
Duale Hochschule Baden-Württemberg Mannheim

## Portfolioprüfung Data Visualization

# Projektdokumentation Dashboard

## Studiengang Wirtschaftsinformatik

### Studienrichtung Data Science


| | |
|---|---|
| Verfasser(in): | André-Anan Gilbert, Valentin Müller, Viet Duc Kieu |
| Matrikelnummer: | 3465546, 4616344, 9588548 |
| Firma: | SAP SE |
| Kurs: | WWI-20-DSB |
| Studiengangsleiter: | Prof. Dr. Bernhard Drabant |
| Wissenschaftliche(r) Betreuer(in): | Steffen Fleischer |
| Bearbeitungszeitraum: | 23.07.2021 – 31.08.2021 |

# Ehrenwörtliche Erklärung

Wir versichern hiermit, dass wir die vorliegende Arbeit mit dem Titel "*Projektdokumentation Dashboard*" selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben.

31.08.2021

Ort, Datum

André-Anan Gilbert, Valentin Müller, Viet Duc Kieu

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Quelltextverzeichnis

# Abkürzungsverzeichnis

| | |
|---|---|
| **API** | Application Programming Interface |
| **CI** | Continuous Integration |
| **GUI** | Graphical User Interface |
| **IBCS** | International Business Communication Standards |
| **KI** | Künstliche Intelligenz |
| **MVC** | Model-View-Controller bzw. Model-View-Controller-Architektur |
| **UI** | User Interface |

# 1 Einleitung

## 1.1 Überblick über die Anwendung

Das Ziel dieser Projektdokumentation ist die Erstellung eines Dashboards in englischer Sprache, um einem fiktiven Unternehmen einen anschaulichen Überblick über seine Lieferungen hinsichtlich Volumen und Qualität zu verschaffen.

Dabei soll es verschiedene Seiten oder Sichten geben, von denen die erste die Gesamtsumme der Ausgaben, im Folgenden als Ordered Spend bezeichnet, und die Anzahl der Bestellungen für das ganze Jahr, nach Monat, nach Einkaufsorganisation und für die Top 10 Lieferanten nach Ordered Spend im aktuellen Jahr darstellen soll. Für jede Darstellung soll auf dieser Seite der Vergleich zum Vorjahr dargestellt werden.

Auf der zweiten Seite soll der Fokus auf abweichenden Lieferungen liegen und eine Darstellung von Ordered Spend und Anzahl der Bestellungen dieser erfolgen, allerdings nur für das aktuelle Jahr. Dabei wird die Gesamtsumme abweichender Bestellungen und deren Anteil an der Gesamtheit dargestellt. Wie bereits auf der ersten Seite soll auch hier eine Darstellung nach Monat, Einkaufsorganisation und Top 10 Lieferanten erfolgen, zusätzlich sollen aber Informationen über Grund und Art der Abweichung dargestellt werden.

Insgesamt soll ein Umschalten zwischen Ordered Spend und Anzahl der Bestellungen sowie eine Filterung nach Buchungskreis, Einkaufsorganisation, Werk und Materialgruppe seitenübergreifend möglich sein.

Eine Zusammenstellung visueller Elemente zur Darstellung von Daten bezeichnet man als Dashboard. Das Dashboard grenzt sich dadurch, dass es visuelle Elemente statt einer einfachen Auflistung der Daten beinhaltet und das Ziel besitzt, eine Geschichte über die zugrundeliegenden Daten zu erzählen oder eine Frage zu diesen zu beantworten, von anderen Darstellungsarten wie beispielsweise Reports oder Tabellen ab. Dabei besitzt ein Dashboard verschiedene sogenannte Key Performance Indicators, denen zentrale Bedeutung für die Beantwortung der gestellten Frage zugemessen wird (Vgl. Kusleika, 2021, S. 5f). Im Falle des Dashboards, das Thema dieser Projektdokumentation sein soll, sind die Key Performance Indicators sowohl Ordered Spend und Anzahl der Bestellungen als

auch verzögerte Lieferungen, da besonders letztere sehr aussagekräftig für Probleme beim qualitativen Ziel rechtzeitiger, vollständiger und zuverlässiger Lieferung sind.

In dieser Projektdokumentation sollen nun sowohl theoretische Grundlagen als auch Umsetzung in Form von Implementierung einer Webanwendung zu diesem Zweck dargelegt werden. Die Implementierung erfolgt mithilfe der Bibliotheken NumPy, Pandas und Dash. Zur Darstellung der im Dashboard enthaltenen Charts kommt die Bibliothek Plotly zum Einsatz.

## 1.2 Verwendete Bibliotheken und technische Voraussetzungen

Dash ist eine Bibliothek für Python, R und Julia und ermöglicht komplexe, wissenschaftliche und interaktive Visualisierung von Daten als Web-Applikation und ist somit besonders im Data Science-Kontext geeignet. Dabei basiert Dash auf weiteren Bibliotheken. Für das Back-End sowie Caching wird Flask verwendet, während die Applikation im Browser als React-App gerendert wird. Zum Erstellen der Charts können verschiedene Bibliotheken verwendet werden, wobei anzumerken ist, dass Plotly den umfangreichtsten Support genießt und am besten integriert ist (Vgl. Dabbas, 2021, Chapter 1).

Dabei besteht Dash nicht aus einem einzelnen, umfangreichen Package sondern enthält mehrere, nach ihrer Funktionalität getrennte Pakete, von denen die im Folgenden beschriebenen in dieser Anwendung zum Einsatz kommen. Dash selbst bildet das zentrale Element und enthält zusätzlich Funktionen für Interaktivität wie die in Kapitel 2.3 beschriebenen Callbacks. Dash Core Components stellt interaktive Komponenten, die später durch den User beeinflusst werden können, bereit, und enthält somit neben den Graph-Objekten, in denen die Charts gerendert werden, auch Dropdown-Menüs, Datepicker oder Eingabefelder. Weiterhin stellt das Paket Dash HTML Components HTML-Tags in Form von Python-Klassen bereit. Im Browser werden diese dann in die korrespondierenden Tags konvertiert und als dieser gerendert. Zuletzt kommt noch Dash Bootstrap Components zum Einsatz, welches dazu dient, Funktionalitäten der verbreiteten HTML, CSS und JavaScript-Bibliothek Bootstrap in Dash nutzbar zu machen (Vgl. Dabbas, 2021, Chapter 1).

Zum Einlesen der in Form einer Excel-Datei vorliegenden Lieferungsdaten sowie zur Data Preparation und Filterung zur Laufzeit kommt die Python-Bibliothek Pandas zum Ein-

satz. Pandas stellt tabulare Datenstrukturen zur Verfügung und bietet darüber hinaus viele Möglichkeiten zur Analyse und Manipulation (Vgl. Navlani et al., 2021, Chapter 1), wie beispielsweise Filterung und Aggregation.

Eine vollständige Liste der neben den genannten Bibliotheken notwendigen Packages findet sich in der dem Projekt beiliegenden Datei *requirements.txt*, damit mit einem Paketmanager wie beispielsweise pip eine unkomplizierte Installation durchgeführt werden kann. Da einige Funktionaliäten wie beispielsweise Type Hinting Generics verwendet werden, die in älteren Python-Versionen nicht zur Verfügung stehen (Vgl. „What's New In Python 3.9 — Python 3.9.6 documentation", 24/08/2021), wird Python 3.9 zur Ausführung der Web-Applikation benötigt.

# 2 Grundlagen von Web-Apps

## 2.1 Model-View-Controller-Architektur

Die Model-View-Controller-Architektur macht die Entwicklung komplexer Anwendungen zu einem handhabbaren Prozess und ermöglicht es mehreren Entwicklern gleichzeitig an der Anwendung zu arbeiten. Dabei wird die Komponente einer Anwendung in drei eigenständige Module aufgeteilt:

- **Model:** Enthält das Back-End
- **View:** Enthält das Front-End oder die graphische Benutzeroberfläche
- **Controller:** Steuert die Daten in der Anwendung

Das Modell (Model) stellt in einer MVC-Anwendung den Status, die Daten und die zugehörige Geschäftslogik der Anwendung dar. Zum Beispiel werden Kundeninformationen aus der Datenbank abgerufen, manipuliert und aktualisiert und anschließend zurück in die Datenbank geschrieben oder sie werden auf der Benutzeroberfläche gerendert.

Ansichten (View) werden aus Daten, die durch das Modell bereitgestellt werden, erstellt. Diese Daten können auf der Benutzeroberfläche in Form von Diagrammen und Tabellen dargestellt werden und enthalten zudem alle UI-Komponenten wie Textfelder, Dropdowns und Tabs.

Controller verarbeiten Benutzerinteraktionen wie Maus- und Tastatureingaben des Benutzers und informieren das Modell, die Ansicht entsprechend zu ändern. Beispielsweise verarbeitet der Controller alle Interaktionen und Eingaben aus der Kundenansicht und aktualisiert die Datenbank unter Verwendung des Modells.

Die Idee Code nach seiner Funktion zu organisieren, wird als Trennung von Belangen (engl.: seperation of concerns) bezeichnet und erleichtert das Verständnis, das Debuggen und das Testen des Codes (Vgl. Ardalis, 30/08/2021).

## 2.2 Front-End und Back-End

Front-End und Back-End sind die am häufigsten verwendeten Begriffe in der Webentwicklung und unterscheiden sich stark voneinander. Das Front-End bezeichnet den Teil der Anwendung, den Benutzer sehen und mit dem sie interagieren können, und das Backend bezeichnet die Logik hinter der Benutzeroberfläche.

Das Front-End wird auch als Kundenseite bezeichnet und enthält alles, was Benutzer direkt erleben: Farben, Text, Bilder, Tabellen und das Navigationsmenü. Front-End-Entwickler benutzen bei der Entwicklung der UI Tools wie HTML, CSS und JavaScript, um das Design, das Verhalten und den Inhalt festzulegen. Dabei muss sichergestellt werden, dass Webseiten responsive sind, d. h. sie werden auf allen Geräte-Größen korrekt angezeigt und sollen performant Inhalte anzeigen können. Heutzutage sieht man vermehrt den Einsatz von Frameworks oder Bibliotheken wie React, Vue und Angular, die es möglich machen, Komponenten wiederzuverwenden.
Zu verbreiteten Front-End-Tools gehören:

- **Programmiersprachen:** JavaScript, TypeScript
- **Frameworks/Bibliotheken:** React, Vue, Angular, Gatsby
- **Compiler:** Babel
- **Bundler:** Webpack, Parcel
- **CSS-Tools:** Sass, Styled Components

Das Back-End, die Serverseite der Applikation, enthält die gesamte Geschäftslogik. Es ist der Teil der Software, der nicht in direkten Kontakt mit den Benutzern kommt, da auf sie nur indirekt über eine Front-End-Anwendung zugegriffen werden kann. Auch das Schreiben von APIs, das Erstellen von Bibliotheken und das Arbeiten mit Systemkomponenten sind im Back-End enthalten.
Zu verbreiteten Back-End-Tools gehören:

- **Programmiersprachen:** Java, C++, Python, Node.js
- **Datenbankensprachen:** NoSQL, MySQL
- **Frameworks/Bibliotheken:** Spring Boot, Django, Flask
- **Cloud Server:** Amazon Web Services, Google Cloud Plattform, Microsoft Azure

Unabhängig vom Stapel wird es immer die Trennung der Belange geben. Konkret bezeichnet Front-End den Browser und das Back-End, den Server oder vermehrt auch die Cloud (Vgl. Fayock, 18/06/2020).

## 2.3 Callbacks

Callback-Funktionen oder Rückruffunktionen werden häufig in Bibliotheken und Frameworks verwendet, beispielsweise in JavaScript Anwendungen, React und Dash. Sie sollen Funktionen erweitern und werden bei bestimmten Ereignissen zurückgerufen. Der wesentliche Unterschied zu einer normalen Funktion besteht darin, dass eine Callback-Funktion zunächst nur definiert und erst aufgerufen wird, wenn ein bestimmtes Ereignis eingetreten ist. Ereignishändler aus Webseiten sind ein gutes Beispiel für eine Art von Callback-Funktionen. Ein Ereignis kann demnach ein Mausklick sein, wodurch eine Callback-Funktion ausgeführt wird, um beispielsweise eine neue Seite aufzurufen oder ein Popup Fenster zu öffnen (Vgl. Eygi, 17/03/2020).
Der folgende Quelltext zeigt einen Python-Callback:

```
1  from typing import Callable
2
3  def sum_callback(number: int) -> None:
4      print(f"Sum = {number}")
5
6  def add(a: int, b: int, callback: Callable[int]) -> None:
7      print(f"Adding {a} + {b}")
8      callback(a + b)
9
10 add(1, 2, sum_callback)
```

Quelltext 2.1: `Callback.py`

In Dash hingegen wird ein *@app.callback* decorator verwendet, der dafür sorgt, dass Callback-Funktionen aufzurufen sind, wenn sich der Wert einer Input-Komponente verändert. Dash stellt der Callback-Funktion den Input-Wert als Eingabeargument bereit und aktualisiert die Output-Komponente mit dem Rückgabewert der Funktion. Durch diese Interaktionsmöglichkeit lassen sich beispielsweise Kinder einer HTML-Komponente oder die Abbildung

einer dcc.Graph-Komponente aktualisieren (Vgl. „Basic Callbacks | Dash for Python Documentation | Plotly“, 30/08/2021).

Der folgende Quelltext zeigt einen simplen Dash Callback:

```
1  import dash_core_components as dcc
2  import dash_html_components as html
3  from dash.dependencies import Input, Output
4
5  app.layout = html.Div([
6          html.H1(id="user-output"),
7          dcc.Input(id="user-input", type="text")
8  ])
9
10 @app.callback(
11     Output(component_id="user-output", component_property="children"),
12     Input(component_id="user-input", component_property="value")
13 )
14 def update_page_title(new_title: str) -> str:
15     return new_title
```

Quelltext 2.2: DashCallback.py

# 3 App

## 3.1 Architektur

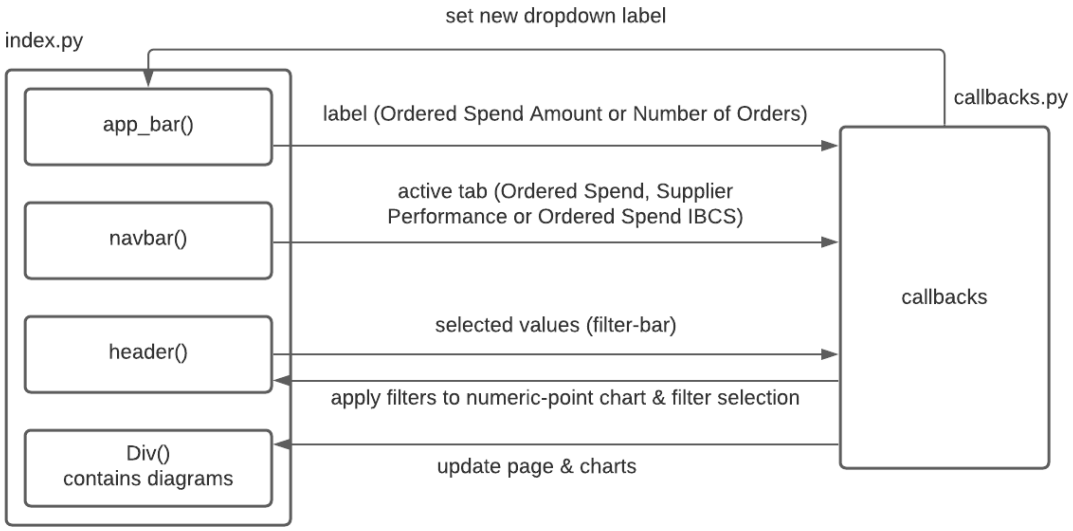


Abbildung 3.1: Gliederung von index.py

Der Startpunkt der Anwendung ist die Datei index.py. Dort wird das Layout sowie die Struktur der verschiedenen Komponenten festgelegt. app_bar(), navbar() und header() enthalten Komponenten mit auswählbaren Werten, die bei jeder Wertänderung Callback-Funktionen aufrufen. Die Callback-Funktionen registrieren die Wertänderungen und passen die Seiten sowie die Charts entsprechend der Eingaben an. Zusätzlich werden nur Filterungskombinationen zurückgegeben, für die auch Daten existieren und somit Charts angezeigt werden können.

Im Ordner *Charts* befinden sich die Dateien mit den Funktionen, die die Charts konstruieren. Die Callback-Funktionen geben die ausgewählten Werte als Parameter weiter an die Charts-Funktionen, die die Charts an die Eingaben anpassen und zurückgeben.

Neben den Chart-Funktionen existiert noch der Store. Dieser ist auf der Oberfläche der Anwendung nicht sichtbar und dient ausschließlich der Speicherung von benutzerdefinierten

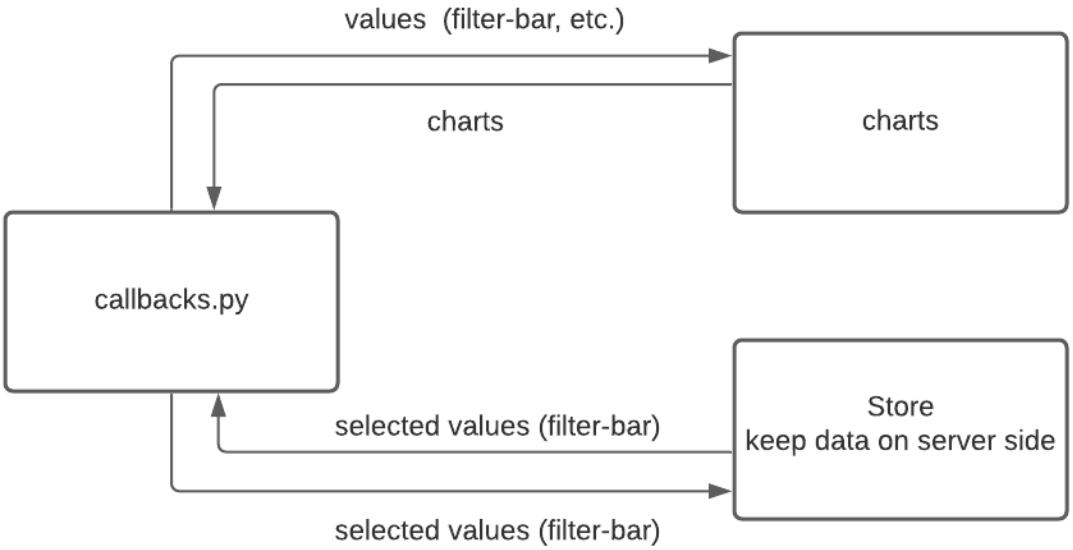


Abbildung 3.2: Zusammenhang zwischen Callbacks, Charts und Store

Filtern. Diese Filter sind durch den Store in der Anwendung global verfügbar und leicht zugänglich.

Die Ordnerstruktur gliedert sich in fünf Bereiche. Im Folgenden wird der Zweck sowie die Funktion jedes einzelnen Ordners dargelegt:

- **assets:** Enthält alle Styling-Dateien sowie Logos und Bilder.
- **charts:** Enthält alle Funktionen zur Generierung der Charts. Jede Datei enthält Charts, die zu einem Tab korrespondieren.
- **components:** Enthält Seitenlayouts als modulare Komponenten, die wiederverwendet werden können.
- **pages:** Enthält die Struktur der Seite sowie die Containers, in denen die Charts platziert werden.
- **utils:** Enthält Callbacks und Funktionen zur Data Preparation und Formatierung der in Charts angezeigten Zahlenwerte.

## 3.2 Design

Für die Anwendung wurden die SAP Fiori Guidelines, die im Jahr 2013 von der SAP SE veröffentlich wurden, verwendet. Ziel und Fokus von Fiori waren die Harmonisierung aller SAP-Produkte sowie die Vereinheitlichung der Nutzeroberfläche. Darüber hinaus gewährt Fiori ein responsives Design mit einem geräteübergreifenden Nutzererlebnis.
Das Konzept basiert auf fünf Kernprinzipien:

- **Rollenbasiert:** Die Anwendung passt sich an den Nutzern an, sodass nur die relevanten Informationen angezeigt werden.
- **Adaptiv:** Fiori lässt sich auf allen stationären und mobilen Endgeräten bedienen. Dabei wird immer ein einheitliches Design gewährt.
- **Einfach:** Durch das intuitive Design bleibt die Aufmerksamkeit auf relevante Informationen und zentrale Aufgaben gerichtet.
- **Kohärent:** Fiori deckt alle Anwendungsfälle ab, sodass es überall verwendet werden kann.
- **Personalisiert:** Der Nutzer kann die Fiori-Apps nach seinen spezifischen Bedürfnissen individualisieren. Dadurch verbessert sich die Arbeitserfahrung und die Produktivität wird erhöht.

Des Weiteren wurde das 1-1-3 Prinzip eingeführt. Dies ist der architektonische Ansatz, dass jede Anwendung einen speziellen Business Use Case für genau eine spezifische Benutzergruppe mit Hilfe von maximal drei Navigationsschritten abbilden soll (Vgl. „SAP Fiori Design Guidelines“, 31/08/2021 and „SAP Fiori“, 21/06/2021).

## 3.3 Funktionalität

### 3.3.1 Data Preparation und Filterung

Wie in Kapitel 1.2 bereits erwähnt, liegen die Daten in einer Excel Datei vor und werden aus dieser mithilfe der Bibliothek Pandas eingelesen, die im weiteren Vorgehen auch zur Data Preparation verwendet wird. Zunächst werden zur leichteren und konsistenteren Bearbeitung sowie auch zur späteren Anzeige in der GUI die Spaltennamen so bearbeitet, dass sie eine einheitliche Schreibweise aufweisen.

In den vorliegenden Daten müssen zunächst noch einige Werte ergänzt werden, so ergeben sich beispielsweise die Daten für Year und Month der Bestellungen aus dem Document Date und können später zur Vereinfachung einiger Abfragen und der dazugehörigen graphischen Darstellungen verwendet werden. Die Delivery Deviation in Tagen ergibt sich aus der Differenz zwischen Supplier Date und Delivery Date. Aus dieser kann dann der Deviation Indicator, der auf der Seite zur Supplier Performance dargestellt werden soll, bestimmt werden. Wenn die Abweichung in Tagen größer 0 ist, wird sie in verschiedene Kategorien der Verspätung weniger als fünf Tage, zwischen fünf und zehn Tagen und mehr als zehn Tagen eingeteilt.

Um in den Charts aggregierte Daten für beispielsweise Monate, Einkaufsorganisationen oder Lieferanten darstellen zu können, kann die Pandas-Funktion groupby verwendet werden, die in ihrer Funktionsweise der aus SQL bekannten GROUPBY-Anweisung gleicht und somit die Werte numerischer Spalten durch bestimmte Aggregatsfunktionen zusammenfassen kann (Vgl. Navlani et al., 2021, Section 1). Dabei wird Ordered Spend durch Aufsummierung der Net Value-Werte der einzelnen Bestellungen mithilfe der sum-Funktion ermittelt, während für die Anzahl der Bestellungen nur die vorhandenen einzigartigen Werte der Spalte Purchasing Doc. mithilfe der nunique-Option der Aggregationen ermittelt werden.

Aus Gründen der Einfachheit und Peformance der Callbacks werden aus den eingelesenen Daten separate DataFrames für einzelne oder Gruppen sich inhaltlich überschneidender Charts erstellt, von denen bei Bedarf Kopien erstellt werden, die gefiltert und gegebenenfalls weiter bearbeitet werden. Das Erstellen einer Kopie ist notwendig, um die ursprünglichen Daten beispielsweise bei Einschränkungen durch Filter nicht zu verändern. Um die in die GUI integrierten Filter anzuwenden, wird die loc-Funktion verwendet. Pandas-DataFrames liegen in tabellarischer Form vor und mit genannter Funktion kann der Zugriff auf bestimmte Zeilen und Spalten anhand einer Bedingung ermöglicht werden (Vgl. „pandas.DataFrame.loc — pandas 1.3.2 documentation“, 15/08/2021). So kann beispielsweise, wenn ein bestimmter Buchungskreis in der GUI ausgewählt wird, mittels loc auf Zeilen zugegriffen werden, bei denen die entsprechende Spalte Company Code mit der Auswahl übereinstimmt und somit der Filter angewandt werden.

Weiterhin ist es für Diagramme, in denen kategorisch unterteilte Daten mehrerer Jahre dargestellt werden, notwendig, die Reihenfolge der Kategorien mittels Pandas zu bestimmen, da die in Plotly integrierten Sortierungsfunktionen den Anforderungen nicht genügen. Ebenfalls werden die Top Ten Lieferanten für die entsprechenden Charts gemäß Ordered

Spend in 2020 mit Pandas bestimmt.

Zuletzt werden den DataFrames, die zur Darstellung von Balkendiagrammen noch Spalten zur Darstellung der Texte in den Charts hinzugefügt, in die Zahlen je nach Größenordnung mit Zusätzen k für tausend, M für Millionen, etc. versehen und den SAP-Fiori Guidelines folgend auf eine Nachkommastelle gerundet werden.

### 3.3.2 App Views und Pages

Das Dashboard erlaubt mithilfe einer Bootstrap-Dropdown-Liste in der Anwendungsleiste das Wechseln zwischen den folgenden zwei Ansichten:

- Ordered Spend Amount
- Number of Orders

Bei diesem Callback werden Trigger aus dem Paket dash-extensions verwendet, die einen Callback auslösen, ohne dabei den Wert als Eingabeargument zu übergeben (Vgl. PyPI, 31/08/2021). Besonders gut funktioniert der Trigger in Kombination mit dem *dash.callback_context*, da dieser keine Parameter benötigt, um den auslösenden Input zu bestimmen (Vgl. „Advanced Callbacks | Dash for Python Documentation | Plotly“, 31/08/2021).

In der darunterliegenden Navigationsleiste können Nutzer mit einer Bootstrap-Tabs-Komponente zwischen drei verschiedenen Seiten wechseln:

- Ordered Spend
- Supplier Performance
- Ordered Spend IBCS

Hierbei wird darauf geachtet, welcher Tab aktiv ist, um die benötigten Komponenten wie den Titel, die Numeric Point Charts und den Seiteninhalt per Rückruffunktion zu rendern.

### 3.3.3 Filter Bar

Die Filter Bar, die sich unterhalb der Navigationsleiste und des Titels der Seite befindet, ermöglicht es dem Nutzer, die Daten nach folgenden vier Kriterien zu filtern:

- Company Code
- Purchasing Organization
- Plant
- Material Group

Durch einen User Input werden die Diagramme der Ordered Spend und Supplier Performance Seite aktualisiert, wobei zunächst überprüft wird, welche Seite aktiv ist, um die Aktualisierung beider Seiten zu vermeiden. Zudem kann ein Benutzer mehrere Filter nach Belieben auswählen, um die Daten entsprechend zu visualisieren. Da nicht nicht zu jeder Kombination an Filtern auch Zeilen im DataFrame gibt, die diese Bedingungen erfüllen, sorgt eine Callback-Funktion dafür, dass die Optionen bei jedem Input des Users dynamisch angepasst werden. Sollte dennoch der Fall eintreten, dass keine Werte für die Plots vorhanden sind, beispielsweise durch weitere Einschränkungen in den Chart-Funktionen, greift Plotly auf einen vordefinierten leeren Graph zurück.

### 3.3.4 Charts

Die Anwendung verfügt über drei Tabs: Ordered Spend, Supplier Performance und Ordered Spend IBCS. Für jeden Tab existieren drei Charts, das entweder ein Line-Chart oder ein Bar-Chart ist. Das erste Chart in Ordered Spend ist ein Line-Chart, das die Ausgabensumme zwischen dem Jahr 2020 und dem Jahr 2019 im Laufe des Jahres darstellt. Das Line-Chart eignet sich hier besonders gut, da die Differenzen zum Vorjahr auf der Zeitfolge direkt sichtbar sind. Zusätzlich ist die Farbe für das Vorjahr in neutralem Grau gehalten, sodass die Werte des aktuellen Jahres, die in einer Primärfarbe dargestellt sind, hervorgehoben werden.
Das nächste Chart visualisiert die Ausgaben nach Einkaufsorganisationen. Der horizontale Barchart wurde ausgewählt, weil der Nutzer so die Summen vom aktuellen Jahr direkt mit denen vom Vorjahr vergleichen kann. Des Weiteren sind die Balken nach dem Wert des aktuellen Jahrs sortiert. Dies hebt die Werte für dieses Jahr hervor. Das letzte Chart ähnelt dem zweiten Chart mit dem Unterschied, dass es sich um die zehn Toplieferanten

handelt. Bei Ordered Spend IBCS sind die Charts gleich angeordnet, wobei für IBCS keine Farben aus den Fiori-Guidelines, sondern Schwarz- und Graustufen verwendet werden. Die Farbpalette ist in der config.py verfügbar.
Bei der Supplier Performance sind die drei Charts, die bei Ordered Spend verwendet wurden, ebenfalls zu finden, allerdings werden nur Daten für das aktuelle Jahr angezeigt, aber sowohl in die Balken- als auch die Line-Charts wurde eine Stapelung nach Gründen für die Abweichung der Orders (Deviation Cause) integriert. Dabei wird eine erweiterte Farbpalette verwendet und sichergestellt, dass aus Gründen der Konsistenz und besseren Übersichtlichkeit der gleiche Deviation Cause in allen Charts in der gleichen Farbe dargestellt wird. Zusätzlich gibt es auf der Supplier Performance-Seite auch noch zwei Bar-Charts, in denen Ordered Spend oder Number of Orders nach Deviation Cause und dem Deviation Indicator, der Auskunft über den Grad der Abweichung gibt, ablesbar sind. Auch in diesem Graphen sind die Deviation Causes farblich konsistent dargestellt.

# 4 Zusammenfassung

## 4.1 Fazit

Zusammenfassend lässt sich festhalten, dass die Verwendung von Plotly für die Entwicklung des Dashboards sehr gut geeignet ist. Das Dashboard erfüllt alle Aufgabenbedingungen und beinhaltet alle Charts, die auf die Filterungen des Nutzers responsive aktualisieren. Darüber hinaus wurde bei der Entwicklung die Fiori-Guideline strikt befolgt, sodass die Anwendung ein angenehmes intuitives Nutzererlebnis bietet. Ergänzend dazu ist die Anwendung responsive und lässt sich auf allen Geräten bedienen. Darüber hinaus hat die Anwendung eine kurze Ladedauer, die durch die Nutzung von Flask-Caching ermöglicht wurde.
Neben den vorgelegten Aufgabenstellungen, beinhaltet das Dashboard weitere Details, die die Nutzererfahrung verbessert. So gibt es ein animiertes Icon beim Laden und einen Warntext, wenn keine Charts für die nutzerspezifische Filterung existieren. Dies verbessert das Nutzungserlebnis, da üblicherweise nur ein leeres Chart zurückgegeben wird, was der Nutzer mit dem Laden der Charts verwechseln kann. Darüber hinaus kleben die Filter-bar und die App-bar beim Scrollen, sodass der Nutzer beim Auswählen der Filterung bei unteren Charts nicht nach oben scrollen muss. Darüber hinaus beinhaltet die Anwendung einen Tab mit IBCS-konformen Charts.

## 4.2 Ausblick

In Zukunft kann die Anwendung durch weitere Funktionalitäten ergänzt werden. Eine mögliche Funktionalität wäre eine Data Pipeline aufzubauen, sodass neue Daten automatisch zur Anwendung hinzugefügt werden. Auch ist es denkbar, die Anwendung auf bessere Anpasstbarkeit auf mobilen Geräten zu optimieren. Darüber hinaus lässt sich die Anwendung durch KI ergänzen, sodass anhand der Daten Handlungsempfehlungen herausgegeben werden können.

Sofern die Anwendung kommerziell genutzt wird, gilt es, CI in den Workflow einzubinden, um die hohe Softwarequalität zu sichern.

Sollte noch weiterer Informationsbedarf Ihrerseits bestehen, so ist ein Blick in den Python-Code empfehlenswert. Dieser ist vollständig dokumentiert und gibt daher Auskunft über die Funktionalität und den Zweck jeder einzelnen Funktion.

# Literaturverzeichnis


Advanced Callbacks | Dash for Python Documentation | Plotly. (31/08/2021). Verfügbar unter https://dash.plotly.com/advanced-callbacks

Ardalis. (30/08/2021). Übersicht über ASP.NET Core MVC. Verfügbar unter https://docs.microsoft.com/de-de/aspnet/core/mvc/overview?WT.mc_id=dotnet-35129-website&view=aspnetcore-5.0

Basic Callbacks | Dash for Python Documentation | Plotly. (30/08/2021). Verfügbar unter https://dash.plotly.com/basic-callbacks

Dabbas, E. (2021). *Interactive Dashboards and Data Apps with Plotly and Dash: Harness the power of a fully fledged frontend web framework in Python – no JavaScript required / Elias Dabbas*. Packt Publishing.

Eygi, C. (17/03/2020). JavaScript Callback Functions – What are Callbacks in JS and How to Use Them. *freeCodeCamp.org*. Verfügbar unter https://www.freecodecamp.org/news/javascript-callback-functions-what-are-callbacks-in-js-and-how-to-use-them/

Fayock, C. (18/06/2020). Front End Developer vs Back End Developer – Definition and Meaning In Practice. *freeCodeCamp.org*. Verfügbar unter https://www.freecodecamp.org/news/front-end-developer-vs-back-end-developer-definition-and-meaning-in-practice/

Kusleika, D. (2021). *Data visualization with Excel dashboards and reports* (1st). John Wiley & Sons.

Navlani, A., Fandango, A. & Idris, I. (2021). *Python Data Analysis: Perform data collection, data processing, wrangling, visualization, and model building using Python / Avinash Navlani, Armando Fandango, Ivan Idris* (Third edition). Packt Publishing.

pandas.DataFrame.loc — pandas 1.3.2 documentation. (15/08/2021). Verfügbar unter https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.loc.html

PyPI. (31/08/2021). dash-extensions. Verfügbar unter https://pypi.org/project/dash-extensions/

SAP Fiori. (21/06/2021). *Mission Mobile*. Verfügbar unter https://mission-mobile.de/knowhow/sap-fiori/#:~:text=%C3%97-,Vorteile%20von%20Fiori,Arbeitserfahrung%20mit%20der%20entwickelten%20Technologie.

SAP Fiori Design Guidelines. (30/08/2017). *mindsquare*. Verfügbar unter https://mindsquare.de/knowhow/sap-fiori-design-guidelines/

SAP Fiori Design Guidelines. (31/08/2021). Verfügbar unter https://experience.sap.com/fiori-design-web/

What's New In Python 3.9 — Python 3.9.6 documentation. (24/08/2021). Verfügbar unter https://docs.python.org/3/whatsnew/3.9.html