DOMAIN NAME: CLOUD APPLICATION DEVELOPMENT

PROJECT NAME: E-COMMERCE APPLICATION ON IBM CLOUD FOUNDRY

PHASE : 3

DEVELOPMENT PART-1

Problem Statement:

Begin building the artisanal e-commerce platform on IBM Cloud Foundry.

Design the platform layout and create a database to store product information.

STEP 1: Building on E-commerce platform on IBM Cloud Foundry

1. Plan Your Application: - Define the scope and features of your eCommerce application. - Determine the technologies and tools you'll use, such as programming languages, databases, and frameworks.
2. Set Up an IBM Cloud Account: - If you don't already have one, sign up for an IBM Cloud account.

3. Create a Cloud Foundry Space: - Log in to your IBM Cloud account and create a Cloud Foundry space to host your application.

4. Develop Your Application: - Write the code for your eCommerce application. - Use frameworks like Node.js, Ruby on Rails, or Java, and integrate with databases for storing product and customer data.

5. Database Setup: - Choose a database service on IBM Cloud, such as IBM Db2 or Cloudant, and configure it for your application's data storage.

6. Application Testing: - Test your eCommerce application to ensure it works as expected.

7. Containerization (Optional): - If you prefer using containers, create a Docker image of your application.

8. Deploy to IBM Cloud Foundry: - Use the IBM Cloud CLI or web interface to deploy your application to the Cloud Foundry space you created.

9. Configure Environment Variables: - Set up environment variables for your application, including database credentials, API keys, and other configuration details.

10. Secure Your Application: - Implement security measures to protect sensitive customer data, such as using HTTPS, authentication, and encryption.

11. Scaling and Load Balancing: - Configure auto-scaling and load balancing to handle traffic fluctuations.

12. Monitoring and Logging: - Set up monitoring and logging to keep track of the application's performance and troubleshoot issues.

13. Backup and Recovery: - Implement backup and recovery strategies to safeguard your data in case of failures.

14. Continuous Integration/Continuous Deployment (CI/CD): - Set up CI/CD pipelines to automate application updates and deployments.

15. Optimize for Production: - Tune your application for production-level performance, considering factors like caching, content delivery, and database optimization.

16. Domain and DNS Configuration: - Configure custom domains and DNS settings for your eCommerce site.

17. Compliance and Regulations: - Ensure your application complies with relevant regulations, especially if you handle customer data.

18. Launch and Marketing: - Once everything is set up and tested, officially launch your eCommerce application.

19. Monitor and Maintain: - Continuously monitor your application's performance and security, and regularly update it with new features and improvements.

20. Customer Support: - Offer customer support to address inquiries, issues, and feedback.

STEP 2: Design the Platform Layout.

1.Create a table to store the platform layout data. This table should include columns for the following:

- platform_id: A unique identifier for the platform.
- platform_name: The name of the platform.
- platform_type: The type of platform (e.g., web, mobile, etc.).
- platform_version: The version of the platform.
- platform_layout: The JSON representation of the platform layout.

2.Insert the platform layout data into the table. The JSON representation of the platform layout should include the following information:

- The position of each element on the platform.
- The size of each element on the platform.
- The type of each element on the platform.

3.Query the table to retrieve the platform layout data for the desired platform. The following SQL query can be used to retrieve the platform layout data for a platform :

Here is an example of a SQL code to design the platform layout:

```
CREATE TABLE platform_layout_table (
  platform_id INT NOT NULL AUTO_INCREMENT,
  platform_name VARCHAR(255) NOT NULL,
  platform_type VARCHAR(255) NOT NULL,
  platform_version VARCHAR(255) NOT NULL,
  platform_layout JSON NOT NULL,
  PRIMARY KEY (platform_id)
);
 INSERT INTO platform_layout_table (platform_name, platform_type, platform_version,
platform_layout)
VALUES ('Web Platform', 'Web', '1.0', '{
  "elements": [
    { "type": "Header",
      "position": {
        "x": 0,
        "y": 0
      },"size": {
        "width": 100,
        "height": 50
      }
    },
    {
      "type": "Content",
      "position": {
        "x": 0,
        "y": 50
```

```
            },
            "size": {
                "width": 100,
                "height": 500
            }
        }
    ]
  }');
SELECT platform_layout
  FROM platform_layout_table
  WHERE platform_id = 1;
```

The output of the query will be the following JSON representation of the platform layout:

```
{
    "elements": [
        {  "type": "Header",
            "position": {
                "x": 0,
                "y": 0
            },
            "size": {
                "width": 100,
                "height": 50
            }
        },
        {
```

```
        "type": "Content",
        "position": {
          "x": 0,
          "y": 50
        },
        "size": {
          "width": 100,
          "height": 500
        }
      }
  ]
}
```

STEP 3: Create a database to store product information.

1. Choose a database management system (DBMS). Some popular DBMSs include MySQL, PostgreSQL, and Microsoft SQL Server.

2. Create a database schema. This is a blueprint for your database and defines the tables and columns that you will use to store your data. For a product database, you might want to include tables for products, categories, and product attributes.

3. Create your tables. Once you have a database schema, you can create your tables using the SQL CREATE TABLE statement.

4. Import your product data. You can import your product data into your database using a variety of methods, such as a CSV file or a SQL INSERT statement.

5. Create relationships between your tables. This will allow you to query your data in more complex ways. For example, you might want to create a relationship between the products table and the categories table so that you can easily find all of the products in a particular category.

Here is an example of a database schema for a product database

```
CREATE TABLE products (
  product_id INT NOT NULL AUTO_INCREMENT,
  product_name VARCHAR(255) NOT NULL,
  category_id INT NOT NULL,

  price DECIMAL(10,2) NOT NULL,

  description TEXT NULL,

  PRIMARY KEY (product_id),

  FOREIGN KEY (category_id) REFERENCES categories(category_id)

);


CREATE TABLE categories (

  category_id INT NOT NULL AUTO_INCREMENT,

  category_name VARCHAR(255) NOT NULL,

  PRIMARY KEY (category_id)

);


CREATE TABLE product_attributes (

  product_attribute_id INT NOT NULL AUTO_INCREMENT,

  product_id INT NOT NULL,

  attribute_name VARCHAR(255) NOT NULL,

  attribute_value TEXT NULL,

  PRIMARY KEY (product_attribute_id),

  FOREIGN KEY (product_id) REFERENCES products(product_id)

);
```

This schema creates three tables:

- Products: This table stores the basic information about each product, such as its name, category, price, and description.
- Categories: This table stores the names of all of the product categories.
- Product attributes: This table stores the values of the different attributes for each product, such as size, color, and weight.
- Once you have created your database schema and imported your product data, you can start using your database to store and manage your product information.

some tips for creating a database to store product information:

- Normalize your data. This means organizing your data into tables in a way that minimizes redundancy and makes it easy to maintain.
- Use foreign keys to create relationships between your tables. This will allow you to query your data in more complex ways.
- Create indexes on your tables. This will improve the performance of your database queries.
- Back up your database regularly. This will protect your data in case of a hardware failure or other disaster.