

Quiz Questions

(1) Mono vs Micro Q1 – Microservices

← Classroom

Microservice
True/False

?

Problem Submissions Doubts

✓ Mono vs Micro Q1 - Microservices - Spring boot

Easy • Score 20/20

Send feedback

Problem statement

A sizeable multinational company plans to develop a comprehensive employee management system to streamline HR processes. The system will include employee profiles, payroll management, leave management, and performance evaluation modules. The company is considering two architectural approaches: microservices and monolithic architecture.
Is the given statement true/false?
"If there are different modules and their interactions, the company should adopt microservices architecture as it is more suitable for the employee management system"

Options: Pick one correct answer from below

☒ True

☐ False

Solution description

The microservices architecture allows for independent development and deployment of each module, providing flexibility and modularity in managing the HR processes.

(2) Mono vs Micro Q2 – Microservices

← Classroom

Microservice
True/False

?

Problem Submissions Doubts

✓ Mono vs Micro Q2 - Microservices - Spring boot

Easy • Score 20/20

Send feedback

Problem statement

Is the given statement true/false for the scenario stated in the previous question?
"In terms of complexity, the microservices architecture will likely have a lower level of complexity for the employee management system if the company has a small team of developers working on the application and the application is not intended to be scalable."

Options: Pick one correct answer from below

☐ True

☒ False

Solution description

With a few developers and a non-scalable application, the microservices architecture is likely to have a higher level of complexity. Managing multiple services with their databases, communication mechanisms, and coordination adds complexity compared to a monolithic architecture with a more straightforward structure with all components within a single codebase.

(3) Mono vs Micro Q3 – Microservices

← Classroom

Microservice
True/False

?

Problem Submissions Doubts

✓ Mono vs Micro Q3 - Microservices - Spring boot

Easy • Score 20/20

Send feedback

Problem statement

Is the given statement true/false?
"The monolithic architecture ensures that in the event of a failure in any application, the failure is isolated and does not impact other modules."

Options: Pick one correct answer from below

☐ True

☒ False

Solution description

In a monolithic architecture, a failure in one module could impact the entire system. The microservices architecture provides better fault isolation, ensuring that failures in one module do not affect the others.

(4) Mono vs Micro Q4 – Microservices

The screenshot shows a Classroom interface for a problem titled "Mono vs Micro Q4 - Microservices - Spring boot". The problem is marked as "Easy" with a score of 20/20. The problem statement asks: "Is the given statement true/false? 'The microservices architecture allows less efficient communication and interaction between the different modules in Web Applications.'" The options are "True" and "False", with "False" selected and marked correct. The solution description states: "The microservices architecture allows more efficient communication and interaction between the modules. Each module can communicate via well-defined APIs, facilitating better coordination and flexibility."

(5) Travel booking platform

The screenshot shows a Classroom interface for a problem titled "Travel booking platform - Microservices - Spring Boot". The problem is marked as "Easy" with a score of 20/20. The problem statement asks: "A travel booking platform uses the Microservice architecture where each module (flight booking, hotel reservation, car rental) becomes a separate microservice with its own SQL or NoSQL database based on its needs. What design principle is being demonstrated in this case?" The options are "Single Responsibility Principle", "Loose Coupling", "Autonomy and Decentralization", and "Polyglot Persistence", with "Polyglot Persistence" selected and marked correct. The solution description states: "Polyglot Persistence uses different database technologies to store data based on the specific needs of each microservice. In this scenario, utilising a NoSQL database for the flight booking microservice and a relational database for the hotel reservation microservice demonstrates the principle of Polyglot Persistence."

(6) Car Rental Service

The screenshot shows a Classroom interface for a problem titled "Car Rental Service". The problem is marked as "Easy" with a score of 20/20. The problem statement asks: "Suppose the car rental microservice in the previous question exposes a set of well-defined APIs for other microservices to interact with it. For allowing independent development and deployment of the other microservices. Which design principle is being demonstrated here?" The options are "Single Responsibility Principle", "Loose Coupling", "Autonomy and Decentralization", and "Polyglot Persistence", with "Loose Coupling" selected and marked correct. The solution description states: "By exposing well-defined APIs, the car rental microservice lets other microservices interact with it decoupled. This promotes loose coupling, as each microservice can be developed and deployed independently without depending on the internal implementation details of the car rental microservice."

(7) Flight Booking Service – Microservices

← Classroom

Microservice
Flight Booking Service

?

Problem Submissions Doubts

Flight Booking Service - Microservices - Spring Boot

Easy Score 20/20

Send feedback

Problem statement

If the flight booking microservice mentioned previously incorporates fault tolerance mechanisms to handle service failures gracefully and maintain the system's overall stability. Which design principle does this align with?

Options: Pick one correct answer from below

☐ Single Responsibility Principle

☐ Loose Coupling

☐ Autonomy and Decentralization

☒ Resilience and Fault Isolation

Solution description

The flight booking microservice demonstrates the design principle of resilience and fault isolation by implementing fault tolerance mechanisms. It ensures that failures within the microservice do not disrupt the system's overall stability and provides graceful handling of errors.

(8) Designing Principles – Microservices

← Classroom

Microservice
Choose Designing Principles

?

Problem Submissions Doubts

Designing Principles - Microservices - Spring Boot

Easy Score 20/20

Send feedback

Problem statement

Which design principles is/are being demonstrated based on the statements below?
Statement 1: The hotel reservation service stated previously can handle a high volume of requests and scale horizontally by adding more instances to meet the increased demand.
Statement 2: The travel booking platform has a continuous deployment approach, enabling rapid and frequent updates to individual microservices without impacting the entire system.
This question may have multiple answers

Options: One or more answers may be correct

☐ Single Responsibility Principle

☐ Loose Coupling

☒ Continuous Deployment

☒ Scalability

Solution description

- The ability of the hotel reservation microservice to handle a high volume of requests and scale horizontally by adding more instances aligns with the design principle of scalability. It allows the microservice to accommodate increased demand and maintain performance without being a bottleneck.
- Embracing continuous deployment practices enables the platform to update individual microservices independently and frequently. This aligns with the design principle of continuous deployment, which promotes faster delivery of changes without impacting the entire system.

(9) Food Delivery Application I

← Classroom

Microservice
Food Delivery Application I

?

Problem Submissions Doubts

Problem statement

Scenario: A developer is creating a Food Delivery Application which has two services associated with it, UserService (Handling user profiles and data) and RestaurantService (Managing restaurant information, menu items, and orders). However, during the compilation time, the userService gives the error message "cannot find the method update()".
Based on the codes given below for the UserService, identify the correct implementation of the updateUser() method in the UserService to update a user's information.

```
***
// User Controller
@RestController
@RequestMapping("/users")
public class UserController {
    @Autowired
    private UserService userService;
    @PostMapping("/")
    public User create(@RequestBody User user) {
        return userService.createUser(user);
    }
    @PostMapping("/{id}")
    public User updateUser(@PathVariable Long id, @RequestBody User updatedUser) {
        return userService.updateUser(id, updatedUser);
    }
}
```

```
***
// User Service
public class UserService {
```

☐

```
***
public User updateUser(Long id, User updatedUser) {
    return userRepository.save(id, updatedUser);
}
```

☒

```
***
public User updateUser(Long id, User updatedUser) {
    User user = userRepository.findById(id).orElse(null);
    if (user != null) {
        user = User.builder()
            .name(updatedUser.getName())
            .email(updatedUser.getEmail())
            .build();
        return userRepository.save(user);
    }
    return null;
}
```

Solution description

- We first need to find if the user is already in the database to update the user's information. We use the findById() method to find the user through the user id. If the user is present, we update its details using the builder() method and then save it to the database.
- Both Options B and D can be used. However, it's not advisable to change the id of an existing entity (user, in this case) during an update, as in Option B. The id is typically a unique identifier and should remain constant throughout the entity's lifecycle. Changing the id could lead to unexpected behaviour or data inconsistencies.
- Therefore, Option D is a safer and more appropriate implementation for updating a user's information. It only updates the attributes that are allowed to change (name and email) and does not modify the id. This way, the user retains the same identity in the system.

(10) Food Delivery Application II

← Classroom

Microservice
Food Delivery Application II

Problem Submissions Doubts

Food Delivery Application II - Microservices - Spring Boot

Easy • Score: 20/20

Send feedback

Problem statement

In continuation to the previous problem, when the developer tests the RestaurantService and calls the API to create a new restaurant, he receives the following error:
"org.springframework.web.bind.MissingPathVariableException"
Choose the correct implementation of the createRestaurant() method in the Restaurant Controller.

```
***  
// Restaurant Entity  
@Entity  
public class Restaurant {  
    private String id;  
    private String name;  
    private String address;  
    private String workingHours;  
}
```

```
***  
// Order Entity  
@Entity  
public class Order {  
    private String id;  
    private String userId;  
    private String dishId;  
    private String quantity;  
}
```

Options: Pick one correct answer from below

☐

```
***  
@PathVariable("/{restaurantId"})  
public Restaurant createRestaurant(@PathVariable({@PathVariable Restaurant restaurant}) {  
    return restaurantService.createRestaurant(restaurant);  
}
```

☐

```
***  
@PathVariable("/{")  
public Restaurant createRestaurant(@PathVariable({@PathVariable Restaurant restaurant}) {  
    return restaurantService.save(restaurant);  
}
```

☒

```
***  
@PathVariable("/{")  
public Restaurant createRestaurant(@PathVariable({@PathVariable Restaurant restaurant}) {  
    return restaurantService.createRestaurant(restaurant);  
}
```

☐ None of the above

Solution description

Sending an entity as a PathVariable is generally not considered good practice in RESTful API design. The PathVariable is a part of the URL, used to extract values from the URL path, and it is commonly used for simple scalar data types like integers, strings, or other primitive types. Sending an entire entity as a PathVariable goes against some fundamental principles of RESTful API design.

```
***  
// Restaurant Controller  
@RestController  
@RequestMapping("/restaurant")  
public class RestaurantController {  
    @Autowired  
    private RestaurantService restaurantService;  
    @PostMapping("/{")  
    public Restaurant createRestaurant(@PathVariable Restaurant restaurant) {  
        return restaurantService.createRestaurant(restaurant);  
    }  
    @PostMapping("/{placeOrder}")  
    public OrderResponse placeOrder(@RequestBody OrderRequest request) {  
        return restaurantService.placeOrder(request);  
    }  
}
```

```
***  
// Restaurant Service  
@Service  
public class RestaurantService {  
    @Autowired  
    private RestaurantRepository restaurantRepository;  
    @Autowired  
    private OrderRepository orderRepository;  
    public Restaurant createRestaurant(Restaurant restaurant) {  
        return restaurantRepository.save(restaurant);  
    }  
    public OrderResponse placeOrder(OrderRequest request) {  
        try {  
            orderRepository.save(order);  
        } catch (Exception e) {  
            return new OrderResponse("Error");  
        }  
        return new OrderResponse("Order Placed successfully")  
    }  
}
```

(11) Food Delivery Application III

← Classroom

Microservice
Food Delivery Application III

Problem Submissions Doubts

easy 20/20

Problem statement

Send feedback

As mentioned, there has been a new requirement for the Food Delivery Application. Whenever the user places an order, he should get an order confirmation/rejection message of the order.
Below are codes for OrderRequest, OrderResponse classes, and User Controller for placing and confirming the orders. Choose the correct implementation of placeOrder() method in the User Service.

```
***
// Order Request Entity
@Data
public class OrderRequest {
    private String userId;
    private String dishId;
    private String quantity;
}

// Order Response
@Data
public class OrderResponse {
    private String message;
}
```

```
***
@RestController
@RequestMapping("/orders")
public class UserController {

    @Autowired
    private UserService userService;

    @PostMapping("/user/{id}/place-order")
    public ResponseEntity<Order> placeOrder(@PathVariable String userId, @RequestBody
    OrderRequest orderRequest) {
        OrderResponse response = userService.placeOrder(userId, orderRequest);
        return response;
    }
}
```

Options: Pick one correct answer from below

☒

```
***
public class UserService {

    private final RestTemplate restTemplate;

    public UserService(RestTemplateBuilder restTemplateBuilder) {
        this.restTemplate = restTemplateBuilder.build();
    }

    public OrderResponse placeOrder(String userId, OrderRequest orderRequest) {
        String url = "http://restaurant-service/";
        orderRequest.setUserId(userId);
        orderRequest.setDishId("restaurant-placeOrder");
        ResponseEntity<OrderResponse> response = restTemplate.exchange(
            url, HttpMethod.POST,
            new RequestEntity<OrderRequest>(orderRequest, HttpHeaders.EMPTY),
            OrderResponse.class,
            orderRequest);
        return response;
    }
}
```

☐

```
***
public class UserService {

    private final RestTemplate restTemplate;

    public UserService(RestTemplateBuilder restTemplateBuilder) {
        this.restTemplate = restTemplateBuilder.build();
    }

    public OrderResponse placeOrder(String userId, OrderRequest orderRequest) {
        String url = "http://restaurant-service/";
        orderRequest.setUserId(userId);
        orderRequest.setDishId("restaurant-placeOrder");
        ResponseEntity<OrderResponse> response = restTemplate.exchange(
            url, HttpMethod.POST,
            new RequestEntity<OrderRequest>(orderRequest, HttpHeaders.EMPTY),
            OrderResponse.class,
            orderRequest);
        return response;
    }
}
```

☐

```
***
public class UserService {

    private final RestTemplate restTemplate;

    public UserService(RestTemplateBuilder restTemplateBuilder) {
        this.restTemplate = restTemplateBuilder.build();
    }

    public OrderResponse placeOrder(String userId, OrderRequest orderRequest) {
        String url = "http://restaurant-service/";
        orderRequest.setUserId(userId);
        orderRequest.setDishId("restaurant-placeOrder");
        ResponseEntity<OrderResponse> response = restTemplate.exchange(
            url, HttpMethod.POST,
            new RequestEntity<OrderRequest>(orderRequest, HttpHeaders.EMPTY),
            OrderResponse.class,
            orderRequest);
        return response;
    }
}
```

(12) Statement Regarding Eureka Server

← Classroom

Microservice
Statements Regarding Eureka

110% - + Reset

Problem Submissions Doubts

easy 20/20

Statement Regarding Eureka Server - Microservices - Spring boot

Send feedback

Which of the following statements about Eureka Server is true?

Options: Pick one correct answer from below

☐ Eureka Server is primarily used for managing user authentication.

☒ Eureka Server is part of the Spring Cloud ecosystem for microservices.

☐ Eureka Server is designed for complex database queries.

☐ Eureka Server doesn't support service registration and discovery.

Solution description

The Eureka Server is an integral component of the Spring Cloud ecosystem that addresses the challenges of service registration and discovery in microservices architectures. It allows services to dynamically register and discover each other, leading to efficient communication and improved overall system reliability.

(13) Client in Eureka Server – Microservice

← Classroom

Microservice
Eureka Server Client

Problem Submissions Doubts

Client in Eureka Server - Microservices - Spring boot

Easy • Score 20/20

Send feedback

Problem statement

What does the term "Client" refer to in the context of a Eureka Server?

Options: Pick one correct answer from below

☒ The microservice that registers itself with the Eureka Server for service discovery.

☐ The Eureka Server itself.

☐ An external user trying to access a microservice.

☐ The database used by the Eureka Server to store service information.

Solution description

In a Eureka Server setup, the term "Client" refers to the microservice that registers itself with the Eureka Server. This registration allows the Eureka Server to maintain a registry of available services and enables service discovery for other clients. This concept is fundamental in microservices architecture to enable efficient communication and interaction between services.

(14) Circuit Breaker in Food Delivery Application

← Classroom

Microservice
Food Delivery Application IV

80% — + Reset

Problem Submissions Doubts

Circuit Breaker in Food Delivery application - Microservices - Spring Boot

Easy • Score 20/20

Send feedback

Problem statement

Choose the correct implementation of Circuit Breaker for the placeOrder API discussed in the previous question.

Options: Pick one correct answer from below

☐

```
***
@RestController
@RequestMapping("/orders")
public class OrderController {
    @Autowired
    private UserService userService;
    @RequestMapping("placeOrder")
    public ResponseEntity<String> placeOrder(@RequestBody OrderRequest orderRequest) {
        String response = userService.placeOrder(orderRequest);
        return ResponseEntity.ok(response);
    }
}
```

☐

```
***
@RestController
@RequestMapping("/orders")
public class OrderController {
    @Autowired
    private UserService userService;
    @RequestMapping("placeOrder")
    public ResponseEntity<String> placeOrder(@RequestBody OrderRequest orderRequest) {
        String response = userService.placeOrder(orderRequest);
        return ResponseEntity.ok(response);
    }
    public String placeOrderFallback(@RequestBody OrderRequest orderRequest) {
        return "Restaurant Service Unavailable";
    }
}
```

☒

```
***
@RestController
@RequestMapping("/orders")
public class OrderController {
    @Autowired
    private UserService userService;
    @RequestMapping("placeOrder")
    public ResponseEntity<String> placeOrder(@RequestBody OrderRequest orderRequest) {
        String response = userService.placeOrder(orderRequest);
        return ResponseEntity.ok(response);
    }
    public String placeOrderFallback(@RequestBody OrderRequest orderRequest) {
        return "Restaurant Service Unavailable";
    }
}
```

(15) Routing using Eureka

←

Classroom

Microservice

Routing Through Eureka

Problem

Submissions

Doubts

✔ Routing using Eureka

Easy • Score: 20/20

Problem statement

Send feedback

In a microservices architecture, what role does Eureka play in routing?

Options: Pick one correct answer from below

☐ Eureka acts as a reverse proxy for all incoming requests.

☐ Eureka routes requests based on predefined routing rules.

☒ Eureka dynamically discovers and routes requests to appropriate microservices.

☐ Eureka generates API endpoints for microservices communication.

Solution description

Eureka's role in routing involves dynamically discovering available microservice instances, maintaining an up-to-date registry, and assisting in load balancing to route incoming requests to the most suitable instances. This contributes to efficient and reliable communication between microservices in a dynamic and distributed environment.

(16) Feign Client Implementation Microservices

←

Classroom

Problem

Submissions

Details

Microservice

Food Delivery Application V

Problem statement

Send feedback

Continuing the placeOrder API, implemented for the food Delivery Application using RestTemplate. Now you have to implement the API using Feign Client. Below are codes for Restaurant Controller and User Service classes.

```
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/restaurant")
public class RestaurantController {

    @Autowired
    private RestaurantService restaurantService;

    @PostMapping("/placeOrder")
    public ResponseEntity<Order> placeOrder(@PathVariable String user_id,
        @RequestBody OrderRequest orderRequest) {
        OrderResponse response = restaurantService.placeOrder(id, orderRequest);
        return response;
    }
}
```

```
import org.springframework.stereotype.Service;

@Service
public class UserService {

    @Autowired
    private RestaurantService restaurantService;

    public ResponseEntity<OrderResponse> placeOrder(String user_id, OrderRequest orderRequest) {
        return restaurantService.placeOrder(user_id, orderRequest);
    }
}
```

Options: Pick one correct answer from below

☐

```
import org.springframework.cloud.openfeign.*;

@FeignClient(name = "restaurant-service")
public class RestaurantService {

    @PostMapping("/restaurant/placeOrder")
    ResponseEntity<Order> placeOrder(@PathVariable String user_id,
        @RequestBody OrderRequest orderRequest);
}
```

☐

```
import org.springframework.cloud.openfeign.*;

@FeignClient(name = "restaurant-service")
public interface RestaurantService {

    @PostMapping("/restaurant/placeOrder")
    ResponseEntity<Order> placeOrder(@PathVariable String user_id,
        @RequestBody OrderRequest orderRequest);
}
```

☒

```
import org.springframework.cloud.openfeign.*;

@FeignClient(name = "restaurant-service")
public interface RestaurantService {

    @PostMapping("/restaurant/placeOrder")
    ResponseEntity<OrderResponse> placeOrder(@PathVariable String user_id,
        @RequestBody OrderRequest orderRequest);
}
```

☐ None of the above

Solution description

For the Feign Client implementation, first, we have to create an interface with Feign Client name same as the service name to whom we want to make the API call. Secondly, the method declaration in the interface should be the same as the API controller where we will make an API call. Therefore option A is correct. ##### In option B, the RestaurantService is made a class instead of an interface. ##### In option C, the controller method is declared incorrectly.

(17) Load Balancing in API Gateway

←

CC

Classroom

Microservice

Load Balancing

Problem

Submissions

Doubts

✔

Load Balancing in API Gateway

Easy • Score 26/20

Send feedback

What is the primary purpose of load balancing in the context of an API gateway?

Options: Pick one correct answer from below

☐ To manage microservices versioning.

☐ To secure communication between microservices.

☐ To optimize database queries for microservices.

☒ To distribute incoming API requests across multiple service instances.

Solution description

In a microservices architecture, an API gateway is the entry point for client applications to interact with the various microservices. Load balancing is an API gateway's fundamental technique to ensure efficient and reliable distribution of incoming API requests to multiple instances of the same microservice.