Programación con IA Generativa

Documentación con IA

Índice

1 | Documentar código

#DOCUMENTAR CÓDIGO

Trabajar con código heredado puede ser un desafío para los desarrolladores, sobre todo cuando el código es complejo o no está bien documentado. En tales casos, puede ser útil usar un asistente de IA para documentarlo.

Ejemplo: Código sin documentación

COBOL

```
IDENTIFICATION DIVISION.
PROGRAM-ID. INSERT-RECORD.

ENVIRONMENT DIVISION.

DATA DIVISION.
WORKING-STORAGE SECTION.
01  WS-STATUS-FLAGS.
    05 WS-DB-STATUS     PIC X(2).
       88 WS-SUCCESS    VALUE "00".
    05 WS-SQLCODE       PIC S9(9) COMP.
    05 WS-ERROR-MSG     PIC X(50).

LINKAGE SECTION.
01  LS-PARAMETERS.
    05 LS-PERSON-RECORD.
       10 PERSON-ID     PIC 9(6).
       10 PERSON-NAME   PIC X(50).
       10 PERSON-AGE    PIC 9(3).
    05 LS-RESULT        PIC X.
       88 SUCCESS       VALUE 'T'.
       88 FAILED        VALUE 'F'.

PROCEDURE DIVISION USING LS-PARAMETERS.
    PERFORM INSERT-AND-VALIDATE
    GOBACK
    .

INSERT-AND-VALIDATE.
    EXEC SQL
        INSERT INTO persons (id, name, age)
        VALUES (:PERSON-ID, :PERSON-NAME, :PERSON-AGE)
    END-EXEC

    IF SQLCODE = 0
        EXEC SQL COMMIT END-EXEC
        SET SUCCESS TO TRUE
    ELSE
        EXEC SQL ROLLBACK END-EXEC
        SET FAILED TO TRUE
        STRING "DB Error: " SQLCODE
            DELIMITED BY SIZE
            INTO WS-ERROR-MSG
        DISPLAY WS-ERROR-MSG
    END-IF
```

Consideraciones a tener en cuenta

Podemos usar una indicación simple para pedir al asistente de IA que documente el código como por ejemplo: **Documentar este código / Documentar este código en castellano**

Si deseamos que la documentación se genere en otro formato, como **Markdown**, podemos especificar el formato de salida deseado en la indicación. Por ejemplo: **Documentar este código en Markdown**

Ejemplo: Código con documentación

COBOL

```
*****************************************************************
* Program: INSERT-RECORD
* Purpose: Inserts a person record into the database with
*          transaction management and error handling
*****************************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID. INSERT-RECORD.

 ENVIRONMENT DIVISION.

 DATA DIVISION.
 WORKING-STORAGE SECTION.
* Status flags and error handling variables
 01  WS-STATUS-FLAGS.
     05 WS-DB-STATUS     PIC X(2).             *> Database status
        88 WS-SUCCESS    VALUE "00".           *> Success flag
     05 WS-SQLCODE       PIC S9(9) COMP.       *> SQL return code
     05 WS-ERROR-MSG     PIC X(50).            *> Error message

 LINKAGE SECTION.
* Input/Output parameters
 01  LS-PARAMETERS.
     05 LS-PERSON-RECORD.
        10 PERSON-ID     PIC 9(6).             *> Person ID
        10 PERSON-NAME   PIC X(50).            *> Person name
        10 PERSON-AGE    PIC 9(3).             *> Person age
     05 LS-RESULT        PIC X.                *> Operation result
        88 SUCCESS       VALUE 'T'.            *> Success flag
        88 FAILED        VALUE 'F'.            *> Failure flag

 PROCEDURE DIVISION USING LS-PARAMETERS.
     PERFORM INSERT-AND-VALIDATE
     GOBACK
     .

*****************************************************************
* Handles database insertion with transaction management
*****************************************************************
 INSERT-AND-VALIDATE.
     EXEC SQL
         INSERT INTO persons (id, name, age)
         VALUES (:PERSON-ID, :PERSON-NAME, :PERSON-AGE)
     END-EXEC

     IF SQLCODE = 0
         EXEC SQL COMMIT END-EXEC               *> Commit if successful
         SET SUCCESS TO TRUE
     ELSE
         EXEC SQL ROLLBACK END-EXEC             *> Rollback if failed
         SET FAILED TO TRUE
         STRING "DB Error: " SQLCODE
             DELIMITED BY SIZE
             INTO WS-ERROR-MSG
         DISPLAY WS-ERROR-MSG
     END-IF
     .
```

2 | Explicar código

#EXPLICAR CÓDIGO

Uno de los mayores desafíos que plantea el código heredado es ayudar a que los desarrolladores que no están familiarizados con los lenguajes es que puedan comprenderlo. Con un asistente de IA puedes pedir una explicación.

Ejemplo: Código en COBOL que no comprendemos


```
IDENTIFICATION DIVISION.
PROGRAM-ID. INSERT-RECORD.

ENVIRONMENT DIVISION.

DATA DIVISION.
WORKING-STORAGE SECTION.
01  WS-STATUS-FLAGS.
    05 WS-DB-STATUS     PIC X(2).
       88 WS-SUCCESS    VALUE "00".
    05 WS-SQLCODE       PIC S9(9) COMP.
    05 WS-ERROR-MSG     PIC X(50).

LINKAGE SECTION.
01  LS-PARAMETERS.
    05 LS-PERSON-RECORD.
       10 PERSON-ID     PIC 9(6).
       10 PERSON-NAME   PIC X(50).
       10 PERSON-AGE    PIC 9(3).
    05 LS-RESULT        PIC X.
       88 SUCCESS       VALUE 'T'.
       88 FAILED        VALUE 'F'.

PROCEDURE DIVISION USING LS-PARAMETERS.
    PERFORM INSERT-AND-VALIDATE
    GOBACK
    .

INSERT-AND-VALIDATE.
    EXEC SQL
        INSERT INTO persons (id, name, age)
        VALUES (:PERSON-ID, :PERSON-NAME, :PERSON-AGE)
    END-EXEC

    IF SQLCODE = 0
        EXEC SQL COMMIT END-EXEC
        SET SUCCESS TO TRUE
    ELSE
        EXEC SQL ROLLBACK END-EXEC
        SET FAILED TO TRUE
        STRING "DB Error: " SQLCODE
            DELIMITED BY SIZE
            INTO WS-ERROR-MSG
        DISPLAY WS-ERROR-MSG
    END-IF
```

Consideraciones a tener en cuenta

Puedes pedir una explicación del código COBOL desde el punto de vista de un desarrollador de Python. Por ejemplo:

Soy programador Python. Por favor ayúdame a entender este código. Por favor utiliza analogías.

Ejemplo: Analogía con Python


```python
# In Python, the COBOL divisions would be like:

# IDENTIFICATION DIVISION = module/file name
# PROGRAM-ID = function name
def insert_record():
    pass

# DATA DIVISION would be like declaring these variables:
class WorkingStorage:
    db_status = ""  # WS-DB-STATUS
    sqlcode = 0     # WS-SQLCODE
    error_msg = ""  # WS-ERROR-MSG

# LINKAGE SECTION is like function parameters
class PersonRecord:
    def __init__(self, person_id: int, person_name: str, person_age: int):
        self.person_id = person_id     # PERSON-ID
        self.person_name = person_name # PERSON-NAME
        self.person_age = person_age   # PERSON-AGE
        self.success = False           # LS-RESULT
```

2 | Explicar lógica compleja

#EXPLICAR LÓGICA COMPLEJA

Puede haber ocasiones en las que necesites explicar cierta lógica o algoritmos complejos en el código. Esto puede plantear un desafío, sobre todo cuando intentas hacer que sea comprensible para los demás. Un asistende de IA puede ofrecer sugerencias sobre cómo explicar el algoritmo o la lógica de una forma clara y concisa para ayudarte con esta tarea.

Ejemplo: Lógica compleja

```
private static readonly HttpClient _client = new HttpClient();

public async Task<string> FetchDataFromApiWithRetryAsync(string apiUrl, CancellationToken cancellationToken, int maxRetries, int cancellationDelay, Label statusLabel)
{
    var retryCount = 0;
    using var cts = CancellationTokenSource.CreateLinkedTokenSource(cancellationToken);

    while (retryCount < maxRetries)
    {
        try
        {
            cts.CancelAfter(cancellationDelay);
            return await FetchDataFromApiAsync(cts.Token, statusLabel);
        }
        catch (Exception ex) when (!(ex is OperationCanceledException))
        {
            if (retryCount < maxRetries - 1) {
                retryCount++;
                int delay = (int)Math.Pow(2, retryCount) * 1000;
                await Task.Delay(delay, cancellationToken);
                UpdateStatusLabel($"Retrying ({retryCount}/{maxRetries})...", statusLabel);
            } else {
                throw new Exception($"Failed to fetch data after {maxRetries} retries: {ex.Message}", ex);
            }
        }
    }

    throw new OperationCanceledException("The operation was cancelled.", cts.Token);
}

private async Task<string> FetchDataFromApiAsync(string apiUrl, CancellationToken cancellationToken, Label statusLabel)
{
    UpdateStatusLabel("Fetching data...", statusLabel);

    HttpResponseMessage response = await _client.GetAsync(apiUrl, cancellationToken);
    response.EnsureSuccessStatusCode();
    string data = await response.Content.ReadAsStringAsync();

    UpdateStatusLabel("Data fetched.", statusLabel);
    return data;
}

private void UpdateStatusLabel(string message, Label statusLabel)
{
    if (statusLabel.InvokeRequired) {
        statusLabel.BeginInvoke(new Action<string, Label>(UpdateStatusLabel), message, statusLabel);
    } else {
        statusLabel.Text = message;
    }
}
```

Consideraciones a tener en cuenta

Podemos pedir al asistente de IA que haga que este código sea más fácil de entender a través de comentarios. Por ejemplo:

Agregar comentarios al código para que sea más entendible.

Ejemplo: Algoritmo complejo explicado

```csharp
// Shared HttpClient instance for making HTTP requests (best practice for reuse)
private static readonly HttpClient _client = new HttpClient();

/// <summary>
/// Fetches data from an API with retry mechanism and timeout functionality
/// </summary>
/// <param name="apiUrl">The URL to fetch data from</param>
/// <param name="cancellationToken">Token to cancel the operation</param>
/// <param name="maxRetries">Maximum number of retry attempts</param>
/// <param name="cancellationDelay">Timeout duration in milliseconds</param>
/// <param name="statusLabel">UI label to show status updates</param>
public async Task<string> FetchDataFromApiWithRetryAsync(string apiUrl, CancellationToken cancellationToken, int maxRetries, int cancellationDelay, Label statusLabel)
{
    var retryCount = 0;
    // Create a linked cancellation token that combines user cancellation and timeout
    using var cts = CancellationTokenSource.CreateLinkedTokenSource(cancellationToken);

    while (retryCount < maxRetries)
    {
        try
        {
            // Set timeout for this attempt
            cts.CancelAfter(cancellationDelay);
            return await FetchDataFromApiAsync(cts.Token, statusLabel);
        }
        catch (Exception ex) when (!(ex is OperationCanceledException))
        {
            // If not the last retry attempt, implement exponential backoff
            if (retryCount < maxRetries - 1) {
                retryCount++;
                // Exponential backoff: 2^retryCount seconds (1s, 2s, 4s, 8s, etc.)
                int delay = (int)Math.Pow(2, retryCount) * 1000;
                await Task.Delay(delay, cancellationToken);
                UpdateStatusLabel($"Retrying ({retryCount}/{maxRetries})...", statusLabel);
            } else {
                // If all retries failed, throw exception with details
                throw new Exception($"Failed to fetch data after {maxRetries} retries: {ex.Message}", ex);
            }
        }
    }

    // This is reached if maxRetries is 0 or some other unexpected case
    throw new OperationCanceledException("The operation was cancelled.", cts.Token);
}
```

Ejemplo: Algoritmo complejo explicado


```
/// <summary>
/// Makes the actual API call to fetch data
/// </summary>
private async Task<string> FetchDataFromApiAsync(string apiUrl, CancellationToken cancellationToken, Label statusLabel)
{
    UpdateStatusLabel("Fetching data...", statusLabel);

    // Perform HTTP GET request with cancellation support
    HttpResponseMessage response = await _client.GetAsync(apiUrl, cancellationToken);
    response.EnsureSuccessStatusCode(); // Throws if response is not successful
    string data = await response.Content.ReadAsStringAsync();

    UpdateStatusLabel("Data fetched.", statusLabel);
    return data;
}

/// <summary>
/// Updates UI label text safely across threads
/// </summary>
/// <param name="message">The message to display</param>
/// <param name="statusLabel">The label control to update</param>
private void UpdateStatusLabel(string message, Label statusLabel)
{
    // Check if we need to invoke on UI thread
    if (statusLabel.InvokeRequired) {
        // Use BeginInvoke to update label on UI thread
        statusLabel.BeginInvoke(new Action<string, Label>(UpdateStatusLabel), message, statusLabel);
    } else {
        // Direct update if already on UI thread
        statusLabel.Text = message;
    }
}
```

? Dudas