Name : J. Sathya Sneka

College code: 9530

College name: St. Mother Theresa Engineering College

Naan Mudhalvan id : au953021104043

Team id: Proj_201056_Team_1

Project Name: E Commerce Application for IBM cloud foundry

INTRODUCTION :

Electronic commerce or E Commerce is a platform through which trading of products and services can be done using the internet.Online food ordering is a process that delivers food. This style of food delivery is gaining popularity with more and more people especially the younger generation turning to mobile food ordering apps, thereby changing the way food is delivered and picked up. The customer can generate an order without having to explain it to another human being and have the food delivered at his doorstep.

PROBLEM SOLUTION :

The objective of this project is to provide a convenient platform for users to browse menus, place food orders, and have those orders delivered to their doorstep efficiently and quickly. Key goals include improving customer convenience and streamlining the food ordering and delivery process. The main objective is that it allows customers to order food without the need to visit in person, saving time and effort. It eliminates the need for meal preparation and cooking, making it a time-saving option for busy individuals. It's also convenient for ordering meals for groups or parties, as individual orders can be combined and delivered together. A food delivery app's goals are to offer a practical and effective platform that links customers and enabling food ordering and delivery services.

INNOVATION:

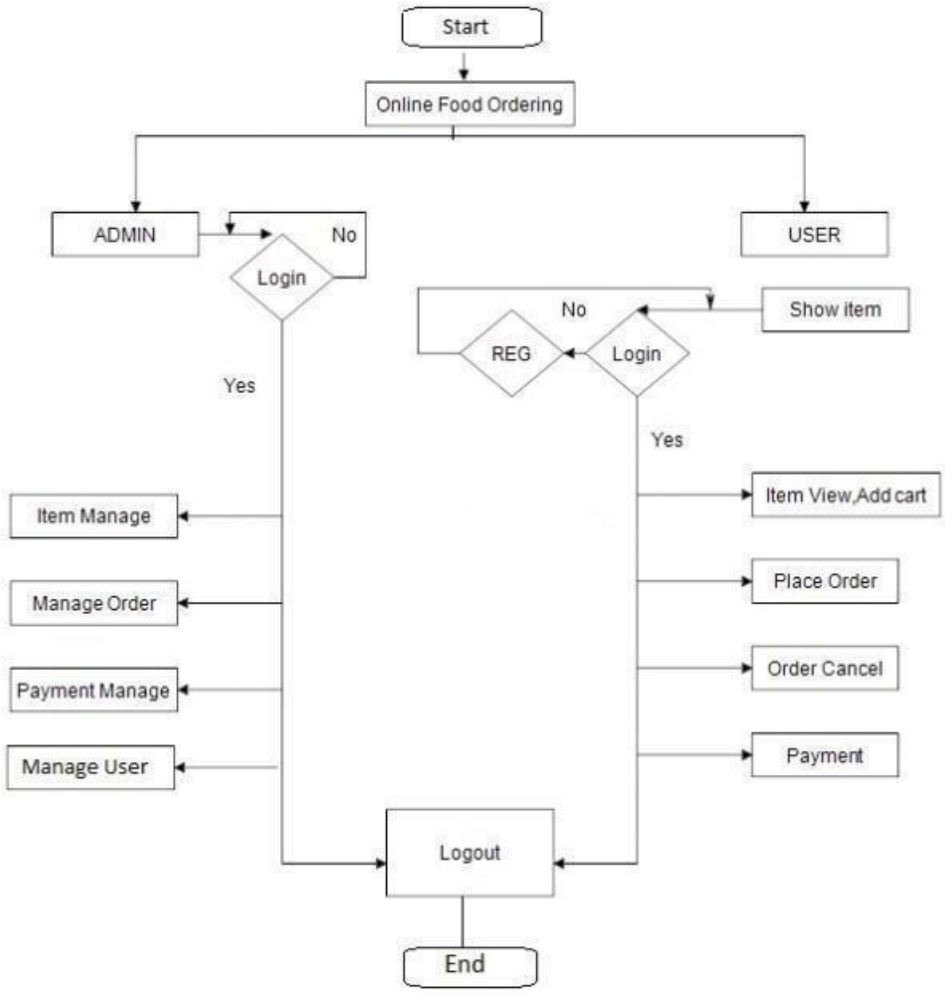
- Using IBM Cloud Functions to implement serverless functions for real-time order tracking, notifications, and automated processes, improving app performance and scalability.
- This E commerce application allows customers to order food without the need to visit in person, saving time and effort which gives the time saving option for busy individuals

SOFTWARE REQUIREMENTS :

Programming Languages:

- Java for Android app development.
- Java or Java-based frameworks for backend development.
- MySQL for database management.

METHODOLOGY :



APPLICATION DESCRIPTION:

There are certain requirements the proposed application must fulfil to meet the objectives of the project.

The requirements to be achieved:

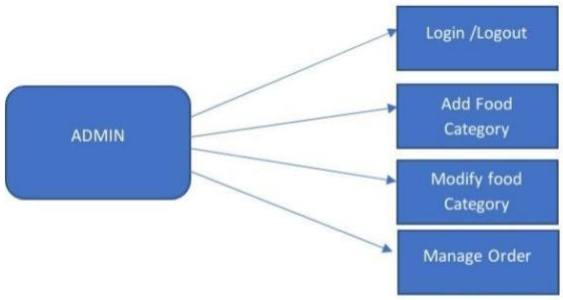
In Background Management Platform:

- Administrator can add and modify food categories.
- Administrator can add, modify and query food information.
- Administrator can add, modify and query employee information.
- Administrator can manage orders produced from the web application and
- Android application.

In the Website Public Page and Android Application:

- Customer can view food information, such as category, name, price, image, description and so on.
- Customer can order food.
- Customer can modify food item, food amount in Shopping Cart.
- Produce food order.

FUNCTIONAL DESCRIPTION :



Background management :

Log In/Out:

Properties:

- Username
- Password

When the administrator connects the background management platform url, a log in interface will be displayed. The administrator needs to input correct username and password to log into the main page of the background management page. One

admin username and password is pre-set when the application is initialized.

Add Food Category:

Properties:

- Category Name
- Note

When the administrator clicks “Category Manage”, a list view of dish category will be displayed. On the bottom of the list view page, there is a button “Add Category”.While clicking the “Add Category” button, the page will be linked to adish-adding interface, the administrator can add a new dish category here.

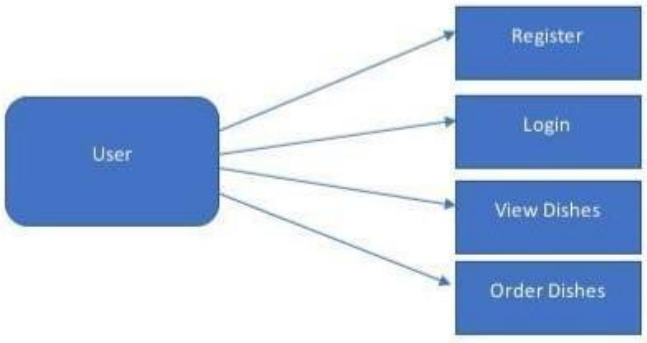
Modify food category:

Properties:

- Category Name
- Note

In addition, in the category list view page, the administrator can modify the existing dish category by clicking “Modify” button.

USER APPLICATION :



Register:

When the customer wants to purchase dishes on the website, first, he needs to register an account by clicking “Register” button to enter the register page, and then just filling in User Name, Password and Repeat Password. If the customer register is successful, a page will display to

inform the customer, also if there is a register fault, such as the User Name already exists, a friendly warning message will be given on the register page.

Log In:

After registering, the customer can log in with the correct User Name and Password to the website to purchase the dishes.

View Dishes:

To click “Online Order” tab, the customer can view the current dishes according to different categories. Every dish item has two buttons: Order and Detail. The customer can click the “Detail” Button to view the detailed information (big image, material of dish, price, etc.) of dishes. By clicking “Order” Button, the dish item you select will be added to the shopping cart.

Shopping Cart:

The dish items purchased by the customer will be added to the shopping cart. The customer can view the dish items he/she purchases, unit price, dish amount and total price. The default amount for each dish item is one, the customer can update the amount according to his own requirement. After modifying the amount,

customer needs to click “Update” button to update the dish amount. Also, if the customer needs to delete some dish item, it is easily implemented by clicking “Delete” Button following each dish item.

CREATING APPLICATION IN IBM CLOUD FOUNDRY:

Step 1:

To start developing your application locally, begin by logging in to the IBM Cloud Platform directly from the command line, as shown in the example. You can specify optional parameters, such as your organization with option -o and the space with option -s. If you’re using a federated account use –sso.

Ibm cloud login

After logging in, when you are asked if you want to install any extensions, you may see an announcement regarding the Cloud Foundry plugin. Type the command as shown in order to download and install the CLI extension used in this tutorial.

Ibmcloud cf install

When you log in you might be asked to choose a region. For this exercise, select us-south as the region, as that same option is used to build a CD Toolchain later in this tutorial.

Next, set the endpoint (if it isn't set already). Other endpoints are possible, and might be preferable for production use. For now, use the code as shown, if appropriate for your account.

Ibmcloud api cloud.ibm.com

Target the Cloud Foundry (cf) aspect of IBM Cloud Platform by using the target command and the –cf option. The cf API is no longer embedded within the CLI Developer Tools and will have to be downloaded separately.

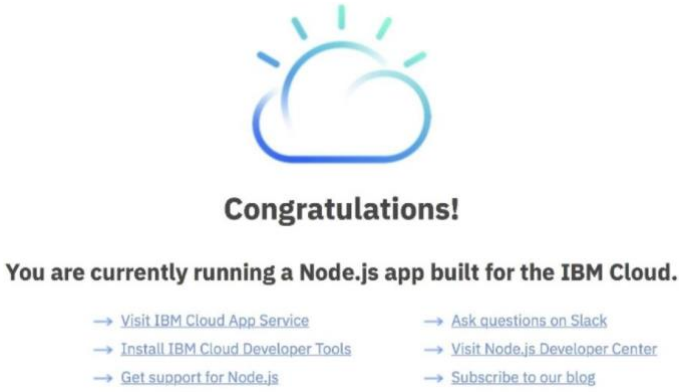
Ibmcloud target –cf

And now, time to create a web application. The dev space is a default option for your organization, but you might prefer to create others for isolating different efforts. For example, keeping 'finance' separate from 'development'.

Ibmcloud dev create

With that command, you're asked a series of questions. You can go back at many points in the process, so if you feel lost you can start over by deleting the existing directory and creating a new directory. Even when you create your application on the command line, you'll still see the results in your IBM Cloud console.

Now there is a option for choosing the application type web or mobile application and have to select the language for our application such as Java or python and then we have to choose the app service after all of this application is created.



Step 2:

Creating food delivery app in IBM cloud foundry. There is a complete solution and methodology about our food delivery application.

Some of the method functions we are using is given below:

IMPLEMENTATION:

Background storage database:

To create a background storage database for food items and their prices in Java, We use SQLite and Java's JDBC (Java Database Connectivity) to create and manage a simple food items database.

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

public class FoodItemsDatabase {
    private static final String DATABASE_URL = "jdbc:sqlite:food_items.db";
    private static final String TABLE_NAME = "food_items";

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

```
        // Establish a connection to the SQLite database
        try (Connection connection = DriverManager.getConnection(DATABASE_URL)) {
            System.out.println("Connected to the database.");

            // Create the food_items table if it doesn't exist
              String createTableSQL = "CREATE TABLE IF NOT EXISTS " + TABLE_NAME + "
(id INTEGER PRIMARY KEY, name TEXT, price REAL)";
                                                try (PreparedStatement createTableStatement =
connection.prepareStatement(createTableSQL)) {
                createTableStatement.executeUpdate();
                System.out.println("Table 'food_items' created.");
            }

            // Insert a food item into the database
            insertFoodItem(connection, "Pizza", 10.99);

            // Retrieve and display food items from the database
            retrieveFoodItems(connection);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    private static void insertFoodItem(Connection connection, String name, double price) throws
SQLException {
        String insertSQL = "INSERT INTO " + TABLE_NAME + " (name, price) VALUES (?, ?)";
        try (PreparedStatement insertStatement = connection.prepareStatement(insertSQL)) {
            insertStatement.setString(1, name);
            insertStatement.setDouble(2, price);
            insertStatement.executeUpdate();
            System.out.println("Inserted food item: " + name);
```

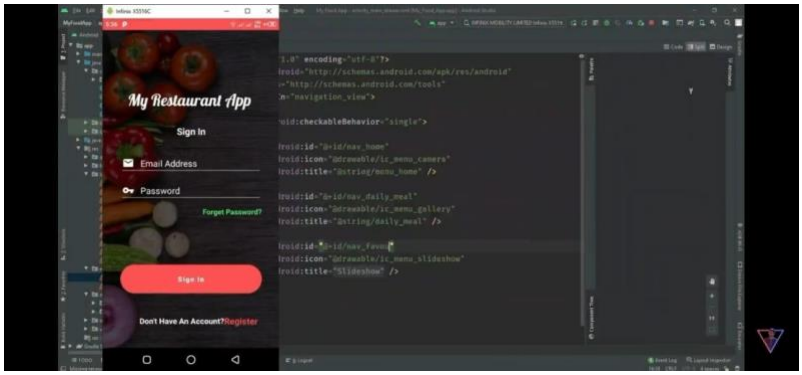
```
        }
    }

    private static void retrieveFoodItems(Connection connection) throws SQLException {
        String selectSQL = "SELECT * FROM " + TABLE_NAME;
        try (PreparedStatement selectStatement = connection.prepareStatement(selectSQL);
            ResultSet resultSet = selectStatement.executeQuery()) {
            System.out.println("Food Items:");
            while (resultSet.next()) {
                int id = resultSet.getInt("id");
                String name = resultSet.getString("name");
                double price = resultSet.getDouble("price");
                System.out.println("ID: " + id + ", Name: " + name + ", Price: $" + price);
            }
        }
    }
}
```

User Registration and Log in :

```
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class UserAuthentication {
    private static Map<String, String> users = new HashMap<>();

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        boolean isRunning = true;

        while (isRunning) {
```

```
        System.out.println("Choose an option:");
        System.out.println("1. Register");
        System.out.println("2. Login");
        System.out.println("3. Exit");
        int choice = scanner.nextInt();
        scanner.nextLine();  // Consume the newline character

        switch (choice) {
            case 1:
                registerUser(scanner);
                break;
            case 2:
                login(scanner);
                break;
            case 3:
                isRunning = false;
                System.out.println("Goodbye!");
                break;
            default:
                System.out.println("Invalid option. Please choose a valid option.");
        }
    }

    scanner.close();
}

private static void registerUser(Scanner scanner) {
    System.out.println("Registration");
    System.out.print("Enter your username: ");
    String username = scanner.nextLine().trim();
    System.out.print("Enter your password: ");
```

```
        String password = scanner.nextLine();

        if (!users.containsKey(username)) {
            users.put(username, password);
            System.out.println("Registration successful!");
        } else {
            System.out.println("Username is already taken. Please choose another.");
        }
    }

    private static void login(Scanner scanner) {
        System.out.println("Login");
        System.out.print("Enter your username: ");
        String username = scanner.nextLine().trim();
        System.out.print("Enter your password: ");
        String password = scanner.nextLine();

        if (users.containsKey(username) && users.get(username).equals(password)) {
            System.out.println("Login successful! Welcome, " + username + "!");
        } else {
            System.out.println("Login failed. Please check your username and password.");
        }
    }
}
```



Backend server for user registration:

Creating a backend server for user registration and authentication in Node.js typically involves using a framework like Express.js and a database to store user information.

```
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');

const app = express();
const port = 3000;

// Connect to MongoDB (Make sure to have MongoDB running locally or provide your database connection URI)
mongoose.connect('mongodb://localhost/user-auth-app', { useNewUrlParser: true, useUnifiedTopology: true });
const db = mongoose.connection;

// Define the User schema and model
const userSchema = new mongoose.Schema({
  username: String,
  password: String,
});

const User = mongoose.model('User', userSchema);

// Middleware for parsing JSON request bodies
app.use(bodyParser.json());
```

```
// User registration endpoint
app.post('/register', async (req, res) => {
const { username, password } = req.body;

  // Check if the user already exists
  const existingUser = await User.findOne({ username });
  if (existingUser) {
    return res.status(400).json({ message: 'User already exists' });
  }

  // Hash the password
  const saltRounds = 10;
  const hashedPassword = await bcrypt.hash(password, saltRounds);

  // Create a new user
  const newUser = new User({ username, password: hashedPassword });
  await newUser.save();

  res.json({ message: 'User registered successfully' });
});

// User login endpoint
app.post('/login', async (req, res) => {
  const { username, password } = req.body;

  // Find the user by username
  const user = await User.findOne({ username });
  if (!user) {
    return res.status(401).json({ message: 'Invalid username or password' });
  }
```

```
  // Check the password
  const passwordMatch = await bcrypt.compare(password, user.password);
  if (!passwordMatch) {
    return res.status(401).json({ message: 'Invalid username or password' });
  }

  // Create and send a JSON Web Token (JWT)
  const token = jwt.sign({ userId: user._id }, 'your-secret-key', { expiresIn: '1h' });
  res.json({ token });
});

// Protected route for testing authentication
app.get('/protected', (req, res) => {
  res.json({ message: 'This is a protected route' });
});

app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

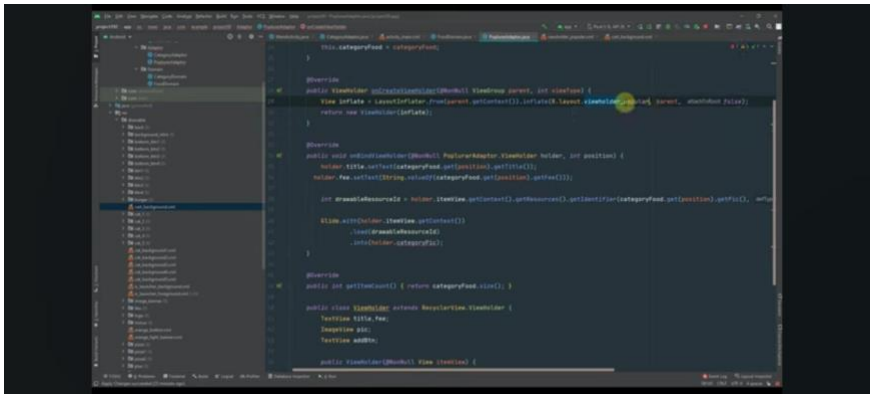
Adding food items by admin:

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

class FoodItem {
    private String name;
    private double price;

    public FoodItem(String name, double price) {
```

```
        this.name = name;
        this.price = price;
    }

    public String getName() {
        return name;
    }

    public double getPrice() {
        return price;
    }
}

public class AdminFoodMenu {
    public static void main(String[] args) {
        List<FoodItem> menu = new ArrayList<>();
        Scanner scanner = new Scanner(System.in);

        System.out.println("Welcome, Admin!");

        while (true) {
            System.out.println("Choose an option:");
            System.out.println("1. Add Food Item");
            System.out.println("2. View Menu");
            System.out.println("3. Exit");

            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume the newline character

            switch (choice) {
                case 1:
```

```
                System.out.print("Enter the food item name: ");
                String name = scanner.nextLine();

                System.out.print("Enter the price: ");
                double price = scanner.nextDouble();
                scanner.nextLine(); // Consume the newline character

                FoodItem foodItem = new FoodItem(name, price);
                menu.add(foodItem);
                System.out.println(name + " has been added to the menu.");
                break;

            case 2:
                System.out.println("Menu:");
                int itemNumber = 1;
                for (FoodItem item : menu) {
                        System.out.println(itemNumber + ". " + item.getName() + " - $" +
item.getPrice());
                    itemNumber++;
                }
                break;

            case 3:
                System.out.println("Goodbye, Admin!");
                scanner.close();
                System.exit(0);

            default:
                System.out.println("Invalid choice. Please select a valid option.");
          }
        }
```

```
    }
}
```



Listing food items and ordering them:

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

class FoodItem {
    private String name;
    private double price;

    public FoodItem(String name, double price) {
        this.name = name;
        this.price = price;
    }

    public String getName() {
        return name;
    }

    public double getPrice() {
        return price;
```

```
    }
}

public class FoodOrderingApp {
    public static void main(String[] args) {
        List<FoodItem> menu = new ArrayList<>();
        menu.add(new FoodItem("Pizza", 10.99));
        menu.add(new FoodItem("Burger", 5.99));
        menu.add(new FoodItem("Sushi", 12.99));
        menu.add(new FoodItem("Salad", 7.99));

        Scanner scanner = new Scanner(System.in);
        List<FoodItem> cart = new ArrayList();
        double totalCost = 0.0;

        System.out.println("Welcome to the Food Ordering App!");
        boolean isOrdering = true;

        while (isOrdering) {
            System.out.println("Menu:");
            for (int i = 0; i < menu.size(); i++) {
                FoodItem foodItem = menu.get(i);
                System.out.println((i + 1) + ". " + foodItem.getName() + " - $" + foodItem.getPrice());
            }
            System.out.println("Enter the item number to add to your cart (0 to finish ordering):");
            int choice = scanner.nextInt();

            if (choice == 0) {
                isOrdering = false;
            } else if (choice >= 1 && choice <= menu.size()) {
                FoodItem selectedFood = menu.get(choice - 1);
```

```
            cart.add(selectedFood);
            totalCost += selectedFood.getPrice();
            System.out.println(selectedFood.getName() + " added to your cart.");
        } else {
            System.out.println("Invalid choice. Please select a valid item.");
        }
    }

    System.out.println("Your Order:");
    for (FoodItem item : cart) {
        System.out.println(item.getName() + " - $" + item.getPrice());
    }

    System.out.println("Total Cost: $" + totalCost);
    System.out.println("Thank you for using the Food Ordering App!");

    scanner.close();
  }
}
```

CONCLUSION :

Nowadays, the traditional way of going to a restaurant and eating has reduced considerably. It's a new age where technology dominates human life. With the software and technological devices, exceptions are reduced and even terminated. Also, people prefer easy, quick and safe access to everything. This project is designed to meet the requirements of a restaurant.

The Online Food Ordering System provides a simple way to store details of the customer. food items available and to generate the bill. It is an interface that allows the customer to order the desired food which he/she can relish within a span of forty-five minutes.

Faculty Evaluator Mail Id: danipackiaseeli@mtec.ac.in