Realizacion de API
Dashboard de Teams Approvals para Usuario Específico

FASE 1: PREPARACIÓN Y PERMISOS (30 minutos)

Paso 1.1: Verificar permisos necesarios

Necesitas tener:
- ✅ Administrador Global de Azure AD (o Application Administrator)
- ✅ Licencia Power Automate Premium (incluida en A5)
- ✅ Acceso a Power BI Pro/Premium (incluido en A5)
- ✅ Permisos de administrador en SharePoint
- ✅ **NUEVO**: Administrador de Teams (para acceso a datos de Teams)

Paso 1.2: Crear estructura de carpetas en SharePoint

1. **Ir a SharePoint**:
   - Ve a https://[tuempresa].sharepoint.com
   - Clic en "Sites" → "Create site"

2. **Crear sitio nuevo**:
   - Tipo: "Team site"
   - Nombre: "Teams Approvals Dashboard"
   - Descripción: "Dashboard de aprobaciones de Teams por usuario"
   - Privacy: "Private"
   - Clic "Next" → "Finish"

3. **Crear biblioteca de documentos**:
   - En tu sitio, clic en "Documents"
   - Clic en "New" → "Folder"
   - Nombre: "UserApprovalData"

4. **Anotar la URL**:
   ```
   📝 GUARDA ESTE DATO:
   SharePoint URL: https://tuempresa.sharepoint.com/sites/TeamsApprovalsUsers
   ```

FASE 2: REGISTRO DE APLICACIÓN EN AZURE AD (25 minutos)

Paso 2.1: Crear aplicación en Azure

1. **Ir a Azure Portal**:
   - Ve a https://portal.azure.com
   - Busca "Azure Active Directory"
   - Clic en "App registrations"

2. **Crear nueva aplicación**:
   - Clic "New registration"
   - **Name**: `TeamsApprovalsExtractor`
   - **Supported account types**: "Accounts in this organizational directory only"
   - **Redirect URI**: Déjalo vacío
   - Clic "Register"

3. **Guardar información**:
   ```
   📝 GUARDA ESTOS DATOS:
   Application (client) ID: [cópialo aquí]
   Directory (tenant) ID: [cópialo aquí]
   ```

Paso 2.2: Crear Client Secret

1. **En tu aplicación**:
   - Clic "Certificates & secrets"
   - Clic "New client secret"
   - **Description**: `Teams Dashboard Secret`
   - **Expires**: "12 months"
   - Clic "Add"

2. **⚠️ COPIAR VALUE INMEDIATAMENTE**:
   ```
   📝 GUARDA ESTE DATO:
   Client Secret: [cópialo AHORA - no se mostrará de nuevo]
   ```

Paso 2.3: Asignar permisos específicos para Teams

1. **Ir a API permissions**:
   - Clic "Add a permission"
   - Selecciona "Microsoft Graph"
   - Clic "Application permissions"

2. **Agregar permisos para Teams Approvals**:

   Permiso 1 - Para leer equipos:
   - Busca "Team"
   - Marca ☑️ `Team.ReadBasic.All`
   - Clic "Add permissions"

   Permiso 2 - Para leer usuarios:
   - Clic "Add a permission" → "Microsoft Graph" → "Application permissions"
   - Busca "User"
   - Marca ☑️ `User.Read.All`
   - Clic "Add permissions"

   Permiso 3 - Para leer instalaciones de apps:
   - Clic "Add a permission" → "Microsoft Graph" → "Application permissions"
   - Busca "TeamworkAppInstallation"
   - Marca ☑️ `TeamworkAppInstallation.Read.All`
   - Clic "Add permissions"

   Permiso 4 - Para directorio:
   - Clic "Add a permission" → "Microsoft Graph" → "Application permissions"
   - Busca "Directory"
   - Marca ☑️ `Directory.Read.All`
   - Clic "Add permissions"

3. **🔴 CRÍTICO - Grant admin consent**:
   - Clic "Grant admin consent for [tu organización]"
   - Clic "Yes"
   - Verifica que todos tengan ✅ verde

FASE 3: CREAR AZURE FUNCTION PARA PROCESAMIENTO (30 minutos)

Paso 3.1: Crear Function App

1. **En Azure Portal**:
   - Busca "Function App"
   - Clic "Create"

2. **Configuración**:
   - **Resource Group**: Crear nuevo → "TeamsApprovals-RG"
   - **Function App name**: `teams-approvals-processor-[tus-iniciales]`
   - **Runtime stack**: "Python"
   - **Version**: "3.9"
   - **Region**: La más cercana
   - Clic "Review + create" → "Create"

Paso 3.2: Configurar Function

1. **Crear función**:
   - En tu Function App → "Functions" → "Create"
   - **Template**: "HTTP trigger"
   - **Function name**: `ProcessUserApprovals`
   - **Authorization level**: "Function"
   - Clic "Create"

2. **Agregar código Python**:

```python
import azure.functions as func
import pandas as pd
import json
import io
from datetime import datetime, timedelta
import logging

def main(req: func.HttpRequest) -> func.HttpResponse:
    logging.info('🔄 Procesando aprobaciones de Teams por usuario')

    try:
        # Obtener datos del request
        req_body = req.get_json()
        csv_data = req_body.get('csv_data')
        target_user = req_body.get('target_user', '')
```

```
if not csv_data:
    return func.HttpResponse(
        json.dumps({"error": "No se proporcionaron datos CSV"}),
        status_code=400
    )

# Convertir CSV a DataFrame
df = pd.read_csv(io.StringIO(csv_data))

# Filtrar por usuario específico si se proporciona
if target_user:
    # Filtrar donde el usuario es solicitante O aprobador
    df_filtered = df[
        (df['RequestorEmail'].str.contains(target_user, case=False, na=False)) |
        (df['ApproverEmail'].str.contains(target_user, case=False, na=False)) |
        (df['RequestorName'].str.contains(target_user, case=False, na=False)) |
        (df['ApproverName'].str.contains(target_user, case=False, na=False))
    ]
else:
    df_filtered = df

# Procesar datos
df_processed = process_user_approvals(df_filtered, target_user)

# Generar análisis específico por usuario
user_analysis = generate_user_analysis(df_processed, target_user)

# Preparar archivos de salida
response_data = {
    "processed_data": df_processed.to_csv(index=False),
    "user_as_requestor": user_analysis['as_requestor'].to_csv(index=False),
    "user_as_approver": user_analysis['as_approver'].to_csv(index=False),
    "monthly_trends": user_analysis['monthly_trends'].to_csv(index=False),
    "response_times": user_analysis['response_times'].to_csv(index=False),
    "metadata": {
        "processing_date": datetime.now().isoformat(),
        "target_user": target_user,
        "total_records": len(df_processed),
        "user_requests": len(user_analysis['as_requestor']),
```

```
            "user_approvals": len(user_analysis['as_approver']),
            "status": "success"
        }
    }

    return func.HttpResponse(
        json.dumps(response_data),
        status_code=200,
        mimetype="application/json"
    )

  except Exception as e:
    logging.error(f"❌ Error en procesamiento: {str(e)}")
    return func.HttpResponse(
        json.dumps({"error": str(e), "status": "failed"}),
        status_code=500,
        mimetype="application/json"
    )

def process_user_approvals(df, target_user):
  """Procesa los datos de aprobación específicos del usuario"""

  # Convertir fechas
  df['CreatedDateTime'] = pd.to_datetime(df['CreatedDateTime'])
  df['CompletedDateTime'] = pd.to_datetime(df['CompletedDateTime'])

  # Calcular tiempo de procesamiento
  df['ProcessingTimeHours'] = (df['CompletedDateTime'] -
df['CreatedDateTime']).dt.total_seconds() / 3600
  df['ProcessingTimeDays'] = df['ProcessingTimeHours'] / 24

  # Agregar campos temporales
  df['Year'] = df['CreatedDateTime'].dt.year
  df['Month'] = df['CreatedDateTime'].dt.month
  df['MonthName'] = df['CreatedDateTime'].dt.strftime('%B')
  df['Quarter'] = df['CreatedDateTime'].dt.quarter
  df['WeekOfYear'] = df['CreatedDateTime'].dt.isocalendar().week
  df['DayOfWeek'] = df['CreatedDateTime'].dt.day_name()
  df['HourOfDay'] = df['CreatedDateTime'].dt.hour
```

```
    # Clasificar velocidad de respuesta
    def classify_response_time(hours):
        if pd.isna(hours):
            return 'Pending'
        elif hours <= 1:
            return 'Immediate (≤1h)'
        elif hours <= 4:
            return 'Very Fast (1-4h)'
        elif hours <= 24:
            return 'Fast (4-24h)'
        elif hours <= 72:
            return 'Medium (1-3d)'
        elif hours <= 168:
            return 'Slow (3-7d)'
        else:
            return 'Very Slow (>7d)'

    df['ResponseTimeCategory'] =
df['ProcessingTimeHours'].apply(classify_response_time)

    # Identificar rol del usuario objetivo
    df['UserRole'] = 'Other'
    if target_user:
        mask_requestor = (df['RequestorEmail'].str.contains(target_user, case=False,
na=False)) | \
                          (df['RequestorName'].str.contains(target_user, case=False, na=False))
        mask_approver = (df['ApproverEmail'].str.contains(target_user, case=False,
na=False)) | \
                         (df['ApproverName'].str.contains(target_user, case=False, na=False))

        df.loc[mask_requestor, 'UserRole'] = 'Requestor'
        df.loc[mask_approver, 'UserRole'] = 'Approver'
        df.loc[mask_requestor & mask_approver, 'UserRole'] = 'Both'

    # Clasificar urgencia por hora de solicitud
    def classify_urgency(hour):
        if 9 <= hour <= 17:
            return 'Business Hours'
        elif 18 <= hour <= 22:
            return 'Evening'
```

```
        elif 23 <= hour or hour <= 6:
            return 'Off Hours'
        else:
            return 'Early Morning'

    df['RequestUrgency'] = df['HourOfDay'].apply(classify_urgency)

    return df

def generate_user_analysis(df, target_user):
    """Genera análisis específicos por usuario"""

    # Filtrar datos del usuario como solicitante
    user_as_requestor = df[df['UserRole'].isin(['Requestor', 'Both'])].copy()

    # Filtrar datos del usuario como aprobador
    user_as_approver = df[df['UserRole'].isin(['Approver', 'Both'])].copy()

    # Tendencias mensuales
    monthly_trends = df.groupby(['Year', 'Month', 'MonthName', 'UserRole']).agg({
        'ApprovalId': 'count',
        'ProcessingTimeHours': ['mean', 'median'],
        'Status': [
            lambda x: (x == 'approved').sum(),
            lambda x: (x == 'rejected').sum(),
            lambda x: (x == 'pending').sum()
        ]
    }).round(2)

    monthly_trends.columns = [
        'TotalApprovals', 'AvgProcessingHours', 'MedianProcessingHours',
        'ApprovedCount', 'RejectedCount', 'PendingCount'
    ]
    monthly_trends = monthly_trends.reset_index()
    monthly_trends['ApprovalRate'] = (monthly_trends['ApprovedCount'] /
monthly_trends['TotalApprovals'] * 100).round(2)

    # Análisis de tiempos de respuesta por categoría
    response_times = df.groupby(['UserRole', 'ResponseTimeCategory']).agg({
        'ApprovalId': 'count',
```

```
        'ProcessingTimeHours': 'mean'
    }).reset_index()
    response_times.columns = ['UserRole', 'ResponseCategory', 'Count', 'AvgHours']

    # Resúmenes por usuario como solicitante
    if len(user_as_requestor) > 0:
        requestor_summary = user_as_requestor.groupby(['Category', 'Status']).agg({
            'ApprovalId': 'count',
            'ProcessingTimeHours': 'mean'
        }).reset_index()
        requestor_summary.columns = ['Category', 'Status', 'Count',
'AvgProcessingHours']
    else:
        requestor_summary = pd.DataFrame(columns=['Category', 'Status', 'Count',
'AvgProcessingHours'])

    # Resúmenes por usuario como aprobador
    if len(user_as_approver) > 0:
        approver_summary = user_as_approver.groupby(['TeamName',
'ApproverResponse']).agg({
            'ApprovalId': 'count',
            'ProcessingTimeHours': 'mean'
        }).reset_index()
        approver_summary.columns = ['TeamName', 'Response', 'Count',
'AvgResponseTime']

        # Calcular eficiencia como aprobador
        approver_summary['EfficiencyScore'] = (100 -
approver_summary['AvgResponseTime']).clip(lower=0)
    else:
        approver_summary = pd.DataFrame(columns=['TeamName', 'Response',
'Count', 'AvgResponseTime', 'EfficiencyScore'])

    return {
        'as_requestor': requestor_summary,
        'as_approver': approver_summary,
        'monthly_trends': monthly_trends,
        'response_times': response_times
    }
```

3. **Obtener URL de la función**:
  ```
  📝 GUARDA ESTE DATO:
  Function URL: [copiar la URL completa con el código]
  ```

FASE 4: CREAR POWER AUTOMATE DESKTOP FLOW (25 minutos)

Paso 4.1: Crear Desktop Flow para Teams

1. **En Power Automate Desktop**:
  - Crear nuevo flow: `Extract-Teams-User-Approvals`

2. **Variables de entrada**:
  - `TenantId` (Text)
  - `ClientId` (Text)
  - `ClientSecret` (Text)
  - `TargetUserEmail` (Text) - **NUEVO**
  - `OutputPath` (Text)

3. **Script PowerShell para Teams Approvals**:

```powershell
param($TenantId, $ClientId, $ClientSecret, $TargetUserEmail, $OutputPath)

try {
   Write-Host "🔄 Iniciando extracción de Teams Approvals para: $TargetUserEmail" -ForegroundColor Green

   # Instalar módulos necesarios
   $Modules = @('Microsoft.Graph.Authentication', 'Microsoft.Graph.Teams', 'Microsoft.Graph.Users')
   foreach ($Module in $Modules) {
      if (!(Get-Module -ListAvailable -Name $Module)) {
         Install-Module $Module -Force -AllowClobber -Scope CurrentUser
      }
      Import-Module $Module
```

```
    }

    # Autenticación
    $SecureSecret = ConvertTo-SecureString $ClientSecret -AsPlainText -Force
    $Credential = New-Object
System.Management.Automation.PSCredential($ClientId, $SecureSecret)

    Connect-MgGraph -TenantId $TenantId -ClientSecretCredential $Credential
    Write-Host "✅ Conectado a Microsoft Graph" -ForegroundColor Green

    # Buscar el usuario objetivo
    $TargetUser = Get-MgUser -Filter "userPrincipalName eq '$TargetUserEmail'" -
ErrorAction SilentlyContinue
    if (-not $TargetUser) {
        $TargetUser = Get-MgUser -Filter "displayName eq '$TargetUserEmail'" -
ErrorAction SilentlyContinue
    }

    if (-not $TargetUser) {
        throw "No se encontró el usuario: $TargetUserEmail"
    }

    Write-Host "👤 Usuario encontrado: $($TargetUser.DisplayName)" -
ForegroundColor Cyan

    # Obtener equipos donde el usuario es miembro
    Write-Host "🔍 Buscando equipos del usuario..." -ForegroundColor Yellow
    $UserTeams = Get-MgUserJoinedTeam -UserId $TargetUser.Id

    $AllApprovals = @()
    $TeamsProcessed = 0

    foreach ($Team in $UserTeams) {
        try {
            Write-Host "📋 Procesando equipo: $($Team.DisplayName)" -
ForegroundColor Cyan

            # Verificar si tiene la app Approvals instalada
            $Apps = Get-MgTeamInstalledApp -TeamId $Team.Id -ErrorAction
SilentlyContinue
```

```
        $ApprovalsApp = $Apps | Where-Object {
            $_.TeamsAppDefinition.DisplayName -like "*Approval*" -or
            $_.TeamsAppDefinition.DisplayName -like "*Aprobación*"
        }

        if ($ApprovalsApp) {
            Write-Host "📝 App Approvals encontrada" -ForegroundColor Green

            # Intentar obtener aprobaciones (API Beta)
            try {
                $Uri =
"https://graph.microsoft.com/beta/teams/$($Team.Id)/approvals"
                $ApprovalResponse = Invoke-MgGraphRequest -Uri $Uri -Method GET -
ErrorAction Stop

                foreach ($Approval in $ApprovalResponse.value) {
                    # Verificar si el usuario está involucrado en esta aprobación
                    $IsUserInvolved = $false
                    $UserRole = "None"

                    # Verificar como solicitante
                    if ($Approval.requestor.user.userPrincipalName -eq
$TargetUser.UserPrincipalName -or
                        $Approval.requestor.user.displayName -eq
$TargetUser.DisplayName) {
                        $IsUserInvolved = $true
                        $UserRole = "Requestor"
                    }

                    # Verificar como aprobador
                    if ($Approval.responses) {
                        foreach ($Response in $Approval.responses) {
                            if ($Response.approver.user.userPrincipalName -eq
$TargetUser.UserPrincipalName -or
                                $Response.approver.user.displayName -eq
$TargetUser.DisplayName) {
                                $IsUserInvolved = $true
                                if ($UserRole -eq "Requestor") {
                                    $UserRole = "Both"
                                } else {
```

```
                    $UserRole = "Approver"
                }
            }
        }
    }

    # Solo incluir si el usuario está involucrado
    if ($IsUserInvolved) {
        $ApprovalRecord = [PSCustomObject]@{
            TeamId = $Team.Id
            TeamName = $Team.DisplayName
            ApprovalId = $Approval.id
            DisplayName = $Approval.displayName
            Status = $Approval.status
            CreatedDateTime = $Approval.createdDateTime
            CompletedDateTime = $Approval.completedDateTime
            RequestorName = if ($Approval.requestor.user) {
$Approval.requestor.user.displayName } else { "N/A" }
            RequestorEmail = if ($Approval.requestor.user) {
$Approval.requestor.user.userPrincipalName } else { "N/A" }
            ApproverName = "Pending"
            ApproverEmail = "N/A"
            ApproverResponse = "pending"
            Comments = ""
            Description = if ($Approval.details.description) {
$Approval.details.description } else { $Approval.displayName }
            Category = if ($Approval.details.category) {
$Approval.details.category } else { "General" }
            UserRole = $UserRole
            TargetUser = $TargetUser.DisplayName
            TargetUserEmail = $TargetUser.UserPrincipalName
            ExtractedDate = Get-Date -Format "yyyy-MM-dd HH:mm:ss"
        }

        # Agregar información del aprobador si existe respuesta
        if ($Approval.responses -and $Approval.responses.Count -gt 0) {
            $LastResponse = $Approval.responses | Sort-Object
createdDateTime | Select-Object -Last 1
```

```
                    $ApprovalRecord.ApproverName = if
($LastResponse.approver.user) { $LastResponse.approver.user.displayName } else {
"N/A" }
                    $ApprovalRecord.ApproverEmail = if
($LastResponse.approver.user) { $LastResponse.approver.user.userPrincipalName }
else { "N/A" }
                    $ApprovalRecord.ApproverResponse = $LastResponse.response
                    $ApprovalRecord.Comments = if ($LastResponse.comments) {
$LastResponse.comments } else { "" }
                }

                $AllApprovals += $ApprovalRecord
            }
        }

        $TeamsProcessed++
    }
    catch {
        Write-Warning "⚠️ No se pudieron obtener aprobaciones:
$($_.Exception.Message)"
    }
}
else {
    Write-Host "⏭️ Sin app Approvals" -ForegroundColor Gray
}
}
catch {
    Write-Warning "❌ Error procesando equipo $($Team.DisplayName):
$($_.Exception.Message)"
}
}

# Exportar datos
$CsvPath = Join-Path $OutputPath "teams_user_approvals_raw.csv"
$AllApprovals | Export-Csv -Path $CsvPath -NoTypeInformation -Encoding UTF8

# Crear resumen
$Summary = @{
    TargetUser = $TargetUser.DisplayName
    TargetUserEmail = $TargetUser.UserPrincipalName
```

```
        TotalApprovals = $AllApprovals.Count
        TeamsProcessed = $TeamsProcessed
        TotalTeams = $UserTeams.Count
        AsRequestor = ($AllApprovals | Where-Object { $_.UserRole -in
@("Requestor", "Both") }).Count
        AsApprover = ($AllApprovals | Where-Object { $_.UserRole -in @("Approver",
"Both") }).Count
        PendingApprovals = ($AllApprovals | Where-Object { $_.Status -eq "pending"
}).Count
        ApprovedCount = ($AllApprovals | Where-Object { $_.Status -eq "approved"
}).Count
        RejectedCount = ($AllApprovals | Where-Object { $_.Status -eq "rejected"
}).Count
        ExtractionDate = Get-Date -Format "yyyy-MM-dd HH:mm:ss"
        Status = "Success"
        OutputFile = $CsvPath
    }

    $SummaryPath = Join-Path $OutputPath "user_extraction_summary.json"
    $Summary | ConvertTo-Json | Out-File -FilePath $SummaryPath -Encoding UTF8

    Write-Host "✅ Extracción completada para $($TargetUser.DisplayName):" -
ForegroundColor Green
    Write-Host "   📊 Total aprobaciones: $($AllApprovals.Count)" -ForegroundColor
White
    Write-Host "   👤 Como solicitante: $(($AllApprovals | Where-Object {
$_.UserRole -in @("Requestor", "Both") }).Count)" -ForegroundColor White
    Write-Host "   ✅ Como aprobador: $(($AllApprovals | Where-Object {
$_.UserRole -in @("Approver", "Both") }).Count)" -ForegroundColor White
    Write-Host "   🏢 Equipos procesados: $TeamsProcessed de
$($UserTeams.Count)" -ForegroundColor White
    Write-Host "   📁 Archivo: $CsvPath" -ForegroundColor White

    Disconnect-MgGraph

    # Retornar para Power Automate
    "Success:$($AllApprovals.Count):$CsvPath:$($TargetUser.DisplayName)"

} catch {
    $ErrorMessage = $_.Exception.Message
```

```
    Write-Error "❌ Error durante la extracción: $ErrorMessage"

    $ErrorSummary = @{
        TargetUser = $TargetUserEmail
        TotalApprovals = 0
        ExtractionDate = Get-Date -Format "yyyy-MM-dd HH:mm:ss"
        Status = "Error"
        ErrorMessage = $ErrorMessage
    }

    $ErrorPath = Join-Path $OutputPath "user_extraction_error.json"
    $ErrorSummary | ConvertTo-Json | Out-File -FilePath $ErrorPath -Encoding UTF8

    "Error:$ErrorMessage"
}
```

FASE 5: CREAR FLUJO PRINCIPAL EN POWER AUTOMATE (35 minutos)

Paso 5.1: Crear flujo automatizado

1. **En Power Automate**:
   - Crear "Scheduled cloud flow"
   - **Name**: `Teams-User-Approvals-Dashboard`
   - **Frequency**: Daily at 6:00 AM

Paso 5.2: Configurar variables

```yaml
Variables a crear:
1. TenantId: [tu-tenant-id]
2. ClientId: [tu-client-id]
3. ClientSecret: [tu-client-secret]
4. SharePointSite: [tu-sharepoint-url]
5. TargetUserEmail: usuario@empresa.com  # ← NUEVO
6. FunctionURL: [tu-azure-function-url]
```

Paso 5.3: Estructura del flujo

1. **Initialize variables** (6 variables)
2. **Run Desktop Flow**: `Extract-Teams-User-Approvals`
   - Parameters:
     - TenantId: `@{variables('TenantId')}`
     - ClientId: `@{variables('ClientId')}`
     - ClientSecret: `@{variables('ClientSecret')}`
     - TargetUserEmail: `@{variables('TargetUserEmail')}`
     - OutputPath: `C:\temp`

3. **Get file content** (SharePoint):
   - File: `teams_user_approvals_raw.csv`

4. **HTTP Request** (Azure Function):
   - Method: POST
   - URI: `@{variables('FunctionURL')}`
   - Body:
     ```json
     {
       "csv_data": "@{base64ToString(body('Get_file_content'))}",
       "target_user": "@{variables('TargetUserEmail')}"
     }
     ```

5. **Parse JSON** (resultado de función)

6. **Create files** (Parallel branches):
   - `user_approvals_current.csv`
   - `user_as_requestor_current.csv`
   - `user_as_approver_current.csv`
   - `monthly_trends_current.csv`
   - `response_times_current.csv`

7. **Send notification email**:
   ```
   Subject: ✅ Dashboard Usuario @{variables('TargetUserEmail')} Actualizado

   Body:
   Dashboard actualizado para: @{variables('TargetUserEmail')}
   ```

📊 Resumen:
- Total aprobaciones: @{body('Parse_JSON')?['metadata']?['total_records']}
- Como solicitante: @{body('Parse_JSON')?['metadata']?['user_requests']}
- Como aprobador: @{body('Parse_JSON')?['metadata']?['user_approvals']}
- Fecha: @{formatDateTime(utcNow(), 'dd/MM/yyyy HH:mm')}
```

---

## FASE 6: CREAR DASHBOARD EN POWER BI (30 minutos)

### Paso 6.1: Conectar a datos

1. **Power BI Desktop** → **Get Data** → **SharePoint Online List**
2. **Site URL**: Tu URL de SharePoint
3. **Seleccionar archivos**:
   - `user_approvals_current.csv`
   - `user_as_requestor_current.csv`
   - `user_as_approver_current.csv`
   - `monthly_trends_current.csv`

### Paso 6.2: Crear medidas DAX

```dax
// Medidas para el dashboard por usuario

Total Approvals = COUNTROWS(user_approvals_current)

As Requestor =
CALCULATE(
    COUNTROWS(user_approvals_current),
    user_approvals_current[UserRole] IN {"Requestor", "Both"}
)

As Approver =
CALCULATE(
    COUNTROWS(user_approvals_current),
    user_approvals_current[UserRole] IN {"Approver", "Both"}
)
```
```

```
Avg Response Time (Hours) =
AVERAGE(user_approvals_current[ProcessingTimeHours])

Approval Rate =
DIVIDE(
   CALCULATE(COUNTROWS(user_approvals_current),
user_approvals_current[Status] = "approved"),
   COUNTROWS(user_approvals_current)
) * 100

Pending Count =
CALCULATE(
   COUNTROWS(user_approvals_current),
   user_approvals_current[Status] = "pending"
)
```

Paso 6.3: Crear visualizaciones

1. **Cards superiores**:
   - Total Approvals
   - As Requestor
   - As Approver
   - Avg Response Time
   - Approval Rate

2. **Gráfico de barras**: Status por UserRole

3. **Línea de tiempo**: Aprobaciones por mes (CreatedDateTime)

4. **Donut chart**: Distribución por ResponseTimeCategory

5. **Tabla detallada**:
   - Columns: TeamName, DisplayName, Status, CreatedDateTime, ProcessingTimeHours, UserRole

6. **Gráfico de barras horizontal**: Top 5 equipos con más aprobaciones

Paso 6.4: Agregar filtros

1. **Slicer por fechas**: CreatedDateTime
2. **Slicer por equipo**: TeamName
3. **Slicer por categoría**: Category
4. **Slicer por rol**: UserRole

FASE 7: CONFIGURAR PARÁMETRO DE USUARIO DINÁMICO (15 minutos)

Paso 7.1: Crear flujo para cambiar usuario

1. **Nuevo flujo**: `Change-Target-User`
2. **Trigger**: "When an HTTP request is received"
3. **Actions**:
   - Get target user from request body
   - Update environment variable
   - Trigger main flow

Paso 7.2: Crear interfaz simple

```html
<!-- Simple HTML form para cambiar usuario -->
<!DOCTYPE html>
<html>
<head>
   <title>Dashboard Usuario - Teams Approvals</title>
</head>
<body>
   <h2>Seleccionar Usuario para Dashboard</h2>
   <form action="[URL-del-flujo-HTTP]" method="POST">
      <label for="userEmail">Email del Usuario:</label>
      <input type="email" id="userEmail" name="userEmail" required>
      <button type="submit">Actualizar Dashboard</button>
   </form>
</body>
</html>
```

📋 CHECKLIST ESPECÍFICO PARA USUARIO

- ✅ Azure AD App con permisos Teams específicos
- ✅ Azure Function para procesamiento por usuario
- ✅ Desktop Flow que filtra por usuario específico
- ✅ Flujo principal con parámetro de usuario
- ✅ Dashboard Power BI personalizado por usuario
- ✅ Archivos CSV con prefijo "user_" para claridad
- ✅ Notificaciones con información del usuario objetivo

🎯 DATOS QUE OBTENDRÁS

```
📊 DASHBOARD MOSTRARÁ:
├── 👤 Perfil del usuario objetivo
├── 📈 Aprobaciones como solicitante
├── ✅ Aprobaciones como aprobador
├── ⏱️ Tiempos de respuesta promedio
├── 📅 Tendencias mensuales
├── 🏢 Equipos más activos
├── 🔄 Estados de aprobaciones
└──── 📋 Categorías de solicitudes
```

🚀 PARA EJECUTAR

1. **Manual**: Cambiar `TargetUserEmail` en variables del flujo
2. **Automático**: Usar el flujo HTTP para cambiar usuario dinámicamente
3. **Programado**: Ejecutará diariamente para el usuario configurado

¿En qué paso específico necesitas ayuda o dónde te encuentras actualmente en la implementación?