Staff TeamLead Technical LiveCoding Test

# Pre-requisites

The candidate is expected to have access to a computer, email, telephone/zoom account (in order to speak to the organiser should the need arise) and suitable software for developing .NET or .NET Core applications in C# -.

# Instructions

You are expected to work on this task on your own without direct help and advice from others, although you may use online sources freely.

# Assignment

We are developing an ecommerce solution. As part of the system, a team of front-end developers are developing an app to inform customers about when they should expect the delivery of their most recent order and a list of customers with their respective orders for the Customer Service team.

You are provided with an implementation made by a team member. You are tasked with reviewing the implementation and make changes to make it production ready.

## GetLatestOrder endpoint

It accepts a HTTP POST request with the customer’s email address and customer ID in JSON in the body, like this:

{

"user": "johndoe@cf.co.uk",

"customerId": "CUST-001"

}

And it returns the details of the most recent order in JSON like this:

{

"customer": {

"firstName": "John",

"lastName": "Doe"

},

{

"orderNumber": "ORD-001",

"orderDate": "07-December-2024",

"deliveryAddress": "1A Lansdowne Way, London, UK",

"deliveryExpected": "14-December-2024",

"orderItems": [

{

"product": "Classic White T-Shirt",

"quantity": 2,

"price": 25.99

},

{

"product": "Denim Jeans",

"quantity": 1,

"price": 79.99

}

]

},

}

The order data should be the most recent order placed by the customer (for clarity, in this case “most recent” means the order with the latest order date, even if that order date is in the future).

The information about the orders is contained in the provided SQL database. This should be combined with the customer detail information which is returned from the provided API.

## GetCustomersWithOrders

It accepts a HTTP GET request

And it returns a list of customers with their orders in JSON like this:

{

"customers": [

{

"customerId": "CUST-001",

"firstName": "John",

"lastName": "Doe",

"orders": [

{

"orderNumber": "ORD-001",

"orderDate": "07-December-2024",

"deliveryAddress": "1A Lansdowne Way, London, UK",

"deliveryExpected": "14-December-2024",

"orderItems": [

{

"product": "Classic White T-Shirt",

"quantity": 2,

"price": 25.99

},

{

"product": "Denim Jeans",

"quantity": 1,

"price": 79.99

}

]

},

{

"orderNumber": "ORD-002",

"orderDate": "10-December-2024",

"deliveryAddress": "1A Lansdowne Way, London, UK",

"deliveryExpected": "15-December-2024",

"orderItems": [

{

"product": "Gift",

"quantity": 1,

"price": 129.99

}

]

}

]

}

]

}

# Restrictions

* Where the user's email address does not match the customer number, you should treat this as an invalid request and return an appropriate response.
* Where the user does not have an order, you should return a successful response with the user details and leave the order content blank.
* Orders which are marked as "contains a gift" should return "Gift" in place of the product name.

# Customer Details API

The API for retrieving the customer details.

it requires an API key to be passed in a querystring parameter called “code”. You will find the API key at the end of this document.

The customer details API accepts the customer email as input, either in a querystring parameter called “email” in a HTTP GET, or as JSON in a HTTP POST:

{ "email": "johndoe@cf.co.uk" }

And returns the customer details in this form:

{

"email": "johndoe@cf.co.uk",

"customerId": "CUST-001",

"website": true,

"firstName": "John",

"lastName": "Doe",

"lastLoggedIn": "03-May-2024 09:15",

"houseNumber": "1a",

"street": "Landsdowne Way",

"town": "London",

"postcode": "SW8 2ED",

"preferredLanguage": "en-gb"

}

Where the email address supplied is not a valid customer in the system, the API will return a 404 status.

Valid email addresses in the test data are:

johndoe@cf.co.uk (ID= CUST-001)

# General considerations

Your code should be well-structured and make appropriate use of testing, comments, error handling, security and production readiness

# Connection Details

The following information is sensitive, and you should take care to keep it safe.

The API Key for the Customer Details API is:

uu2ToG/dcsg3DI8CGlpLro1PyLhZNUWHpdPv8VmWFLBaxM0fvUZvkA==

# What you need to do

You should spend approximately 40 minutes on this task.

1. Review existing code, point out any potential bugs and make changes that meet the specification, retrieving the order data from the database and provided customer API.
2. What additional tasks might need to be performed before your API could be deployed in a production environment?
3. Consider any changes that you might have made if you had more time, or changes you might make to the specifications or implementation to improve it.

# Existing Bugs and Issues:

## 1. API Client Implementation Issue

A screen shot of a computer code

Description automatically generated

- Creating new HttpClient instances instead of using the injected one, which can cause socket exhaustion

## 2. N+1 Query Problem in CustomerRepository

A computer screen shot of a program

Description automatically generated

- Inefficient querying causing multiple database hits

## 3. Incorrect value of OrderItem

A screen shot of a computer code

Description automatically generated

- Requires a check if the order contains a gift then set the product as “Gift” otherwise ProductName

## 4. Missing Validation in OrderRequest

A screenshot of a computer code

Description automatically generated

- User field validation is commented out

## 5. Security Issue in Exception Handler

A computer screen shot of a program

Description automatically generated

- Exposing sensitive error information

## 6. Incorrect Order Modification in Repository

A computer screen shot of a code

Description automatically generated

- Hardcoding ContainsGift value instead of using database value

## 7. Moving uri to Settings

A screenshot of a computer program

Description automatically generated

# Production Readiness

## 1. API Authentication & Authorization

- Implement authentication and authorization mechanisms to control access to API endpoints based on user roles or permissions.

### a) Implement JWT Authentication:

A computer screen shot of text

Description automatically generated

### b) Add Authorization Policies:

A screenshot of a computer program

Description automatically generated

## 2. SQL Server Database Handling

- Implement Sql Server instead of inmemory sqlserver with proper indexes, database connection resilience and query timeouts

A screen shot of a computer code

Description automatically generated

## 3. Unit tests

- Add appropriate unit tests for

* GetCustomersWithOrdersQueryHandler
* OrdersService

## 3. Error Handling

Remove sensitive data from error response

A screen shot of a computer

Description automatically generated

## 4. Resilience Policies

- Introduce resilience policies like retry, fallback, and circuit breaker patterns to handle failures gracefully in third-party integrations and database connections.

A screen shot of a computer code

Description automatically generated

## 5. Logging Enhancement

- Implement logging mechanisms using frameworks like Serilog to capture application events, errors, and diagnostics for monitoring and troubleshooting.

### a) Configure structured logging:

A screen shot of a computer code

Description automatically generated

### b) Add correlation ID middleware:

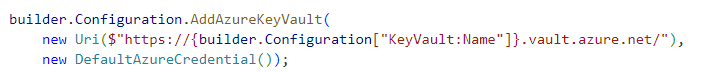
A screen shot of a computer code

Description automatically generated

## 6. Secure Configuration Management

- Securely manage sensitive information such as API keys and connection strings using environment variables or secure configuration stores.

Add Azure Key Vault integration:



## 7. Performance Optimization

- Optimize database queries, introduce caching mechanisms, and minimize network round-trips to improve application performance and scalability.

### a) Implement response caching:

A screenshot of a computer program

Description automatically generated

### b) Add database query optimization:

A computer screen shot of a program

Description automatically generated

## 8. Monitoring and Analytics Integration

- Integrate with Azure Application Insights or AWS CloudWatch to monitor application performance, track usage metrics, and gain insights into application behavior.

### a) Add Application Insights:

A computer code on a white background

Description automatically generated

### b) Add health checks:

A computer code with text

Description automatically generated with medium confidence

## 9. Additional Enhancements

### a) API Versioning:

A white background with black and white text

Description automatically generated

### b) API Pagination:

A screenshot of a computer program

Description automatically generated

### c) Rate Limiting:

A computer screen shot of a program code

Description automatically generated