# Architectural Blueprint: Automating Financial Accruals and Unassigned Transaction Retrieval in SAP Concur via Azure and FastAPI

## 1. Executive Summary

The modernization of corporate financial operations demands near real-time visibility into organizational liabilities. For enterprises utilizing SAP Concur for expense management, the "accrual gap"—the latency between a credit card transaction occurring and its assignment to a submitted expense report—presents a critical challenge to accurate financial forecasting and month-end closing. This report provides an exhaustive technical analysis and architectural blueprint for automating the retrieval of **unassigned credit card transactions** to populate an accrual dashboard hosted in Microsoft SharePoint, powered by a Python FastAPI backend on Microsoft Azure.

The core technical challenge addressed in this analysis is the security model of the **SAP Concur Cards v4 API**, which explicitly mandates **User Level Access Tokens** for retrieving transaction data via standard REST endpoints. This requirement creates a "N+1 authentication" bottleneck, preventing a standard service account from iterating through an employee directory to pull card feeds without individual user consent or session instantiation.

This report evaluates the complete SAP Concur API landscape, including **Identity v4**, **Reports v4**, **Cards v4**, and the **Financial Integration Service (FIS)**. It dissects the available OAuth2 grant types—specifically the **Password Grant** and **Company Request Token** flows—and analyzes their applicability to the user-impersonation problem. The analysis confirms that while direct API access to unassigned transactions is restricted by the **User JWT** requirement in standard endpoints, alternative architectural patterns exist that respect the platform's security boundaries while delivering the required data.

The recommended solution shifts from a "Direct API Traversal" model (which is rate-limited and architecturally brittle) to a **"Report-as-a-Source" (RaaS)** pattern using SAP Concur Intelligence and SFTP automation. We detail the implementation of a FastAPI "Ingestion Engine" that acts as the middleware between these asynchronous data sources and the synchronous demands of a SharePoint dashboard, leveraging **Microsoft Graph API** for the final presentation layer.

## 2. The Accrual Data Challenge in Modern Expense Management

To understand the necessity of the proposed architecture, one must first dissect the lifecycle of a corporate expense and identify where the data visibility gap—the "Unassigned" bucket—exists within the SAP Concur ecosystem.

### 2.1 The Taxonomy of Expense Data States

In the context of corporate travel and expense (T&E) management, an accrual represents a financial liability that the company has incurred but has not yet recorded in the general ledger (GL). Within SAP Concur, a credit card transaction traverses several distinct states, each governed by different API availabilities and visibility scopes:

1. **Authorization & Clearing:** The employee swipes the corporate card. The bank authorizes the charge and settles it.
2. **Posted / Unassigned (The "Dark Matter"):** The transaction data arrives in SAP Concur via a bank feed (e.g., Amex, Visa, Mastercard). At this stage, the transaction sits in the user's "Available Expenses" pool. It is visible to the user but often invisible to the finance team via standard reporting until the user takes action. This is the **Unassigned Transaction**.
3. **Assigned / Unsubmitted:** The user adds the transaction to an expense report header but has not yet clicked "Submit." The liability is now associated with a report ID.
4. **Submitted / Workflow:** The report enters the approval workflow (Manager -> Processor -> Audit).
5. **Extracted / Posted:** The report is finalized, extracted via the Standard Accounting Extract (SAE) or Financial Integration Service (FIS), and posted to the ERP.1

The specific challenge articulated in the user query is the retrieval of data in **State 2 (Unassigned)**. This data represents "Shadow Accruals"—spend that has occurred, and for Company Billed/Company Paid (CBCP) cards, likely already paid by the corporation, yet remains unallocated to cost centers, project codes, or general ledger accounts.

### 2.2 The User-Centric Security Model Constraint

SAP Concur’s architecture is fundamentally user-centric, designed to comply with global data privacy regulations (GDPR, CCPA) and internal security policies. The platform treats the individual employee as the custodian of their transaction data until it is formally submitted to the organization via an expense report.

This philosophy manifests technically in the API scopes. As detailed in the **Cards v4** documentation, the endpoint GET /cards/v4/users/{userId}/transactions is the primary mechanism for reading credit card data.2 However, the documentation explicitly states: **"This endpoint supports user level access tokens only."**

This is a deliberate architectural constraint. Unlike the **Reports v4 API**, which offers broader visibility for processors and approvers via the expense.report.read scope (which can sometimes be accessed via Company JWT for specific operations or administrative contexts), the raw card feed is strictly locked to the user context.

### 2.3 The Automation Dilemma

For a developer building an automated dashboard in SharePoint, this constraint creates a significant hurdle. To build a system that automatically retrieves unassigned transactions for 10,000 employees, the backend application (FastAPI) would essentially need to:

1. Authenticate as User A.
2. Retrieve User A's transactions.
3. Authenticate as User B.
4. Retrieve User B's transactions.
5. Repeat for 10,000 users.

Since the **Password Grant** generally requires the user's explicit credentials (username and password) 2, and the **Authorization Grant** requires user interaction (browser redirect), there is no standard "System Account" mechanism to iterate through this data via the REST API without possessing the credentials of every single employee—an impossible and insecure requirement.

## 3. Comprehensive Analysis of SAP Concur API Architecture

To engineer a solution that bypasses these constraints, we must perform a deep dive into the authentication mechanisms, principal types, and specific capabilities of the relevant Concur APIs.

### 3.1 Authentication and Principal Types

The SAP Concur OAuth2 service distinguishes between three distinct principal types, as outlined in the authentication documentation.2 Understanding these principals is key to determining what data can be accessed programmatically.

#### 3.1.1 The Principal Hierarchy

* **User:** An individual employee. Tokens issued to this principal have access only to that user's data (e.g., their receipts, their reports, their card transactions).
* **Application:** A third-party integration or partner application.
* **Company:** The corporate entity itself. Tokens issued to this principal allow for administrative actions and access to company-wide data, *where permitted by the specific API endpoint*.

#### 3.1.2 Grant Types and Limitations

The documentation lists four primary grant types for obtaining tokens 2:

1. **Authorization Grant:** The standard 3-legged OAuth2 flow. It requires the user to authenticate via a login page. This is suitable for user-facing apps but unsuitable for a background daemon process gathering accruals for all users, as it requires manual intervention.
2. **Password Grant:** This grant is used when a "trust relationship" exists. It allows an application to exchange credentials for a token directly.
   * *Credential Type "Password":* Requires username and password. This obtains a **User Token**.
   * *Credential Type "AuthToken":* Used for App Center connections.
3. **Client Credentials Grant:** Typically used for machine-to-machine communication to get an **Application** or **Company** token. The snippet confirms this is supported for obtaining a token.2
4. **One Time Password Grant:** Leveraging geolocation and email links.

**The "Impersonation" Myth:** A common misconception is that a "Web Services Admin" can use the Password Grant with their own credentials to generate tokens for *other* users. The documentation does not support this. The Password Grant requires the credentials of the *resource owner*. Therefore, you cannot use an Admin's credentials to generate a User Token for "John Doe" to access John Doe's card transactions.

### 3.2 Deep Dive: Cards v4 API Constraints

The **Cards v4 API** is the modern interface for credit card data. A granular review of its capabilities 2 reveals why the direct approach fails:

* **Write Capabilities:** The API supports **Company Level Access Tokens** for *writing* data. Endpoints like POST.../transactions/bulk allow partners (banks, fintechs) to push transactions into Concur for the entire company using the scope cards.transaction.writeonly.
* **Read Capabilities:** The endpoint GET /cards/v4/users/{userId}/transactions is the only mechanism to read transactions. The documentation is explicit: **"This endpoint supports user level access tokens only."**
* **Scope Mismatch:** The required scope is cards.transaction.read. There is no "company-wide read" scope listed for transactions.

This asymmetry (Company Write / User Read) confirms that a Company JWT cannot be used to scrape unassigned transactions from the Cards v4 API.

### 3.3 Deep Dive: Reports v4 API Capabilities

The **Reports v4 API** 2 allows reading and modifying expense report headers.

* **Token Support:** It supports both **Company** and **User** access tokens.
* **Endpoint:** GET /expensereports/v4/users/{userID}/context/{contextType}/reports.
* **Relevance:** This API can retrieve data for reports that have already been created (Assigned/Unsubmitted). Using a Company Token, a backend service *could* theoretically iterate through users to fetch their unsubmitted reports.
* **The Gap:** Unassigned credit card transactions, by definition, are **not yet associated with a report**. Therefore, they do not exist as resources within the Reports v4 API. This API can solve the "Unsubmitted" part of the accrual equation but fails for the "Unassigned" part.

### 3.4 Deep Dive: Identity v4 API

The **Identity v4 API** 2 is critical for system enumeration.

* **Endpoint:** GET /profile/identity/v4.1/Users.
* **Token Support:** Supports **Company Level Access Tokens**.
* **Utility:** This API allows the FastAPI backend to retrieve a complete list of userIds (UUIDs) for every employee in the company.
* **Application:** While this allows the system to know *who* the users are, it does not grant the privilege to read their card data via the Cards v4 API. However, it is essential for mapping Employee IDs to names and cost centers in the final dashboard.

## 4. Architectural Solution: The "Report-as-a-Source" Pattern

Since the direct API path for unassigned transactions is blocked by the User Token requirement, the solution must pivot to an alternative data ingress method that provides company-wide visibility. The robust, SAP-sanctioned pattern for this is **Intelligence Reporting via SFTP**, orchestrated by the FastAPI backend.

### 4.1 Solution Architecture Overview

The proposed architecture implements a **Hybrid Data Ingestion** strategy, combining API calls for user metadata with SFTP ingestion for transaction data.

1. **Orchestrator (FastAPI on Azure):** The central brain that manages schedules, data ingestion, and transformation.
2. **Data Source A (SAP Concur Intelligence):** A scheduled "Unassigned Credit Card Transactions" report delivered to SFTP.
3. **Data Source B (Identity v4 API):** Provides user metadata (Department, Cost Center) via Company Token.
4. **Storage Layer (Azure SQL / Cosmos DB):** Stores the normalized accrual data.
5. **Presentation Layer (SharePoint):** Displays the data, populated via Microsoft Graph API.

### 4.2 The "Unassigned Credit Card Transactions" Report

SAP Concur provides a standard report within its Analysis/Intelligence module specifically designed for this use case: **"Unassigned Credit Card Transactions"**.3

* **Content:** This report captures all corporate card transactions that have been posted to Concur but have not yet been assigned to an expense report.4
* **Data Fields:** It includes critical fields such as Merchant Code, Description, Transaction Date, Posted Amount, Reimbursement Currency, Employee Name, and Employee ID.5
* **Availability:** This report is available in the "Accrual" folder of the standard reports catalog.6
* **Automation:** The Intelligence platform allows this report to be **scheduled**. It can be configured to run daily (e.g., at 01:00 UTC) and output a formatted file (CSV, XLS, XML) to an **SAP Concur-hosted SFTP server**.7

### 4.3 Why This Solves the Problem

This approach bypasses the OAuth2 limitation entirely. Instead of the FastAPI application asking the API "Give me User A's transactions" (which requires User A's token), the application asks the Intelligence engine "Give me a dump of ALL unassigned transactions," which is an administrative privilege. The data is delivered securely via SFTP, and the FastAPI application ingests it using a Service Account (SSH Key), effectively decoupling data retrieval from individual user sessions.

### 4.4 The "Company Bill Statements" Consideration

The research material mentions the **Company Bill Statements** feature.8 This feature centralizes p-card reconciliation.

* If the client uses Company Bill Statements, unassigned transactions may also be visible in the **Standard Accounting Extract (SAE)** under the "Company Bill Statements" section.10
* However, SAEs are typically generated for *posting* purposes (i.e., ready for the ERP). Relying on the SAE for *accruals* (unprocessed spend) can be risky if the configuration excludes unassigned items.
* Therefore, the **Intelligence Report** remains the most reliable source for raw, unassigned liability data.

## 5. Technical Implementation Guide

This section details the step-by-step implementation of the solution using **Python FastAPI** hosted on **Azure**.

### 5.1 Infrastructure Components

* **Azure App Service (Linux Plan):** Hosting the FastAPI Docker container.
* **Azure Key Vault:** Securely storing the Concur Client ID, Client Secret, Refresh Token, SFTP Host, SFTP Username, and SSH Private Key.
* **Azure SQL Database:** For persisting the current state of accruals.
* **Concur SFTP Account:** Provisioned by SAP Concur support.

### 5.2 Step 1: Configuring SAP Concur Intelligence

Before writing code, the data source must be configured.

1. **Log in** to SAP Concur as an Administrator with the "Cognos Professional Author" or "Business Author" role.11
2. **Navigate** to Reporting > Intelligence.
3. **Locate** the standard report: Public Folders > Intelligence - Standard Reports > Accrual > Unassigned Credit Card Transactions.
4. **Create a View:** Save a copy of this report to your company's folder. Modify it to ensure it exports as **CSV** (preferred for machine parsing) or XML.
5. **Schedule the Report:**
   * Set the interval to **Daily**.
   * Set the delivery method to **File System** (which targets the SFTP server).
   * Define a consistent naming convention (e.g., accrual\_unassigned\_%Y%m%d.csv).

### 5.3 Step 2: The FastAPI "Ingestion Engine"

The FastAPI application will act as a background worker. While FastAPI is primarily a web framework, it can handle scheduled tasks using libraries like APScheduler or fastapi-utils.

#### 5.3.1 Dependencies

Python

# requirements.txt  
fastapi  
uvicorn  
paramiko # For SFTP connection  
pandas # For CSV data processing  
httpx # For API calls (Graph, Identity v4)  
apscheduler # For scheduling the ingest job  
azure-identity # For Key Vault integration  
azure-keyvault-secrets  
sqlalchemy # For database persistence  
msgraph-core # For SharePoint integration

#### 5.3.2 The SFTP Ingestion Logic

This function connects to the Concur SFTP, looks for new files, and processes them.

Python

import paramiko  
import pandas as pd  
from io import BytesIO  
from datetime import datetime  
  
class ConcurSftpConnector:  
 def \_\_init\_\_(self, host, username, key\_content):  
 self.host = host  
 self.username = username  
 self.key = paramiko.RSAKey.from\_private\_key(BytesIO(key\_content.encode()))  
  
 def fetch\_latest\_accrual\_report(self, directory='/out'):  
 transport = paramiko.Transport((self.host, 22))  
 transport.connect(username=self.username, pkey=self.key)  
 sftp = paramiko.SFTPClient.from\_transport(transport)  
   
 try:  
 files = sftp.listdir\_attr(directory)  
 # Filter for accrual reports and sort by modification time  
 accrual\_files = [f for f in files if "accrual\_unassigned" in f.filename]  
 accrual\_files.sort(key=lambda x: x.st\_mtime, reverse=True)  
   
 if not accrual\_files:  
 return None  
   
 latest\_file = accrual\_files  
 with sftp.open(f"{directory}/{latest\_file.filename}", "r") as f:  
 # Load CSV content into Pandas  
 df = pd.read\_csv(f)  
 return df  
 finally:  
 sftp.close()  
 transport.close()

#### 5.3.3 Data Normalization

The CSV data from Concur must be normalized before storage. Key fields identified in the report snippet 5 include <Merchant Code>, <Description>, <Transaction Date>, and <Posted Amount>.

Python

def normalize\_accrual\_data(df):  
 # Mapping CSV columns to internal schema  
 normalized = df.rename(columns={  
 'Merchant': 'merchant\_name',  
 'Transaction Date': 'transaction\_date',  
 'Posted Amount': 'amount',  
 'Reimbursement Currency': 'currency',  
 'Employee': 'employee\_name',  
 'Employee ID': 'employee\_id'  
 })  
   
 # Add status field  
 normalized['status'] = 'Unassigned'  
 normalized['ingestion\_date'] = datetime.utcnow()  
   
 return normalized.to\_dict('records')

## 6. Integrating User Metadata via API

While the SFTP report provides the transaction data, the dashboard may require richer user metadata (e.g., Department, Division, Manager) that isn't on the transaction report. For this, we use the **Identity v4 API** with a **Company Token**.

### 6.1 Authentication: Managing the Company Token

To call the Identity API, we need a valid Access Token. The **Company Request Token** 12 is a temporary credential used once to bootstrap the **Refresh Token**. The FastAPI app must manage the lifecycle of this Refresh Token.

**Critical Security Note:** Concur Refresh Tokens expire after 6 months but can rotate on use.2 The application **must** persist the new Refresh Token to Azure Key Vault every time it is used to get a new Access Token.

Python

# auth\_service.py  
import httpx  
from azure.identity import DefaultAzureCredential  
from azure.keyvault.secrets import SecretClient  
  
class ConcurAuth:  
 def \_\_init\_\_(self, vault\_url):  
 self.credential = DefaultAzureCredential()  
 self.client = SecretClient(vault\_url=vault\_url, credential=self.credential)  
   
 async def get\_access\_token(self):  
 # Retrieve secrets  
 client\_id = self.client.get\_secret("ConcurClientId").value  
 client\_secret = self.client.get\_secret("ConcurClientSecret").value  
 refresh\_token = self.client.get\_secret("ConcurRefreshToken").value  
   
 url = "https://us.api.concursolutions.com/oauth2/v0/token"  
 data = {  
 "client\_id": client\_id,  
 "client\_secret": client\_secret,  
 "refresh\_token": refresh\_token,  
 "grant\_type": "refresh\_token"  
 }  
   
 async with httpx.AsyncClient() as client:  
 resp = await client.post(url, data=data)  
 resp.raise\_for\_status()  
 tokens = resp.json()  
   
 # CRITICAL: Update Key Vault if a new refresh token is returned  
 if 'refresh\_token' in tokens:  
 self.client.set\_secret("ConcurRefreshToken", tokens['refresh\_token'])  
   
 return tokens['access\_token']

### 6.2 Enriching Data

Once the transaction data is loaded from SFTP, the FastAPI app can query the Identity v4 API to enrich the records.

* **Endpoint:** GET /profile/identity/v4.1/Users.2
* **Logic:** Fetch all users, create a lookup map (employee\_id -> Department), and merge this with the transaction data. This ensures the dashboard can slice and dice accruals by department, even if the raw card feed lacks that context.

## 7. SharePoint Integration via Microsoft Graph

The final requirement is to visualize this data in a SharePoint dashboard. The most effective method is to push the processed data into a **SharePoint List**, which can then be visualized using a Power BI web part or a custom SPFx view.

### 7.1 Microsoft Graph Implementation

We will use the Microsoft Graph API 13 to create list items.

1. **Azure AD Registration:** Register the FastAPI app in Azure AD. Grant it Sites.ReadWrite.All application permission.
2. **Authentication:** Use ClientSecretCredential from azure-identity to get a Microsoft Graph token.
3. **Push Logic:**

Python

# graph\_service.py  
async def push\_to\_sharepoint(site\_id, list\_id, accruals):  
 token = await get\_graph\_token()  
 headers = {"Authorization": f"Bearer {token}", "Content-Type": "application/json"}  
   
 async with httpx.AsyncClient() as client:  
 for item in accruals:  
 payload = {  
 "fields": {  
 "Title": f"Txn-{item['employee\_id']}-{item['date']}",  
 "Amount": item['amount'],  
 "Currency": item['currency'],  
 "Merchant": item['merchant\_name'],  
 "Employee": item['employee\_name'],  
 "Status": "Unassigned"  
 }  
 }  
 # POST to create item  
 await client.post(  
 f"https://graph.microsoft.com/v1.0/sites/{site\_id}/lists/{list\_id}/items",  
 headers=headers,   
 json=payload  
 )

### 7.2 Dashboard Design

* **SharePoint List:** Acts as the database. Columns: Merchant, Amount, Currency, Date, Employee, Department.
* **Power BI:** Connect Power BI to this SharePoint List. Create visuals showing "Total Unassigned Liability," "Liability by Department," and "Aging of Unassigned Transactions."
* **Embed:** Publish the Power BI report and embed it into the SharePoint page.

## 8. Operational Considerations and Constraints

### 8.1 API Rate Limiting

SAP Concur imposes global rate limits (e.g., 50 concurrent requests per second).14 While the SFTP download consumes no API quota, the Identity API calls do.

* **Mitigation:** Cache the User Identity list in the Azure SQL database and only refresh it weekly. Do not call the Identity API for every transaction row.

### 8.2 Token Security

The **Refresh Token** is highly sensitive. It grants long-term access to the Concur instance.

* **Storage:** Always use Azure Key Vault. Never hardcode it.
* **Rotation:** Implement logic to detect invalid\_grant errors.12 If the refresh token expires or is revoked, the system should alert administrators to re-run the Company Request Token flow.

### 8.3 Data Freshness

The Intelligence Report is typically generated overnight (e.g., reflecting data up to the previous day). This means the dashboard will have a **24-hour latency**.

* **Managing Expectations:** Ensure stakeholders understand this dashboard represents "Yesterday's Liability." Real-time API access for unassigned transactions is not currently supported by Concur for this architectural pattern.

## 9. Conclusion

Automating the retrieval of unassigned credit card transactions from SAP Concur for an accrual dashboard requires a strategic bypass of the **Cards v4 API's** user-level security constraints. By adopting a **"Report-as-a-Source"** architecture, utilizing SAP Concur Intelligence to schedule transaction dumps to SFTP, and leveraging **Azure FastAPI** as the integration middleware, organizations can achieve total visibility into "Shadow Accruals."

This solution avoids the operational nightmare of managing thousands of user tokens, respects SAP Concur's rate limits, and utilizes the robust **Microsoft Graph API** to deliver a seamless user experience within SharePoint. The integration of **Identity v4** data allows for rich organizational context, transforming raw transaction lists into actionable financial intelligence.

### Key Implementation Checklist

| **Component** | **Action Required** |
| --- | --- |
| **Concur Intelligence** | Schedule "Unassigned Credit Card Transactions" report to SFTP (CSV format). |
| **Concur Auth** | Generate Company Request Token; Exchange for initial Refresh Token via Password Grant. |
| **Azure Key Vault** | Store Client ID, Secret, SFTP Credentials, and *Mutable* Refresh Token. |
| **FastAPI** | Implement APScheduler for SFTP polling and Token Rotation logic. |
| **SharePoint** | Create "Accruals" List and register Azure App for Graph API access. |
| **Data Policy** | Define data retention rules in Azure SQL to match corporate compliance. |

This blueprint provides a production-ready, scalable, and secure path to solving the unassigned transaction visibility challenge.

#### Works cited

1. preview.developer.concur.com/src/api-reference/financial-integration/v4.financial-integration.markdown at main · SAP-docs/preview.developer.concur.com · GitHub, accessed on January 2, 2026, <https://github.com/SAP-docs/preview.developer.concur.com/blob/main/src/api-reference/financial-integration/v4.financial-integration.markdown>
2. Concur APIs
3. Report: Unassigned Company Billed Transactions - SAP Help Portal, accessed on January 2, 2026, <https://help.sap.com/docs/SAP_CONCUR/d314a525472c49548219b560516c9b8d/1c497991700b1014a46a108435d32877.html>
4. Credit Card Transactions - UCSD Blink, accessed on January 2, 2026, <https://blink.ucsd.edu/finance/bi-financial-reporting/concur-reports/concur-cctransactions.html>
5. Report: Unassigned Credit Card Transactions | SAP Help Portal, accessed on January 2, 2026, <https://help.sap.com/docs/SAP_CONCUR/8ab8e8f3cf69412e9c6517d0266282c6/1c49954d700b1014a46a108435d32877.html?locale=en-US&state=PRODUCTION&version=2025_06>
6. Report: Expense Accrual | SAP Help Portal, accessed on January 2, 2026, <https://help.sap.com/docs/SAP_CONCUR/8ab8e8f3cf69412e9c6517d0266282c6/1c3f6182700b1014a46a108435d32877.html>
7. Cognos Report Net Delivery with SAP Concur SFTP, accessed on January 2, 2026, <https://community.sap.com/t5/spend-management-blog-posts-by-sap/cognos-report-net-delivery-with-sap-concur-sftp/ba-p/13801829>
8. Company Bill Statements: Business P-Card Reconciliation | SAP ..., accessed on January 2, 2026, <https://www.concur.com/products/company-bill-statements>
9. March 2021 Expense Professional Edition Admin Summary - SAP Help Portal, accessed on January 2, 2026, <https://help.sap.com/docs/CONCUR_EXPENSE/85f3db6f3c5945fe8414a08fc5f9e3fc/41ace0c87f7d4241a7d952e69f1b1c74.html?locale=en-GB&version=2022_09>
10. File\_Data - SAP Concur Developer Center, accessed on January 2, 2026, <https://developer.concur.com/api-guides/ERP-integration/standard-accounting-extract-expense.xlsx>
11. Intelligence Summary and FAQ - SAP Concur Community, accessed on January 2, 2026, <https://community.concur.com/marav43842/attachments/marav43842/ReportingHub/41/1/3584_CON_IntelligenceSummary_FAQ_r2.pdf>
12. preview.developer.concur.com/src/api-reference/authentication/company-auth.markdown at ... - GitHub, accessed on January 2, 2026, <https://github.com/SAP-docs/preview.developer.concur.com/blob/main/src/api-reference/authentication/company-auth.markdown>
13. Create a new entry in a SharePoint list - Microsoft Graph v1.0 ..., accessed on January 2, 2026, <https://learn.microsoft.com/en-us/graph/api/listitem-create?view=graph-rest-1.0>
14. Rate Limiting and Throttling | SAP Help Portal, accessed on January 2, 2026, <https://help.sap.com/docs/cloud-identity-services/cloud-identity-services/rate-limiting-and-throttling>