potential logical errors in the matrix multiplication algorithm.

4. Applicability:

- Program inspection is useful for catching syntax and computation errors, but for more comprehensive testing and fixing logical errors, additional testing techniques (e.g., debugging and test cases) are required.

Debugging:

1. Errors:

- Error 1: In the nested loop that calculates the product of matrices, change `sum = sum + first[c-1][c-k]*second[k-1][k-d];` to `sum = sum + first[c][k] * second[k][d];`.
- Error 2: Initialize variables `c`, `d`, and `k` properly within the for loops.

2. Breakpoints Needed:

- At least two breakpoints are needed to address these errors.

Corrected Code:

```
import java.util.Scanner;

class MatrixMultiplication {

   public static void main(String args[]) {
        int m, n, p, q, sum = 0, c, d, k;

        Scanner in = new Scanner(System.in);

        System.out.println("Enter the number of rows and columns of the first matrix");

        m = in.nextInt();

        n = in.nextInt();

        int first[][] = new int[m][n];


        System.out.println("Enter the elements of the first matrix");

        for (c = 0; c < m; c++) {

            for (d = 0; d < n; d++) {

                first[c][d] = in.nextInt();

            }

        }

        System.out.println("Enter the number of rows and columns of the second matrix");

        p = in.nextInt();

        q = in.nextInt();

        if (n != p) {

            System.out.println("Matrices with entered orders can't be multiplied with each other.");

        } else {

            int second[][] = new int[p][q];

            int multiply[][] = new int[m][q];


            System.out.println("Enter the elements of the second matrix");

            for (c = 0; c < p; c++) {
```

```
            for (d = 0; d < q; d++) {
                second[c][d] = in.nextInt();
            }
        }
        for (c = 0; c < m; c++) {
            for (d = 0; d < q; d++) {
                sum = 0;
                for (k = 0; k < p; k++) {
                    sum = sum + first[c][k] * second[k][d];
                }
                multiply[c][d] = sum;
            }
        }
        System.out.println("Product of entered matrices:-");
        for (c = 0; c < m; c++) {
            for (d = 0; d < q; d++) {
                System.out.print(multiply[c][d] + "\t");
            }
            System.out.println();
        }
    }
  }
}
```

Q7) Quadratic Probing

```
/**

 *   Java Program to implement Quadratic Probing Hash Table

 **/



import java.util.Scanner;




/** Class QuadraticProbingHashTable **/

class QuadraticProbingHashTable

{

    private int currentSize, maxSize;

    private String[] keys;

    private String[] vals;
```

```
/** Constructor **/

public QuadraticProbingHashTable(int capacity)

{

    currentSize = 0;

    maxSize = capacity;

    keys = new String[maxSize];

    vals = new String[maxSize];

}




/** Function to clear hash table **/

public void makeEmpty()

{

    currentSize = 0;
    keys = new String[maxSize];

    vals = new String[maxSize];

}




/** Function to get size of hash table **/

public int getSize()

{

    return currentSize;

}




/** Function to check if hash table is full **/

public boolean isFull()

{
    return currentSize == maxSize;

}



/** Function to check if hash table is empty **/

public boolean isEmpty()

{

    return getSize() == 0;

}



/** Fucntion to check if hash table contains a key **/

public boolean contains(String key)

{

    return get(key) !=  null;
```

```
}



/** Functiont to get hash code of a given key **/

private int hash(String key)

{

    return key.hashCode() % maxSize;

}



/** Function to insert key-value pair **/
public void insert(String key, String val)
{
    int tmp = hash(key);
    int i = tmp, h = 1;
    do
    {

        if (keys[i] == null)
        {
            keys[i] = key;
            vals[i] = val;
            currentSize++;
            return;
        }
        if (keys[i].equals(key))
        {
            vals[i] = val;
            return;
        }
        i + = (i + h / h--) % maxSize;
    } while (i != tmp);
}
/** Function to get value for a given key **/
public String get(String key)
{
    int i = hash(key), h = 1;
    while (keys[i] != null)
    {
        if (keys[i].equals(key))
            return vals[i];
        i = (i + h * h++) % maxSize;
        System.out.println("i "+ i);
    }
    return null;
}
```

```
/** Function to remove key and its value **/
public void remove(String key)
{
    if (!contains(key))
        return;
    /** find position key and delete **/
    int i = hash(key), h = 1;
    while (!key.equals(keys[i]))
        i = (i + h * h++) % maxSize;
    keys[i] = vals[i] = null;
    /** rehash all keys **/
    for (i = (i + h * h++) % maxSize; keys[i] != null; i = (i + h * h++) % maxSize)
    {
        String tmp1 = keys[i], tmp2 = vals[i];
        keys[i] = vals[i] = null;
        currentSize--;
        insert(tmp1, tmp2);
    }
    currentSize--;
}
/** Function to print HashTable **/
public void printHashTable()
{
    System.out.println("\nHash Table: ");
    for (int i = 0; i < maxSize; i++)
        if (keys[i] != null)
```

```
                System.out.println(keys[i] +" "+ vals[i]);
            System.out.println();
        }
    }
    /** Class QuadraticProbingHashTableTest **/
    public class QuadraticProbingHashTableTest
    {
        public static void main(String[] args)
        {
            Scanner scan = new Scanner(System.in);
            System.out.println("Hash Table Test\n\n");
            System.out.println("Enter size");
            /** maxSizeake object of QuadraticProbingHashTable **/
            QuadraticProbingHashTable qpht = new
QuadraticProbingHashTable(scan.nextInt() );



            char ch;
            /**  Perform QuadraticProbingHashTable operations  **/
            do
            {
                System.out.println("\nHash Table Operations\n");
                System.out.println("1. insert ");
                System.out.println("2. remove");
                System.out.println("3. get");
                System.out.println("4. clear");
System.out.println("5. size");
int choice = scan.nextInt();
switch (choice)
{
case 1 :
   System.out.println("Enter key and value");
   qpht.insert(scan.next(), scan.next() );
   break;
case 2 :
   System.out.println("Enter key");
   qpht.remove( scan.next() );
   break;
case 3 :
   System.out.println("Enter key");
   System.out.println("Value = "+ qpht.get( scan.next() ));
   break;
case 4 :
   qpht.makeEmpty();
   System.out.println("Hash Table Cleared\n");
   Break:
case 5 :
   System.out.println("Size = "+ qpht.getSize() );


   break;
default :
   System.out.println("Wrong Entry \n ");
```

```
                    break;
                }
                /** Display hash table **/
                qpht.printHashTable();
                System.out.println("\nDo you want to continue (Type y or n) \n");
                ch = scan.next().charAt(0);
            } while (ch == 'Y'|| ch == 'y');
        }
    }
```

Input:

```
Hash table test

Enter size: 5
Hash Table Operations
1. Insert
2. Remove
3. Get
4. Clear
5. Size

1

Enter key and value
c computer
```

```
d desktop
h harddrive
```

```
Output:
Hash Table:
c computer
d desktop
h harddrive
```

Program Inspection:

1. Errors:

- Error 1: In the `insert` method, there is a logical error in the line `i + = (i + h / h--) %maxSize;`. The correct statement should be `i = (i + h * h++) % maxSize;` to implement quadratic probing.
- Error 2: In the `get` method, the loop condition should consider the case when the table is full. Change `while (keys[i] != null)` to `for (int j = 0; j < maxSize; j++)` to ensure that the loop will eventually stop.
- Error 3: In the `remove` method, there is an issue with the loop that rehashes keys. Change `for (i = (i + h * h++) % maxSize; keys[i] != null; i = (i + h * h++) % maxSize)` to `for (int j = 0; j < maxSize; j++)` to ensure it rehashes properly.

2. Effective Category:

- Category C (Computation Errors) and Category E (Control-Flow Errors).

3. Unidentified Error Types:

- Program inspection has identified computation errors, but it might not catch potential logical errors in the hash table operations.

4. Applicability:

- Program inspection is useful for catching syntax and computation errors, but for more comprehensive testing and fixing logical errors, additional testing techniques (e.g., debugging and test cases) are required.

Debugging:

1. Errors:

- Error 1: In the `insert` method, change `i + = (i + h / h--) % maxSize;` to `i = (i + h * h++) % maxSize;`.
- Error 2: In the `get` method, change the loop condition to `for (int j = 0; j < maxSize; j++)` to ensure it doesn't run indefinitely.
- Error 3: In the `remove` method, change the rehash loop to `for (int j = 0; j < maxSize; j++)` to ensure it rehashes properly.

2. Breakpoints Needed:

- At least three breakpoints are needed to address these errors.

Corrected Code:

```
import java.util.Scanner;


class QuadraticProbingHashTable {
  private int currentSize, maxSize;
  private String[] keys;
  private String[] vals;


  public QuadraticProbingHashTable(int capacity) {
    currentSize = 0;
    maxSize = capacity;
    keys = new String[maxSize];
    vals = new String[maxSize];
  }
}

public void makeEmpty() {
   currentSize = 0;
   keys = new String[maxSize];
   vals = new String[maxSize];
}

public int getSize() {
   return currentSize;
}

public boolean isFull() {
   return currentSize == maxSize;
}

public boolean isEmpty() {
   return getSize() == 0;
}

public boolean contains(String key) {
   return get(key) != null;
}

private int hash(String key) {
   return key.hashCode() % maxSize;
}
```

```
public void insert(String key, String val) {
    int tmp = hash(key);
    int i = tmp, h = 1;
    do {
        if (keys[i] == null) {
            keys[i] = key;
            vals[i] = val;
            currentSize++;
            return;
        }
        if (keys[i].equals(key)) {
            vals[i] = val;
            return;
        }
        i = (i + h * h++) % maxSize;
    } while (i != tmp);
}

public String get(String key) {
    int i = hash(key), h = 1;
    for (int j = 0; j < maxSize; j++) {
        if (keys[i] != null) {
            if (keys[i].equals(key)) {
                return vals[i];
            }
            i = (i + h * h++) % maxSize;
        } else {
            return null;
        }
    }
    return null;
}

public void remove(String key) {
    if (!contains(key)) return;
    int i = hash(key), h = 1;
    while (!key.equals(keys[i])) i = (i + h * h++) % maxSize;
    keys[i] = vals[i] = null;
    for (int j = 0; j < maxSize; j++) {
        if (keys[i] != null) {
            String tmp1 = keys[i], tmp2 = vals[i];
            keys[i] = vals[i] = null;
            currentSize--;
            insert(tmp1, tmp2);
        } else {
            currentSize--;
            break;
        }
    }
}

public void printHashTable() {
        System.out.println("\nHash Table: ");
        for (int i = 0; i < maxSize; i++) {
            if (keys[i] != null) {
                System.out.println(keys[i] + " " + vals[i]);
            }
        }
        System.out.println();
    }
}

public class QuadraticProbingHashTableTest {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Hash Table Test\n\n");
        System.out.println("Enter size");
        QuadraticProbingHashTable qpht = new
QuadraticProbingHashTable(scan.nextInt());
        char ch;

        do {
            System.out.println("\nHash Table Operations");
            System.out.println("1. insert ");
            System.out.println("2. remove");
            System.out.println("3. get");
            System.out.println("4. clear");
            System.out.println("5. size");
            int choice = scan.nextInt();
```

```
switch (choice) {
    case 1:
        System.out.println("Enter key and value");
        qpht.insert(scan.next(), scan.next());
        break;
    case 2:
        System.out.println("Enter key");
        qpht.remove(scan.next());
        break;
    case 3:
        System.out.println("Enter key");
        System.out.println("Value = " + qpht.get(scan.next()));
        break;
    case 4:
        qpht.makeEmpty();
        System.out.println("Hash Table Cleared\n");
        break;
    case 5:
        System.out.println("Size = " + qpht.getSize());
        break;
    default:
        System.out.println("Wrong Entry \n");
        break;
}
qpht.printHashTable();
System.out.println("\nDo you want to continue (Type y or n) \n");
```

```
            ch = scan.next().charAt(0);
        } while (ch == 'Y' || ch == 'y');
    }
}
```

Q8) Sorting Array

```
// sorting the array in ascending order
import java.util.Scanner;
public class Ascending _Order
{
    public static void main(String[] args)
    {
        int n, temp;
        Scanner s = new Scanner(System.in);
        System.out.print("Enter no. of elements you want in array:");
        n = s.nextInt();
        int a[] = new int[n];
        System.out.println("Enter all the elements:");
        for (int i = 0; i < n; i++)
        {
            a[i] = s.nextInt();
        }
        for (int i = 0; i >= n; i++);
        {
            for (int j = i + 1; j < n; j++)
            {
```

```
            if (a[i] <= a[j])
            {
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
        }
    }
    System.out.print("Ascending Order:");
    for (int i = 0; i < n - 1; i++)
    {
        System.out.print(a[i] + ",");
    }
    System.out.print(a[n - 1]);
  }
}
```

```
Input: Enter no. of elements you want in array: 5
       Enter all elements:
       1 12 2 9 7
       1 2 7 9 12
```

Program Inspection:

1. **Errors**:
   - **Error 1**: In the first for loop, the condition for(int i=0; i>=n; i++); is incorrect. It uses i>= n, which will never be true, and as a result, the loop's body will not execute.

- **Error 2**: The loop condition in the second for loop should be i<n, not i>=n. The same issue is present in this loop as well.

2. **Effective Category**:
    - Category C (Computation Errors)
    - Category D (Comparison Errors).
3. **Unidentified Error Types**:
    - The program inspection process identified computation errors, but it might not catch potential logical errors in the sorting logic.
4. **Applicability**:
    - Program inspection is useful for catching syntax and computation errors, but for more comprehensive testing and fixing logical errors, additional testing techniques (e.g., debugging and test cases) are required.

Debugging:

1. **Errors**:
    - **Error 1**: In the first for loop, change for(int i=0; i>=n; i++); to for(int i=0; i<n; i++) to correctly iterate through the array.
    - **Error 2**: In the second for loop, change for (int i = 0; i >= n; i++) to for (int i = 0; i < n; i++) to correctly iterate through the array.
2. **Breakpoints Needed**:
    - Two breakpoints are needed to address these errors.
3. **Steps Taken to Fix Errors**:
    - In the first for loop, change the condition to i < n to iterate through the array.
    - In the second for loop, change the condition to i < n to iterate through the array.

Corrected Code:

```java
import java.util.Scanner;
public class Ascending_Order {
    public static void main(String[] args) {
        int n, temp;
        Scanner s = new Scanner(System.in);
        System.out.print("Enter no. of elements you want in the array: ");
        n = s.nextInt();
        int a[] = new int[n];
        System.out.println("Enter all the elements: ");

        for (int i = 0; i < n; i++) {
            a[i] = s.nextInt();
        }
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                if (a[i] > a[j]) {
                    temp = a[i];
                    a[i] = a[j];
                    a[j] = temp;
                }
            }
        }
        System.out.print("Ascending Order: ");
        for (int i = 0; i < n - 1; i++) {
            System.out.print(a[i] + ", ");
        }
        System.out.print(a[n - 1]);
    }
}
```

Q9) Stack Implementation

```
//Stack implementation in java
import java.util.Arrays;

public class StackMethods {
    private int top;
    int size;
    int[] stack ;

    public StackMethods(int arraySize){
        size=arraySize;
        stack= new int[size];
        top=-1;
    }

    public void push(int value){
        if(top==size-1){
            System.out.println("Stack is full, can't push a value");
        }
        else{

            top--;
            stack[top]=value;
        }
    }

    public void pop(){
        if(!isEmpty())
            top++;
        else{
            System.out.println("Can't pop...stack is empty");
        }
    }

    public boolean isEmpty(){
        return top==-1;
    }

    public void display(){

        for(int i=0;i>top;i++){
            System.out.print(stack[i]+ " ");
        }
        System.out.println();
    }
}
public class StackReviseDemo {

    public static void main(String[] args) {
        StackMethods newStack = new StackMethods(5);
        newStack.push(10);
        newStack.push(1);
        newStack.push(50);
        newStack.push(20);
        newStack.push(90);

        newStack.display();
        newStack.pop();
        newStack.pop();
        newStack.pop();
        newStack.pop();
        newStack.display();
    }
}
```

output:

```
    10
      1
      50
      20
      90
      10
```

Program Inspection:

1. Errors:

- Error 1: In the `push` method, the top should be incremented before pushing the value. However, it's currently being decremented, leading to incorrect behavior.
- Error 2: In the `display` method, the loop condition is using `>` instead of `<`, which will not display the stack correctly.

2. Effective Category:

- Category C (Computation Errors) and Category D (Comparison Errors).

3. Unidentified Error Types:

- While this inspection identified computation errors, it might not catch potential logical errors in the behavior of the stack (e.g., handling overflow or underflow).

4. Applicability:

- Program inspection is useful for catching syntax and computation errors, but for more comprehensive testing, additional testing techniques are required.

Debugging:

1. Errors:

- Error 1: In the `push` method, change `top--` to `top++` to increment the top before pushing the value.
- Error 2: In the `display` method, change the loop condition from `for(int i=0; i > top; i++)` to `for (int i = 0; i < top; i++)` to correctly display the stack.

2. Breakpoints Needed:

- Two breakpoints are needed to address these errors.

3. Steps Taken to Fix Errors:

- In the `push` method, change `top--` to `top++`.- In the `display` method, change the loop condition to use `<` instead of `>`.

Corrected Code:

```
import java.util.Arrays;


public class StackMethods {
   private int top;
   int size;
   int[] stack;


   public StackMethods(int arraySize) {
      size = arraySize;
      stack = new int[size];
      top = -1;
   }


   public void push(int value) {
      if (top == size - 1) {
         System.out.println("Stack is full, can't push a value");
      } else {
         top++;
         stack[top] = value;
      }
   }


   public void pop() {
      if (!isEmpty()) {
         top--;
```

```
        } else {
            System.out.println("Can't pop...stack is empty");
        }
    }

    public boolean isEmpty() {
        return top == -1;
    }

    public void display() {
        for (int i = 0; i <= top; i++) {
            System.out.print(stack[i] + " ");
        }
        System.out.println();
    }
}

public class StackReviseDemo {
    public static void main(String[] args) {
        StackMethods newStack = new StackMethods(5);
        newStack.push(10);
        newStack.push(1);
        newStack.push(50);
        newStack.push(20);
        newStack.push(90);
        newStack.display();
        newStack.pop();
        newStack.pop();
        newStack.pop();
        newStack.pop();
        newStack.display();
    }
}
```

Q10) Tower of Hanoi

```
//Tower of Hanoi
public class MainClass {
  public static void main(String[] args) {
    int nDisks = 3;
    doTowers(nDisks, 'A', 'B', 'C');
  }
  public static void doTowers(int topN, char from,
  char inter, char to) {
    if (topN == 1){
      System.out.println("Disk 1 from "
      + from + " to " + to);
    }else {
      doTowers(topN - 1, from, to, inter);
      System.out.println("Disk "
      + topN + " from " + from + " to " + to);
      doTowers(topN ++, inter--, from+1, to+1)
      }
   }
}
```

```
Output: Disk 1 from A to C
      Disk 2 from A to B
      Disk 1 from C to B
      Disk 3 from A to C
      Disk 1 from B to A
      Disk 2 from B to C
      Disk 1 from A to C
```

Program Inspection:

1. Errors:

- Error 1: Incorrect usage of post-increment and post-decrement operators (`topN++`, `inter--`).
- Error 2: Incorrect order of parameters in the recursive call to `doTowers`.

2. Effective Category:

- Category C (Computation Errors).

3. Unidentified Error Types:

- More complex logic errors or algorithmic issues may not be identified by this program inspection checklist.

4. Applicability:

- The program inspection technique is worth applying to catch syntax and logical errors, but it should be complemented with other testing methods.

Debugging:

1. Errors:

- Error 1: Incorrect usage of post-increment and post-decrement operators (`topN++`, `inter- `).
- Error 2: Incorrect order of parameters in the recursive call to `doTowers`.

2. Breakpoints Needed:

- At least two breakpoints are needed to address these errors.

3. Steps Taken to Fix Errors:

- Replace `topN++` with `topN--`.- Swap the order of `from` and `inter` in the recursive call.

Corrected Code:

```
public class MainClass {
public static void main(String[] args) {
   int nDisks = 3;
   doTowers(nDisks, 'A', 'B', 'C');
}

public static void doTowers(int topN, char from, char inter, char to) {
   if (topN == 1) {
      System.out.println("Disk 1 from " + from + " to " + to);
   } else {
      doTowers(topN - 1, from, to, inter);
      System.out.println("Disk " + topN + " from " + from + " to " + to);
      doTowers(topN - 1, inter, from, to);
   }
}
}
```

2000 LOC:

Code1:

```
const pool = require('../config/db');
const axios = require('axios');
const Bottleneck = require('bottleneck');

// Create a limiter with a specified rate limit (requests per second)
const limiter = new Bottleneck({
  maxConcurrent: 5,
  minTime: 1000,   // Minimum time to wait between each request in
milliseconds
});

module.exports.nearby_hospitals = async function (req, res) {
  try {
    let hospitals = await pool.query(`SELECT
                    name,
                    email,
                    location,
                    (icu + iicu + operation_theatre + general_ward) AS
total_beds,
                    (nurse + intern + ot_technician) AS total_staff
                  FROM
                    hospital
```

```
                        ORDER BY
                          name;`);
    hospitals = hospitals.rows;
    let nearbyHospitals = [];
    let distanceTime = [];

    if (req.query) {
      const lat = req.query.lat;
      const lng = req.query.lng;

      // find the nearby hospitals
      for (const hospital of hospitals) {
        await limiter.schedule(async () => {
          const apiKey = process.env.apiKey;

          const startCoordinates = lat + ',' + lng;
          const endCoordinates = hospital.location.replace(/\s+/g, ''); // Remove
spaces
          const traffic = true;

          const tomtomApiEndpoint =
'https://api.tomtom.com/routing/1/calculateRoute/';
          const url =
${tomtomApiEndpoint}${startCoordinates}:${endCoordinates}/json?key=${apiK
ey}&traffic=${traffic};

          try {
```

```
          const response = await axios.get(url);
          const data = response.data;
          const route = data.routes && data.routes[0];

          if (route) {
            const distance = route.summary.lengthInMeters / 1000; // in km
            const travelTime = route.summary.travelTimeInSeconds / 3600; // in
hrs

            if (distance < 20) {
              nearbyHospitals.push(hospital);
              distanceTime[hospital.email] = {
                distance: distance,
                travelTime: travelTime
              };
            }
          } else {
            console.error('No route found.');
          }
        } catch (error) {
          console.error('Error fetching TomTom data:', error.message);
        }
      });
    }
  }

  return res.render('ambulance-nearby-hospitals', {
      title: 'Nearby Hospitals',
      hospitals: nearbyHospitals,
      distanceTime: distanceTime
    });
  } catch (error) {
    console.error('Error: ', error);
    return res.status(500).json({ error: 'Server Error!' });
  }
};
```

Debugging:

1. Check pool.query()

Make sure that your pool from '../config/db' is properly configured and the query is correct.

Debugging the SQL Query

- Ensure the SQL table names (hospital) and the column names are correct.
- Add error handling to verify if the pool.query() is returning the expected result.

2. Ensure the Bottleneck Limiter is Working Correctly

The Bottleneck limiter is designed to prevent sending more than 5 concurrent requests. Make sure the limiter is properly applied around the axios.get() request.

Potential Issue:

- If the request does not respect the rate limit, you could hit the API request limit, resulting in failed API calls.

3. Check the URL Formatting

Ensure that the url being used for the API request is properly constructed. In JavaScript, template literals need backticks (`), but you are missing them in the url assignment.

4. Check for Environment Variables (process.env.apiKey)

Ensure that the apiKey is correctly set in the environment variables. If process.env.apiKey is undefined, the API calls will fail.

5. Check for Proper Rendering of View (res.render)

Ensure that the rendering of the view ambulance-nearby-hospitals is done correctly. You are passing the nearbyHospitals and distanceTime arrays to the view, but ensure that these values are as expected.

6. Error Handling

Ensure all potential errors are caught and handled appropriately in all asynchronous operations (await, axios.get, pool.query, etc.).

Code2:

```
const pool = require('../config/db');



// Render the Upcoming Appointments page

module.exports.upcoming_appointments = async (req, res) => {

  try {

      // Find out those appointments that are not cancelled and are not completed,i.e., is_pending bit is 1; from the appoints table

      let appointment = await pool.query(SELECT * FROM appoints WHERE is_pending = '1' AND doc_email = $1, [req.user.email]);

      appointment = appointment.rows;
```

```
        return res.render('doctor-appointments', {
            title: 'Upcoming Appointments',
            appointments: appointment
        })
    } catch (error) {
        console.error(error.message);
        return res.status(500).json({ error: 'Server Error' });
    }
}

// Render the Past Appointments page
module.exports.past_appointments = async (req, res) => {
    try {
        // Find out those appointments that are completed,i.e., is_pending bit is 0;
from the appoints table
        let appointment = await pool.query(SELECT * FROM appoints WHERE
is_pending = '0' AND doc_email = $1, [req.user.email]);
        appointment = appointment.rows;

        return res.render('doctor-appointments', {
            title: 'Past Appointments',
            appointments: appointment
        })
    } catch (error) {
        console.error(error.message);
        return res.status(500).json({ error: 'Server Error' });
    }
}


// Cancel an appointment
module.exports.open_appointment = async (req, res) => {
    try {
        const id = req.params.id.toString();
        // Given the appointment id, fetch the appointment details from the
appoints table, i.e., the patient name, doctor name, date, time, etc.
        // You can use multiple queries to fetch the desired results like using
doctors table for fetching the name of the doctor from
        // the doctor's table but make sure you fetch it by making variables with
appropriate names like doctor_name, patient_name, etc.
        let appointment = await pool.query(`with appoints_detail(patient_email,
date, start_time, end_time, prescription, is_pending) as (
                                    select patient_email, date, start_time, end_time,
prescription, is_pending
                                    from appoints
                                    where id = $1 )
                                select name as patient_name, email, date, start_time,
end_time, prescription, is_pending, gender, height, weight, blood_group,
diseases, past_history, contact
                                from appoints_detail, patient
```

```
                        where patient_email = email`, [id]);

        appointment = appointment.rows[0];

        return res.render('doctor-open-appointment', {
            title: 'Edit Appointment',
            appointment: appointment,
            id: id
        });

    } catch (error) {
        console.log('Error: ', error.message);
        return res.status(500).json({ error: 'Server Error' });
    }
}

// Update an appointment
module.exports.update_appointment = async (req, res) => {
    try {

        // update the is_pending bit to 0 in the appoints table for the given
appointment id
        // details are available in req.body like req.body.id, req.body.date, etc.
```

```
      if(req.body.pending === 'N'){
         await pool.query(`update appoints
                    set is_pending = 0
                    where id = $1`, [req.params.id]);

         // update the prescription in the appoints table for the given
appointment id
         await pool.query(`update appoints
         set prescription = $1
         where id = $2`, [req.body.prescription, req.params.id]);

      } else {
         await pool.query(`update appoints
                    set is_pending = 1
                    where id = $1`, [req.params.id]);
      }

      req.flash('success', 'Appointment updated successfully');
      return res.redirect('/');

   } catch (error) {
      console.log('Error: ', error.message);
      return res.status(500).json({ error: 'Server Error' });
```

```
    }
}
```

Debugging:

1. SQL Query Syntax Error in upcoming_appointments and past_appointments

The SQL queries are missing backticks or quotes around the SELECT statement. JavaScript requires you to wrap your SQL strings in backticks (`) or single/double quotes, and placeholders for dynamic values need to be outside the query string.

2. Error in Query of open_appointment

In the query for open_appointment, you're doing a complex query using WITH. While the query itself seems logically sound, ensure the columns and table names (appoints_detail, patient) match the structure of your database.

Potential Issue: If there is any typo or mismatch between the schema and the query, it will cause a SQL error.

3. Error Handling in update_appointment

There are multiple SQL updates happening in this block, and some issues could arise:

- If one query fails, it won't stop the others from running.
- If there's an error, the second update may not execute properly.
- Potential SQL Injection risk if inputs are not sanitized.

Potential Issue: The flow of if/else logic could lead to partial updates or skipped updates.

4. Add Proper Logging for Easier Debugging

At multiple places, you can add console.log() to track the flow and data being fetched/updated:

- Log the req.user.email to check the user's email being passed.
- Log the result of pool.query() to see what data is returned from the database.

- Log the req.body in update_appointment to confirm the data being sent.

5. Check the Flash Messages

Ensure req.flash() is properly configured in your application. If it's not working as expected, you may not see the success messages. Flash messages require connect-flash middleware and session setup in Express.

Code3:

```
const pool = require('../config/db');



// Rendering the page for displaying doctor's name, email, contact and reg_no

module.exports.doctors = async (req, res) => {

   try {

      let doctors = await pool.query(`select name, email, contact, reg_no,
specialization

                              from doctor

                              order by name`);

      doctors = doctors.rows;



      return res.render('govt_agency-doctor', {

         title: 'Doctors',

         doctors: doctors

      });

   } catch (error) {

      console.log('Error: ', error.message);
```

```
        return res.status(500).json({error: 'Server Error!'});
    }
}


// Rendering the page for displaying hospital's name, specialization, total beds
and total staff at the hospital
module.exports.hospitals = async (req, res) => {
    try {
        let hospitals = await pool.query(`SELECT
        name,
        email,
        specialities,
        (icu + iicu + operation_theatre + general_ward) AS total_beds,
        (nurse + intern + ot_technician) AS total_staff
    FROM
        hospital
    ORDER BY
        name;
    `);
        hospitals = hospitals.rows;


        return res.render('govt_agency-hospital', {
            title: 'Hospitals',
         hospitals: hospitals
      });
   } catch (error) {
      console.log('Error: ', error.message);
      return res.status(500).json({error: 'Server Error!'});
   }
}
```

Debugging:

1. Add Logging for Data Returned by Queries

- Ensure the data returned by the database query is valid before rendering the page.
- Log the result of the queries before sending them to the res.render() method.

2. SQL Query Debugging

Make sure your queries are correct based on your database schema, especially the column names (specialities, icu, iicu, etc.). These could lead to errors if any of them are misspelled or if they don’t exist.

3. Handle Edge Cases

Ensure that the code handles cases where there are no doctors or hospitals returned from the query.

4. Render Templates

Make sure that the templates govt_agency-doctor and govt_agency-hospital exist and are correctly configured to display the data passed to them.

5. Check for DB Connection Issues:

Ensure that the database connection (pool.query) is working correctly. If there is any issue with the connection, the catch block should log it, but make sure the connection configuration in ../config/db is correct.

Code4:

```
const contact_mailer = require('../mailers/contacts');

const pool = require('../config/db');

const bcrypt = require('bcryptjs');


module.exports.home = async function(req, res){


   return res.render('home', {

      title: "MediAssist | Home"

   });

}


module.exports.about = function(req, res){

   return res.render('about', {

      title: "MediAssist | About"

   });

}


module.exports.contact = function(req, res){

   return res.render('contact', {
        title: "MediAssist | Contact US"

    });

}


module.exports.team = function(req,res){

    return res.render('team', {

        title: "MediAssist | Team"

    });

}


module.exports.submit_contact = async function(req, res){


    try{


        if(!req.user){

            req.flash('error', 'You need to Login to send the Mail!');

            return res.redirect('back');

        }


        let user = await pool.query(select email, name from users where email =
$1, [req.user.email]);

        user = user.rows[0];

        contact_mailer.contact(user, req.body.subject, req.body.message);
        return res.redirect('back');


    } catch(err){

        console.log('Error: ', err);

    }


}
```

Debugging:

1. Check SQL Query Syntax:

There is a syntax error in the select query. SQL keywords must be in uppercase, and the SELECT statement requires parentheses around the query parameters array.

2. Ensure User Object is Fetched Properly:

After fetching the user from the database, ensure you’re properly handling the case when no user is found.

3. Check for Email Function Call:

Ensure that the contact_mailer.contact function is properly implemented and can send emails as expected. If there’s an issue with email delivery, ensure you log the appropriate error and return feedback to the user.

4. Properly Handle Errors:

Add proper error handling in case of issues with fetching the user, sending the email, or any other part of the function. Log the error and ensure the user is informed in case something goes wrong.

Code5:

```
const pool = require('../config/db');


// Render the Add Pharmacy page

module.exports.add_pharma = function(req, res){

   return res.render('add-pharma', {

      title: 'Add Pharma'

   });

}


// Render the Add Laboratory page

module.exports.add_lab = function(req, res){

   return res.render('add-lab', {

      title: 'Add Lab'

   });

}


// Register a new Pharmacy

module.exports.register_pharma = async function(req, res){

   if(req.params.email === req.user.email){
        try {

            if(req)

            // You have the data in req.body and can access it like this:
req.body.detail_you_want. Make an INSERT query into the Pharmacy table
using this data.

            // Write the query in the backticks below

            await pool.query(INSERT INTO
Pharmacy(email,username,name,contact,email_hospital) VALUES
($1,$2,$3,$4,$5),[req.body.email,req.body.username,req.body.name,req.body.
contact,req.body.hospital]);

            await pool.query(INSERT INTO Users(email,username,password,role)
VALUES
($1,$2,$3,$4),[req.body.email,req.body.username,req.body.password,'Pharma
cy']);



            req.flash('success', 'Pharmacy added successfully');

            return res.redirect('/');

        } catch (error) {

            console.log('Error: ', err);

            return res.redirect('back');

        }

    } else {

        return res.redirect('back');

    }

}
```

```
// Register a new Laboratory

module.exports.register_lab = async function(req, res){



   if(req.params.email === req.user.email){

      try {

         // You have the data in req.body and can access it like this:
req.body.detail_you_want. Make an INSERT query into the Pharmacy table
using this data.

         // Write the query in the backticks below

         await pool.query(INSERT INTO
Laboratory(email,username,name,contact,instruments,email_hospital) VALUES
($1,$2,$3,$4,$5,$6),[req.body.email,req.body.username,req.body.name,req.b
ody.contact,req.body.instruments,req.body.hospital]);

         await pool.query(INSERT INTO Users(email,username,password,role)
VALUES
($1,$2,$3,$4),[req.body.email,req.body.username,req.body.password,'Laborat
ory']);



         req.flash('success', 'Laboratory added successfully');

         return res.redirect('/');

      } catch (error){

         console.log('Error: ', err);

         return res.redirect('back');

      }

   } else {
        return res.redirect('back');
    }
}


// Render the appointments views page for the hospital
module.exports.appointments = async (req, res) => {
    try {


        // Find out the doctors that are enrolled in the hospital using the works
table
        let appointments = await pool.query(`with hosp_doctors as (
                                        select distinct(doc_email)
                                        from works
                                        where hosp_email=$1)
                                    select *
                                    from appoints natural join hosp_doctors
                                    where (is_pending='1')`, [req.user.email]);
        appointments = appointments.rows;


        return res.render('hospital_appointments', {
            title: 'Appointments',
            appointments: appointments
        })
    } catch (error) {
        console.log('Error: ', error.message);
        return res.status(500).json({ error: 'Server Error' });
    }
}


// Render the add doctors page
module.exports.add_doctor = async (req, res) => {
    return res.render('add-doctor', {
        title: 'Add Doctor'
    });
}


// Register a new Doctor at the hospital
module.exports.register_doctor = async (req, res) => {
    if(req.params.email === req.user.email){
        try {
            // You have the data in req.body and can access it like this: 
req.body.detail_you_want. Make an INSERT query into the Doctor table using 
this data.
            // Write the query in the backticks below
            await pool.query(insert into doctor 
(REG_NO,EMAIL,USESRNAME,NAME,GENDER,QUALIFICATION,EXPERIENCE,CO
NTACT,INSTITUTE,ADDRESS,SPECIALIZATION) values 
($1,$2,$3,$4,$5,$6,$7,$8,$9,$10,$11),
[req.body.reg_no,req.body.email,req.body.username,req.body.name,req.body
.name,req.body.gender,req.body.qualification,req.body.experience,req.body.c
ontact,req.body.institute,req.body.address,req.body.specialization]);


            // Insert into the works table

            await pool.query(insert into works
(doc_email,hosp_email,start_time,end_time,salary) values
($1,$2,$3,$4,$5),[req.body.doc_email,req.body.hosp_email,req.body.start_tim
e,req.body.end_time,req.body.salary]);


            req.flash('success', 'Doctor added successfully');

            return res.redirect('/');




        } catch (error) {

            console.log('Error: ', error.message);

            return res.status(500).json({ error: 'Server Error' });

        }

    } else {

        return res.redirect('back');

    }

}


// Render the page showing the doctors of the given hospital
```

```
module.exports.view_doctors = async (req, res) => {

   try {

      // Find out the doctors that are enrolled in the hospital using the works
table. Need complete details of doctors from both

      // works as well as the doctor table.

      let doctors = await pool.query(`SELECT *

                              FROM doctor JOIN works ON doctor.email =
works.doc_email

                              WHERE works.hosp_email = $1`, [req.user.email]);

      doctors = doctors.rows;



      return res.render('hospital_doctors', {

         title: 'Doctors',

         doctors: doctors

      })

   } catch (error) {

      console.log('Error: ', error.message);

      return res.status(500).json({ error: 'Server Error' });

   }

}



module.exports.manage_patients = async (req, res) => {

   try {
        // Find out the patients that are enrolled in the hospital using the

        // doctors details from the appoints table.

        let patients = await pool.query(select
a.start_time,a.end_time,a.doc_email,a.date,a.id , p.name,p.email,p.contact
from appoints a join Patient p on a.patient_email = p.email where
a.is_pending='0' AND (a.doc_email in (select doc_email from works where
hosp_email=$1)), [req.user.email]);

        patients = patients.rows;



        return res.render('hospital-patients', {

            title: 'Manage Patients',

            patients: patients

        })

    } catch (error) {

        console.log('Error: ', error.message);

        return res.status(500).json({ error: 'Server Error' });

    }

}



// Cancel an appointment

module.exports.cancel_appointment = async (req, res) => {

    try {

        const { id } = req.params;

        // Given the appointment id, delete the appointment from the appoints
table
```

```
        await pool.query(delete from appoints where id= $1, [id]);



        req.flash('success', 'Appointment cancelled successfully');

        return res.redirect('back');

    } catch (error) {

        console.log('Error: ', error.message);

        return res.status(500).json({ error: 'Server Error' });

    }

}
```

Debugging:

1. SQL Query Syntax Errors:

- SQL keywords like INSERT INTO and DELETE should be in uppercase.
- The SQL query string should be wrapped in backticks (`) when spanning multiple lines, and query parameters should be placed within square brackets [].
- Ensure you are using correct SQL syntax for values and parameter placeholders like $1, $2, etc.

2. Query Parameter Placement:

- You must properly pass the query parameters in the correct format as an array after the SQL string.

3. **req.user.email Verification:**

- Ensure that req.user.email is correctly validated before running database queries, and consider adding extra validation to prevent errors.

4. Missing Error Handling:

- Ensure all caught errors are properly logged, and meaningful messages are returned to the client.

Code6:

```
const pool = require('../config/db');

const axios = require('axios');



// Render the appointment booking page

module.exports.find_hospitals_doctors = async (req, res) => {



    try {

        let hospitals = await pool.query(select * from hospital);

        hospitals = hospitals.rows;

        let nearbyHospitals = [];

        let doctorsInNearbyHospitals = [];



        if (Object.keys(req.query).length > 0) {

            const lat = req.query.lat;

            const lng = req.query.lng;



            // find the nearby hospitals

            for (const hospital of hospitals) {

                const apiKey = process.env.apiKey;
            const startCoordinates = lat + ',' + lng;

            if (!hospital.location) continue;

            const endCoordinates = hospital.location.replace(/\s+/g, ''); //
Remove spaces

            const traffic = true;



            const tomtomApiEndpoint =
'https://api.tomtom.com/routing/1/calculateRoute/';

            const url =
${tomtomApiEndpoint}${startCoordinates}:${endCoordinates}/json?key=${
apiKey}&traffic=${traffic};



            const response = await axios.get(url);

            const data = response.data;

            const route = data.routes && data.routes[0];



            if (route) {

              const distance = route.summary.lengthInMeters / 1000; // in km

              const travelTime = route.summary.travelTimeInSeconds / 3600;
// in hrs



              if (distance < 20) {

                await nearbyHospitals.push(hospital);
```

```
                  let doctors = await pool.query(`select * from doctor where
email in (

                        select doc_email from works where hosp_email=$1

                     )`, [hospital.email]);

                  doctors = doctors.rows;


                  doctorsInNearbyHospitals[hospital.email] = {

                     doctors: doctors,

                     distance: distance,

                     travelTime: travelTime

                  };

               }

            } else {

               console.error('No route found.');

            }

         }

      } else {


         let location = await pool.query(select location from patient where
email=$1, [req.user.email]);

         location = location.rows[0].location;


         if (!location) {
```

```
        req.flash('error', 'Please update your location to find nearby
hospitals and doctors');

        return res.redirect('/users/profile/' + req.user.email);

    }



    // find the nearby hospitals

    for (const hospital of hospitals) {

        const apiKey = process.env.apiKey;


        const startCoordinates = location;

        if (!hospital.location) continue;

        const endCoordinates = hospital.location.replace(/\s+/g, ''); //
Remove spaces

        const traffic = true;


        const tomtomApiEndpoint =
'https://api.tomtom.com/routing/1/calculateRoute/';

        const url =
${tomtomApiEndpoint}${startCoordinates}:${endCoordinates}/json?key=${
apiKey}&traffic=${traffic};


        const response = await axios.get(url);

        const data = response.data;

        const route = data.routes && data.routes[0];
```

```
            if (route) {
                const distance = route.summary.lengthInMeters / 1000; // in km
                const travelTime = route.summary.travelTimeInSeconds / 3600;
// in hrs


                if (distance < 25) {
                    nearbyHospitals.push(hospital);


                    let doctors = await pool.query(`select * from doctor where
email in (
                            select doc_email from works where hosp_email=$1
                        )`, [hospital.email]);
                    doctors = doctors.rows;


                    doctorsInNearbyHospitals[hospital.email] = {
                        doctors: doctors,
                        distance: distance,
                        travelTime: travelTime
                    };
                }
            } else {
                console.error('No route found.');
                }
            }
        }

        return res.render('patient-find-hospitals-doctors', {
            title: 'Find Hospitals and Doctors',
            hospitals: nearbyHospitals,
            doctorsInNearbyHospitals: doctorsInNearbyHospitals
        });
    } catch (error) {
        console.log('Error: ', error.message);
        return res.status(500).json({ error: 'Server Error!' });
    }


}

// Render the appointments page
module.exports.check_slots = function (req, res) {
    return res.render('appointment', {
        title: "Appointment | MediAssist",
        date: null,
        available: false,

        doctor: req.params.email

    });

}



// Check if the doctor is available on the given date

module.exports.check_availability = async function (req, res) {

    try {

        let date = req.body.date;

        let slots = await pool.query(select * from appoints where date=$1 AND
doc_email=$2, [date, req.params.email]);

        slots = slots.rows;



        if (slots.length == 0) {



            let startTime = await pool.query(select start_time from works where
doc_email=$1, [req.params.email]);

            startTime = startTime.rows[0].start_time;

            let endTime = await pool.query(select end_time from works where
doc_email=$1, [req.params.email]);

            endTime = endTime.rows[0].end_time;

            const newSlots = [];



            const slotDuration = 60 * 60 * 1000; // 1 hour in milliseconds
        let formattedStartTime = "";

        let formattedEndTime = "";


        if(!startTime || !endTime) {

          start_time = new Date(date + 'T09:00:00');

          end_time = new Date(date + 'T17:00:00');

          const numberOfSlots = 8;


          for (let i = 0; i < numberOfSlots; i++) {

            const slotStartTime = new Date(startTime.getTime() + i *
slotDuration);

            const slotEndTime = new Date(slotStartTime.getTime() +
slotDuration);


            // Extract hours and minutes from Date object using getHours()
and getMinutes() methods

            const start_hours = slotStartTime.getHours();

            const start_minutes = slotStartTime.getMinutes();

            formattedStartTime = ${start_hours.toString().padStart(2,
'0')}:${start_minutes.toString().padStart(2, '0')};


            const end_hours = slotEndTime.getHours();

            const end_minutes = slotEndTime.getMinutes();
```

```
                formattedEndTime = ${end_hours.toString().padStart(2,
'0')}:${end_minutes.toString().padStart(2, '0')};


                newSlots.push({
                    start_time: formattedStartTime,
                    end_time: formattedEndTime,
                    is_booked: false
                });
            }
        } else {
            const numberOfSlots = parseInt(endTime.substring(0, 2), 10) -
parseInt(startTime.substring(0, 2), 10);


            const start_time = new Date(date + 'T' + startTime + ':00');
            const end_time = new Date(date + 'T' + endTime + ':00');


            for(let i = 0; i < numberOfSlots; i++){
                const slotStartTime = new Date(start_time.getTime() + i *
slotDuration);
                const slotEndTime = new Date(slotStartTime.getTime() +
slotDuration);


                // Extract hours and minutes from Date object using getHours()
and getMinutes() methods
                const start_hours = slotStartTime.getHours();
                const start_minutes = slotStartTime.getMinutes();

                formattedStartTime = ${start_hours.toString().padStart(2,
'0')}:${start_minutes.toString().padStart(2, '0')};


                const end_hours = slotEndTime.getHours();

                const end_minutes = slotEndTime.getMinutes();

                formattedEndTime = ${end_hours.toString().padStart(2,
'0')}:${end_minutes.toString().padStart(2, '0')};


                newSlots.push({

                    start_time: formattedStartTime,

                    end_time: formattedEndTime,

                    is_booked: false

                });

            }

        }


        return res.render('appointment', {

            title: 'Appointment | MediAssist',

            slots: newSlots,

            date: date,

            available: true,

            doctor: req.params.email
```

```
    });

  } else {

    let startTime = await pool.query(select start_time from works where
doc_email=$1, [req.params.email]);
    startTime = startTime.rows[0].start_time;
    let endTime = await pool.query(select end_time from works where
doc_email=$1, [req.params.email]);
    endTime = endTime.rows[0].end_time;
    const newSlots = [];

    const slotDuration = 60 * 60 * 1000; // 1 hour in milliseconds

    let formattedStartTime = "";
    let formattedEndTime = "";

    if(!startTime || !endTime) {
      start_time = new Date(date + 'T09:00:00');
      end_time = new Date(date + 'T17:00:00');
      const numberOfSlots = 8;

      for (let i = 0; i < numberOfSlots; i++) {
          const slotStartTime = new Date(startTime.getTime() + i *
slotDuration);

          const slotEndTime = new Date(slotStartTime.getTime() +
slotDuration);


          // Extract hours and minutes from Date object using getHours()
and getMinutes() methods

          const start_hours = slotStartTime.getHours();

          const start_minutes = slotStartTime.getMinutes();

          formattedStartTime = ${start_hours.toString().padStart(2,
'0')}:${start_minutes.toString().padStart(2, '0')};


          const end_hours = slotEndTime.getHours();

          const end_minutes = slotEndTime.getMinutes();

          formattedEndTime = ${end_hours.toString().padStart(2,
'0')}:${end_minutes.toString().padStart(2, '0')};


          if(slots.some(slot =>

            slot.start_time === formattedStartTime && slot.end_time ===
formattedEndTime

          )) {

            newSlots.push({

              start_time: formattedStartTime,

              end_time: formattedEndTime,

              is_booked: true
```

```
                        });

                    } else {

                        newSlots.push({

                            start_time: formattedStartTime,

                            end_time: formattedEndTime,

                            is_booked: false

                        });

                    }

                }

            } else {

                const numberOfSlots = parseInt(endTime.substring(0, 2), 10) -
parseInt(startTime.substring(0, 2), 10);



                const start_time = new Date(date + 'T' + startTime + ':00');

                const end_time = new Date(date + 'T' + endTime + ':00');



                for(let i = 0; i < numberOfSlots; i++){

                    const slotStartTime = new Date(start_time.getTime() + i *
slotDuration);

                    const slotEndTime = new Date(slotStartTime.getTime() +
slotDuration);



                    // Extract hours and minutes from Date object using getHours()
and getMinutes() methods
```

```
                const start_hours = slotStartTime.getHours();

                const start_minutes = slotStartTime.getMinutes();

                formattedStartTime = ${start_hours.toString().padStart(2,
'0')}:${start_minutes.toString().padStart(2, '0')};



                const end_hours = slotEndTime.getHours();

                const end_minutes = slotEndTime.getMinutes();

                formattedEndTime = ${end_hours.toString().padStart(2,
'0')}:${end_minutes.toString().padStart(2, '0')};



                if(slots.some(slot =>

                   slot.start_time === formattedStartTime && slot.end_time ===
formattedEndTime

                )) {

                   newSlots.push({

                      start_time: formattedStartTime,

                      end_time: formattedEndTime,

                      is_booked: true

                   });

                } else {

                   newSlots.push({

                      start_time: formattedStartTime,

                      end_time: formattedEndTime,

                      is_booked: false
```

```
                    });
                }
            }
        }

        return res.render('appointment', {
            title: 'Appointment | MediAssist',
            slots: newSlots,
            date: date,
            available: true,
            doctor: req.params.email
        });
    }

  } catch (err) {
      console.log('Error: ', err);
      return res.redirect('back');
  }
}

// Make an appointment
module.exports.make_appointment = async (req, res) => {
   try {


       // Generating 15digits id with current date and time so that each
appointment id is unique

       // assuming that no two appointments are booked at the same time

       const currentDate = new Date();

       const options = { timeZone: 'Asia/Kolkata' };

       const formattedDate = currentDate.toLocaleDateString('en-IN',
options).replace(/\//g, '');

       const formattedTime = currentDate.toLocaleTimeString('en-IN', {
hour12: false, timeZone: 'Asia/Kolkata' }).replace(/:/g, '');

       const formattedDateTime = ${formattedDate}T${formattedTime};


       const id = formattedDateTime; // I will generate unique id using any
predefined functions


       const [start_time, end_time] = req.body.slot.split(',');


       // Update the appoints table by making the is_pending bit 1 and adding
the patient's email and id

       // req.user.email is the patient's email and req.params.email is the
doctor's email

       await pool.query(INSERT INTO appoints(is_pending, patient_email, id,
doc_email, date, start_time, end_time) VALUES  ('1', $1, $2, $3, $4, $5, $6),
[req.user.email, id, req.query.email, req.query.date, start_time, end_time]);

       let slot = await pool.query(select * from appoints where id=$1, [id]);
```

```
        slot = slot.rows[0];

        let name = await pool.query(select name from patient where email=$1,
[req.user.email]);

        name = name.rows[0].name;



        req.flash('success', 'Appointment booked successfully');

        return res.render('appointment_display', {

            title: 'Appointment Booked | MediAssist',

            slot: slot,

            name: name,

            email: req.user.email

        });


    } catch (error) {

        console.log('Error: ', error.message);

        return res.status(500).json({error: 'Server Error!'});

    }

}


module.exports.track_appointment = async (req, res) => {

    try {

        // given the patient's email in req.user.email, find out the
appointments booked by the patients
```

```
}

        // All the appointments, i.e., pending bit 0 as well as 1

        // write two diiferent different queries for upcoming and past
appointments

        let upcoming_appointments = await pool.query(select * from appoints
where patient_email=$1 AND is_pending ='1',[req.user.email]);

        upcoming_appointments = upcoming_appointments.rows;



        let past_appointments = await pool.query(select * from appoints
where patient_email=$1 AND is_pending ='0',[req.user.email]);

        past_appointments = past_appointments.rows;

       //

        return res.render('patient-appointments', {

            title: 'Appointments',

            upcoming_appointments: upcoming_appointments,

            past_appointments: past_appointments

        });


    } catch (error) {

        console.log('Error: ', error.message);

        return res.status(500).json({error: 'Server Error!'});

    }

}


// Cancel an appointment
```

```
module.exports.cancel_appointment = async (req, res) => {

   try {

      const { id } = req.params;

      // Given the appointment id, delete the appointment from the
appoints table

      await pool.query(delete from appoints where id= $1, [id]);



      req.flash('success', 'Appointment cancelled successfully');

      return res.redirect('back');

   } catch (error) {

      console.log('Error: ', error.message);

      return res.status(500).json({ error: 'Server Error' });

   }

}
```

Debugging:

1. SQL Query Syntax

In your SQL queries, the SELECT and DELETE statements are missing quotes around the query strings. They need to be enclosed in double or backticks when passed into the pool.query() function.

2. Template Literal Formatting in Strings

There are a few template literals (${}) that are incorrectly used in normal strings.

3. SQL Query Variables

In some places, your SQL queries are missing placeholders or parentheses, which could lead to syntax errors.

4. Conditional Logic for Doctor Slot Timing

You are using two different ways to fetch doctor timings from the works table based on whether or not startTime and endTime exist. The logic is duplicated and can be streamlined for better readability and maintainability.

5. Creating Unique Appointment IDs

The code for generating unique appointment IDs based on the current date and time looks good, but you can use libraries like uuid to generate unique IDs in a more standard way.

6. Slot Duration Calculation

The code is calculating slots in a 1-hour format, but what if you want to offer shorter or longer durations? You may want to introduce a variable for slotDuration to easily adjust the slot length.

7. Debugging Suggestions

Here are common ways to debug:

- **Console Logging:** Use console.log() extensively to trace values.
- **Error Handling:** Add more detailed error handling and log stack traces to get insights into issues.

Code7:

```
const pool = require('../config/db');



// Rendering the page showing the medicine stocks present at the pharmacy
and respective quantities
```

```
}

module.exports.stocks = async (req, res) => {
   try {

       // req.user.email has the email of pharmacy logged in from which you can
use the stores table to
       // find out the medicines and their quantities present in the pharmacy
       let medicines = await pool.query(SELECT name, brand_name, stock FROM
stores WHERE email_pharm = $1,[req.user.email]);

       medicines = medicines.rows;

       return res.render('pharmacy_stocks', {
          title: 'Stocks',
          medicines: medicines
       });

   } catch (error) {
       console.log('Error: ', error.message);
       return res.status(500).json({ msg: 'Internal Server Error' });
   }
}
```

```
// Rendering the page for pharmacy to update the stocks details

module.exports.update_stocks = async (req, res) => {

   try {


      // req.user.email has the email of pharmacy logged in from which you can
use the stores table to

      // find out those medicines from the pharmacy whose quantity is greater
than 0


      let medicines = await pool.query(SELECT name, brand_name, stock FROM
stores WHERE email_pharm = $1,[req.user.email]);


      medicines = medicines.rows;


      let brands = await pool.query(SELECT distinct(brand_name) FROM stores
WHERE email_pharm = $1, [req.user.email]);

      brands = brands.rows;


      return res.render('pharmacy-update-stocks', {

         title: 'Update Stocks',

         medicines: medicines,

         brands: brands

      });
```

```
    } catch (error) {

        console.log('Error: ', error.message);

        return res.status(500).json({ msg: 'Internal Server Error' });

    }

}



// Update Stock of a given medicine

module.exports.update_medicine_stocks = async function(req, res){

    if(req.params.email === req.user.email){

        try {

            // write a query wherein you are having all the details in req.body and
you need to update the stock

            // of a medicine given its brand_name as req.body.brand and name as
req.body.medicine. Also the new stock value is in req.body.stock

            await pool.query(UPDATE stores SET stock=$3 WHERE brand_name = $1
AND name = $2,[req.body.brand,req.body.medicine,req.body.stock]);



            req.flash('success', 'Data Updated Successfully!');

            return res.redirect('/');

        } catch (error){

            console.log('Error: ', error);

            return res.redirect('back');

        }

    } else {
        return res.redirect('back');

    }

}


// Add a new medicine

module.exports.add_medicine = async (req, res) => {

    try {



        // write a query to insert a medicine into the medicne table and the stores
table with

        // name in req.body.name, brand_name in req.body.brand_name, stock in
req.body.stock and

        // pharma email in req.user.email

        await pool.query(INSERT into medicine(name,brand_name) VALUES
($1,$2),INSERT INTO stores (email_pharma,name,brand_name,stock) VALUES
($3,$1,$2,$4),[req.body.name,req.body.brand_name,req.body.email,req.body.
stock]);



        req.flash('success', 'Medicine added Successfully!');

        return res.redirect('back');



    } catch (error) {

        console.log('Error: ', err);

        return res.status(500).json({ msg: 'Internal Server Error'});

    }
```

```
}
```

Debugging:

1. SQL Query Syntax

- SQL keywords should be in uppercase.
- Table names and placeholders in the queries are not properly quoted.
- In some places, parentheses and commas in the queries are incorrectly used.

2. Query Interpolation and Structure

- The query INSERT into medicine(...) VALUES (...), INSERT INTO stores(...) VALUES (...) is invalid because you cannot combine two insert statements like this.
- You are querying distinct brands and medicines in update_stocks but also need to handle SELECT clauses properly.

3. Error Logging

- In the last catch block, you are logging err but returning error.

4. Data Validation

- You should check for valid stock values before updating to avoid inserting invalid data.