SUNSTONE

“Professional Portfolio”

Task - 5

Your Project Name

- Developers build portfolio websites as full-stack developer sample projects to showcase their skills and impress clients.
- In this Portfolio I Use the following Technology.
- HTML, CSS, JS, MongoDB, SRS
- ODBC is an SQL-based Application Programming Interface (API) created by Microsoft that is used by Windows software applications to access databases via SQL.

Campus Name	Student Name	Batch
Ajeenkya D.Y Patil University	Omkar Bhure	2022-25

Task 5 :: Hosting

Hosting the website so that it can be accessed from anywhere

- Host backend on aws with all environment setup
- Make sure to whitelist api ports and host with the database
- Migrate database to mongodb atlas
- Build react app to be hosted on the server
- Set hosted backed url to the frontend
- Test entire frontend and backend

Evaluation Metric:

100% Completion of the above tasks

Learning outcome

- Understand how to host a web api
- How to deploy production ready react applications
- Setting up environment for production

Java Database Connectivity (JDBC)

ODBC is an SQL-based Application Programming Interface (API) created by Microsoft that is used by Windows software applications to access databases via SQL. JDBC is an SQL-based API created by Sun Microsystems to enable Java applications to use SQL for database access.
Java™ database connectivity (JDBC) is the JavaSoft specification of a standard application programming interface (API) that allows Java programs to access database management systems. The JDBC API consists of a set of interfaces and classes written in the Java programming language.

ODBC	JDBC
ODBC Stands for Open Database Connectivity.	JDBC Stands for java database connectivity.
Introduced by Microsoft in 1992.	Introduced by SUN Micro Systems in 1997.
We can use ODBC for any language like C,C++,Java etc.	We can use JDBC only for Java languages.
We can choose ODBC only windows platform.	We can Use JDBC in any platform.
Mostly ODBC Driver developed in native languages like C,C++.	JDBC Stands for java database connectivity.
For Java applications it is not recommended to use ODBC because performance will be down due to internal conversion and applications will become platform Dependent.	For Java application it is highly recommended to use JDBC because there are no performance & platform dependent problem.
ODBC is procedural.	JDBC is object oriented.

The JDBC API defines the Java™ interfaces and classes that programmers use to connect to databases and send queries. A JDBC driver implements these interfaces and classes for a particular DBMS vendor.
A Java program that uses the JDBC API loads the specified driver for a particular DBMS before it actually connects to a database. The JDBC **DriverManager** class then sends all JDBC API calls to the loaded driver.

JDBC-ODBC bridge plus ODBC driver, also called Type 1 driver
Translates JDBC API calls into Microsoft ODBC calls that are then passed to the ODBC driverThe ODBC binary code must be loaded on every client computer that uses this type of driver.

JDBC-Net, pure-Java driver, also called Type 3 driver
Sends JDBC API calls to a middle-tier server that translates the calls into the DBMS-specific network protocol. The translated calls are then sent to a particular DBMS.

Native-protocol, pure-Java driver, also called Type 4 driver
Converts JDBC API calls directly into the DBMS-specific network protocol without a middle tier this driver allows the client applications to connect directly to the database server.

Check JDBC MySQL Connection

This Java program checks if a connection to a MySQL database can be established. The program starts by importing the necessary classes from the java.sql package. These classes include DriverManager, Connection, and SQLException.

The main method of the CheckConnection class defines the database URL, username, and password as string variables. The database URL specifies the protocol (jdbc:mysql:), the location of the database server (localhost:3306), and the name of the database (MCA_JDBC).

The program then attempts to establish a connection to the database using the getConnection method of the DriverManager class. This method takes the database URL, username, and password as arguments and returns a Connection object if the connection is successful.

```
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.SQLException;

public class CheckConnection {
    public static void main(String[] args) {
        String dbURL = "jdbc:mysql://localhost:3306/MCA_JDBC";
        String username = "root";
        String password = "Password";

        try {

            Connection conn = DriverManager.getConnection(dbURL,
username, password);

            if (conn != null) {
                System.out.println("Connected");
            }
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
    }
}
```

JDBC Creates a New Table

This is a Java program that creates a new table named contacts in a MySQL database. The program starts by importing the necessary classes from the java.sql package. These classes include DriverManager, Connection, and Statement.

The main method of the CreateTable class defines the database URL, username, and password as string variables. The database URL specifies the protocol (jdbc:mysql:), the location of the database server (localhost:3306), and the name of the database (MCA_JDBC).

The program then attempts to establish a connection to the database using the getConnection method of the DriverManager class. This method takes the database URL, username, and password as arguments and returns a Connection object if the connection is successful.

```
import java.sql.DriverManager;
import java.sql.Connection;
// import java.sql.SQLException;
import java.sql.Statement;

public class CreateTable {
    public static void main(String[] args) {
        String dbURL = "jdbc:mysql://localhost:3306/MCA_JDBC";
        String username = "root";
        String password = "Password";
        try {
            // Class.forName("com.mysql.jdbc.Driver");
            Connection con = DriverManager.getConnection(dbURL,
username, password);


// Use an SQL CREATE TABLE statement to define the structure of
 the table
            String createTableSql = "CREATE TABLE contacts ("
                    + "id INT AUTO_INCREMENT PRIMARY KEY,"
                    + "name VARCHAR(255),"
                    + "number VARCHAR(255),"
                    + "subject VARCHAR(255),"
                    + "message TEXT)";


// Use the executeUpdate method to execute the CREATE TABLE stat
ement
            Statement stmt = con.createStatement();
            stmt.executeUpdate(createTableSql);

            con.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Once a connection is established, the program defines an SQL CREATE TABLE statement as a string variable. This statement specifies the new table's structure, including its column names and data types.

The program then creates a Statement object using the create Statement method of the Connection object. It uses the execute Update method of the Statement object to execute the CREATE TABLE statement.

Finally, the program closes the connection to the database using the close method of the Connection object. If an exception occurs (for example, if there is an error in the SQL statement), the program catches it and prints its stack trace.

JDBC Insert

This is a Java program that inserts data into a `contacts` table in a MySQL database. The program starts by importing the necessary classes from the `java.sql` package. These classes include `DriverManager`, `Connection`, and `PreparedStatement`.

The `main` method of the `InsertMysql` class defines the database URL, username, and password as string variables. The database URL specifies the protocol (`jdbc:mysql:`), the location of the database server (`localhost:3306`), and the name of the database (`BCA_JDBC`).

The program then attempts to establish a connection to the database using the `getConnection` method of the `DriverManager` class. This method takes the database URL, username, and password as arguments and returns a `Connection` object if the connection is successful.

Once a connection is established, the program defines an SQL `INSERT INTO` statement as a string variable. This statement specifies the name of the table and its columns, and uses placeholders (`?`) for the values to be inserted.

```
// import java.sql.Statement;
import java.sql.DriverManager;
import java.sql.Connection;
// import java.sql.SQLException;
import java.sql.PreparedStatement;

public class InsertMysql {
    public static void main(String[] args) {
        String dbURL = "jdbc:mysql://localhost:3306/MCA_JDBC";
        String username = "root";
        String password = "Password";
        // Connection conn = DriverManager.getConnection(dbURL, username, password);
        //
        try {
            Class.forName("com.mysql.jdbc.Driver");

            Connection con = DriverManager.getConnection(dbURL, username, password);

            // Use an SQL INSERT INTO statement with placeholders for the values
            String insertSql =
"INSERT INTO contacts (name, number, subject, message) VALUES (?, ?, ?, ?)";

            // Use a PreparedStatement to set the values for the placeholders
            PreparedStatement pstmt = con.prepareStatement(insertSql);

            // Add the first row of data to the batch
            pstmt.setString(1, "Narendra Modi");
            pstmt.setString(2, "1234567890");
            pstmt.setString(3, "Question");
            pstmt.setString(4, "How do I use JDBC?");
            pstmt.addBatch();

            // Add the second row of data to the batch
            pstmt.setString(1, "Jane Doe");
            pstmt.setString(2, "0987654321");
            pstmt.setString(3, "Answer");
            pstmt.setString(4, "You can use JDBC to interact with a database from a Java application.");
            pstmt.addBatch();

            // Execute the batch of INSERT INTO statements
            pstmt.executeBatch();

            con.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

The program then creates a `PreparedStatement` object using the `prepareStatement` method of the `Connection` object. It uses the various `setXXX` methods of the `PreparedStatement` object to set the values for each placeholder in the SQL statement.

The program adds two rows of data to a batch using the `addBatch` method of the `PreparedStatement` object. It then executes the batch of `INSERT INTO` statements using the `executeBatch` method.

Finally, the program closes the connection to the database using the `close` method of the `Connection` object. If an exception occurs (for example, if there is an error in the SQL statement), it catches it and prints its stack trace.

JDBC Update

This Java program updates data in a `contacts` table in a MySQL database. The program starts by importing the necessary classes from the `java.sql` package. These classes include `Driver Manager`, `Connection`, and `Prepared Statement`.

The `main` method of the `Update Mysql` class defines the database URL, username, and password as string variables. The database URL specifies the protocol (`jdbc:mysql:`), the location of the database server (`localhost:3306`), and the name of the database (`BCA_JDBC`).

The program then attempts to establish a connection to the database using the `getConnection` method of the `DriverManager` class. This method takes the database URL, username, and password as arguments and returns a `Connection` object if the connection is successful.

Once a connection is established, the program defines an SQL `UPDATE` statement as a string variable. This statement specifies the changes to make to the data in the table, and uses placeholders (`?`) for the values to be updated.

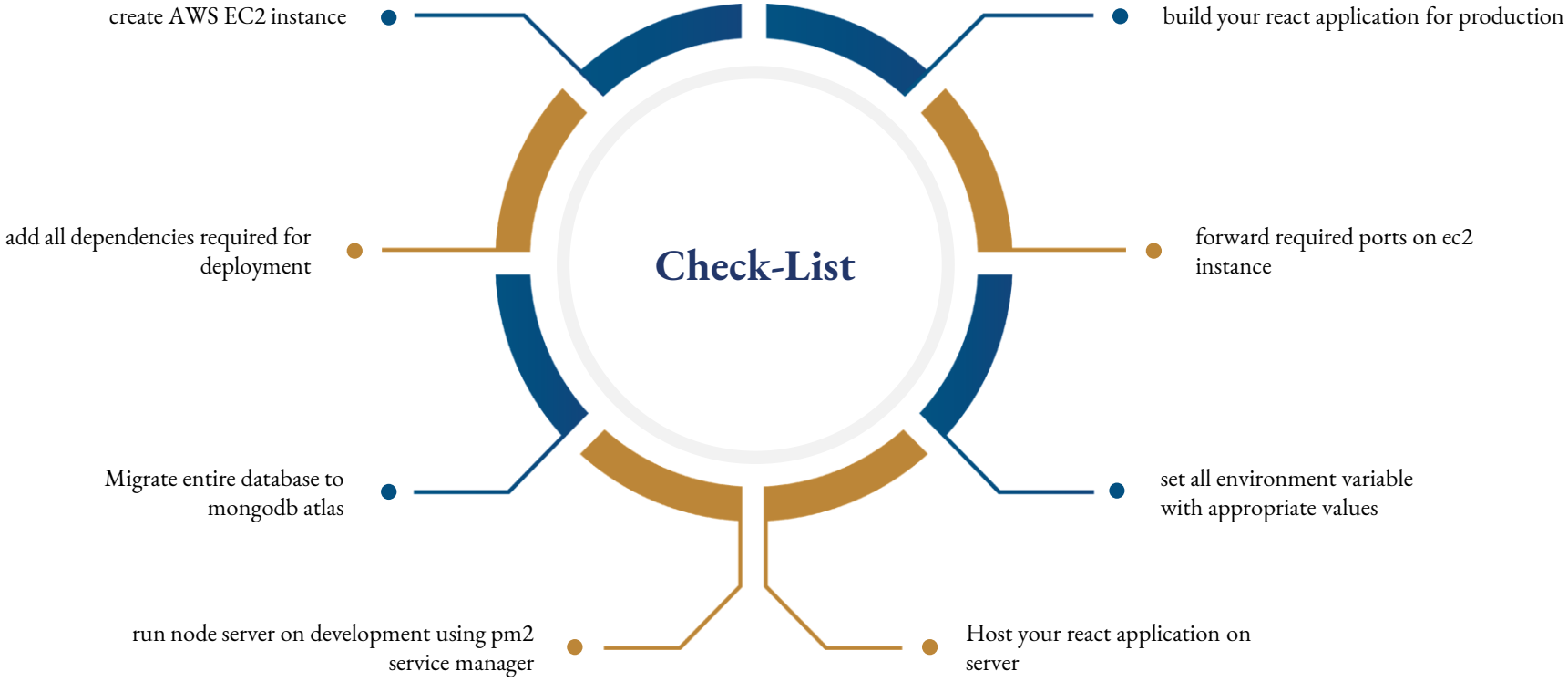
```java
import java.sql.DriverManager;
import java.sql.Connection;
// import java.sql.SQLException;
import java.sql.PreparedStatement;

public class UpdateMysql {
    public static void main(String[] args) {
        String dbURL = "jdbc:mysql://localhost:3306/MCA_JDBC";
        String username = "root";
        String password = "password";
        try {
            Class.forName("com.mysql.jdbc.Driver");

// Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/database_name", "username",
 "password");
            Connection con = DriverManager.getConnection(dbURL, username, password);
            // Use an SQL UPDATE statement to specify the changes to make
            String updateSql = "UPDATE contacts SET name = ? WHERE number = ?";

            // Use a PreparedStatement to set the values for the placeholders
            PreparedStatement pstmt = con.prepareStatement(updateSql);
            pstmt.setString(1, "Yogi Adityanath");
            pstmt.setString(2, "0987654321");
            pstmt.executeUpdate();

            con.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

The program then creates a `Prepared Statement` object using the `Prepare Statement` method of the `Connection` object. It uses the various methods of the `Prepared Statement` object to set the values for each placeholder in the SQL statement.

The program executes the `UPDATE` statement using the `execute Update` method of the `Prepared Statement` object.

Finally, the program closes the connection to the database using the `close` method of the `Connection` object. If an exception occurs (for example, if there is an error in the SQL statement), it catches it and prints its stack trace.

Assessment Parameter



Submission Github

https://github.com/onkarbb/portfollio

Thank you!