AUSGABE 01/2025

VISUAL BASIC ENTWICKLER

MAGAZIN FÜR DIE PROGRAMMIERUNG VON MICROSOFT OFFICE UND ANDEREN ANWENDUNGEN MIT VB.NET, VBA UND TWINBASIC

IN DIESEM HEFT:

André Minhorst Verlag

Impressum

Visual Basic Entwickler

Borkhofer Str. 17
47137 Duisburg

Redaktion: Dipl.-Ing. André Minhorst, Fach- und Sprachlektorat: Dipl.-Ing. André Minhorst

Tools, die uns Arbeit abnehmen – und wie wir sie steuern

Die Möglichkeiten, Software automatisiert für uns arbeiten zu lassen, nehmen rasant zu – und das ist gut so. Denn während die Welt digitaler, verteilter und vernetzter wird, wollen wir nicht mehr Zeit mit Routineaufgaben verlieren, sondern uns auf das konzentrieren, was uns wirklich voranbringt. Genau hier setzen die Beiträge dieser Ausgabe an.

In der vorherigen Ausgabe haben wir einen ersten Blick auf Tools wie Zapier geworfen – Plattformen, die SaaS-Dienste miteinander verbinden und automatisierte Workflows ermöglichen. Dieses Mal gehen wir einen entscheidenden Schritt weiter: Wir zeigen Dir, wie Du solche Automationen nicht nur grafisch zusammenklickst, sondern direkt aus VBA heraus steuerst. Das heißt: Deine Anwendungen werden zum Auslöser für automatische Aktionen in der Cloud.

Ein einfaches, aber wirkungsvolles Beispiel: Immer wenn eine bestimmte E-Mail eintrifft, wird automatisch eine Aufgabe in Microsoft To Do angelegt – ganz ohne Dein Zutun. Wie Du solche Abläufe per VBA triggert, zeigen wir Dir ausführlich im Artikel **Automationen mit Zapier per VBA starten** ab Seite 19. Und das ist nur der Anfang: In weiteren Ausgaben widmen wir uns intensiver der Verbindung klassischer VBA-Anwendungen mit modernen Automationsplattformen wie **Zapier**, **Make.com** oder **Power Automate**.

Mindestens ebenso spannend ist der Blick auf die eBay-API. Wer regelmäßig auf dem Marktplatz aktiv ist oder Kunden betreut, die dort verkaufen, kennt den Aufwand: Artikel einstellen, Bestellungen abgleichen, Rückgaben verwalten. Vieles davon lässt sich automatisieren – und zwar direkt aus Excel, Access oder einer eigenen VBA-Anwendung heraus. Wie Du die notwendigen Zugangsdaten holst und Dir ein Benutzer-Token besorgst, liest Du im Artikel **eBay per VBA steuern** ab Seite 31. Damit legen wir das Fundament für zahlreiche praxisnahe eBay-Integrationen in den kommenden Ausgaben.

Auch bewährte Themen kommen nicht zu kurz. Im Artikel **Zwischenablage für Texte programmieren** (ab Seite 58) zeigen wir Dir, wie Du die Windows-Zwischenablage gezielt ansteuerst – ob zum automatischen Einfügen häufig verwendeter Textbausteine oder zum Auslesen von Inhalten. Kleine Helfer mit großem Effekt.

Darüber hinaus erfährst Du, wie Du bestehende Kontextmenü-Befehle in Office-Anwendungen recycelst, statt sie mühsam neu zu bauen (**Office: Eingebaute Kontextmenübefehle recyclen**, ab Seite 51). Diese Art der Wiederverwendung spart nicht nur Zeit, sondern sorgt auch für konsistentes Verhalten in Deinen Anwendungen.

In Summe ist dies eine Ausgabe, die klassische VBA-Stärken mit modernen Automatisierungsansätzen verknüpft – genau das, was heute gefragt ist. Und wir machen hier noch lange nicht Halt: In den nächsten Heften bauen wir auf diesen Themen auf und zeigen Dir, wie Du APIs, Automationen und Office-Tools noch intelligenter miteinander verknüpfst.

Viel Freude beim Lesen und Programmieren!

Dein André Minhorst

Automation mit Zapier in der Praxis

Zapier ist neben Make.com oder Microsoft Power Automate ein weit verbreitetes Tool, mit dem wir Automationen definieren können. Mit diesen können wir sowohl verschiedenen SaaS-Produkte untereinander so verbinden, dass diese Daten austauschen und automatische Prozesse angestoßen werden. Wir können uns aber auch von unserer Anwendung aus in diese Prozesse einklicken, indem wir diese per VBA anstoßen. In diesem Artikel schauen wir uns jedoch erst einmal an, was Zapier ist, wie wir ein Konto bei Zapier anlegen und wir erste, einfache Automationen einrichten.

Zapier ist wie die Produkte, die wir damit automatisieren können, eine SaaS-Lösung – also Software as a Service. Diese zeichnen sich dadurch aus, dass sie in der Cloud betrieben werden. Dadurch entstehen viele Vorteile. Wir können über das Internet auf diese zugreifen, wodurch wie die Installation auf unserem Rechner sparen. Damit geht einher, dass wir von überall auf diese Dienste zugreifen können. Die SaaS-Produkte werden automatisch aktualisiert, sind skalierbar und wir zahlen die Nutzung entsprechend der geplanten Nutzung.

Die grundlegende Erläuterung von Automationstools wie Zapier oder Make.com haben wir bereits im Artikel **Automation mit Zapier, Make und Co.** (**www.vbentwickler.de/448**) erläutert.

Nun schauen wir uns am Beispiel von Zapier an, wie die ersten Schritte damit funktionieren.

Dazu rufen wir die Webseite **https://zapier.com** auf und finden direkt ein Beispiel dafür vor, wie die Automationen grafisch abgebildet werden (siehe Bild 1). Hier können wir direkt mit einem Klick auf den Link **Sign up** beginnen.

Vorab eine Information: In einem gewissen Rahmen können wir Zapier sogar kostenlos nutzen. Dabei sind wir allerdings eingeschränkt auf Automationen, die maximal aus zwei Schritten bestehen. Außerdem können wir nur 100 Aufrufe im Monat nutzen. Wir können jedoch bereits so viele verschiedene Automationen programmieren, wie wir möchten.

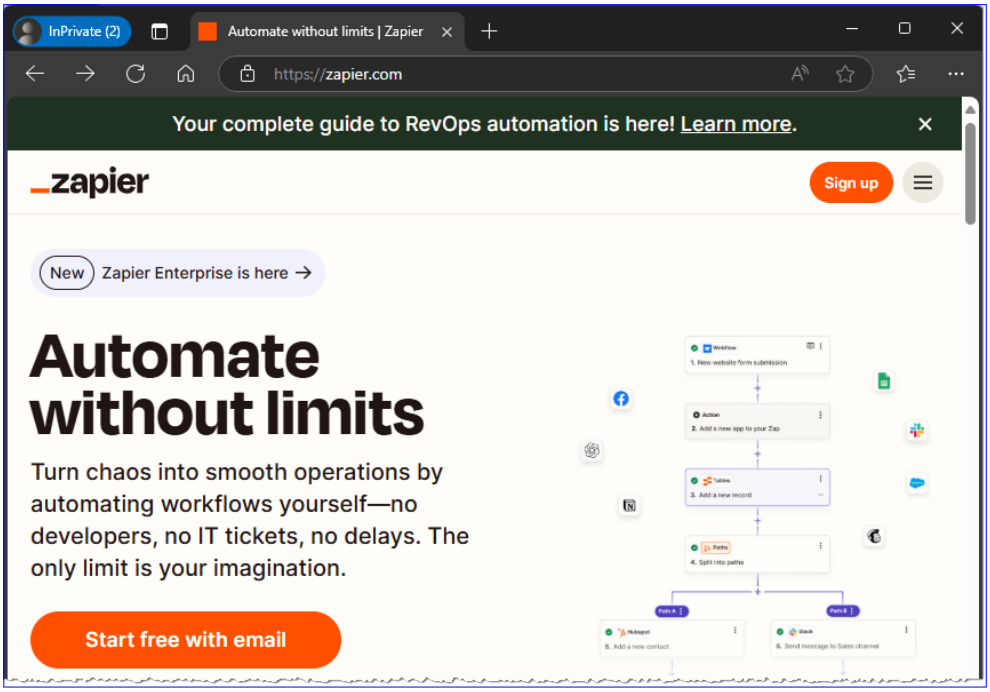

Bild 1: Startseite von Zapier

Zum Ausprobieren reicht das aus, und wenn sich die Nutzung in Grenzen hält, muss man gar nicht zu einem bezahlten Plan wechseln. 100 Aufrufe sind jedoch schnell erreicht, und neben der Beschränkung auf nur zwei Schritte pro Automation können wir bestimmte Trigger und Aktionen auch nur mit einem der höheren Pläne nutzen.

Warum soll ich dafür Geld ausgeben?

Ob man für solche Automationen Geld ausgeben sollte, ist eine berechtigte Frage. Diese kann man sich nur selbst beantworten. Zu prüfen ist aus unserer Sicht dabei: Welche Schritte, die ich bisher manuell ausgeführt habe, bräuchte ich nicht mehr zu erledigen, wenn ich diese durch entsprechende Automationen ersetzen würde? Wieviel Zeit würde es mich sparen? Wieviel Zeit würde ich außerdem sparen, weil die Automationen, einmal korrekt aufgesetzt, immer perfekt arbeiten – und ich keine Schritte mehr korrigieren müsste, weil ich sie manuell nicht korrekt ausgeführt habe?

Hier kommt es auch darauf an, wie viele und welche anderen SaaS-Tools man bereits nutzt oder gegebenenfalls noch hinzunehmen möchte.

Aktuell kostet der Professional-Plan von Zapier im Jahresabonnement 19,51 EUR pro Monat. Wenn Du Deinen Kunden 100 EUR pro Stunde in Rechnung stellst, was gängig ist, sollten Dir die Automationen also mindestens 12 Minuten Zeit im Monat ersparen. Wir können aus Erfahrung sagen, dass diese Ersparnis schnell erreicht ist. Allein das automatische Anlegen eines Kunden, der ein Produkt im Shop bestellt hat, in unserer Kundenplattform kostet immer ein paar Minuten. Diesen Vorgang haben wir komplett automatisiert und bereits damit haben wir die knapp 20 EUR im Monat eingespart.

Der Professional-Plan bietet gegenüber dem kostenlosen Plan einige Vorteile:

- Wir können mehr als zwei Schritte pro Automation definieren.
- Wir können auf alle Anwendungen zugreifen, die mit Zapier zusammenarbeiten.
- Wir können Webhooks nutzen. Das ist für uns besonders wichtig, wenn wir Zapier aus unserer Access-Anwendung heraus per VBA ansteuern wollen.

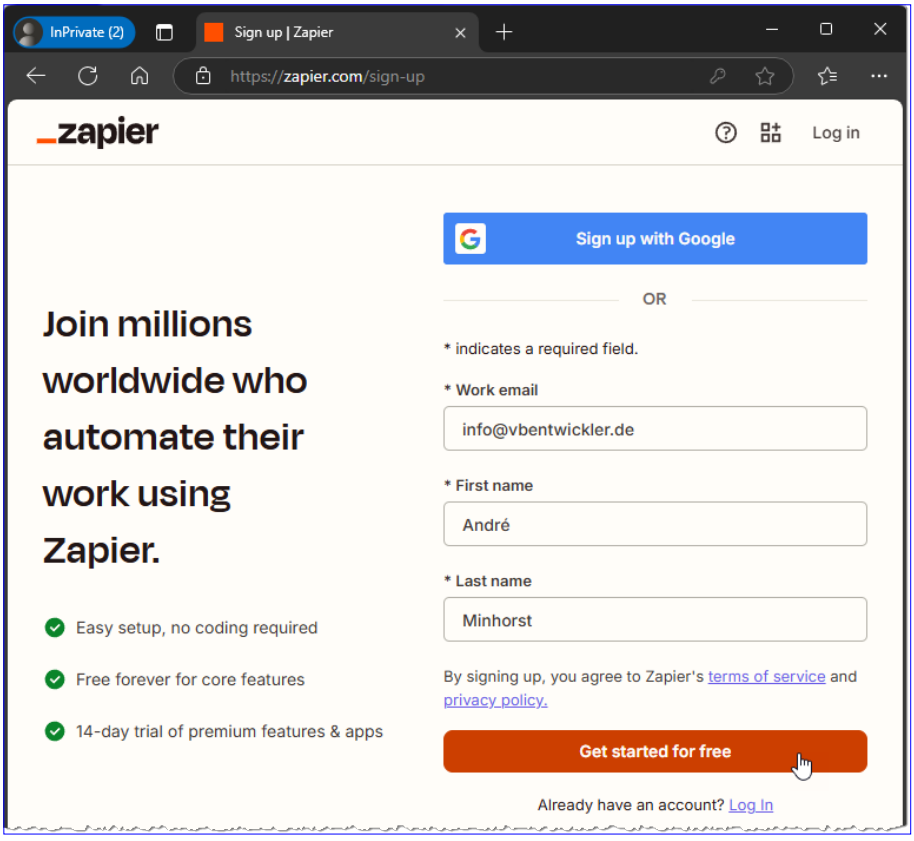

Bild 2: Registrieren bei Zapier

- Im Preis enthalten sind bis zu 750 Aufrufe. Weitere Aufrufe können hinzugebucht werden.

- Das sogenannte Polling erfolgt in einem Zeitintervall von zwei Minuten gegenüber 15 Minuten im kostenlosen Plan. Polling ist interessant, wenn wir beispielsweise auf Änderungen in den Daten einer SQL Server-Tabelle reagieren wollen.

Wenn Du Dich erstmalig bei Zapier registrierst, kannst Du ohnehin zunächst 14 Tage lang den Professional Plan testen.

Wenn Du Dich innerhalb dieser Zeit nicht entscheidest, diesen kostenpflichtig zu buchen, fällst Du einfach in den kostenlosen Plan mit den genannten Einschränkungen zurück.

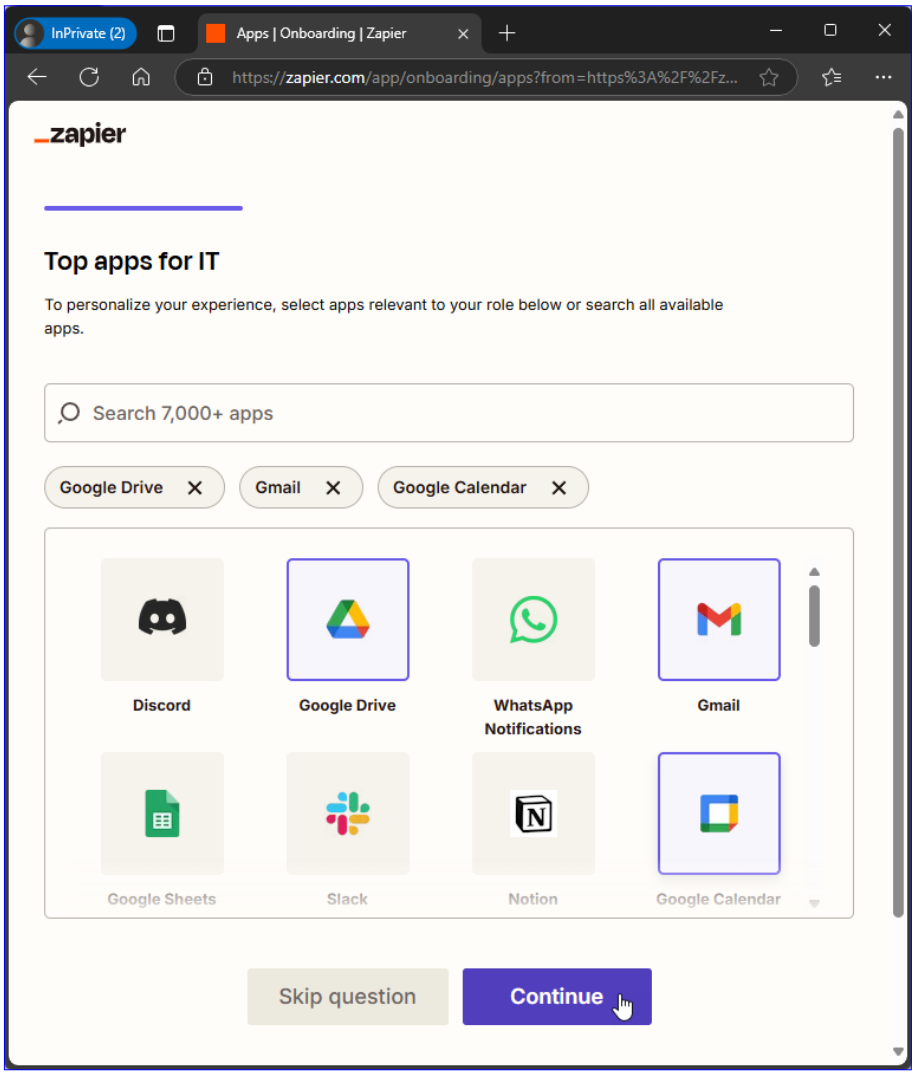

Bild 3: Apps für Zapier auswählen

Bei Zapier registrieren

Die Registrierung ist schnell erledigt. Dazu geben wir unsere E-Mail-Adresse sowie Vor- und Nachname ein und können direkt loslegen (siehe Bild 2). Im folgenden Schritt gegen wir noch ein Kennwort ein.

Danach folgen ein paar Fragen, mit denen Zapier das Nutzungserlebnis für Dich optimieren möchte. Hier können wir zum Beispiel die Anwendungen auswählen, die wir vermutlich oft in unseren Automationen nutzen werden. Außerdem kannst Du Dir bereits einmal einen Überblick verschaffen, welche Anwendungen alle verwendet werden können (siehe Bild 3). Deine Selektion dient nur dazu, die Apps zu wählen, die Dir vorrangig angezeigt werden sollen.

In diesem Dialog kannst du auch gezielt nach Anwendungen suchen. Falls Du zu Beginn dieses Artikels vielleicht noch gedacht hast, dass Du ja ohnehin alles über lokale Desktopanwendungen erledigst und gar keine SaaS-Produkte nutzt, wirst Du eventuell einige entdecken, die Du doch bereits verwendest.

Google Drive, Gmail, Google Calendar oder die entsprechenden Microsoft-Pendants sind nur einige

populäre Beispiele. Vielleicht nutzt Du ein Tool wie Notion zur Projektverwaltung oder Jotform oder Typeform, um professionele Formulare für Kundenumfragen zu nutzen? Oder Du verwendest ein E-Mail-Marketing-Tool wie ActiveCampaign oder Brevo? Nimm Dir ein paar Minuten Zeit, um einmal die verfügbaren Apps durchzusehen.

Vielleicht fallen Dir dabei bereits Möglichkeiten auf, wie Du diese durch Automationen miteinander verbinden kannst. Und wie gesagt: Die hier getroffene Auswahl ist keine Einschränkung, sondern dient nur der präferierten Anzeige von Apps in späteren Schritten.

Erstellen einer ersten Automation

Für das Erstellen der ersten Automation gibt es verschiedene Möglichkeiten. Zunächst einmal sehen wir jedoch, dass es nicht nur Automationen gibt, die unter Zapier die Bezeichnung »Zap« tragen (siehe Bild 4).

Wir finden noch weitere Elemente wie **Table**, **Interface**, **Chatbot** oder **Canvas**. Diese benötigen wir allerdings zunächst nicht. Am interessantesten sind hier noch die **Table**-Elemente: Damit können wir Daten, die während der Durchführung von Automationen anfallen, in Tabellen speichern. Auf diese können wir in der gleichen Automation oder auch in späteren Automationen zugreifen.

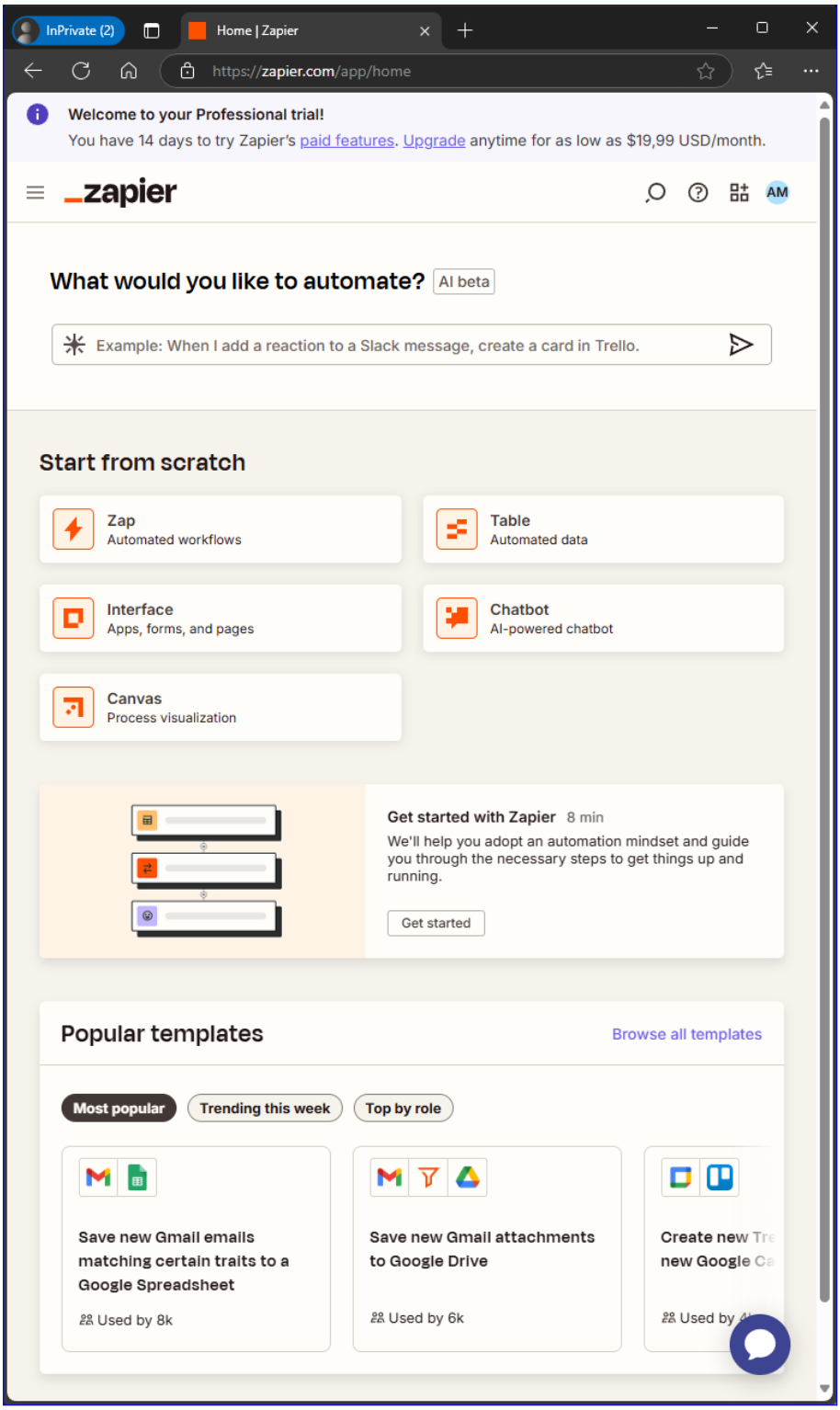

Bild 4: Elemente bei Zapier

Ganz unten unter Popular Templates finden wir Vorlagen für oft verwendete Automationen.

Da es Millionen von Nutzern gibt, die bereits Automationen mit Zapier programmiert haben, kann Zapier oft verwendete Kombinationen bereits als Vorlage bereitstellen. Beispiele:

- Hinzufügen von neue Gmail-E-Mails mit bestimmten Eigenschaften in ein Google Spreadsheet
- Speichern von Gmail-Anhängen in Google Drive
- Erstellen neuer Trello-Aufgaben, wenn neue Ereignisse im Google Calendar eingetragen werden
- Senden einer Gmail-E-Mail, wenn eine Zeile in einem Google Sheet aktualisiert wurde

Du siehst hier schon, dass die Vorlagen anhand der von uns präferierten Apps zusammengestellt wurden.

Wir können uns hier auch noch weitere Anregungen holen, indem wir die Automationen betrachten, die in dieser Woche oft verwendet wurden oder indem wir einfach alle Vorlagen durchsuchen.

Bei Null starten

Wir wollen allerdings mit einer neuen, leeren Automation starten, denn das Anpassen von Vorlagen wird in aller Regel nicht unseren Alltag beim Programmieren von Automationen widerspiegeln. Also klicken wir hier unter **Start from scratch** auf **Zap**. Damit landen wir in der Ansicht aus Bild 5. Hier sehen wir drei verschiedene Elemente:

- Ganz oben können wir dem Copilot von Zapier mitteilen, was für eine Art von Automation wir erstellen wollen. Hier kannst Du einfach einmal ausprobieren, was geschieht, wenn Du verschiedene Ideen eingibst.
- Darunter sehen wir ein Element mit der Beschriftung **Trigger**. Dies ist der auslösende Schritt in unserer Automation.
- Noch ein Stück weiter unten finden wir ein weiteres Element, diesmal mit der Beschriftung **Action**. Damit legen wir fest, welche Aktion ausgeführt werden soll, nach dem der Trigger aus dem ersten Element ausgelöst wurde.

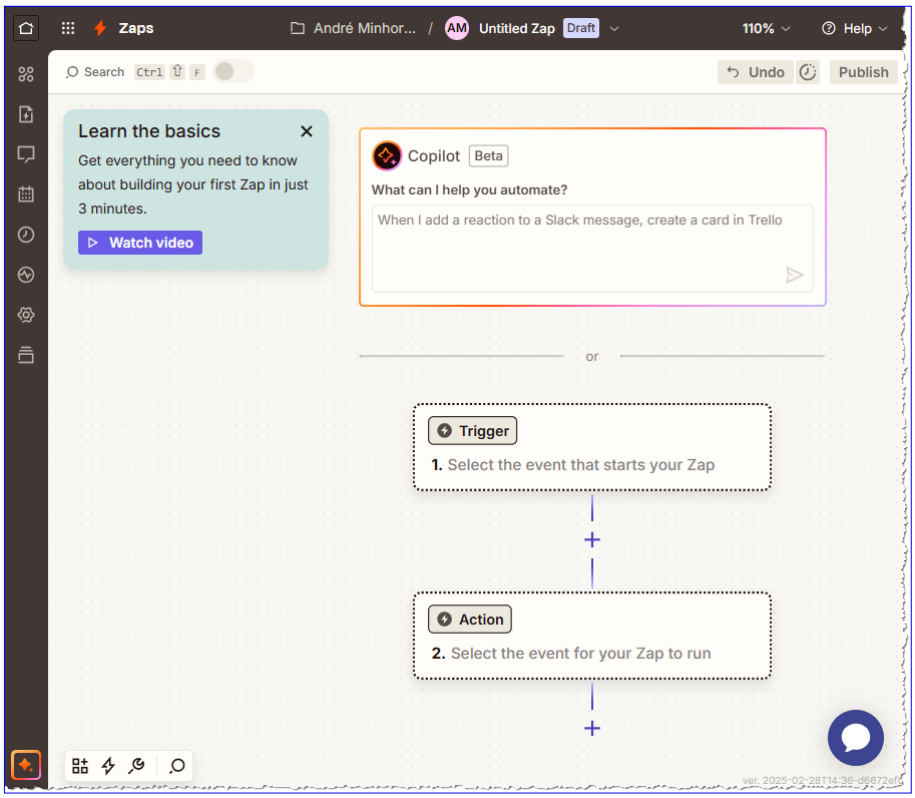

Bild 5: Start mit einem neuen, leeren Zap

Dem Copilot können wir beispielsweise die folgende Anweisung mitgeben:

```
Wenn ein Kunde ein Anmeldeformular mit Jotform ausfüllt,
soll ein neuer Datensatz in der Tabelle tblInteressenten
in der SQL Server-Datenbank angelegt werden.
```

Zur Erklärung: Jotform ist ein SaaS-Produkt, mit dem wir benutzerdefinierte Formulare erstellen können, um Umfragen zu erstellen, Kundendaten einzusammeln und mehr.

Der Copilot wird nun, soweit es ohne weitere Eingaben durch uns möglich ist, die Vorbereitungen für das Erstellen der Automation durchführen. In diesem Fall erkennt er korrekt die zu verwendenden Apps sowie das entsprechende Trigger-Ereignis und auch die passende App für die durchzuführende Aktion (siehe Bild 6).

Links sehen wir das Feedback des Copilots. In der Mitte hat Zapier automatisch den Trigger und die Aktion angelegt. Hier können wir uns nun einmal die verschiedenen Bestandteile dieser Elemente ansehen.

Bestandteile des Triggers

Die Bestandteile des Triggers sehen wir im rechten Bereich, der als Icon das Jotform-Icon enthält und mit New Submission bezeichnet ist.

Darunter sehen wir drei Bereiche:

- **Setup**: Hier legen wir fest, welche App verwendet wird, welches Trigger Event dieser App genutzt werden soll und von welchem Account der auslösenden App auf Trigger gelauscht werden soll. Dieser muss noch ausgewählt werden, bevor wir überhaupt zum nächsten Schritt wechseln und die weitere Konfiguration vornehmen können.

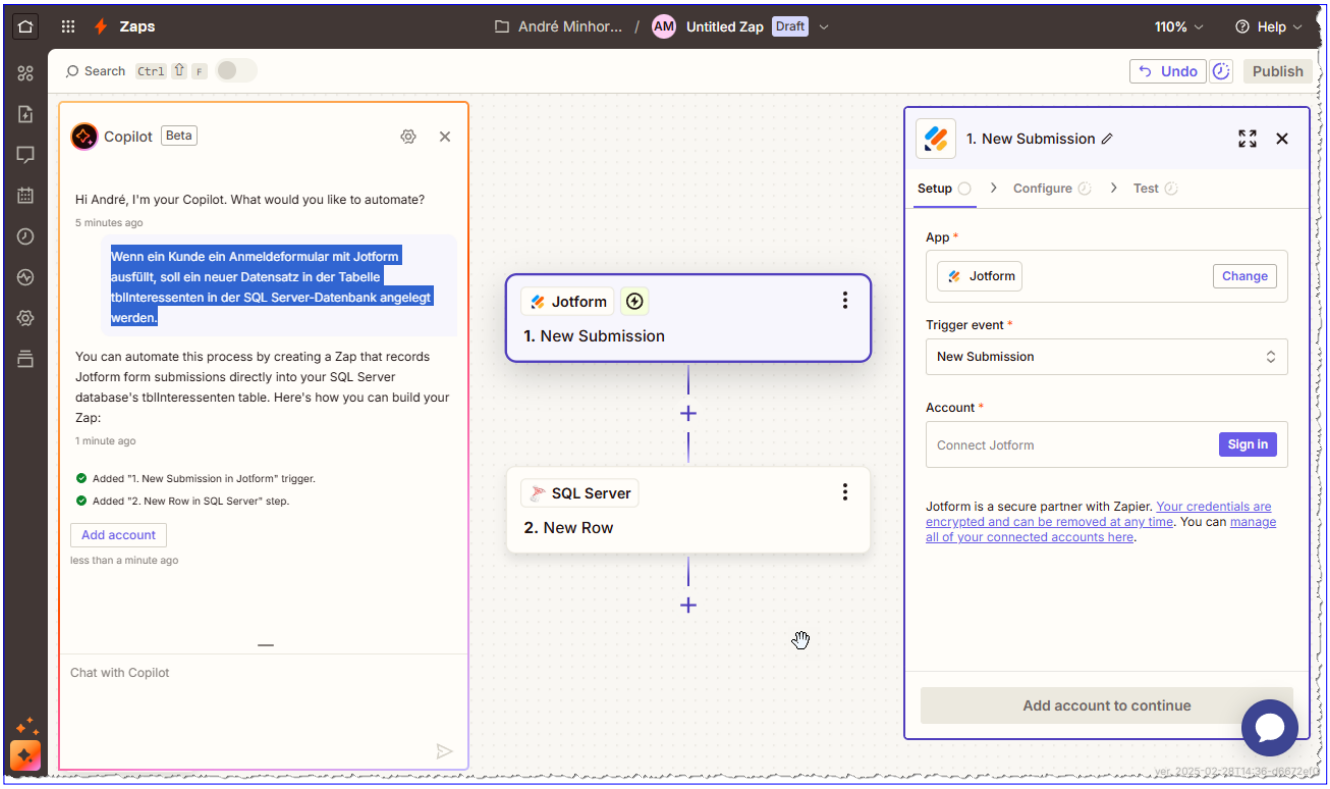

Bild 6: Anpassen eines durch den Copilot erzeugten Zaps

- **Configure**: Im zweiten Bereich legen wir genau fest, was geschehen soll.
- **Test**: Im dritten Bereich können wir den Trigger testen.

Diese Elemente schauen wir uns im Anschluss genauer an. Zunächst werfen wir noch auf die entsprechenden Informationen zur durchzuführenden Aktion.

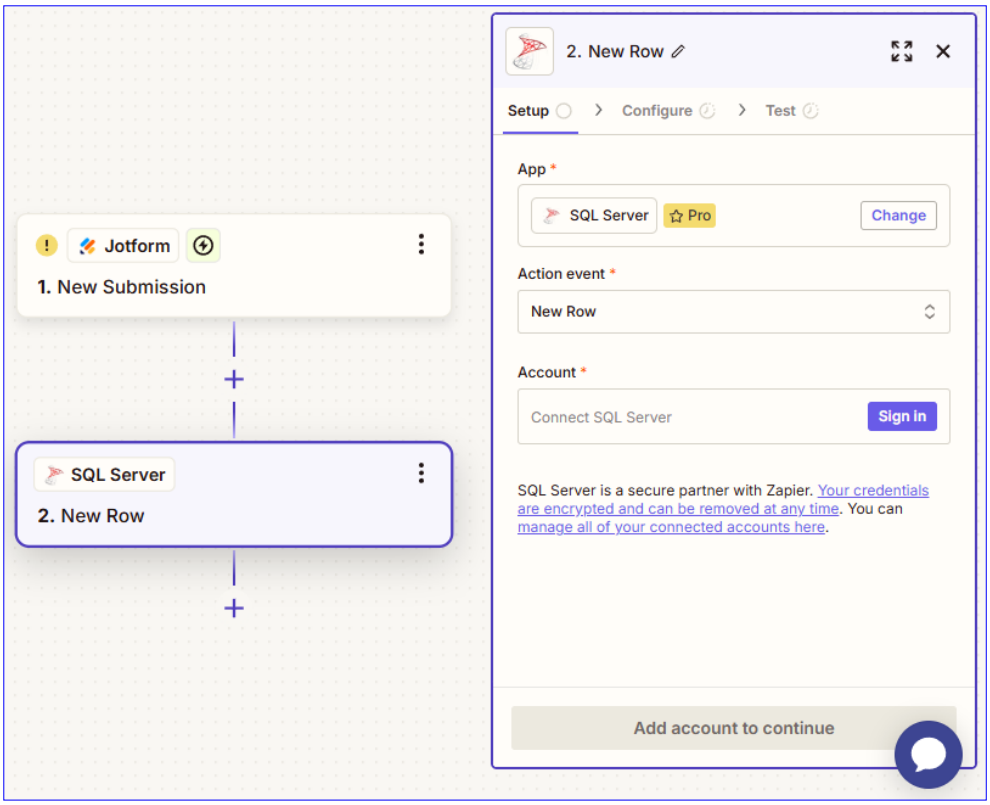

Bild 7: Anpassen der Aktion des Zaps

Bestandteile der Aktion

Klicken wir auf das untere Element mit dem Titel **SQL Server**, sehen wir im rechten Bereich die gleichen drei Bereiche **Setup**, **Configure** und **Test** (siehe Bild 7).

Hier wurde im Bereich **Setup** bereits **SQL Server** als App ausgewählt. Außerdem wurde mit **New Row** auch bereits die durchzuführende Aktion definiert. Unten müssen wir nun noch den SQL Server-Account auswählen, um weiter zu den nächsten Schritten navigieren zu können.

Eine komplett leere Automation anlegen

Wir wollen allerdings diesen Vorschlag nicht weiterbearbeiten, sondern eine ganz neue Automation anlegen. Dazu gehen wir zur Seite **Home** und klicken oben auf **Create** und dann auf **Zaps** (siehe Bild 8).

Hier finden wir wieder die beiden leeren Elemente **Trigger** und **Action** vor und klicken nun auf **Trigger**.

Dies öffnet den Bereich aus Bild 9 zur Auswahl der Anwendung, die den Trigger auslösen soll. Hier sehen wir die Anwendungen nach verschiedenen Kriterien gruppiert. Auf der linken Seite sehen wir die Anwendungen, wie wir sie zu Beginn aus den Vorschlägen

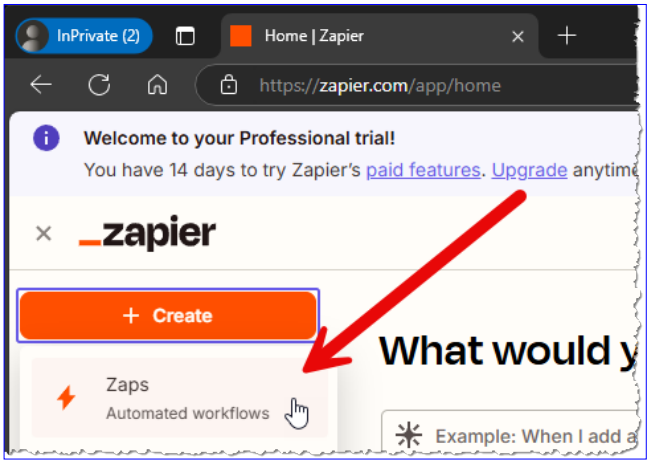

Bild 8: Erstellen eines neuen Zaps

Automationen mit Zapier per VBA starten

Zapier ist ein Tool, mit dem wir SaaS-Anwendungen für verschiedene Bereiche automatisieren können. Das geschieht nach dem Prinzip, dass eine solche Anwendung eine Automation triggert und dadurch Aktionen mit der gleichen oder auch mit anderen Anwendungen angestoßen werden. Wie zum Beispiel das Eintragen einer Aufgabe in eine Aufgabenliste, wenn eine neue E-Mail eingegangen ist. Wie haben in weiteren Artikel bereits gezeigt, wie wir solche Automationen mit Tools wie Zapier realisieren können. Aber welchen Bezug gibt es zum eigentlichen Thema dieses Magazins, nämlich dem Programmieren mit Sprachen wie VBA, VB6, twinBASIC et cetera – abgesehen davon, dass wir diese Automationen als Selbstständiger, Freiberufler, aber auch als Angestellter für unser Unternehmen gewinnbringend einsetzen können? Genau: Wie können diese Automationen natürlich auch per Code triggern. Wie das im Falle von Zapier funktioniert, erklären wir im vorliegenden Artikel.

Wenn Du Dich bereits mit Zapier zur Automation von SaaS-Anwendungen beschäftigt hast, kannst Du direkt in diesen Artikel einsteigen. Falls nicht, empfehlen wir zuvor die Lektüre der folgenden Artikel:

- **Automation mit Zapier, Make und Co.** (**www.vbentwickler.de/448**)
- **Automation mit Zapier in der Praxis** (**www.vbentwickler.de/449**)

Hier findest Du einige Grundlagen allgemein zum Thema Automatisierung und ein erstes Beispiel für eine Automation auf Basis von Microsoft Outlook und Microsoft To Do.

Automation per VBA nutzen

Wie Du in den obigen Artikeln erfährst, gibt es verschiedene Möglichkeiten, wie Du Automationen mit den verschiedenen Tools von VBA aus starten kannst. Für das Tool Zapier sind die Möglichkeiten ein klein wenig eingeschränkter, denn wir können damit keine Ergebnisse über die Erfolgsmeldung hinaus als Rückgabewert erhalten. Bei dem anderen Tool Make.com sieht dies anders aus – hier können wir explizit festlegen, welche Informationen nach dem Durchlaufen einer Automation an den aufrufenden VBA-Code zurückgeliefert werden sollen. Dies schauen wir uns in weiteren Artikeln an.

Dafür liefert Zapier wieder mehr Möglichkeiten bei der Zusammenarbeit mit dem SQL Server und anderen Datenbanksystemen, denn hier können wir mit dem sogenannten Polling auch auf Änderungen im Datenbestand reagieren.

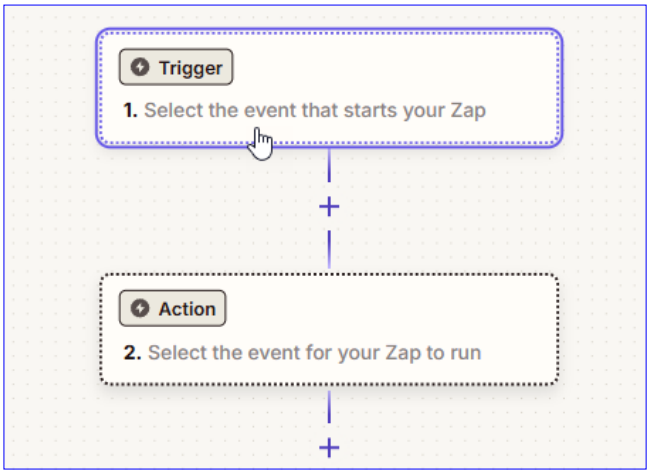

Bild 1: Diesen Trigger wollen wir per VBA ansteuern.

VBA und Software wie Access, Excel, Outlook und Co. als Trigger für Zapier

Wie in den oben angegebenen Artikeln beschrieben, haben Automationen einen Trigger und anschließend auszuführende Aktionen. Außerdem hat jedes namhafte SaaS-Tool Schnittstellen, die man von Zapier aus nutzen kann.

Wenn wir einen neuen Zap anlegen, finden wir hier die beiden mindestens notwendigen Elemente **Trigger** und **Action** (siehe Bild 1).

Klicken wir hier auf Trigger, erscheint der Dialog zur Auswahl der verfügbaren Trigger (siehe Bild 2). Hier können wir in der Suchleiste **Visual Basic**, **Microsoft Access** oder andere Office-Anwendungen eingeben – wir finden keinen passenden Trigger, der uns weiterhilft. Es gibt zwar welche für Outlook, Excel oder Word, aber dies betrifft die Online-Anwendungen, nicht die gleichnamigen Desktop-Programme.

Die Lösung: Webhooks

Wir sehen hier jedoch einen Eintrag namens Webhooks. Das ist unser Weg zum Triggern von Zapier-Automationen per Visual Basic.

Webhooks sind nichts anderes als URLs, die von Zapier zur Verfügung gestellt werden und die wir von beliebigen anderen Anwendungen aus aufrufen können.

Diese können wir mit Visual Basic genauso aufrufen, wie wir es bereits mit verschiedenen Rest-APIs gemacht haben. Dazu

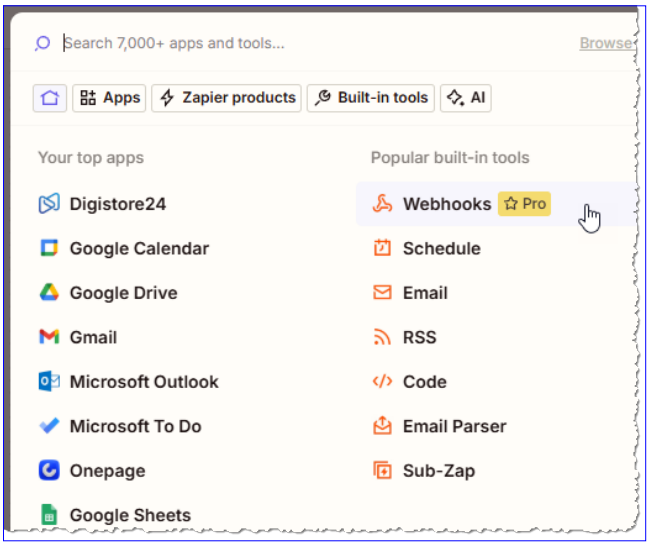

Bild 2: Einen Trigger für VBA oder eine der Office-Anwendungen finden wir nicht. Dafür aber einen für Webhooks.

sind grundsätzlich erst einmal nur wenige Zeilen Code notwendig.

Bevor wir uns dem Code zuwenden, führen wir aber erst einmal die notwendigen Arbeiten mit Zapier durch. Nach einem Klick auf den Eintrag **Webhooks** finden wir drei verschiedene Trigger vor (siehe Bild 3). Diese können wir wie folgt verwenden:

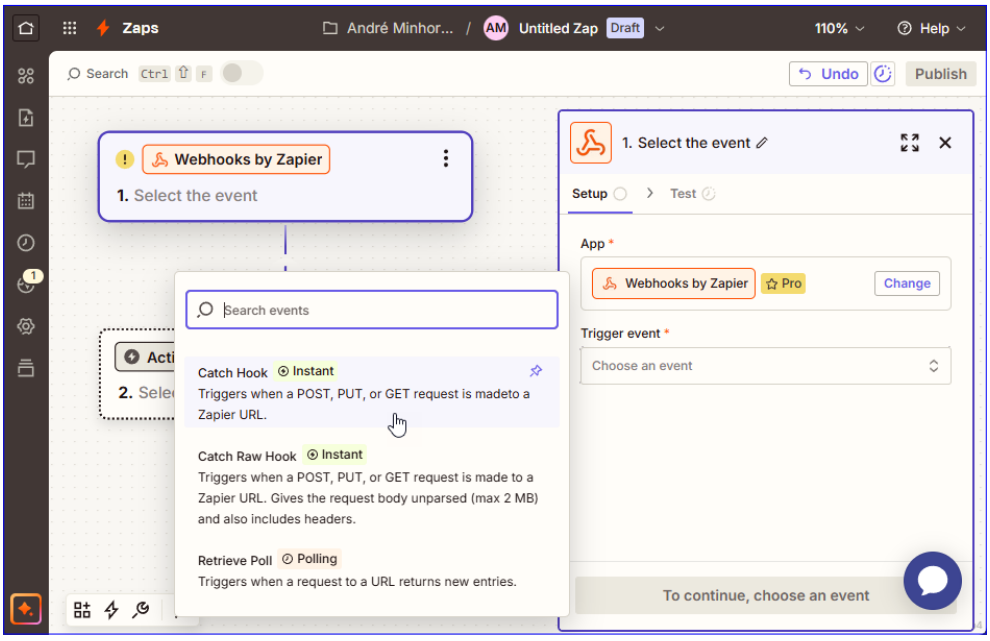

Bild 3: Verschiedene Webhooks

- **Catch Hook**: Hier erhalten wir eine URL von Zapier, die wir von außen aufrufen und so einen Trigger auslösen können. Mitgelieferte Daten beispielsweise im JSON-Format werden automatisch geparst und die enthaltenen Daten können in weiteren Schritten direkt verwendet werden.

- **Catch Raw Hook**: Hier erhalten wir ebenfalls eine Zapier-URL. Der Unterschied ist, dass mitgelieferte Daten nicht direkt von Zapier verarbeitet werden. Damit können wir beispielsweise vollständige Dateien an Zapier schicken.

- **Retrieve Poll**: Hier geben wir einen externen Webhook an, also eine URL einer weiteren Anwendung, die regelmäßig abgerufen werden soll.

Webhook mit Catch Hook verwenden

Wir schauen uns zunächst die Variante mit **Catch Hook** an. Damit können wir zum nächsten Schritt namens Configure weitergehen. Hier sehen wir die Option **Pick off a Child Key**.

Damit können wir festlegen, dass nur bestimmte Elemente eines eventuell übergebenen JSON-Dokuments weiterverarbeitet werden sollen. Aktuell wollen wir dies nicht nutzen und klicken direkt auf **Continue**.

Damit gelangen wir zum Bereich **Test**, wo wir die URL vorfinden, die wir zum Auslösen des Triggers nutzen können (siehe Bild 4).

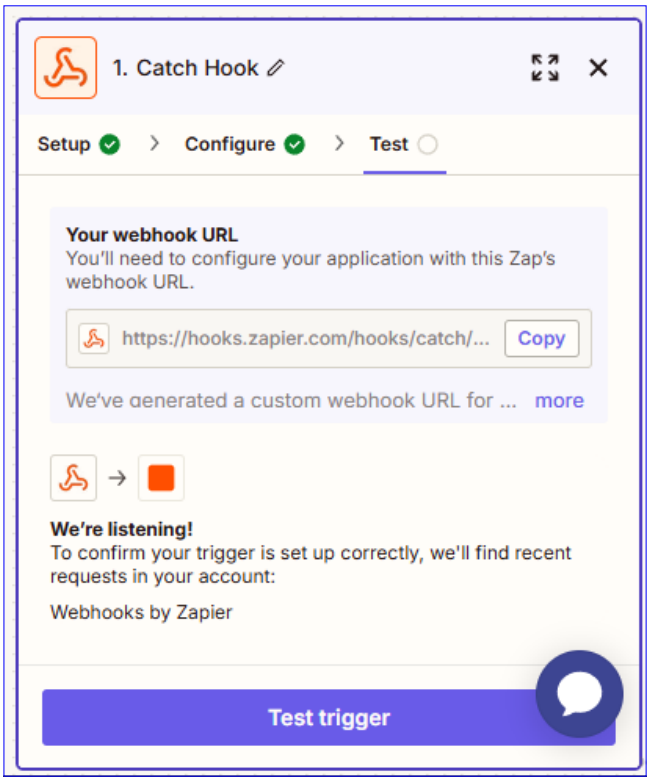

Bild 4: Die fertige Webhook-URL

Klicken wir nun auf **Test trigger**, erhalten wir als Antwort **No request found**. Das ist logisch, denn diese URL wurde speziell für uns definiert und wir haben sie noch nicht aufgerufen. Das können wir nun erst einmal über den Webbrowser erledigen. Dazu kopieren wir die URL in die Zwischenablage und fügen sie in die URL-Leiste des Browsers ein.

Die Antwort sehen wir in Bild 5. Wichtig ist das unterste Name-Wert-Paar namens **status**, das den Wert **success** liefert.

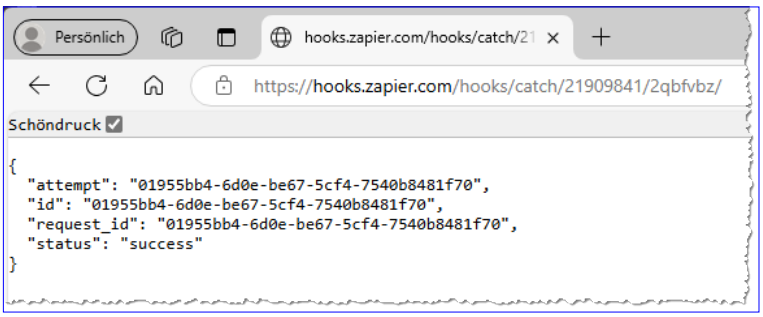

Bild 5: Aufruf mit der Antwort von Zapier

Wechseln wir nun zurück nach Zapier, können wir erneut testen und erhalten auch hier einen Eintrag namens **request A**.

Daten an Zapier übergeben

Auch wenn wir einfach nur eine Automation starten können, indem wir die URL des Webhooks aufrufen, wollen wir vermutlich in den

meisten Fällen Daten übergeben, die in den folgenden Aktionen weiterverarbeitet werden sollen.

Dazu haben wir zwei Möglichkeiten:

- wir hängen die Informationen als URL-Parameter an die URL an oder
- wir fügen dem Aufruf eine JSON-Datei mit den benötigten Informationen hinzu.

Unabhängig davon, welche Variante wir nutzen, ist es wichtig, dass wir die Informationen immer in der gleichen Form übergeben und dass wir diese auch beim Einrichten der Automation so angeben.

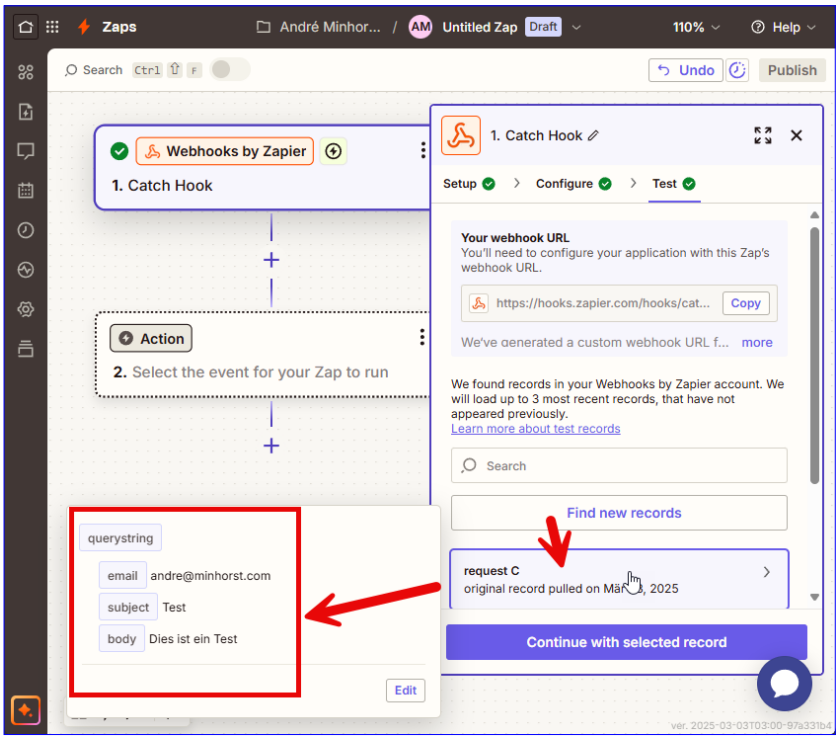

Bild 6: Ein Aufruf mit einem Parameter

Wir können in den folgenden Aktionen nur auf Daten zugreifen, die in einem der vorherigen Testaufrufe auch übergeben wurden. Wir schauen uns das an einem Beispiel an und übergeben einige Parameter, den wir an die URL anhängen – hier **email**, **subject** und **body**. Die URL sieht nun wie folgt aus:

```
https://hooks.zapier.com/hooks/
catch/21909841/2qbfvbz/?email=andre@minhorst.com&sub-
ject=Test&body=Dies%20ist%20ein%20Test
```

Nun können wir in Zapier auf **Find new records** klicken und sehen den neuen Eintrag. Klicken wir diesen an, erscheint ein Bereich, der alle mit diesem Aufruf gelieferten Informationen anzeigt (siehe Bild 6).

Dies können wir nun als Basis für eine weitere Aktion nutzen. Dazu klicken wir auf **Continue with selected record** und landen in der Auswahl der App, die wir für den nächsten Schritt nutzen wollen.

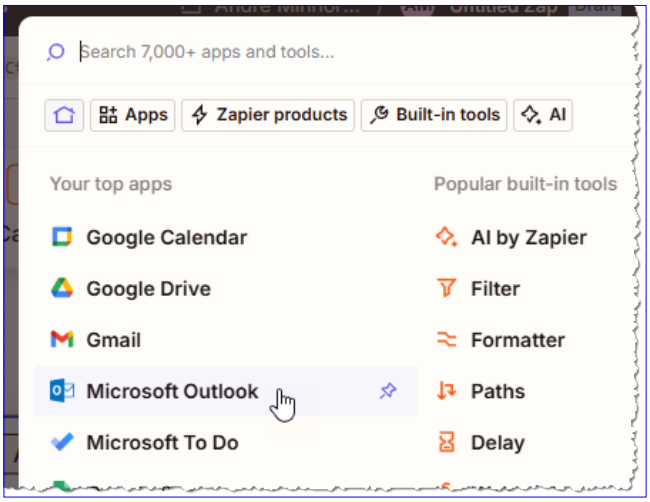

Bild 7: Auswahl von Microsoft Outlook für die Aktion

E-Mail per Outlook versenden

Wie den Parametern unschwer zu entnehmen ist, wollen wir als Aktion eine E-Mail versenden.

Dazu wählen wir im nächsten Schritt, wo es um die App für die folgende Aktion geht, Microsoft Outlook aus (siehe Bild 7).

Nun wählen wir unter **Action event** den Eintrag **Send Email** aus (siehe Bild 8).

Unter Account wählen wir nun den Outlook-Account aus, mit dem wir die E-Mails versenden wollen. Wenn Du bereits das Beispiel aus dem Artikel **Automation mit Zapier in der Praxis** (**www.vbentwickler.de/449**) ausprobiert hast, findest Du hier bereits einen Account vor.

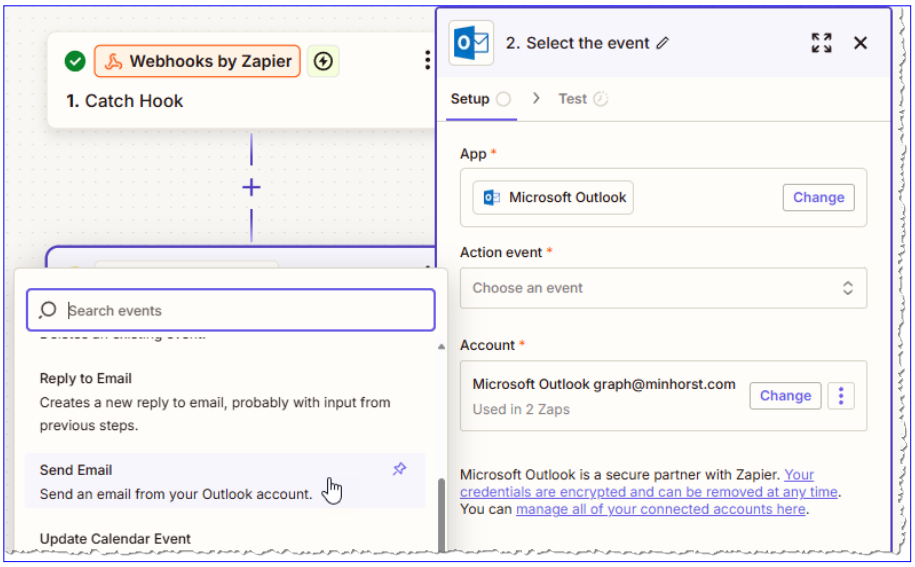

Bild 8: Anlegen der Aktion zum Senden einer E-Mail

Anschließend weisen wir der Aktion **Send Email** die Parameter zu, die wir der URL hinzugefügt haben (siehe Bild 9):

- Für **From Email** legen wir den Absender als Zeichenkette an.
- Unter **To Email(s)** wählen wir **Querystring Email** aus.
- Entsprechende Platzhalter tragen wir für **Subject** und **Body** ein und stellen außerdem die Eigenschaft **Body Format** auf **Text** ein.

Mit einem Klick auf **Continue** landen wir wieder im Test-Bereich. Hier können wir nun mit einem Klick auf **Test Step** einen erneuten Test durchführen.

Wenn wir alles korrekt konfiguriert haben, erscheint kurz danach bereits die versendete E-Mail in unserem Posteingang. Damit sind die ersten Voraussetzungen erfüllt, um das Auslösen des Triggers per VBA zu realisieren

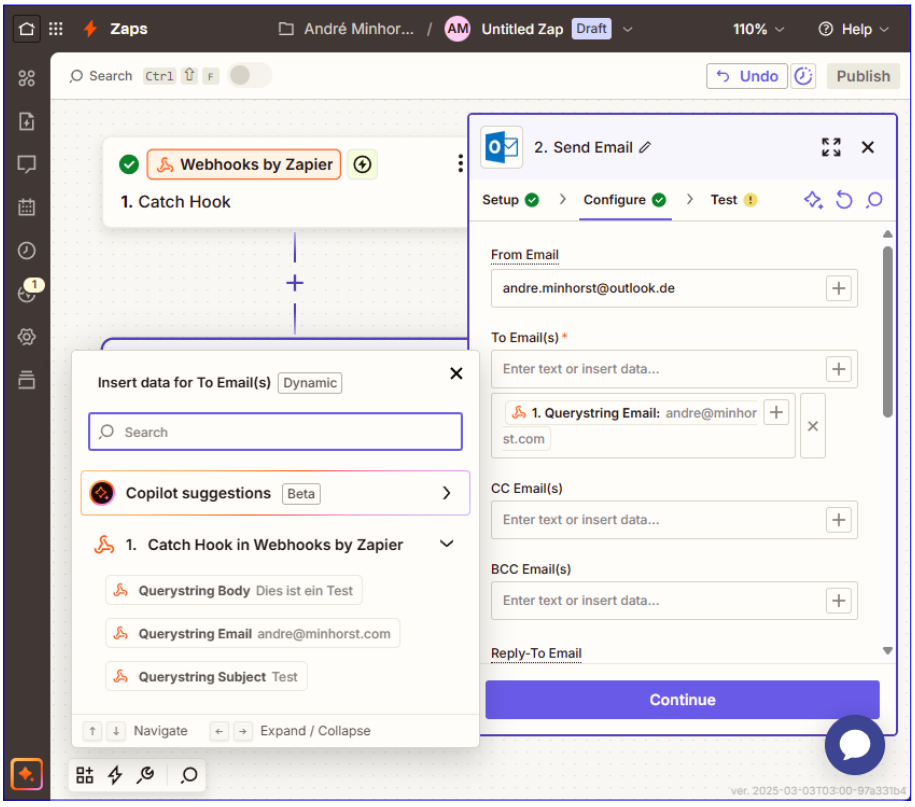

Bild 9: Hinzufügen der Parameter aus dem Trigger

eBay per VBA steuern: Zugangsdaten holen

Im ersten Teil einer Artikelreihe zum Thema Steuerung von eBay mit VBA zeigen wir, wie man einen Entwickler-Account anlegt, eine neue Anwendung bei eBay erstellt, grundlegende Zugriffsdaten holt und die für die Authentifizierung im Kontext eines bestimmten Benutzerkontos notwendigen Informationen holt – hier speziell das Authentifizierungstoken. Damit schaffen wir die Basis, um per VBA auf die Rest API von eBay zuzugreifen. In weiteren Artikeln beschreiben wir den Zugriff und wie wir verschiedene Operationen wie das Anlegen von Angeboten realisieren können.

Um eBay mit VBA zu steuern, brauchen wir als Erstes ein Konto im **ebay developers program**. Um dorthin zu gelangen, reicht eine Websuche nach den Begriffen **ebay developer**.

Für die Registrierung werden die folgenden Informationen benötigt, die wir in ein Formular wie in Bild 1 eintragen: Benutzername, Kennwort, E-Mail-Adresse und Telefonnummer. Nachdem wir diese eingetragen und die Schaltfläche **Join** betätigt haben, landen wir auf der nächsten Seite im Registrierungsprozess.

Hier erhalten wir den Hinweis, dass eBay uns eine E-Mail zugesendet hat. Diese enthält einen Bestätigungslink, mit dem wir den Vorgang fortsetzen können.

Nachdem wir diesen betätigt haben, landen wir auf der Willkommen-Seite des Entwicklerprogramms und können uns dort mit unseren Daten anmelden (siehe Bild 2).

Die Freude währt in der Regel nur kurz, denn gegebenenfalls bekommen wir hier die Meldung, dass die Regis-

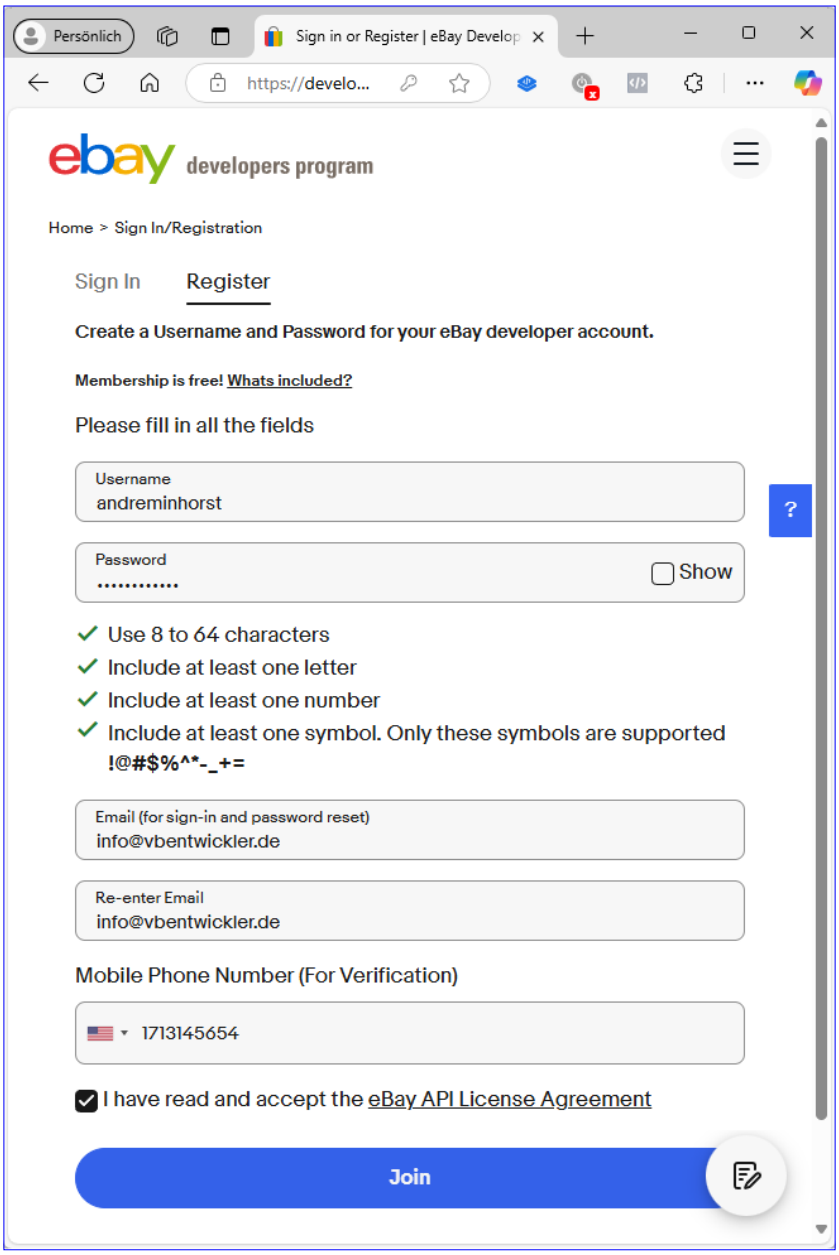

Bild 1: Registrieren für das Developer-Program von eBay

trierung nun überprüft wird und in ca. 24 Stunden freigeschaltet wird.

Einen Tag später können wir uns dann ohne Probleme beim eBay-Entwicklerprogramm anmelden.

Erstellen einer neuen Anwendung

eBay macht es uns dann erst einmal einfach: Da wir das Entwickler-Konto gerade eröffnet haben und folglich noch keine Anwendungen hinzugefügt haben, ohne eine Anwendung aber der Developer-Account keinen Sinn macht, landen wir direkt auf der Seite zum Erstellen einer Anwendung.

Dies beginnt mit der Eingabe eines Anwendungstitels, der vor allem eine Bedingung erfüllen muss: eBay darf nicht im Namen vorkommen (siehe Bild 3).

Wir entscheiden uns für **amvBay** und noch während wir diesen Namen eingeben, erscheinen bereits verschiedene Links für weitere Möglichkeiten (siehe Bild 4). Diese sind aufgeteilt in Sandbox und Production. Was bedeutet das?

- In der **Sandbox** können wir testen, ohne dass wir etwas an bestehenden Konten, Angeboten et cetera kaputtmachen.

- **Production** heißt: Hier arbeiten wir mit echten Benutzern und Angeboten. Hier sollten wir keine Fehler mehr produzieren und beispielsweise beim Experimentieren mit der **Löschen**-Funktion auf Kundendaten losgehen.

Beide Settings sind gleich aufgebaut, für beide benötigen wir entsprechenden Daten für den Zugriff. Diese heißen in diesem Fall **Keysets**.

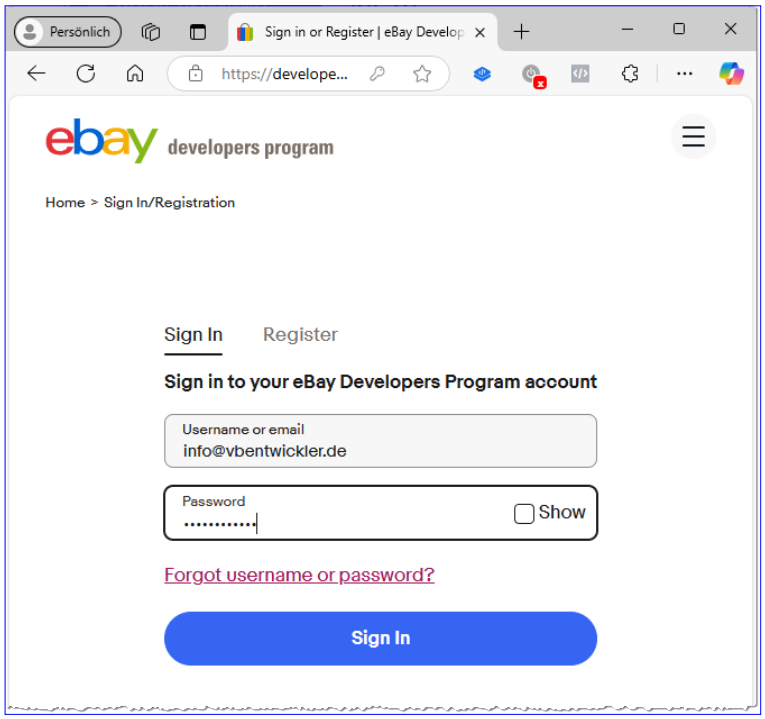

Bild 2: Anmelden beim Developer-Program von eBay

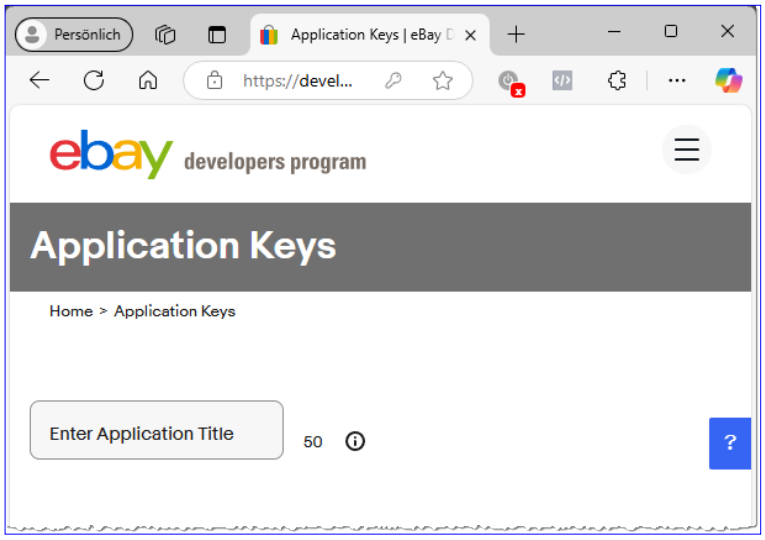

Bild 3: Erstellen einer ersten Anwendung

Wir können hier Keysets erstellen oder weitere Keysets anfragen. Was hat es damit auf sich? Wir klicken also erst einmal auf den Link **Create a keyset** unter Sandbox.

Daraufhin erscheint der Bereich **Confirm the Primary Contact**, wo wir weitere Daten wie Name, E-Mail und

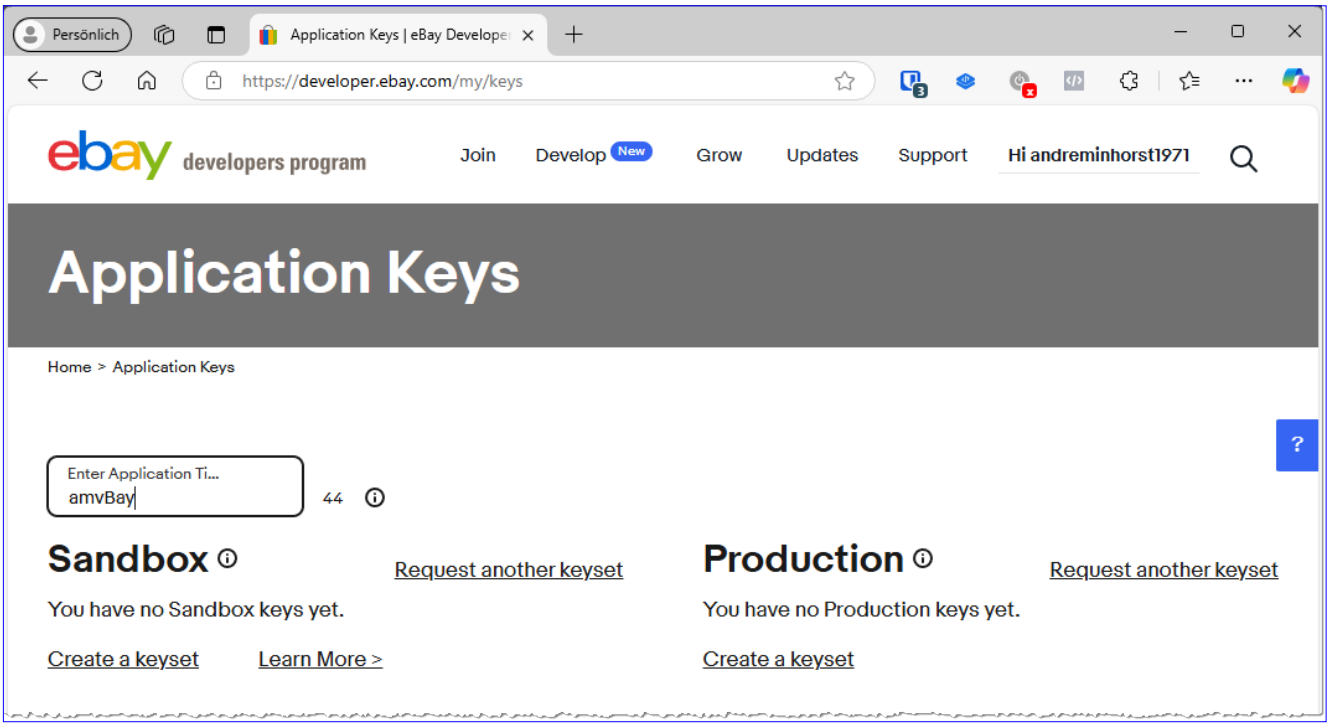

Bild 4: Noch während der Eingabe des Namens erscheinen die Optionen zum Erstellen von Keysets.

Telefonnummer eingeben beziehungsweise bestätigen (siehe Bild 5).

Ein Klick auf **Continue to Create Keys** lässt uns wieder zur aufrufenden Seite zurückkehren, wo wir nun allerdings einige neue Informationen vorfinden (siehe Bild 6).

Dabei handelt es sich um unser Keyset, mit dem wir nun fortfahren können:

- **App ID**
- **Dev ID**
- **Cert ID**

Diese Informationen können wir nun bereits in einem VBA-Modul sichern, am einfachsten in Konstanten. Das sieht wie folgt aus:

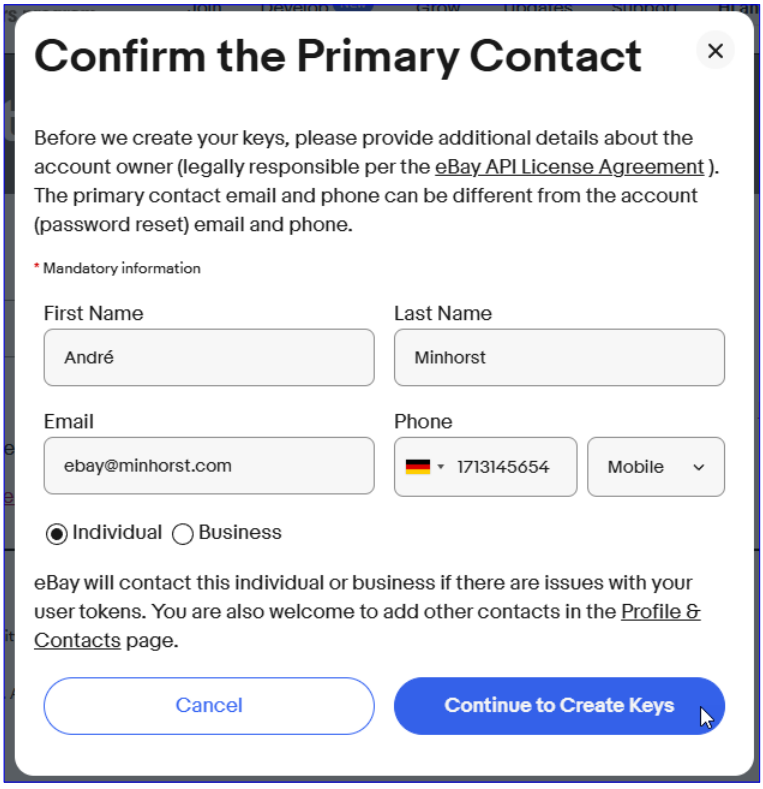

Bild 5: Eingabe weiterer Daten

```
Const cStrAppID As String = "AndrMinh-amvBay-
xxx-xxxxxxxxx-3e3a632f"
Const cStrDevID As String = "e7df1909-ee64-
xxxxxxxx-32138d2b1899"
Const cStrCertID As String = "SBX-4b5baea0d7a2-
xxxx-xxxx-8644-6824"
```

Token holen

Damit haben wir allerdings noch nicht alles, was wir benötigen. Das Wichtigste ist das Authentifizierungstoken. Hier ist eine kurze Erklärung nötig:

Wir erstellen hier eine Anwendung, für die eigentlich zwei Konten erforderlich sind: das Entwicklerkonto, das wir soeben erstellt haben, sowie das eigentliche eBay-Konto.

Um eBay komplett fernsteuern zu können, müssen wir uns ein Authentifizierungstoken für unsere Anwendung im Kontext des eigentlichen Benutzers holen.

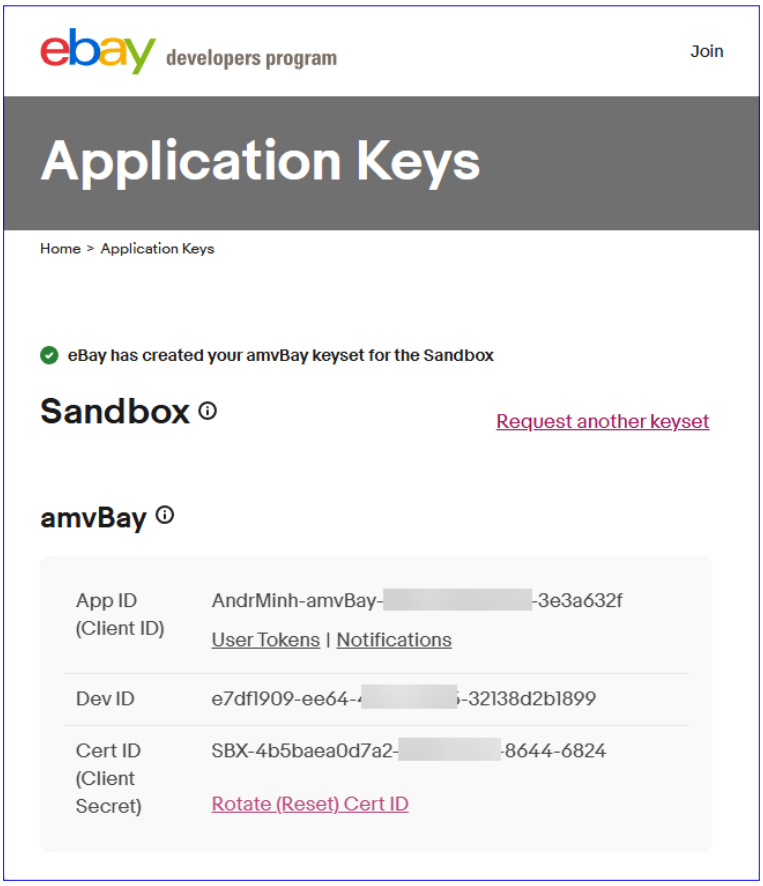

Bild 6: Unser Keyset

Damit können wir dann die Anwendung im Kontext des jeweiligen Benutzers verwenden.

Wir gehen an dieser Stelle davon aus, dass wir eine Anwendung für den Zugriff auf eBay erstellen, um diese beispielsweise in die Warenwirtschaft eines Kunden zu implementieren, damit dieser seine Produkte automatisiert bei eBay einstellen kann.

Nur wollen wir nun zunächst nicht im Kontext

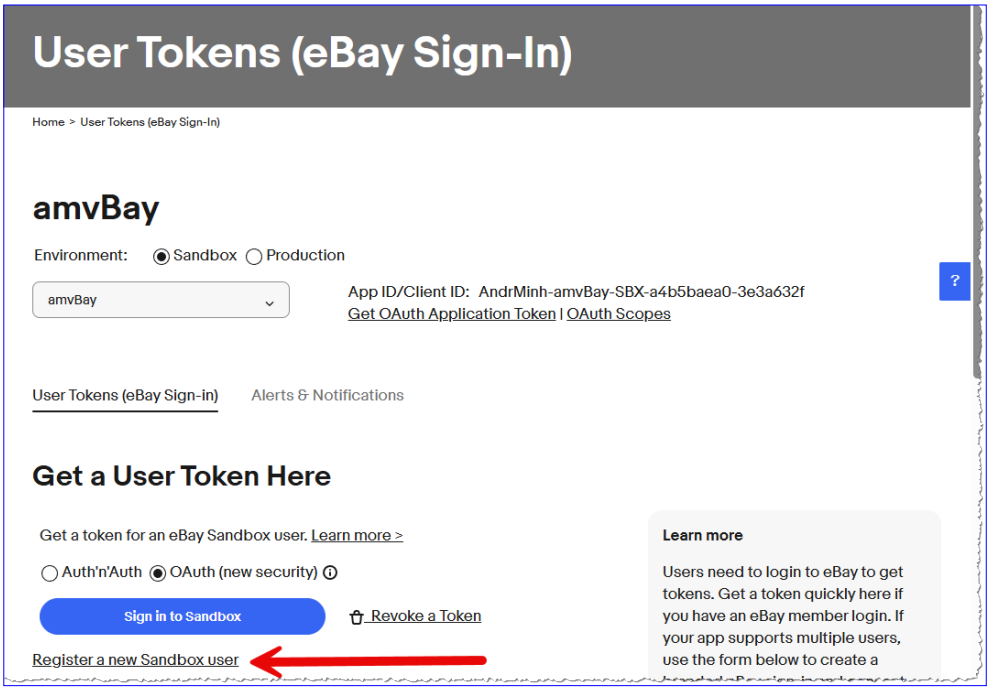

Bild 7: Aufruf der Registrierung eines Sandbox-Benutzers

des echten Benutzers auf eBay zugreifen, sondern erst einmal die Sandbox nutzen. Da die Sandbox praktisch ein eigenes eBay zum Testen ist, können wir logischerweise dort auch nicht mit unserem normalen eBay-Konto arbeiten, sondern müssen uns ein neues Konto anlegen.

Dazu klicken wir erst einmal unter unserer **App ID** (**Client ID**) auf den Link **Register a new Sandbox user** (siehe Bild 7).

Neuen Sandbox-Benutzer anlegen

Das Anlegen des neuen Sandbox-Benutzers führen wir in dem Formular aus Bild 8 durch. Nach dem Registrieren können wir uns an der Sandbox anmelden.

Hier finden wir allerdings zunächst keine Angebote vor.

Token für diesen Benutzer holen

Nun wollen wir das Token holen, indem wir uns über unsere eBay-Anwendung mit den Daten unseres soeben erstellen Sandbox-Benutzers bei der Sandbox anmelden.

Dazu klicken wir im vorherigen Dialog auf **Sign in to Sandbox** (siehe Bild 9).

Damit erscheint nochmals ein Bereich, in dem wir unsere Adresse und weitere Daten eingeben müssen (siehe Bild 10).

Nun erscheint der Anmeldedialog von der eBay-Sandbox, wo wir uns nun mit den Daten unseres Sandbox-Benutzers anmelden (nicht mit dem Developer-Account!).

Schließlich erhalten wir eine Meldung, die uns als Sandbox-Benutzer darüber aufklärt, dass wir der App die Berechtigungen geben, unser Konto für verschiedene Aktionen auf

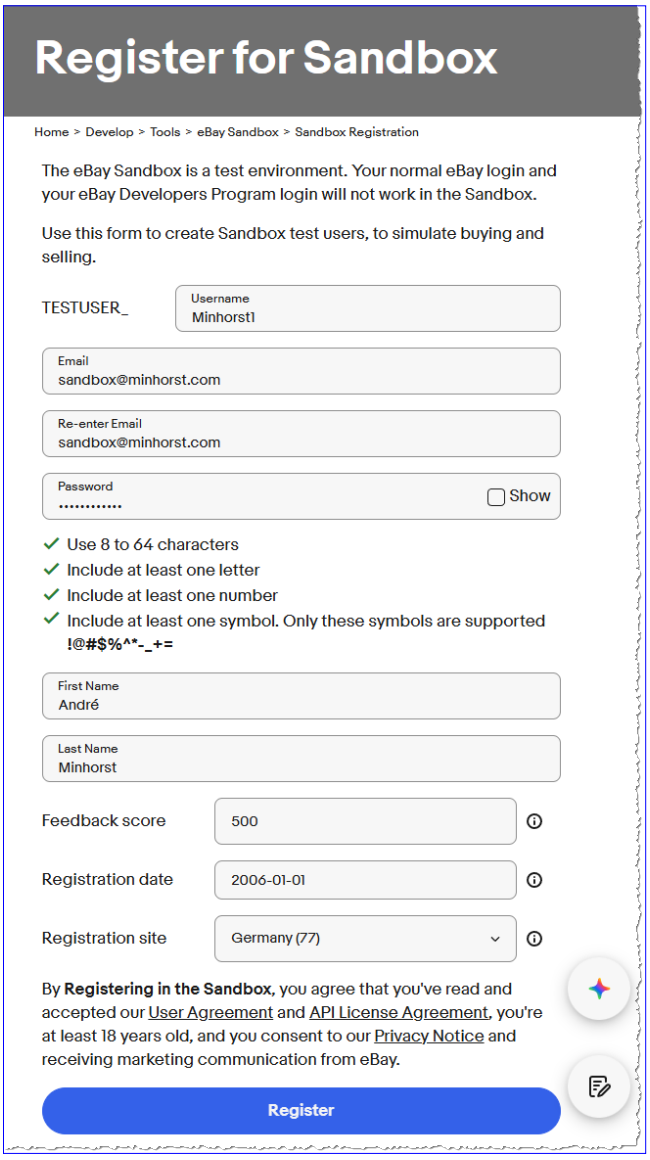

Bild 8: Eingabe der Daten für einen Sandbox-Benutzer

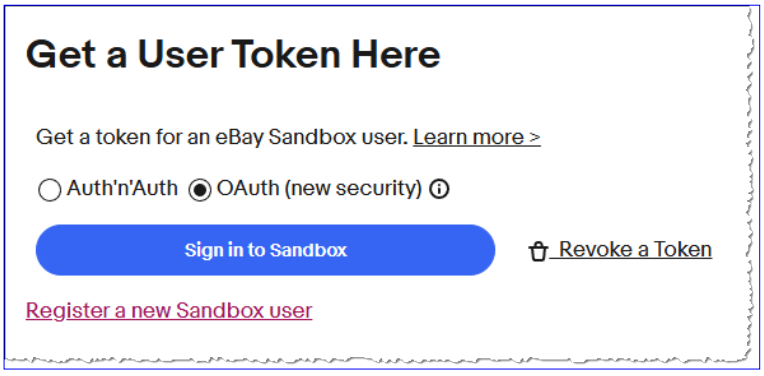

Bild 9: Anmeldung an der Sandbox

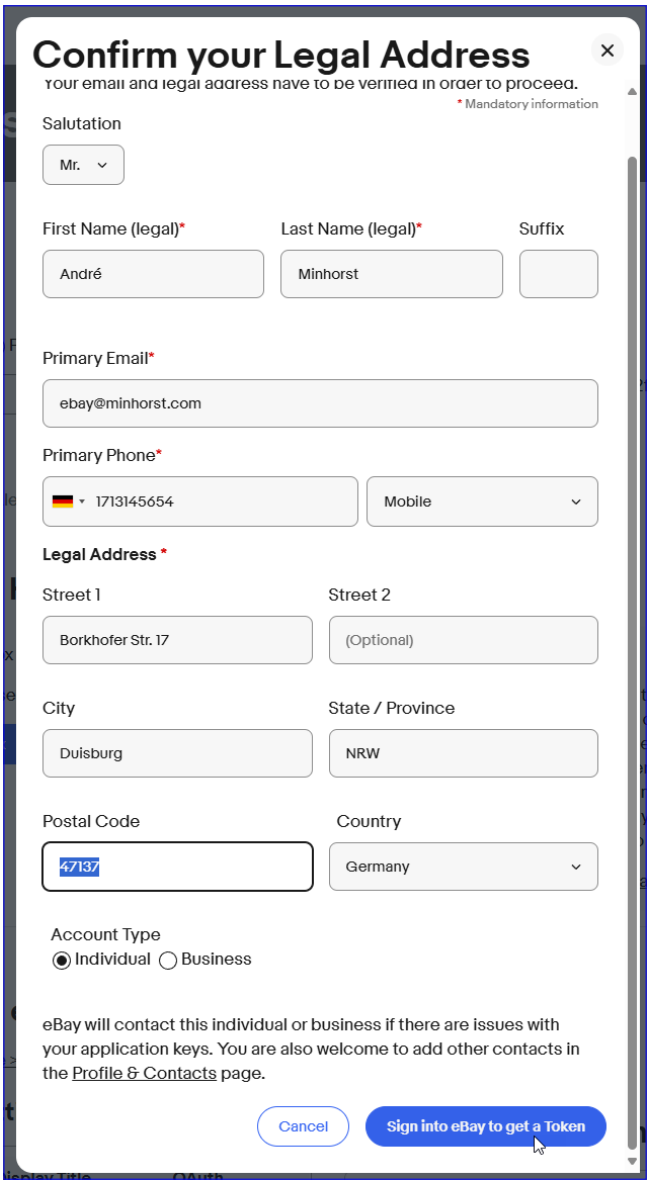

Bild 10: Eingabe unserer Adressdaten für die Token-Generierung

der Sandbox von eBay zu nutzen. Im Anschluss erhielten wir jedoch eine Fehlermeldung (siehe Bild 11).

Bei unseren Recherchen zu diesem Artikel waren wir in der Folge nicht in der Lage, ein Authentifizierungstoken für die Sandbox zu holen, also haben wir es mit der **Production**-Version probiert.

Hier erschien beim Versuch, ein Keyset zu holen, zunächst die Meldung aus Bild 12.

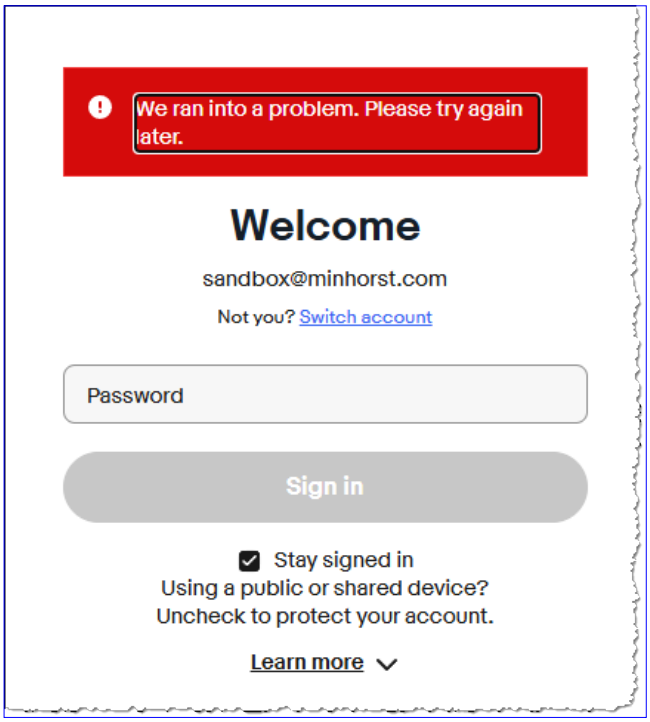

Bild 11: Fehlermeldung beim Versuch, das Token zu holen

Daraufhin haben wir uns, um weiterarbeiten zu können, für die Beantragung einer Ausnahme beworben (siehe Bild 13). Dazu haben wir unter **Event Notification Delivery Method** die Option **Marketplace Account Deletion** gewählt und **Exempted from Marketplace Account Deletion** aktiviert. Nach dem Absenden mit **Submit** wurde dies umgehend akzeptiert.

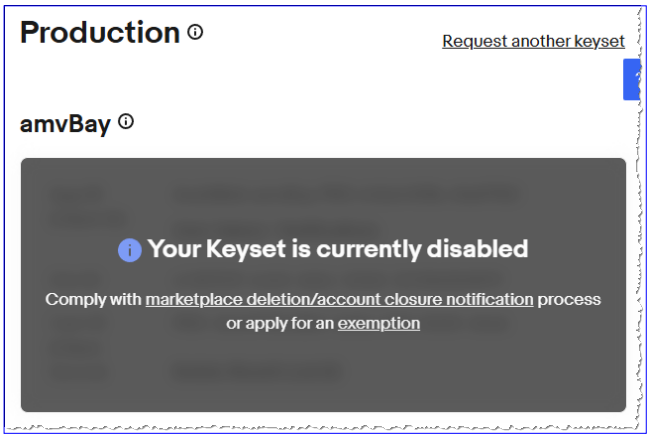

Bild 12: Das Keyset ist aktuell deaktiviert.

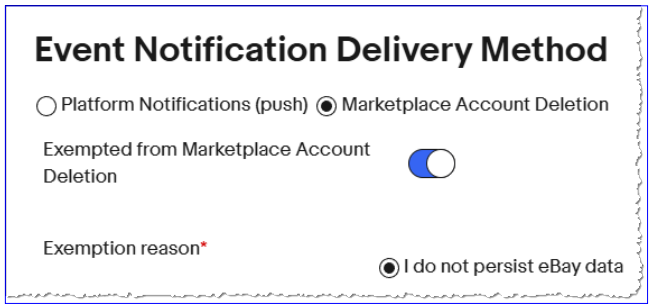

Bild 13: Optionen für eine Ausnahme

Nach der Authentifizierung und der Zustimmung, dass unsere eBay-Anwendung **amvBay** im Kontext unseres Benutzers auf eBay zugreifen darf, landen wir wieder auf der Token-Seite. Hier können wir nun erneut auf **Get OAuth Application Token** klicken und erhalten endlich das Token (siehe Bild 15), das wir in die Zwischenablage kopieren und in einer weiteren Konstanten in unserem VBA-Projekt speichern:

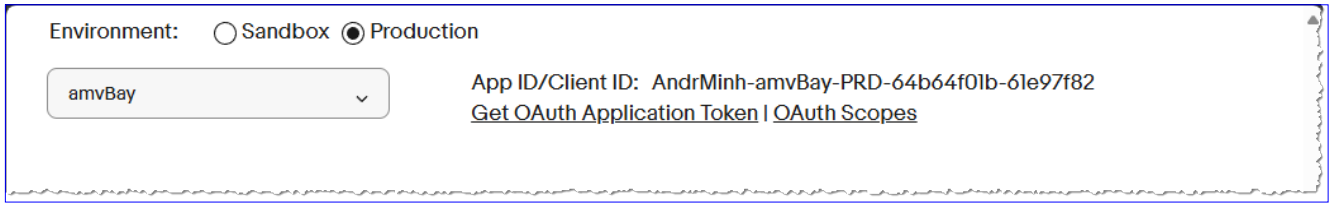

Bild 14: Link zum Holen des Authentifizierungstokens

Danach haben wir im oberen Bereich der Webseite den Link **Get OAuth Application Token** vorgefunden (siehe Bild 14).

Danach mussten wir uns erneut authentifizieren, um das Token zu holen. Dazu haben wir diesmal nicht den Sandbox-Benutzer verwendet, den wir zuvor angelegt haben, sondern einen echten eBay-Benutzeraccount.

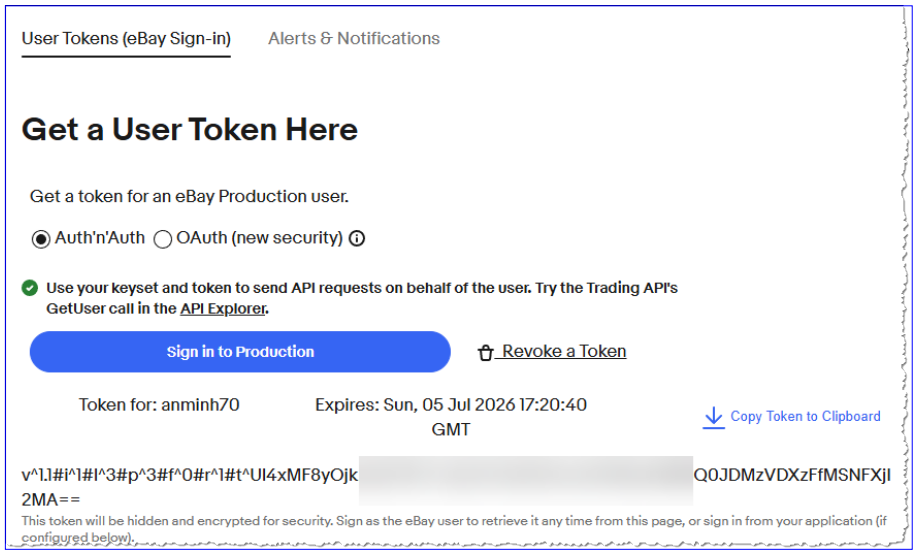

Bild 15: Das ersehnte OAuth Application Token

```
Const cStrUserToken As String = "v^1.1#i^1#p^1#I^3#r^..."
```

Das Token ist zeitlich begrenzt gültig, in diesem Fall 18 Monate. Du solltest dies, wenn Du eine eBay-Anwendung für einen Kunden programmierst, auf jeden Fall im Blick behalten und gegebenenfalls rechtzeitig für eine Aktualisierung des Tokens über den hier vorgestellten Weg sorgen.

Anwendungstoken holen

Das Benutzertoken können wir nutzen, um benutzerspezifische Aufrufe der Rest API zu authentifizieren. Es gibt jedoch auch einige Rest API-Befehle, die allgemein verfügbar sind und nicht im Kontext eines speziellen Benutzers stehen.

In diesem Fall können wir das Benutzertoken nicht verwenden, sondern wir benötigen ein Anwendungstoken. Das holen wir uns auf der gleichen Seite, auf der wir auch die Erstellung

des Benutzertokens initiiert haben. Hier klicken wir nun im oberen Bereich auf **Get OAuth Application Token** (siehe Bild 16).

Bild 16: Ermitteln des Anwendungstokens

Nun öffnet sich ein Dialog namens **OAuth Application Token**, dessen Inhalt wir über die Schaltfläche **Copy Token** in die Zwischenablage kopieren können (siehe Bild 17).

Bild 17: Kopieren des Anwendungstokens

Auch dieses Token wollen wir in einer Konstanten speichern, diesmal unter dem Namen **cStrAppToken**.

Das ist etwas komplizierter, denn eine Zeile zur Definition einer Konstanten kann nur 1.023 Zeichen aufnehmen, das Token hat allerdings eine Länge von über 1.900 Zeichen.

Am einfachsten kopierst Du es zunächst in eine Textdatei, schneidest dort die ersten 950 Zeichen aus und fügst sie der ersten Zeile der folgenden Anweisung hinzu. Den Rest trägst Du dann in die zweite Zeile ein, die Du mit dem Unterstrich und dem Und-Zeichen verbindest:

```
Const cStrAppToken As String = "v^1.1#i^1#r^0#p^1#..." _
    & "EAGuOQLyjgblGU8zkk+nJIBIAGIx8Vg4P9TJre...=="
```

Nachteil: Gültigkeitsdauer dieser Daten

Ein großer Nachteil, wenn wir die Daten so über die Benutzeroberfläche holen, ist die geringe Haltbarkeit:

- Das Anwendungstoken ist für ca. zwei Stunden gültig.
- Das Benutzertoken ist nur ca. eine Stunde gültig.
- Ein Refresh-Token ist 18 Monate gültig und kann zum Erneuern des Benutzertokens verwendet werden.

Was bedeutet das für uns? Nach aktuellem Stand müssten wir alle zwei Stunden oder, wenn wir beispielsweise

Office: Eingebaute Kontextmenübefehle recyclen

Wir haben bereits in einigen weiteren Artikel beschrieben, wie wir unseren Anwendungen in Word, Excel oder Access benutzerdefinierte Kontextmenü-Einträge oder sogar vollständige Kontextmenüs hinzufügen können. Was wird dort noch nicht berücksichtigt haben: Manchmal möchte man vielleicht ein eingebautes Element in einem benutzerdefinierten Kontextmenü oder gar ein vollständiges Menü nachbauen, um dann selbst genauer zu steuern, wann dieses angezeigt werden soll. Ein gutes Beispiel sind die Befehle für die Verwendung der Zwischenablage, also Ausschneiden, Kopieren und Einfügen. Es kann sinnvoll sein, diese einem benutzerdefinierten Kontextmenü hinzuzufügen. In diesem Artikel beschreiben wir, wie Du herausfindest, welche Informationen über das eingebaute Kontextmenü-Element benötigst und wie Du dieses in einem benutzerdefinierten Kontextmenü abbildest.

In den vorherigen Ausgaben haben wir bereits in einigen Artikeln die Programmierung von Kontextmenüs beschrieben:

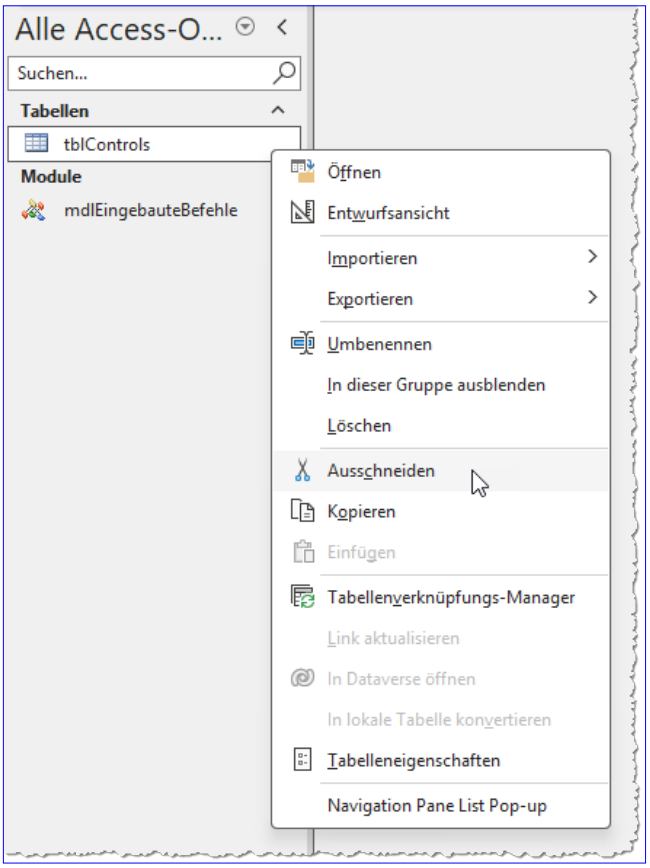

Bild 1: Beispiel für ein eingebautes Kontextmenü

- **Menüs im VBA-Editor anpassen** (**www.vbentwickler.de/434**)
- **Menüs per VBA programmieren** (**www.vbentwickler.de/435**)
- **Kontextmenüs per VBA programmieren** (**www.vbentwickler.de/368**)

Auch auf die Anpassung von Kontextmenüs in Outlook sind wir bereits eingegangen – siehe **Outlook: Kontextmenüs anpassen** (**www.vbentwickler.de/369**).

In diesem Artikel nun schauen wir uns im Detail an, wie wir die eingebauten Kontextmenü-Befehle in benutzerdefinierte Kontextmenüs einbauen können – zum Beispiel die aus Bild 1.

Dazu sind die folgenden Schritte nötig:

- Ermitteln von Informationen, mit denen wir die eingebauten Kontextmenü-Einträge identifizieren können
- Einbau der Einträge in ein benutzerdefiniertes Kontextmenü

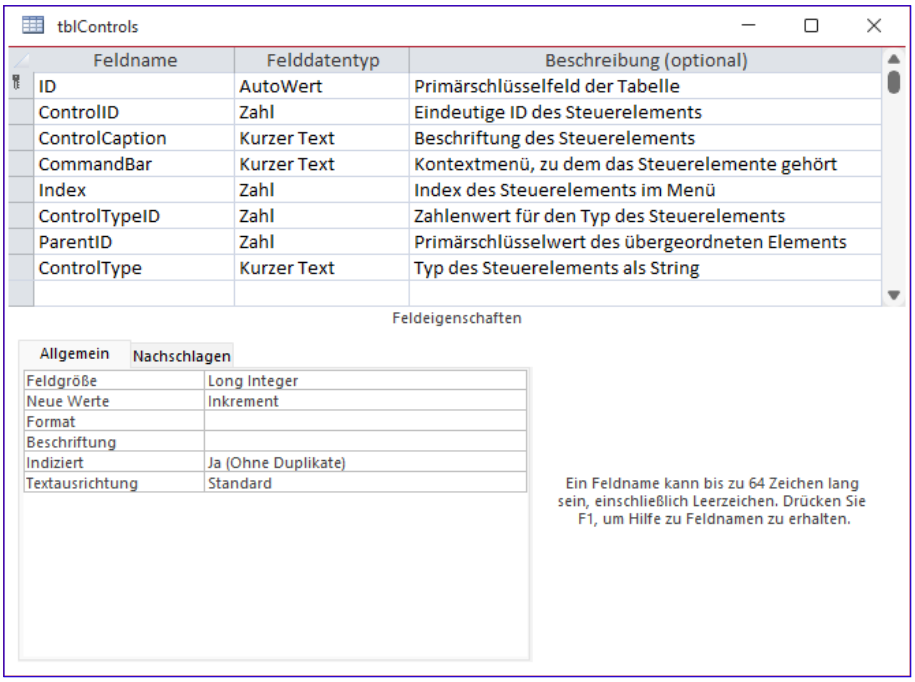

Bild 2: Entwurf der Tabelle zum Speichern der Kontextmenü-Einträge mit Eigenschaften

Identifizieren der eingebauten Kontextmenü-Befehle

Der erste Schritt ist relativ aufwendig. Wir können nicht einfach ein Kontextmenü öffnen und dieses beispielsweise anhand des Namens identifizieren. Stattdessen benötigen wir einen anderen Wert, nämlich die ID dieses Eintrags.

An diesen kommen wir jedoch auch nicht so leicht. Es gibt keine Möglichkeit, diese ID direkt auszulesen.

Daher bauen wir uns eine Tabelle und eine Prozedur, um diese Tabelle mit den Informationen zu den Kontextmenü-Einträgen zu füllen.

Tabelle zum Speichern der Kontextmenü-Einträge

Die Tabelle heißt **tblControls** und sieht in der Entwurfsansicht wie in Bild 2 aus.

Nachdem wir die Tabelle per Code gefüllt haben, sollte diese etwa wie in Bild 3 aussehen.

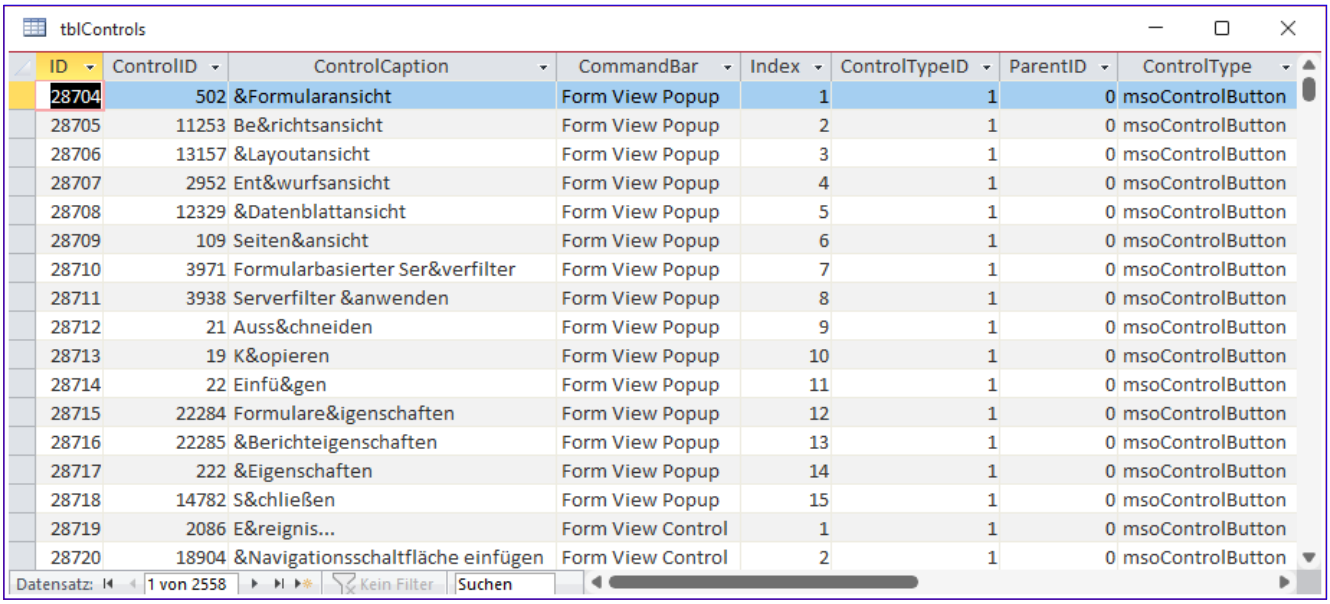

Bild 3: Datenblattansicht der Tabelle zum Speichern der Kontextmenü-Einträge mit Eigenschaften

Hauptprozedur zum Einlesen der Kontextmenü-Elemente

Die Prozedur **KontextmenuesEinlesen** aus Listing 1 ist die Hauptprozedur zum Einlesen der Steuerelemente der Kontextmenüs.

Um diese zu nutzen, müssen wir zuvor einen Verweis auf die Bibliothek **Microsoft Office 16.0 Object Library** setzen (siehe Bild 4). Diese enthält das Objektmodell für den Zugriff auf die **CommandBar**-Elemente, also die Menüs.

Die Prozedur **KontextmenuesEinlesen** startet damit, die Tabelle **tblControls** vollständig zu leeren. Das erledigt sie durch den Aufruf einer **DELETE**-Anweisung.

```
Public Sub KontextmenuesEinlesen()
    Dim cbr As CommandBar
    Dim cbc As CommandBarControl
    Dim db As dao.Database
    Dim lngID As Long
    Set db = CurrentDb
    db.Execute "DELETE FROM tblControls", dbFailOnError
    For Each cbr In Application.CommandBars
        If cbr.BuiltIn = True Then
            Select Case cbr.Type
                Case msoBarTypePopup
                    For Each cbc In cbr.Controls
                        If cbc.BuiltIn = True Then
                            On Error Resume Next
                            db.Execute "INSERT INTO tblControls(ControlID, ControlCaption, CommandBar, Index, " _
                                "ControlTypeID, ControlType) VALUES(" & cbc.Id & ", '" & cbc.Caption & "', '" _
                                & cbr.Name & "', " & cbc.Index & ", " & cbc.Type & ", '" _
                                & ControlTypeString(cbc.Type) & "')", dbFailOnError
                            lngID = db.OpenRecordset("SELECT @@IDENTITY").Fields(0)
                            On Error GoTo 0
                        End If
                        Select Case cbc.Type
                            Case msoControlButton, msoControlComboBox, msoControlExpandingGrid, msoControlEdit, _
                                    msoControlGrid
                            Case msoControlPopup, msoControlButtonPopup, msoControlSplitButtonMRUPopup, _
                                    msoControlSplitButtonPopup
                                Call UntermenuesEinlesen(db, cbc, lngID, cbr.Name)
                            Case Else
                                Debug.Print cbc.Type
                        End Select
                    Next cbc
            End Select
        End If
    Next cbr
End Sub
```

Listing 1: Die Prozedur **KontextmenuesEinlesen** liest die Elemente der ersten Ebene in die Tabelle **tblControls** ein.

Zwischenablage für Texte programmieren

Die Zwischenablage ist ein praktisches Werkzeug, wenn es darum geht, Texte zu kopieren und an anderer Stelle wieder einzufügen. Das ist insbesondere sinnvoll, wenn es sich um umfangreichere Texte handelt. Dabei können wir Funktionen programmieren, die den markierten Text in die Zwischenablage kopieren und solche, die den Inhalt der Zwischenablage als Funktionsergebnis liefern. In diesem Artikel schauen wir uns an, wo man solche Funktionen sinnvoll einsetzen kann und wie sie programmiert werden.

Texte in die Zwischenablage einfügen

In unserer Praxis konnten wir Funktionen zum Kopieren von Texten in die Zwischenablage hauptsächlich dort nutzen, wo eine Ausgabe in den Direktbereich zu umfangreich war, als dort komplett dargestellt zu werden. Das heißt, wir lassen uns beispielsweise das Ergebnis des Aufrufs einer Rest-API im Direktbereich ausgeben und stellen dann beim Scrollen zum Beginn der Ausgabe fest, dass ein Teil abgeschnitten wurde. Hier gibt es dann zwei Möglichkeiten:

- Entweder wir interessieren uns nur für einen bestimmten Teil der Ausgabe wie beispielsweise den Anfang. Dann können wir die Ausgabe des Ergebnisses beispielsweise mit der **Left**-Funktion abschneiden und nur so viel ausgeben, dass der Direktbereich diesen fassen kann.

- Oder wir kopieren einfach die vollständige Ausgabe mit einer dafür vorgesehenen Funktion in die Zwischenablage und geben diese dann in einem entsprechenden Programm wie einem Text-Editor oder einem Code-Editor zur weiteren Untersuchung aus.

Und genau für letzteren Zweck benötigen eine Routine, welche die Rückgabe einer Funktion oder auch den Inhalt einer Variablen in die Zwischenablage kopiert.

Texte aus der Zwischenablage auslesen

Der umgekehrte Anwendungsfall tritt auf, wenn wir den Inhalt der Zwischenablage in eine Variable füllen oder diesen direkt einer Funktion zur weiteren Verarbeitung zuweisen wollen.

Auch hier ein Beispiel: Wir holen aus der Benutzeroberfläche beispielsweise von eBay oder einer anderen Rest-API ein Token, das oft so lang ist, dass man es in einer Konstanten nicht in einer einzelnen Zeile verarbeiten kann (Codezeilen in VBA haben eine natürliche Grenze, und zwar 1.024 Zeichen). Dann müsste man dieses wie folgt auf mehrere Zeichen umbrechen:

```
Public Const cStrBeispiel = "1234567890" _
    & "12345678901234567890123456789012345678901" _
    & "12345678901234567890123456789012345678901" _
    & "12345678901234567890123456789012345678901"
```

Das macht keinen Spaß und vor allem ist es fehleranfällig, weil man hier schnell mal ein Zeichen zu viel oder zu wenig hat.

Also könnten wir, wenn wir das Token aus der Zwischenablage direkt verarbeiten könnten, direkt die Hilfsfunktion aus Listing 1 für diesen Zweck nutzen. Die Funktion nimmt zwei Parameter entgegen. Der erste erwartet den Namen der zu definierenden Konstanten, der zweite die Anzahl der Zeichen je Zeile für die Konstante.

Die Funktion holt mit der später noch zu beschreibenden Funktion **AusZwischenablage** den Inhalt der Zwi-

```
Public Function ZwischenablageInKonstante(strKonstante As String, lngZeichen As Long) As String
    Dim strTemp As String
    Dim strCode As String
    Dim lngLaenge As Long
    strTemp = AusZwischenablage
    strCode = "Public Const " & strKonstante & " As String = "
    strCode = strCode & """ & Left(strTemp, lngZeichen) & """ _" & vbCrLf
    strTemp = Mid(strTemp, lngZeichen)
    lngLaenge = Len(strTemp)
    Do While Not lngLaenge = 0
        strCode = strCode & "    & """ & Left(strTemp, lngZeichen) & """ _" & vbCrLf
        strTemp = Mid(strTemp, lngZeichen)
        lngLaenge = Len(strTemp)
    Loop
    If Not Len(strCode) = 0 Then
        strCode = Left(strCode, Len(strCode) - 4)
    End If
    ZwischenablageInKonstante = strCode
End Function
```

Listing 1: Funktion zum Zerlegen einer Zeichenkette in eine mehrzeilige Konstante

schenablage in die Variable **strTemp**. Dann stellt sie den ersten Teil des Codes der ersten Zeile zusammen, der nur aus der Deklaration der eigentlichen Konstanten besteht.

Anschließend fügt sie die ersten Zeichen entsprechend dem Wert des Parameters **lngZeichen** zusammen, indem sie mit der **Left**-Funktion die entsprechende Teilzeichenkette ausliest. Damit diese Zeichen nicht nochmals verwendet werden, verkürzen wir die Zeichenkette aus **strTemp** vorne um die entsprechende Anzahl Zeichen. Danach gehen wir in eine **Do While**-Schleife, die wir erst verlassen, wenn die Länge der Zeichenkette in **strTemp** gleich **0** ist – also, wenn alle Teilzeichenketten in die Variable **strCode** gelangt sind und entsprechend vorn aus **strTemp** herausgeschnitten wurden.

Diese Schritte führen wir für alle verbleibenden Zeichen durch und erhalten beispielsweise ein Ergebnis wie in Bild 1. Der Code überschreitet so sicher nicht die zulässige Höchstanzahl an Zeichen pro Zeile.

Bild 1: Ergebnis des Aufteilens der Zwischenablage in eine mehrzeilige Konstante