Bidirektionaler Datenfluss zwischen Oracle APEX und Desktop-Anwendungen

Sebastian Reinhard, merlin.zwo InfoDesign

Der Artikel zeigt, wie Oracle APEX mithilfe einer schlanken .NET-Brücke und SignalR ein nahtloser Dateiaustausch mit Desktop-Programmen ermöglicht wird. Zudem gehen wir auf die Hürden ein, welche eine Verlagerung der Dokumente in ein zentrales webbasiertes Dokumentenmanagement mit sich bringt.

Seit Jahren verlagern sich Anwendungen vom klassischen Desktop hin ins Web. Der Charme dieser Architektur liegt auf der Hand: Eine zentrale Deployment-Stelle sorgt dafür, dass alle Benutzer automatisch mit der neuesten Version arbeiten; zugleich ist die Anwendung plattformunabhängig und läuft auf PC, Mac oder Smartphone gleichermaßen. Doch wo Licht ist, ist bekanntlich auch Schatten: Aus Sicherheitsgründen werden Web-Applikationen in einer Sandbox des Browsers ausgeführt. Das erschwert den direkten Zugriff auf lokale Ressourcen – insbesondere auf Dateien, die Anwender in ihren gewohnten Desktop-Programmen bearbeiten möchten.

Sobald die Dateien zentral in der Datenbank liegen und die Anwender sie über den Browser herunterladen, lokal bearbeiten und erneut hochladen müs-

sen, kippt die anfängliche Begeisterung schnell in Ärger um. Zu viele Klicks, Versionskonflikte und Medienbrüche bremsen Prozesse aus – insbesondere, wenn täglich hunderte Office-Dokumente und PDFs bearbeitet werden müssen. Unser Kunde, ein mittelständischer Betrieb, stellte deshalb die scheinbar einfache Frage:

„Warum kann ich mein Dokument nicht einfach direkt aus dem ERP heraus in Word öffnen, speichern und zurückschreiben – ohne Umweg über Download-Dialoge?"

Wir nahmen die technische Herausforderung, die in der Frage schlummerte, an und schufen im Ergebnis eine Brücke zwischen APEX und dem Desktop des Nutzers, welche die benötigten Dateien lokal zu Verfügung stellt, überwacht und mit dem Server synchronisiert.

Vom Netzwerkshare zur zentralen Dokumentenablage in der Datenbank

Die Historie: Das alte ERP basierte auf Oracle Forms; sämtliche kundenbezogenen Dokumente lagen in einem Samba-Share, der auf jedem Arbeitsplatzrechner unter demselben Laufwerksbuchstaben eingebunden war. Jeder Kunde besaß einen Ordner, dessen Name der Kundennummer entsprach. Im Tagesgeschäft wurden jedoch häufig Dateien in falsche Ordner kopiert, versehentlich lokal verschoben oder gleich ganze Ordner verschleppt. Dadurch gingen Unterlagen verloren, waren für Kollegen nicht auffindbar oder existierten mehrfach in unterschiedlichen Versionen.

Ausgangslage: Nach der Migration des ERP-Systems lagen alle Dokumente in BLOB-Spalten der Oracle-Datenbank. Was als zentraler Single-Point-of-Truth geplant war, entpuppte sich für die Anwender als Klick-Orgie: herunterladen → bearbeiten → gegebenenfalls zurück zum Kunden navigieren, da eine andere Anfrage dazwischen kam → hochladen → Datei schließen. Schon drei Office-Formate (.docx, .xlsx, .pdf) sorgten für Ärger – und es war absehbar, dass noch weitere Dateiformate folgen könnten, zum Beispiel CAD. *(siehe Tabelle 1)*

Keine Alternative überzeugte vollständig – entweder fehlte Funktionsumfang, Kosten/Nutzen standen in keinem Verhältnis oder der Kunde wollte partout kein Abo-Modell. Vor allem aber kam für den Kunden Netzwerkshare nicht mehr in Frage. Also entwickelten wir eine eigene Brückenlösung.

Architektur im Überblick

Die Architektur basiert auf drei ineinandergreifenden Bausteinen *(siehe Abbildung 1)*. Im Zentrum steht die Oracle-Datenbank, in der sämtliche Dokumente abgelegt sind. Zwischen der Datenbank und den Endgeräten vermittelt ein schlanker Bridgeserver, der Anfragen entgegennimmt, Sitzungskontexte verwaltet und die Anfragen an die passenden Clients verteilt. Auf jedem Arbeitsplatzrechner läuft dazu ein Hintergrunddienst im System-Tray, der eine ständige Verbindung zum Bridgeserver hält.

Möchte der Anwender eine Datei bearbeiten, benachrichtigt die Datenbank über den Bridgeserver den Hintergrunddienst und dieser lädt das Dokument in ein temporäres Verzeichnis und öffnet es unmittelbar im zugeordneten Desktop-Programm – egal ob Textverarbeitung, Tabellenkalkulation oder Bildbearbeitung. Eine permanente Überwachung der Datei stellt sicher, dass jede Änderung zuverlässig erkannt und umgehend an die zentrale Ablage zurückgespielt wird, ohne dass der Benutzer zusätzliche Schritte ausführen muss.

Nachdem nun grob klar ist, wie eine Datei ihren Weg vom Server zum Arbeitsplatz findet, stellt sich die nächste Frage: Woher weiß die APEX-Anwendung überhaupt, welcher Benutzer hinter welchem lokalen Client steckt?

Die Antwort findet sich in *Abbildung 2* – über den Tray Client kann sich der Anwender anmelden. Die Authentifizierung findet über die Active Directory Zugangsdaten beziehungsweise die Auth-API der Datenbank (Rest Data Service) statt. Nach erfolgreicher Anmeldung stellt die Auth API ein JWT Sitzungstoken sowie eine zufällig generierte, eindeutige Client ID aus. Beides wird serverseitig hinterlegt und parallel an den Client zurückgegeben. Ab diesem Moment kann der APEX-Benutzer seinem Arbeitsplatz zugeordnet werden.

Technik – APEX

In APEX werden die Dokumente in Reports angezeigt *(siehe Abbildung 3)*. Für jedes Dokument gibt es eine Schaltfläche (Monitor-Icon), um das Dokument lokal zu bearbeiten. Beim Klick stoßen wir per JavaScript ein AJAX-Callback (PL/SQL-Prozess) an, der serverseitig für den angemeldeten Benutzer die zugehörige Client-ID ermittelt und ein kryptographisch starkes Einmaltoken („One-Time-Token") generiert. Dieses Token wird in einer dedizierten Session-Tabelle mit extrem kurzer TTL (Time-To-Live, zum Beispiel 30 Sekunden) persistiert, sodass es

Evaluierte Optionen

Option	Vorteile	Nachteile
Office 365 + SharePoint	Aktuellstes Office, Kollaboration	Abo-Modell, Daten in der Cloud, nur Office-Formate, Mapping ins ERP aufwändig
Web-Editoren	Kein lokales Office nötig	Lizenzkosten, Funktionslücken, Schulungsaufwand, ein Editor pro Dateityp
Rückkehr zum Netzwerk-Share	Geringe Komplexität, kein Abo	Chaotische Ordnerstrukturen, Indexierung nötig

Tabelle 1.

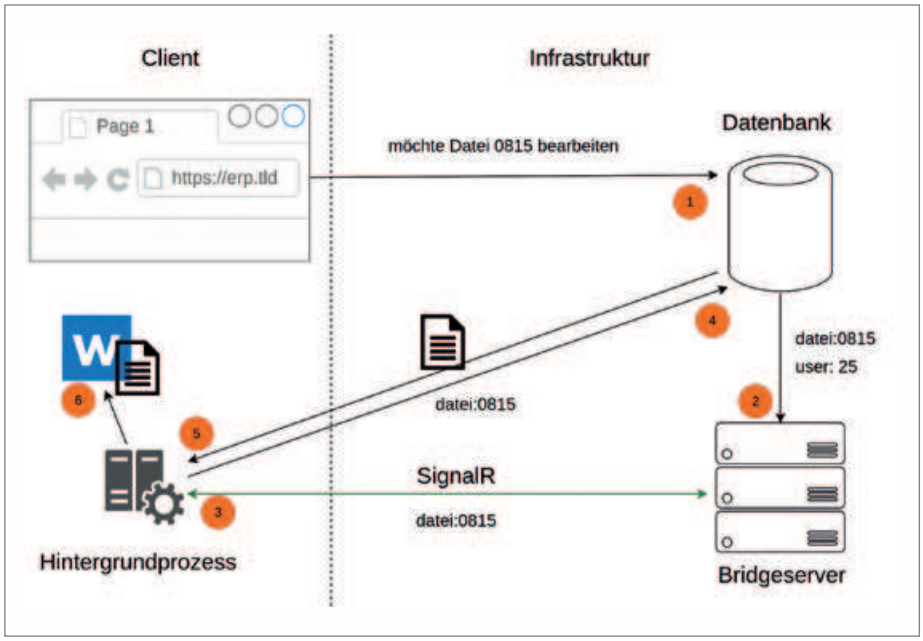

Abbildung 1: Übersicht über den Prozessablauf (Quelle: Sebastian Reinhard)

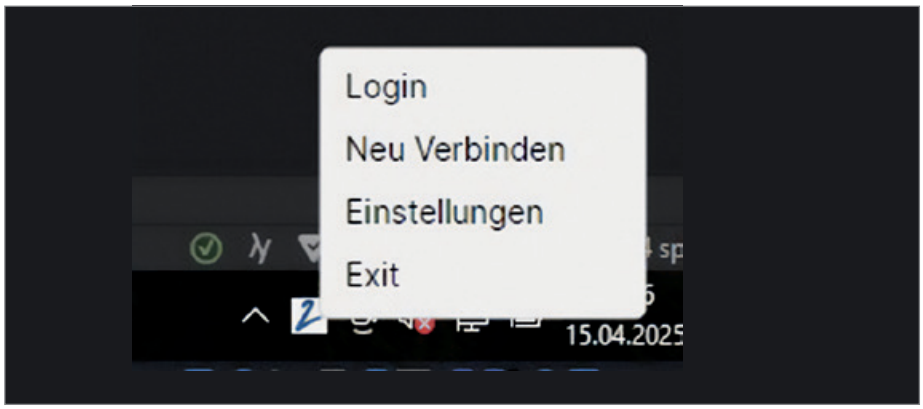

Abbildung 2: Screenshot der Menüpunkte über das Tray-Icon (Quelle: Sebastian Reinhard)

sofort abläuft, falls es nicht genutzt wird. Anschließend initiiert der PL/SQL-Prozess einen REST-Request über apex_rest_request an den Bridge-Server. Hierbei werden alle relevanten Parameter – Client-ID, Einmaltoken, Datei-ID übergeben. Der Bridge-Server validiert daraufhin das Token und leitet die Anforderung über SignalR an den richtigen Client weiter.

Technik – Datenbank

Datenbankseitig stellen wir mittels RESTful Services folgende Endpunkte für den Austausch bereit:

- /login: Authentifizierung, gibt ein JSON Web Token (JWT) und eine eindeutige Client-ID zurück, die den Benutzer identifiziert.
- /token: Endpoint zur Token-Erneuerung oder zur Ausgabe eines neuen Access-Tokens.
- /validate: Über diesen Endpunkt kann der Bridgeserver das Einmaltoken validieren. Das Token ist danach ungültig.
- /file/{id} (GET/PUT): Download oder Upload einer Datei über die ID, mit Validierung des JWT und Prüfung auf Zugriffsrechte.
- /filedetails/{id}: Liefert Metadaten wie MIME-Type und MD5-Hash der Datei, dient unter anderem der Integritätsprüfung und zur Konflikterkennung vor dem Upload.

Technik – Bridgeserver

Um zu erklären, warum wir überhaupt einen Bridgeserver brauchen, machen wir noch einen kleinen Ausflug in die Theorie von HTTP. Das klassische Web Protokoll HTTP arbeitet nach dem Prinzip *„Der Client fragt, der Server antwortet."* Sobald die Antwort versendet ist, gilt die Verbindung als beendet; der Server kann keine Nachrichten mehr initiieren. Für unser Szenario – in dem der Server eine Datei spontan an den Client „schieben" soll – reicht dieses Pull-Modell nicht aus. Deshalb richtet der Tray Dienst unmittelbar nach der Anmeldung eine dauerhafte rückkanalfähige Verbindung zum Bridge Server ein. Technisch basiert dieser Kanal auf einem WebSocket ähnlichen Verfahren mit dem Namen SignalR, das durch Firewalls hindurch funktioniert und lediglich einen einzigen Port benötigt. Der Bridgeserver selbst basiert auf ASP.NET 8 mit einer Minimal-API und dem SignalR Hub. Da hier nur flüchtige Daten anfallen, halten wir die Mappings ClientID → ConnectionID in einem In-Memory-Dictionary, um offene Verbindungen effizient zu adressieren. Ein Persistieren ist nicht notwendig, da die SignalR-Verbindungen nach einem Neustart weg wären beziehungsweise neu aufgebaut werden. Eingehende REST-API-Aufrufe werden an den SignalR Hub weitergeleitet und die FileIDs an die Clients gesendet. Zum Schutz der API werden die Einmaltokens gegen die Datenbank validiert, um Replay- und Injection-Angriffe zu verhindern.

Technik – Client

Der Client-Daemon wurde in .NET 8 entwickelt und setzt Avalonia UI ein, um eine vollständig plattformübergreifende grafische Oberfläche für Windows, macOS und Linux bereitzustellen. Das Tray-Icon ist eine eingebaute Funktionalität von Avalonia und ermöglicht eine plattformübergreifende Integration in die System-Tray-Bereiche der jeweiligen Betriebssysteme. Systemnahe Funktionen wie File-Watching, Auto-Start und Prozessüberwachung sind über abstrahierte Interfaces gekapselt und werden per Dependency Injection injiziert, sodass der Code unabhängig von der Zielplattform bleibt. Über die Konfigurationsoberfläche können Parameter wie die Server-URL, Authentifizierungs-API-Keys und Mapping-Tabellen von Mimetypes sowie Dateiendungen zu Zielanwendungen verwaltet werden.

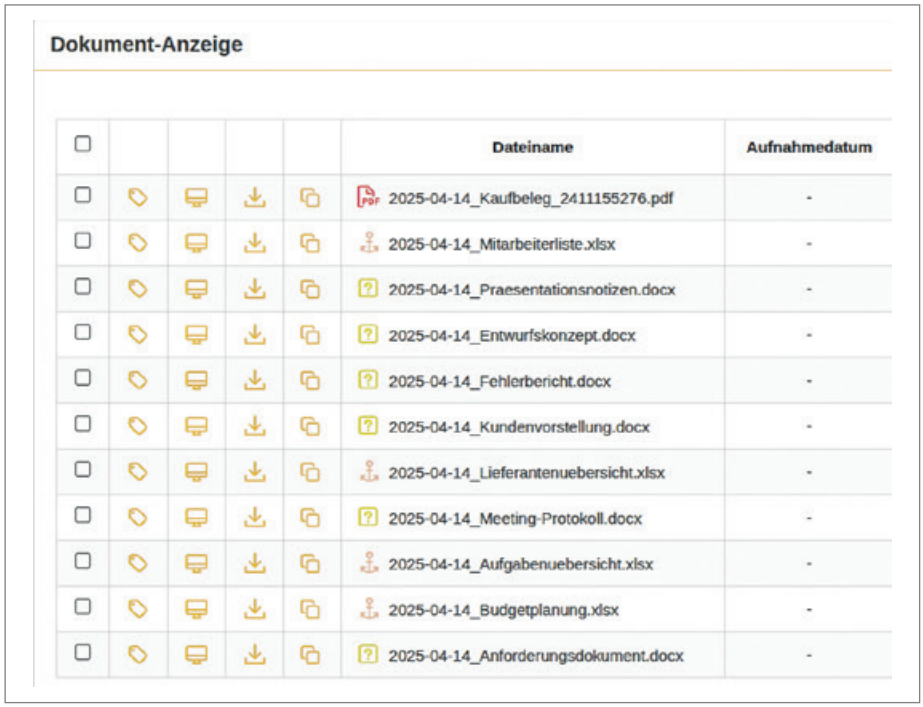

Abbildung 3: Screenshot der Dokumentenverwaltung (Quelle: Sebastian Reinhard)

Da über den Bridgeserver nur die FileID übertragen wird, lädt der Client die Datei mit dem dazugehörigen Hash über die API von der Datenbank herunter und legt die Datei in einem temporären Verzeichnis ab. Der Hash und weitere Metadaten werden in SQLite gespeichert. Vor dem Download werden noch Anfragen zu Dateien mit gewissen Dateitypen wie beispielsweise .exe, .bat oder etc. gefiltert und geloggt. Erst danach werden die Dateien gemäß des Mappings geöffnet.

Ein dedizierter Hintergrundthread setzt den .NET FileSystemWatcher ein, um das temporäre Arbeitsverzeichnis rekursiv zu überwachen. Dateioperationen wie Create und Modify werden in Echtzeit erkannt. Bei jedem Auslösen des Modify-Events wird ein Hash gebildet und dieser mit dem vorherigen Hash verglichen. Erst wenn sich der Hash unterscheidet, gilt eine Datei als geändert. Das hilft uns dabei, Änderungen zuverlässig zu erkennen, verhindert eine Eventflut durch Debouncing (typischerweise bei Office-Speichervorgängen) für die folgende Verarbeitung und hilft beim Erkennen von Konflikten bei der parallelen Bearbeitung einer Datei von mehreren Mitarbeitern.

Sollte eine Änderung an der Datei erkannt werden, wird die Datei über die API automatisch an die Datenbank übertragen. Neben der Datei wird auch der ursprüngliche Hash übertragen, um festzustellen, ob sich die Datei zwischenzeitlich geändert hat. Sollte dies der Fall sein, wird der Nutzer informiert, wer die Datei geändert hat und wann. Er hat dann die Entscheidung, ob die Datei überschrieben werden soll oder nicht.

Beim Starten des Tray-Clients löscht ein automatisches Cleanup-Verfahren temporäre Dateien anhand der Metadaten, wenn diese mehr als 24 Stunden nicht verändert wurden. Für die lokale Persistenz verwendet der Daemon eine SQLite Datenbank mit Write Ahead Logging (WAL), um parallele Zugriffe von der GUI und Hintergrundtasks zu ermöglichen.

Für Logging und Telemetrie kommt Serilog mit strukturiertem JSON Output zum Einsatz. Es gibt Unterstützung für Rolling Files. Das Benachrichtigungssystem informiert den Nutzer, wenn Dateien erfolgreich hochgeladen wurden, und stellt eine Bestätigungsbox bereit, falls es Konflikte gibt, damit der Benutzer über das weitere Vorgehen entscheiden kann. Das Benachrichtigungssystem nutzt plattformspezifische APIs wie Windows Toast Notifications.

Fazit

Die entwickelte Brückenlösung beweist, dass sich durch eine intelligente Architektur nicht nur technische Hürden überwinden, sondern auch handfeste betriebswirtschaftliche Vorteile erzielen lassen. Prozesse werden spürbar beschleunigt, da lästige Download- und Upload-Schritte entfallen, und die Gefahr von Versionskonflikten wird erheblich reduziert. Anwender arbeiten weiterhin in ihrer gewohnten Umgebung. Gleichzeitig bleibt die Datenhaltung zentral, sicher und versioniert, während die Architektur dank Bridgeserver, SignalR und einer schlanken Client-Komponente flexibel und plattformunabhängig einsetzbar ist. So entsteht ein reibungsloser Dokumentenfluss, der sowohl den Komfort der Nutzer als auch die Anforderungen der IT an Sicherheit und Konsistenz in Einklang bringt.

Kurzum: ein pragmatischer Weg, der die Web- und Desktop-Welt ohne Reibungsverluste zusammenzuführt.

Über den Autor

Sebastian Reinhard ist langjähriger Data-Warehouse-Entwickler im Finanzbereich mit Schwerpunkt auf Oracle-Datenbanken, SQL und PL/SQL sowie passionierter .NET-Entwickler und Linux Nerd.

Sebastian Reinhard
sebastian.reinhard@merlin-zwo.de